



istruzioni e
comandi per

BASIC
ASSEMBLER
più **GRAFICA**

SOMMARIO

PREFAZIONE	Pag. 2	RESUME (P)	» 40
SINTASSI DI UN'ISTRUZIONE	» 4	ERROR (P)	» 40
LOAD (D/P)	» 4	ERR (D/P)	» 41
SAVE (D/P)	» 6	ERL (D/P)	» 41
KILL (D/P)	» 6	LINEE DA 1000 A 1020	» 41
ESEMPIO DI PROGRAMMA: PITAGORA	» 7	LINEE 1030-1040	» 42
LINEA 1000	» 8	LINEE DA 1050 A 1080	» 42
CLS (D/P)	» 9	BASIC	» 43
LINEA 1010	» 9	OPEN (D/P)	» 44
PRINT (D/P)	» 9	CLOSE (D/P)	» 46
LINEA 1020	» 11	PRINT # (D/P)	» 46
RND (S/P)	» 11	INPUT # (D/P)	» 47
LINEE 1030-1040	» 12	LINEE DA 1090 A 1170	» 50
PRINT (D/P)	» 12	LINEE DA 1180 A 1440	» 51
LINEA 1050	» 12	GOSUB E RETURN (P)	» 51
LINEE 1060-1070-1080	» 12	LINEE DA 1450 A 1590 E DA 2230 A 2260 ...	» 52
INPUT (P)	» 13	LINEE DA 1600 A 1690	» 52
LINEA 1090	» 13	LINEE DA 1700 A 1770	» 52
LINEA 1100	» 13	LINEE DA 1780 A 1860	» 52
IF... THEN... ELSE (D/P)	» 14	LINEE DA 1870 A 1990	» 53
LINEA 1110	» 14	LPRINT (D/P)	» 53
FOT... TO... STEP... NEXT (D/P)	» 14	PRINTTAB E LPRINTTAB (D/P)	» 53
GOTO (D/P)	» 16	LINEE DA 2000 A 2190	» 54
LINEA 1130	» 17	LINEE DA 2200 A 2220	» 54
LIST (D/P)	» 17	NEW (D/P)	» 55
LLIST (D/P)	» 18	MEM E FRE (D/P)	» 55
AUTO (D/P)	» 18	CLEAR (D/P)	» 56
EDIT (D)	» 19	STOP E CONT (D/P)	» 56
RENUM (D)	» 19	LIET (D/P)	» 56
DELETE (D)	» 23	UN GIOCO ROMPICAPO: MASTERMIND	» 57
SECONDA VERSIONE DEL PROGRAMMA DI PITAGORA	» 25	STRING\$ (D/P)	» 61
DIM (D/P)	» 26	CHR\$ E ASC (D/P)	» 61
LINEE 1000-1010	» 27	LEFT\$ E RIGHT\$ (D/P)	» 63
LINEA 1020	» 28	MIDS (D/P)	» 63
LINEA 1030	» 28	STR\$ (D/P)	» 64
RANDOM (D/P)	» 28	LEN (D/P)	» 64
LINEE 1040-1050-1060	» 28	INSTR (D/P)	» 64
LINEA 1070	» 29	LINEE DA 1000 A 1120	» 65
REVOFF E REVON	» 29	LINEE DA 1130 A 1300	» 65
LINEE DA 1080 A 1160	» 30	LINEE DA 1310 A 1460	» 65
INKEY\$ (P)	» 30	LINEE DA 1470 A 1610	» 66
LINEE 1170-1180	» 30	LINEE DA 1620 A 1730	» 67
LINEA 1190	» 31	ON... GOTO (P)	» 67
VAL (D/P)	» 31	LINEE DA 1740 A 2050	» 68
LINEA 1200	» 31	CONFIGURAZIONI HARDWARE	» 70
LINEA 1210	» 32	CONTABILITÀ SEMPLIFICATA	» 73
LINEA INPUT (P)	» 32	LISTATO ESATTO DEL PROGRAMMA GESTIONE AZIENDALE	» 90
LINEA 1220	» 32	RIFERIMENTI DEL PROGRAMMA GESTIONE AZIENDALE	» 94
LINEA 1230	» 32	STARTREK VIDEOGAME PER COMPUTER Z20	» 96
LINEA 1240	» 33	LA COMUNICAZIONE FRA IL COMPUTER ED IL MONDO ESTERNO	» 99
LINEE 1250-1260	» 33	NOTE AGGIUNTIVE SULLA SCHEDA VIDEO GRAFICA LX.529	» 122
INT (D/P)	» 33	FUNZIONI GRAFICHE DIRETTE	» 123
LINEE DA 1270 A 1340	» 34	IL CIRCUITO ELETTRICO DELLA SCHEDA LX.529	» 126
LINEE DA 1350 A 1450	» 34	LEZIONI BASIC su FLOPPY	» 136
END (P)	» 34		
TERZA VERSIONE DEL PROGRAMMA DI PITAGORA	» 35		
REM (P)	» 39		
ONERRORGOTO (P)	» 40		

PREFAZIONE

Fino ad ora abbiamo visto, in modo molto particolareggiato, tutti i comandi del DOS, mentre del BASIC abbiamo dato, voce per voce, una descrizione succinta.

È stato poi affrontato, con l'ausilio del programma GESTIONE, l'argomento della gestione dei files su disco.

In questo modo coloro che hanno una discreta conoscenza del BASIC hanno potuto iniziare subito a stendere programmi.

Sappiamo però che molti non hanno una buona padronanza del BASIC, o magari sono alle primissime esperienze di programmazione; per costoro quanto detto finora sul BASIC non è certo sufficiente.

Al solito, però, noi intendiamo essere di aiuto proprio a chi ha un bagaglio di nozioni più ristretto, e quindi abbiamo pensato di spiegare in dettaglio tutte le voci del BASIC.

Per fare questo, potremmo limitarci a spiegarle una per una, più diffusamente di quello che è già stato fatto, e potremmo considerare esaurito il nostro compito.

Siamo però convinti che una procedura del genere non aiuterebbe molto coloro che sono alle prime armi, in quanto ogni descrizione teorica, per quanto chiara e dettagliata, resta pur sempre separata dalla pratica. La programmazione in BASIC (come in qualsiasi altro linguaggio) può essere assimilata solo facendo programmi concreti e soprattutto completi. In tal modo, infatti, si impara ad analizzare il problema che si deve risolvere e si capisce, poco per volta, come vanno affrontate le varie fasi della programmazione.

Per tutti questi motivi abbiamo deciso di seguire una strada più lunga e certamente più impegnativa: pubblicheremo volta per volta dei programmi completi e funzionanti, e via via che incontreremo nuove istruzioni del BASIC le spiegheremo in dettaglio. In questa maniera non saranno più puramente teoriche, ma potrete vederle applicate nell'esempio proposto. Molti programmi (specialmente all'inizio) saranno presentati in varie versioni, partendo da una soluzione minima del problema e sviluppandola successivamente fino ad arrivare alla configurazione finale.

Di tutti i programmi proposti forniremo il diagramma di flusso, che è molto utile, specialmente ai meno esperti, per capirne il funzionamento.

Di ogni parola chiave del BASIC daremo poi la SINTASSI (detta anche FORMATO), ossia indicheremo con uno schema grafico il modo in cui può essere impartita la funzione associata alla parola chiave esaminata.



SINTASSI DI UN'ISTRUZIONE

Prima di tutto ricordiamo che le istruzioni BASIC sono di due tipi: quelle che possono essere date solo direttamente dalla tastiera, e quelle che vanno invece inserite nelle linee dei programmi. In effetti non esiste una distinzione così netta, e succede molto spesso che istruzioni dirette possano essere inserite anche in un programma e, inversamente, che istruzioni concepite per essere usate nei programmi possano essere impartite dalla tastiera.

Entrambe le cose possono essere utili, e specialmente in fase di DEBUG la seconda possibilità si fa molto apprezzare.

Per inciso, ricordiamo che DEBUG e DEBUGGING sono termini di derivazione inglese: BUG, in quella lingua, significa PULCE, quindi DEBUG e DEBUGGING, letteralmente, vogliono dire SPULCIARE, SPULCIAMENTO. Sono universalmente usate per indicare l'operazione di ricerca e annullamento degli errori da un programma che non gira come dovrebbe. I programmatori bravi e poco presuntuosi hanno un motto: «There is always one more bug», vale a dire «C'è sempre qualche pulce». Sappiatevi quindi regolare!

Ogni istruzione BASIC, sia essa diretta o da programma, deve essere impartita in un certo modo, altrimenti il computer non la riconosce e segnala errore. Occorre quindi rispettare, per ogni parola chiave, un certo modo di operare ben stabilito, ossia bisogna scrivere le istruzioni rispettando una certa SINTASSI.

Vedremo tra poco i primi esempi; per ora ci limitiamo a introdurre i vari simboli che useremo; sono molto pochi, e sono ormai entrati nell'uso comune. Non crediate che ci divertiamo ad introdurre continuamente nuovi concetti per scoraggiarvi e confondervi le idee; è vero esattamente il contrario. Anche in questo caso il tempo che impiegherete per capire il meccanismo dei grafici di sintassi si tradurrà, dopo pochi esempi, in un ausilio pressoché indispensabile. I simboli usati, tre in tutto, sono nati per essere di comprensione immediata e dopo un po' di pratica vi basterà un'occhiata veloce per capire immediatamente come va impartita un'istruzione, molto meglio che leggendo parecchie righe di spiegazione. Noi vi daremo entrambe le cose, e vi accorgerete, andando avanti, che tutto risulterà di chiara comprensione anche per chi è alle prime armi. Pure i più esperti farebbero bene a seguirci passo passo: c'è sempre qualcosa da imparare, anche dagli esercizi più facili.

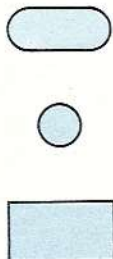


fig. 13 Questi sono i tre simboli usati nei diagrammi di sintassi. All'interno del primo compaiono le istruzioni di volta in volta trattate; il secondo simbolo, quello del cerchio, viene usato per indicare i caratteri che devono necessariamente accompagnare l'istruzione descritta. Il terzo simbolo, quello del rettangolo, conterrà al suo interno i vari elementi via via necessari (una variabile numerica, una stringa, un numero di linea, ecc.).

I tre simboli che useremo per i grafici di sintassi sono rappresentati in FIGURA 13. Il primo è lo stesso che si usa all'inizio e alla fine dei diagrammi di flusso, e al suo interno scriveremo sempre l'istruzione in oggetto. Il secondo simbolo, quello del cerchio, viene usato per indicare i caratteri obbligatori che devono accompagnare l'istruzione: se questi caratteri non vengono messi, il computer va in errore, a meno che essi non siano facoltativi; questa possibilità viene indicata, nel diagramma, con una linea che bypassa il cerchio.

Il terzo ed ultimo simbolo è quello del rettangolo; al suo interno vanno scritti i vari elementi che volta per volta sono necessari: una variabile numerica, oppure un numero o una stringa. Anche in questo caso valgono considerazioni analoghe alle precedenti: se qualche indicazione può essere omessa, ci sarà una linea che evita il passaggio per quel simbolo.

Facciamo subito un esempio pratico.

LOAD (D/P)

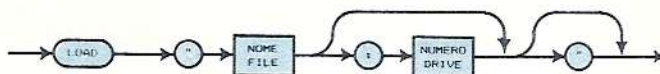
Innanzitutto vediamo il significato delle lettere D e P contenute nella parentesi: esse sono rispettivamente le abbreviazioni di DIRETTO e PROGRAMMA. Per ogni parola chiave del BASIC metteremo le possibilità d'uso: le lettere D e P usate da sole indicano che l'istruzione in esame può essere impiegata solo nel modo indicato; la presenza contemporanea delle due lettere specifica che esiste la possibilità di usare quell'istruzione in entrambi i modi.

Nel caso presente si può quindi inserire LOAD in un programma, anche se non ha molto senso. Supponendo infatti che una linea di programmazione contenga l'istruzione LOAD, arrivato a quel punto il computer va a caricare in memoria il programma col nome indicato; a processo terminato, si torna allo

stato comandi, ossia con la scritta READY. Da osservare poi che in tal caso il programma precedente viene perso, quindi l'eventuale parte di esso successiva a LOAD non verrà mai eseguita.

La sintassi dell'istruzione LOAD è in FIGURA 14.

fig. 14 sintassi di LOAD



Essa va interpretata nel seguente modo: l'istruzione in esame è LOAD, dal momento che nell'ellisse (lasciatecelo chiamare così, anche se non è esatto) c'è appunto la parola LOAD. Seguendo la linea del percorso, si deve passare per il primo cerchio, che contiene il carattere virgolette: ciò significa che la parola chiave LOAD deve essere necessariamente seguita da quel carattere, altrimenti si ha una segnalazione d'errore.

Successivamente si incontra un rettangolo con la scritta NOME FILE al suo interno: vuol dire che dopo aver messo le virgolette bisogna dare il nome del programma che si vuole caricare in memoria.

Procedendo, si incontra dapprima il simbolo del cerchio all'interno del quale c'è il carattere dei due punti, poi un rettangolo con la scritta NUMERO DRIVE; ciò significa che per indicare al computer il drive su cui deve cercare il programma specificato, occorre mettere due punti e poi il numero del drive. Avrete notato che questi ultimi due simboli possono essere bypassati, seguendo la deviazione verso l'alto prima del cerchio contenente i due punti. Ciò significa che quella parte di sintassi è facoltativa: se viene omessa, il comando è valido e viene accettato regolarmente.

Infine troviamo un altro cerchio contenente le virgolette; anche questa indicazione può essere omessa, dal momento che prima di essa esiste una diramazione che la evita.

Se provate ora a riguardare la fig. 14, vi rendete conto che con una sola occhiata si può capire la sintassi di LOAD; si vede, ad esempio, che il modo più conciso per caricare in memoria il programma di nome TRENO è il seguente:

LOAD"TRENO

Dal diagramma di sintassi si capisce che un altro modo ammesso, anche se più lungo del precedente, è questo:

LOAD"TRENO"

Altre due possibilità sarebbero le seguenti:

LOAD"TRENO:1

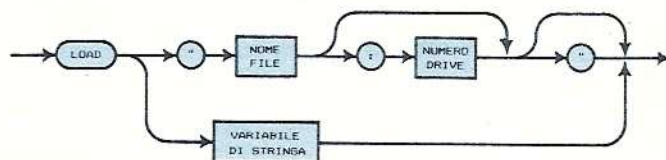
LOAD"TRENO:1"

Esiste però anche un altro modo di assegnare il nome del file, ossia con una variabile di stringa. Nell'esempio precedente si potrebbe anche fare così:

A\$="TRENO":LOAD A\$

Cio, in questo esempio specifico, non serve a molto, ma nel caso del comando RUN, che vedremo tra poco, può risultare utile in parecchie occasioni.

fig. 15 sintassi di LOAD



È meglio allora modificare il diagramma di sintassi di LOAD nel modo indicato in FIGURA 15: è stato aggiunto un ramo inferiore, con una variabile di stringa al posto delle virgolette e di tutto quello che segue. Da notare che scrivendo:

LOAD A\$:1

Il computer va a caricare il programma di nome A\$, iniziandone però la ricerca nel drive zero; se non lo trova e se esiste più di un drive, esso prosegue la ricerca nei drive successivi.

Questo comportamento è evidente dal diagramma di sintassi: l'alternativa inferiore della variabile di stringa rientra nel diagramma precedente solo alla fine, saltando così ogni altra possibilità.

Ancora qualche osservazione importante: gli spazi, nel linguaggio BASIC, non sono generalmente importanti. Ad esempio, le due scritture seguenti si equivalgono:

```
LOAD "PITAGORA"  
LOAD"PITAGORA"
```

Perciò non parleremo mai di spazi, tranne che quando metterli oppure no comporti delle differenze.

Nel caso della sintassi di LOAD c'è da dire che mettendo uno spazio prima dei due punti l'elaboratore si comporta come se non ci fosse nulla dopo il nome del programma: la ricerca inizia sempre dal drive zero. Quindi per specificare in quale drive si trova il programma, non bisogna mettere nessuno spazio prima dei due punti. Se si mette uno spazio dopo i due punti si ha una segnalazione d'errore.

Inoltre, il nome del file può essere scritto sia in caratteri minuscoli che maiuscoli; per il computer non fa differenza.

Per finire, occorre parlare di VALORE DI DEFAULT. Riferiamoci ancora alla figura 14, e precisamente della prima deviazione, quella che esclude i due punti ed il numero del drive. Ebbene, se scegliamo quella deviazione, ossia se non mettiamo quei dati, il computer si comporta come se noi avessimo messo i due punti ed il numero zero; si dice, in questo caso, che il numero del drive, in caso di default, vale zero. La dicitura è chiara pensando che DEFAULT, in inglese, significa MANCANZA. Quindi in molte occasioni, come vedremo andando avanti, il calcolatore assume automaticamente certi valori qualora non gli sia stato ordinato diversamente; quelli sono i valori di default.

Prima di proseguire con un esempio di programmazione, esaminiamo altre parole chiave che sono di sintassi uguale a LOAD: SAVE, KILL, RUN.

SAVE (D/P)

Questo comando serve per registrare su disco il programma esistente in memoria. Il diagramma di sintassi di SAVE è rappresentato in FIGURA 16, che è molto simile alla FIGURA 15; valgono pertanto tutte le considerazioni fatte per LOAD.

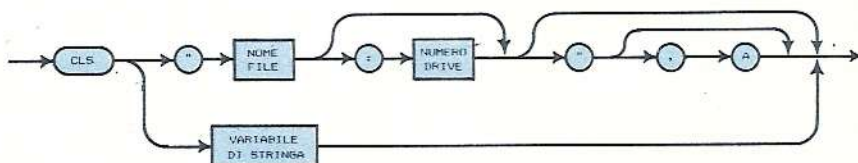


fig. 16 diagramma di sintassi di SAVE

L'unica differenza è nell'opzione finale (A) che serve per registrare un programma in formato ASCII anziché compactato; parleremo di questo a suo tempo.

Resta da aggiungere una cosa, tra l'altro molto importante: se sul disco esiste già un file che ha lo stesso nome di quello messo nel comando SAVE, il calcolatore registra il programma in memoria sul file omonimo già presente. Quindi occorre prestare molta attenzione a questo comportamento, perché può capitare di cancellare involontariamente un programma da un disco, andando a registrarci sopra uno diverso come contenuto ma uguale nel nome. È buona norma, prima di impartire un comando SAVE, sincerarsi che il disco non contenga di già un file col nome che intendiamo dare; tale verifica si fa col comando DIR (appartenente al DOS - vedere la voce relativa). Cogliamo l'occasione per raccomandarvi ancora di fare sempre una copia dei vostri dischi: verrà immancabilmente l'occasione in cui rimpiangerete di non averlo fatto.

Può capitare di impartire il comando SAVE e di avere in linea solo dei dischi che portano il nastro di protezione contro le registrazioni; le possibilità sono due: se esiste già un file con quel nome, il computer manda una segnalazione d'errore e poi torna a livello comandi; se invece non esiste un file con quel nome, il calcolatore perde il controllo e bisogna fare un RESET. In tal modo, ovviamente, il programma presente in memoria centrale va perso; occorre quindi prestare molta attenzione anche a questa evenienza.

KILL (D/P)

Il comando KILL serve per cancellare un file da un disco; per file, in questo caso, si deve intendere qualsiasi tipo di file, e non solo un programma. Si possono cancellare quindi anche file di archivio, sia

sequenziali che random. Il diagramma di sintassi di KILL è riportato in FIGURA 17, ed è del tutto analogo ai precedenti.

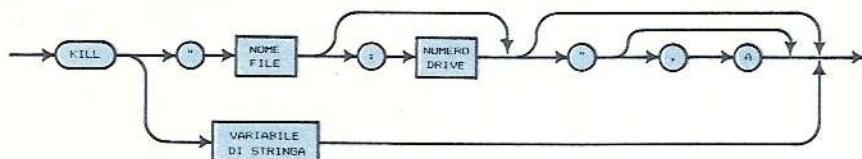


fig. 17 diagramma di sintassi di KILL

Dal momento che si tratta di un'operazione di registrazione, per KILL valgono le stesse considerazioni già fatte per SAVE a proposito del nastro di protezione contro le registrazioni.

RUN (D/P)

RUN è il comando che serve per eseguire il programma presente in memoria: con RUN si "lancia" il programma, come si suol dire.

Il suo diagramma di sintassi è in FIGURA 18.

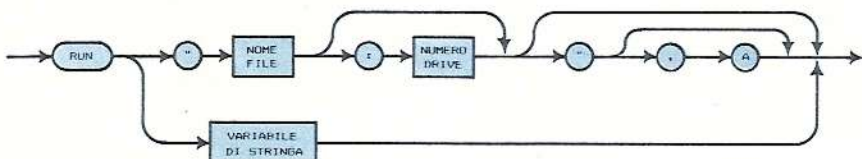


fig. 18 diagramma di sintassi di RUN

Mentre nei casi precedenti la possibilità di usare i comandi anche all'interno di un programma non aveva molto senso, ora il discorso è diverso.

Infatti il comando RUN svolge anche un'altra funzione importantissima, che è quella di concatenare un programma con un altro. Se una linea di programma fosse, ad esempio, la seguente:

1380 RUN "SOFT3"

accadrebbe che il computer, arrivato alla linea 1380, va a cercare su disco il programma di nome SOFT3 e lo manda in esecuzione.

Si capisce quindi l'enorme importanza di poter inserire RUN in un programma; basti pensare che in tal modo diventa possibile elaborare programmi molto lunghi anche avendo a disposizione poca memoria RAM. Infatti si spezzetta il programma originale in tante parti più piccole, ognuna delle quali può rientrare nella memoria centrale. L'unico limite a questo discorso è costituito dal fatto che IL COMANDO RUN AZZERA TUTTE LE VARIABILI NUMERICHE E DI STRINGA. Gli eventuali valori da trasmettere al programma concatenato dovranno essere ripetuti al suo inizio, se si tratta di costanti; se invece si devono trasmettere al secondo programma dei valori variabili perché elaborati dal primo, tali dati andranno registrati su disco in un file apposito che sarà letto all'inizio del programma concatenato.

A dirlo sembra difficile, ma vedremo che è abbastanza agevole realizzarlo in pratica.

Usando ed utilizzando variabili di stringa al posto delle virgolette e del nome, è anche possibile fare in modo che il programma si concateni in vari modi con altri diversi, a seconda del valore che la variabile di stringa avrà assunto.

ESEMPIO DI PROGRAMMA: PITAGORA

Proponiamoci ora di risolvere un problema pratico. Le scelte possibili sono sicuramente infinite e possono trattare gli argomenti più disparati, ma all'inizio vogliamo rimanere sul semplice, per poter essere seguiti facilmente.

Indubbiamente niente di più bello che istruirsi divertendosi, ed è per questo che vedremo spesso qualche giochetto. Le scelte che via via faremo potranno anche non essere condivise, ma non bisogna

dimenticare che il loro scopo principale è quello di fare da strumento didattico per introdurre, poco per volta e con esempi facili e completi, tutte le istruzioni del BASIC.

Cominciamo ... dalle elementari, facendo un programmino che serve a controllare se uno scolaro conosce la tavola di Pitagora: sì, proprio le tabelline!

Vedrete che anche su un argomento così banale (per chi le sa già!) si può costruire un programma abbastanza elaborato.

Siamo anche convinti che molti di voi avranno figli, nipoti o fratelli che frequentano le prime classi della scuola primaria; potranno così dare loro un buon aiuto realizzando questo programma. Se poi c'è tra i possessori del computer di NUOVA ELETTRONICA qualche maestro, allora tanto meglio: vogliamo dimostrargli che l'elaboratore elettronico ha già tutte le carte in regola per entrare anche nella scuola elementare, ad ogni livello. Si tratta, come al solito, di disporre dei programmi (software) più adatti.

Per partire col piede giusto dobbiamo fissare chiaramente gli obiettivi da raggiungere; proponiamoci allora di fare un programma per dei bambini di sette od otto anni, quindi molto facile come presentazione ed uso. Sul monitor dovrà comparire il prodotto di due numeri, ognuno dei quali deve essere compreso tra 1 e 10; il computer resta poi in attesa che il risultato venga introdotto dalla tastiera, e dovrà comparire il responso ESATTO oppure SBAGLIATO a seconda che la risposta sia corretta o meno. Il ciclo si deve ripetere all'infinito.

Si tratta di un problema molto semplice da risolvere, ma vedremo che può essere complicato successivamente, raffinandone il funzionamento.

Và da sé che le combinazioni di numeri presentate dal monitor devono essere svariate e non devono seguire alcun ordine prestabilito.

In FIGURA 19 c'è il listato di un piccolo programma (che chiameremo PITA1) che serve allo scopo e la FIGURA 20 fornisce il relativo diagramma di flusso.

```

1000 CLS
1010 ES=ES+1
1020 PRINT "ESERCIZIO #";ES
1030 A=RND(10)
1040 B=RND(10)
1050 PRINT @128,"QUANTO FA ";
1060 PRINT A;
1070 PRINT " PER ";
1080 PRINT B;
1090 INPUT R
1100 PRINT
1110 IF R=A*B THEN PRINT "ESATTO" ELSE PRINT "SBAGLIATO"
1120 FOR I=1 TO 600 : NEXT
1130 GOTO 1000

```

fig. 19 listato di PITA1

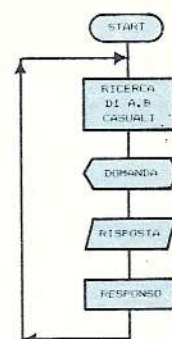


fig. 20 diagramma di flusso di PITA1

Cominciamo proprio ad esaminare quest'ultimo. Dopo lo START, ossia dopo aver lanciato il programma, il computer cerca due variabili numeriche comprese tra 1 e 10, che indicheremo con le lettere A-B. Finito questo, sul monitor appare la domanda: "QUANTO FA A PER B?", dove ovviamente al posto di A e di B compariranno i loro valori effettivi. Successivamente, battendo sulla tastiera il risultato, il programma deve controllare la risposta e dire se è giusta oppure no. Il passo successivo è un ritorno al primo rettangolo, ossia si riparte da capo, e così all'infinito.

Come vedete, l'uso del diagramma di flusso serve egregiamente a schematizzare il funzionamento di un programma.

LINEA 1000

Andiamo ora alla FIGURA 19, ossia al listato. La prima cosa da dire è che abbiamo iniziato dal numero di linea 1000 per motivi tipografici, in modo che le varie righe, essendo contraddistinte sempre da 4 cifre, risultino perfettamente allineate; ciò ne facilita la lettura. È buona norma, comunque, cominciare un programma con un numero di riga abbastanza alto: se è necessario inserire qualche nuova linea, il posto è già disponibile.

Per lo stesso motivo è bene lasciare posto nella numerazione delle linee; è comune l'uso di andare di 10 in 10, anche se nulla vieta di fare in modo diverso.

La prima istruzione che troviamo è CLS, quindi passiamo subito a spiegarla diffusamente. Ci comporteremo così man mano che incontreremo una nuova istruzione.

CLS (D/P)

CLS deriva dalle parole inglesi CLEAR SCREEN (cancella lo schermo) e serve appunto per cancellare il contenuto del monitor e posizionare il cursore, ossia il punto in cui verrà scritto il primo carattere da stampare, in alto a sinistra.

Tale istruzione può essere usata anche da comando diretto, come risulta dalla lettera D messa tra parentesi, ma per tale uso è molto più immediato premere il tasto DEL.

Il diagramma di sintassi di CLS è estremamente semplice e banale, come si vede in FIGURA 21.

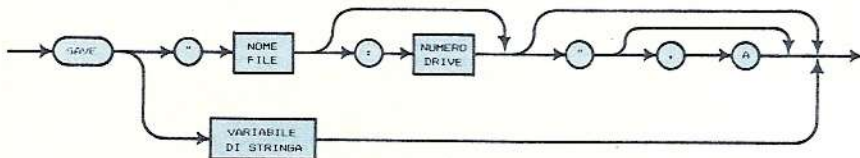


fig. 21 diagramma di sintassi di CLS

LINEA 1010

Riprendiamo allora la lettura del listato.

Alla linea 1010 troviamo $ES = ES + 1$, che in termini aritmetici è sbagliato: non esiste infatti un numero uguale a sé stesso più uno. Quell'espressione va intesa in questo modo: il valore della variabile ES è uguale al valore precedente di ES aumentato di uno. La 1010 è l'espressione di ASSEGNAZIONE del valore alla variabile numerica ES.

Il motivo per cui la riga 1010 è stata fatta così è molto semplice:

dato che ad ogni nuova domanda il programma riparte da capo (vedere ancora il diagramma di flusso nonché la linea 1130 del programma), la variabile ES definita in quel modo rappresenta un numero che tiene il conto dei passaggi; essa ci dà sempre il numero dell'esercizio in corso. Si dice che ES è un CONTATORE.

Ricordiamo che ogni identificatore di variabile numerica deve cominciare con una lettera e può essere lungo a piacere; solo i primi due caratteri, però, sono presi in considerazione dal calcolatore. Se avete qualche dubbio in proposito, andate a rilegervi le righe dove tale argomento è già stato trattato ampiamente.

Dal momento che alla partenza del programma tutte le variabili valgono zero, quando si percorre la linea 1010 per la prima volta si ha, in numeri: $ES = 0 + 1$ ossia ES contiene ora il numero 1; al secondo giro $ES = 1 + 1$, ossia 2, e così via.

Nella riga 1020 c'è la parola chiave PRINT; vediamola in dettaglio.

PRINT (D/P)

PRINT significa STAMPA ed ordina al computer di effettuare un'operazione di scrittura sul monitor. La sintassi di PRINT è riportata in FIGURA 22.

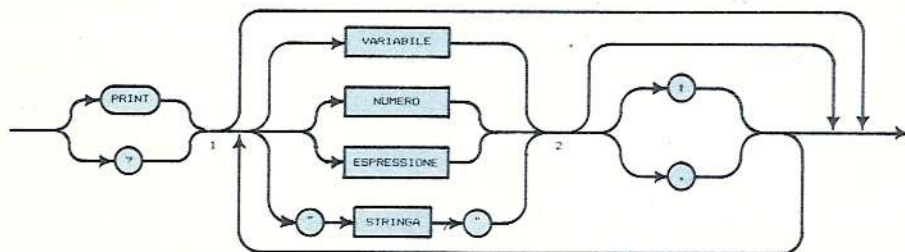


fig. 22 sintassi semplificata di PRINT

A dire il vero quel diagramma è molto semplificato, dal momento che PRINT può essere usato in parecchi modi; poco per volta li vedremo tutti. Per non complicare troppo le cose, limitiamoci a quello che ci serve nell'immediato.

Innanzitutto vediamo che PRINT può essere sostituito dal punto interrogativo; questo viene poi sempre trasformato dal sistema nella corrispondente parola chiave.

Successivamente troviamo un nodo (indicato col numero 1) da cui possiamo scegliere 5 vie diverse.

Seguendo il ramo superiore non si trova niente fino alla fine, dal che si deduce che PRINT può essere dato anche da solo; in tal caso serve per saltare una riga video. Un esempio di questo modo d'uso è nella linea 1100.

Se seguiamo invece il secondo ramo, troviamo un rettangolo con la scritta **VARIABILE**. Non essendo specificato nient'altro, si intende sia una variabile di tipo numerico che di stringa. Un'alternativa a questa possibilità è rappresentata dal terzo ramo: PRINT può essere seguita da un numero. Come quarta possibilità possiamo mettere un'espressione aritmetica, come somme, prodotti od elevamenti a potenza: usato in questo modo il computer si comporta da calcolatrice.

Come ultima possibilità, dopo PRINT possiamo mettere dapprima delle virgolette, poi la dicitura **STRINGA** e quindi ancora virgolette. Ciò significa che con l'istruzione PRINT seguita da una dicitura posta tra virgolette si ottiene la stampa su video della frase specificata.

Siamo così arrivati al nodo 2; qui abbiamo tre diverse alternative.

Se seguiamo quella in alto arriviamo direttamente alla fine: in questo caso una successiva istruzione di stampa avviene partendo dall'inizio della riga video seguente.

Se invece percorriamo la via di mezzo, mettiamo un punto e virgola: così facendo il computer sa che la successiva operazione di stampa deve iniziare dal punto di arrivo di questa. Percorrendo il terzo ramo, mettiamo una virgola anziché un punto e virgola: in tal caso la stampa successiva inizia dopo un certo intervallo.

Ecco alcuni esempi leciti di impiego di PRINT:

```
PRINT A$
PRINT 713
? 98*2.003
PRINT "NUOVA ELETTRONICA"
PRINT F$;" VENDUTO A ";C(NC)
? (A3-3.4*C)/(2*T(4)) -> 5.28
```

Vediamo di spiegare meglio l'ultima parte. Osservate la figura 23;

In essa è raffigurato il monitor, e le linee verticali A e C sono rispettivamente il limite sinistro e destro di scrittura: tra A e C sono compresi quindi 32 caratteri. La linea B è poi tracciata tra il sedicesimo ed il diciassettesimo carattere. In tal modo lo schermo risulta diviso in due bande verticali, larghe ognuna 16 caratteri.

Ebbene, con istruzioni di stampa separate da una virgola, il computer va a scrivere all'inizio di ogni fascia, ossia al primo e al diciassettesimo carattere di ogni riga; si può anche dire che la virgola messa dopo PRINT causa delle tabulazioni fisse all'inizio e a metà riga.

A livello di comandi diretti, ossia fuori da programma, scrivendo:

```
CLS : A=18 : B=-2963 : PRINT "INIZIO", A, B, C, D"FINE"
```

e premendo RETURN si ottiene sul monitor una scrittura come in FIGURA 24.

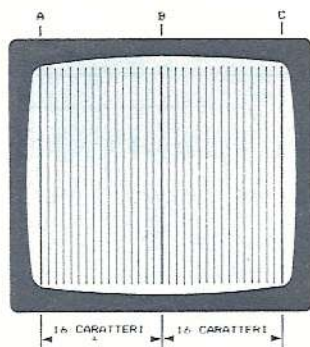


fig. 23 monitor

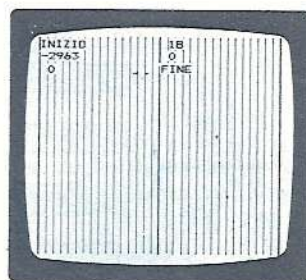


fig. 24 monitor

Innanzitutto si vede che, non essendo state inizializzate, le variabili C e D valgono zero. Si nota poi un non allineamento verticale tra le scritte ed i numeri; ciò è dovuto al fatto che ogni numero è preceduto sempre da un carattere riservato al segno: se questo è negativo viene visualizzato, mentre se risulta positivo viene omissso. Basta guardare come è stato scritto il numero —2963 per rendersene conto.

Provate ora a scrivere:

```
CLS : ? "INIZIO";A;B;C;"FINE"
```

otterrete una scritta come quella di FIGURA 25. Il punto e virgola al posto delle virgole causa una visualizzazione consecutiva dei vari dati.

Ora è evidente anche un'altra cosa: ogni numero viene scritto lasciando un carattere vuoto in coda. In tal modo due numeri stampati consecutivamente non risultano mai attaccati, neppure se negativi.

Vediamo un altro esempio:

```
CLS : ? "PROVA";-67;18*2;730021;"NUOVA ELETTRONICA";"FINE"
```

Si ottiene una stampa come in FIGURA 26. Se si mettono virgole al posto dei punti e virgola, sul monitor appare una scritta come in FIGURA 27: si nota che

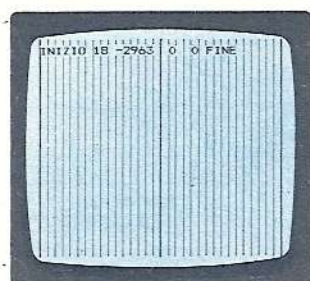


fig. 25 monitor

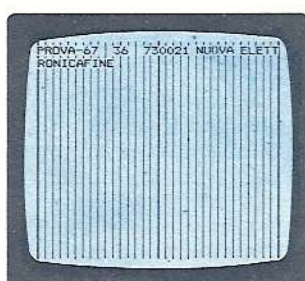


fig. 26 monitor

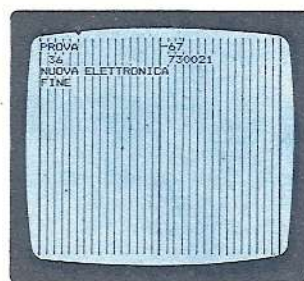


fig. 27 monitor

la parola FINE va a finire a capo della linea seguente, poiché il diciassettesimo carattere della riga precedente è occupato. Questa è una regola generale: quando la stampa è tabulata con l'uso di virgole, si ha uno slittamento alla tabulazione successiva qualora l'inizio della precedente sia già occupato.

Le possibilità d'uso dell'istruzione PRINT non si esauriscono qui; per ora quanto detto ci basta. All'occasione vedremo altri metodi di stampa, ed estenderemo il diagramma di sintassi.

LINEA 1020

Torniamo al nostro programma; eravamo rimasti alla riga 1020, che ora dovrebbe risultare chiara: essa fornisce la dicitura ESERCIZIO #, seguita dal valore della variabile ES. Tra il simbolo # (che significa numero) ed il valore di ES ci sarà sempre uno spazio vuoto, quello occupato dal segno + non visualizzato.

Passiamo ora alle linee 1030 e 1040, formalmente uguali. In esse incontriamo una nuova istruzione: RND.

RND (D/P)

L'istruzione RND (abbreviazione di RANDOM = casuale) genera dei numeri casuali. Vedremo che tale asserzione non è del tutto esatta, ma per il momento prendiamola per buona.

La sintassi di RND è rappresentata in FIGURA 28.

fig. 28 diagramma di sintassi di RND



Per usare RND è necessario mettere anche un numero tra parentesi.

Esso deve essere sempre positivo, e può essere zero. Mettendo tra le parentesi un numero negativo si

ha una segnalazione d'errore, come pure se non si usano le parentesi ed il numero.

Se si mette il numero zero, il computer assegna ad RND (che è una variabile speciale riservata al sistema) un valore casuale sempre compreso tra zero e 1, con questi valori estremi esclusi. Mettendo invece tra le parentesi un numero intero (maggiore di zero), RND restituisce numeri interi compresi tra 1 e quel numero, estremi compresi.

A livello di comandi diretti scrivete quanto segue:

```
FOR I=1 TO 10 : A=RND(0) : B=RND(10) : ? A, B : NEXT
```

Tali istruzioni fanno stampare 10 righe, ognuna delle quali contiene un numero tra zero e uno ed un numero tra uno e dieci. Ripetendo le istruzioni si ottengono valori sempre diversi.

LINEE 1030-1040

Le linee 1030 e 1040 assegnano dei numeri casuali tra 1 e 10 alle variabili numeriche A e B. La linea 1050 contiene un'istruzione di stampa che presenta una novità: PRINT @.

PRINT @ (D/P)

Questa è la prima estensione di PRINT che vediamo. La FIGURA 29 riporta il diagramma di sintassi modificato.

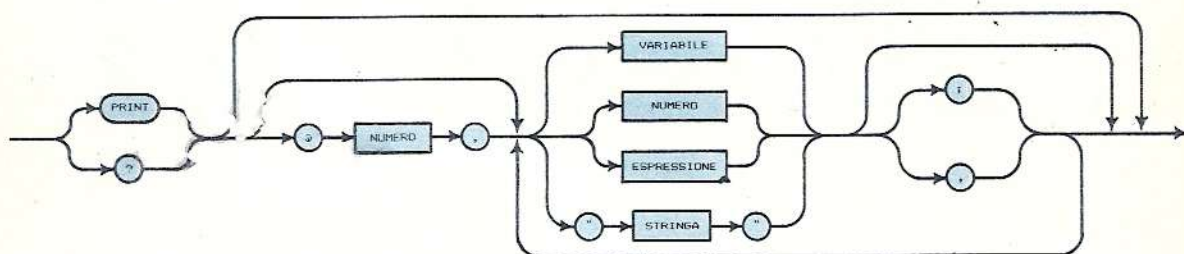


fig. 29 diagramma di sintassi modificato di PRINT

Essa è simile alla FIGURA 22, con l'aggiunta di una parte iniziale, che può essere facoltativa. Se a PRINT si fa seguire il carattere @, un numero ed una virgola, si ordina al computer di stampare quello che segue a cominciare da un ben determinato punto del monitor. Andate a riguardare la figura della mappa video; in essa si vede che il primo carattere in alto a sinistra porta il numero zero, poi ogni carattere ha un numero via via crescente, fino all'ultimo della prima riga, che è il 31; proseguendo, il primo della seconda riga è il 32, e così via fino all'ultimo carattere dell'ultima riga, che corrisponde alla posizione 511 (32 caratteri per 16 righe fa appunto 512; se si comincia a contare da zero, si arriva a 511).

Il numero che segue @ deve essere compreso tra zero e 511, e corrisponde alla posizione scelta per l'inizio della stampa. Se si batte @ in maiuscolo o se si indica un numero di zero o maggiore di 511 si ha una segnalazione d'errore.

LINEA 1050

La linea 1050 fa quindi stampare alla posizione 128 (inizio della quinta riga) la frase QUANTO FA, e predispone affinché la stampa successiva avvenga di seguito (questo per la presenza del punto e virgola).

LINEE 1060-1070-1080

Le righe da 1060 a 1080 fanno stampare il valore della variabile A, poi la dicitura PER, poi il valore assunto dalla variabile B. Ogni dato viene visualizzato di seguito al precedente; inoltre il punto e virgola posto dopo PRINT B indica che la stampa seguente avverrà ancora in coda alle precedenti.

Siamo alla linea 1090, dove compare una nuova istruzione: INPUT..

INPUT (P)

L'istruzione INPUT fa in modo che l'esecuzione del programma si fermi e che il computer resti in attesa che vengano introdotti dei dati da tastiera. In FIGURA 30 c'è il diagramma di sintassi di INPUT;

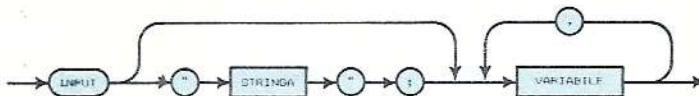


fig. 30 diagramma di sintassi di INPUT

tale istruzione può essere usata in vari modi. Il più semplice è quello della linea 1090, ossia INPUT seguito da un identificatore di variabile (che può essere numerica o di stringa). Tale possibilità si ottiene, dal diagramma di sintassi, percorrendo il ramo superiore della prima deviazione, ed uscendo alla fine dopo VARIABLE, anziché percorrere, come sarebbe possibile, il tratto superiore col carattere della virgola. Come si vede, tale ramo può essere percorso tante volte quante si vuole, ad esempio usando l'istruzione nel seguente modo:

```
INPUT T, ER$, H
```

In questo caso il programma si arresta, ed aspetta che vengano introdotti da tastiera i valori da assegnare alle tre variabili indicate; da notare che la prima e la terza sono numeriche, mentre la seconda è una variabile di stringa. Prima di fare arrestare il programma, l'istruzione INPUT provoca la stampa di un punto interrogativo. I dati digitati alla tastiera devono essere conformi al tipo di variabile che segue INPUT: se è richiesta una variabile numerica, bisogna introdurre un numero, altrimenti l'elaboratore stampa la dicitura:

```
?REDO
```

e resta in attesa che il dato venga ripetuto in modo corretto. Ovviamente, se il dato richiesto è di tipo stringa, digitando ad esempio 23 tale valore è accettato, ma non rappresenta un numero bensì una stringa, ossia un insieme di caratteri alfanumerici.

Qualora i dati richiesti sono più di uno, essi vanno introdotti separandoli con una virgola; se si preme RETURN prima di averli introdotti tutti, il programma non riparte, sul monitor compaiono due punti interrogativi, ed il calcolatore resta in attesa dei dati residui.

Soffermiamoci ora sul caso che venga introdotta una stringa; poniamo che in una linea di programma ci sia ad esempio:

```
INPUT Q$
```

Se digitiamo una stringa che contenga anche una virgola, appare la dicitura:

```
?EXTRA IGNORED
```

ossia il computer ci avvisa che ignora tutto quello che compare dalla virgola in poi; se ad esempio avevamo scritto

```
NUOVA ELETTRONICA, BOLOGNA
```

nella variabile di stringa Q\$ andrà a finire solo NUOVA ELETTRONICA.

LINEA 1090

Torniamo alla linea 1090 del nostro programma; resta da dire che il punto interrogativo generato da INPUT viene stampato di seguito ai dati precedenti a causa del punto e virgola contenuto nella linea 1080. Senza di esso il punto interrogativo sarebbe andato a finire a capo, nella riga video successiva.

LINEA 1100

La linea 1100 serve per lasciare una riga vuota nelle operazioni di visualizzazione dei dati, comportamento che abbiamo già descritto alla parola chiave PRINT.

Arrivati alla linea 1110, troviamo una nuova istruzione: IF...THEN.

IF...THEN...ELSE (D/P)

Con l'istruzione IF (= SE) si fanno dei test relazionali o dei confronti logici; se essi risultano veri il controllo del programma passa alle istruzioni che seguono THEN (= ALLORA), diversamente va a quelle che seguono ELSE (= ALTRIMENTI) oppure, se manca ELSE, alla successiva riga di programma. Il diagramma di sintassi di IF...THEN...ELSE è in FIGURA 31.

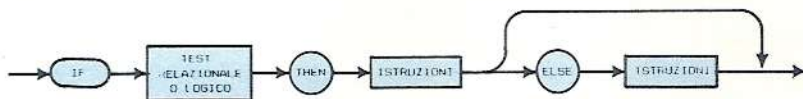


fig. 31 diagramma di sintassi di IF...THEN...ELSE

Da esso si vede che IF deve essere seguito da un confronto di tipo matematico ($<$ $=$ $>$ e tutte le loro possibili combinazioni), oppure logico (AND, NOT, OR). Fanno poi seguito la parola chiave THEN e le istruzioni che il computer deve eseguire nel caso che il test fatto da IF sia verificato.

Ecco alcuni esempi d'uso di IF...THEN:

```
IF DS>56 THEN GOTO 2370
IF A=2 AND G$="VENDITA" THEN PRINT "FATTURA CLIENTI"
IF D OR NOT F THEN GOSUB 8000
IF C)=0 AND H(3)=18 THEN Z3=C
```

Dal diagramma di sintassi si vede, poi, che ELSE può essere messo oppure no. Se c'è, esso deve essere seguito da istruzioni, le quali vengono eseguite nel caso che il test di IF non sia vero. Se ELSE non esiste ed il test di IF non è verificato, il programma va alla linea di programma successiva.

Spesso la parola chiave THEN potrebbe anche essere omessa, ma è preferibile usarla regolarmente sia per rendere più agevole la lettura del programma, sia per motivi di compatibilità del software con altri tipi di computer.

La parola chiave GOTO può essere omessa se si trova assieme a THEN; ad esempio, le tre scritture seguenti sono perfettamente equivalenti:

```
IF J<> 3 THEN GOTO 120
IF J<> 3 THEN 120
IF J<> 3 GOTO 120
```

Rimane da dire che le istruzioni di tipo IF...THEN...ELSE possono essere concatenate le une alle altre, a patto che il numero di IF sia uguale a quello di ELSE. Osserviamo ad esempio questo programma:

```
10 INPUT "INTRODUCI TRE NUMERI";A,B,C
20 PRINT "IL NUMERO PIU' PICCOLO E' ";
30 IF A>B OR A>C THEN IF B>C THEN PRINT C ELSE PRINT B ELSE PRINT A
```

Dopo aver dato il RUN, bisogna introdurre tre numeri, separati con virgole (come abbiamo già visto a proposito di INPUT); la linea 30 cerca il numero più piccolo e lo stampa in coda alla frase della riga 20.

LINEA 1110

Vediamo allora cosa succede quando si arriva alla linea 1110 del nostro programma PITA1.

Quella riga recita così: «se R è uguale al prodotto di A per B, stampa ESATTO, altrimenti stampa SBAGLIATO».

R è il valore introdotto dalla tastiera. A e B sono quelli scelti a caso dal computer nelle linee 1030 e 1040. Vedrete che l'elaboratore si comporterà come un maestro infallibile e velocissimo!

Alla riga 1120 c'è una nuova istruzione, che, come la precedente, è di grande importanza: FOR...NEXT.

FOR...TO...STEP...NEXT (D/P)

L'insieme di queste parole chiave genera un ciclo ripetitivo (LOOP) che fa percorrere per un certo numero di volte la medesima parte di programma.

Il diagramma di sintassi di FOR...NEXT è in FIGURA 32.

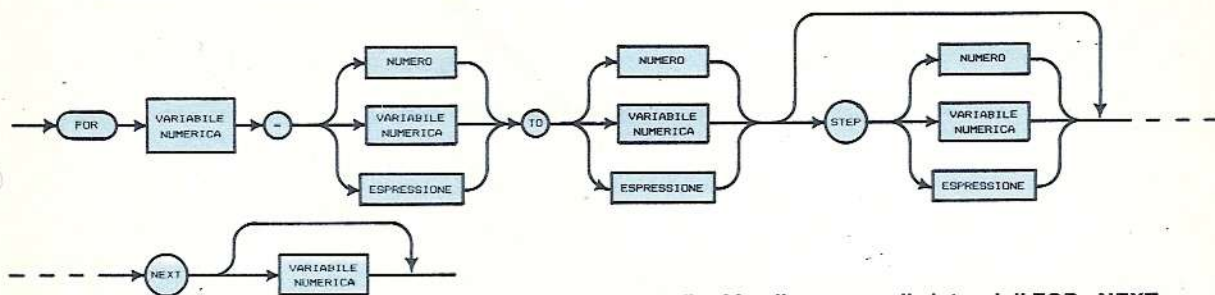


fig. 32 diagramma di sintassi di FOR...NEXT

Sicuramente sembra più complesso di quanto non sia in effetti.

Per spiegarlo, facciamo riferimento alla seguente istruzione, che può essere impartita anche in modo diretto (ossia a livello comandi):

```
FOR A=1 TO 7 STEP 1 : PRINT A : NEXT A
```

Rammentiamo per inciso che in una stessa linea si possono mettere diverse istruzioni, separando le une dalle altre col carattere dei due punti. Spesso faremo uso di questa possibilità, in quanto torna molto, comoda in certe occasioni.

L'ordine impartito al computer è questo: per (FOR) A che parte da 1 e va fino a (TO) 7 con passo (STEP) 1, stampa A; poi ripeti con il valore di A successivo (NEXT).

Dopo aver dato il via all'istruzione, sul monitor appare la sequenza di numeri:

1 2 3 4 5 6 7

Accade quindi che il computer esegue la parte di programma compresa tra FOR e NEXT (in questo caso la visualizzazione del valore di A) fino a che il valore di A non supera il limite di 7.

Quando il valore del passo è 1 esso può essere omissso, come si vede anche dal diagramma di sintassi (linea che bypassa STEP).

La variabile A è quella che controlla il loop facendo da contatore.

Il ciclo FOR...NEXT è consegnato in modo che esso viene percorso, come minimo, almeno una volta. Questo è un fatto molto importante, cui bisogna prestare attenzione: certi risultati errati di programma, che sembrano inspiegabili, derivano talvolta da una non perfetta conoscenza del comportamento del ciclo FOR...NEXT.

Per familiarizzarsi con esso, niente di meglio che eseguire il seguente programmino (consigliamo vivamente a tutti di eseguirlo: anche i più esperti potrebbero scoprire di non saper prevedere i risultati!):

```
100 N=0
110 INPUT "VALORE INIZIALE ";I
120 INPUT "VALORE FINALE ";F
130 INPUT "STEP ";S
140 FOR A=I TO F STEP S
150 N=N+1
160 PRINT "GIRO NUMERO";N;" " ;A
170 NEXT A
180 PRINT
190 PRINT "NUMERO DI PERCORRENZE:";N
200 PRINT "VALORE FINALE DI A:";A
210 PRINT : GOTO 100
```

Dopo aver lanciato questo programma, ossia dopo aver dato il RUN, si deve introdurre un numero in risposta alla prima domanda; esso rappresenta il valore iniziale della variabile di controllo del loop. Successivamente, sempre rispondendo a domande, si introduce il valore finale di quella variabile, nonché il valore che si vuole assegnare a STEP, ossia l'incremento che si desidera dare alla variabile di controllo.

Prima di premere il RETURN che introduce quest'ultimo valore, cercate di dire quante volte il ciclo viene percorso e che valore ha la variabile di controllo dopo l'uscita dal loop. Appena avrete premuto RETURN vedrete una serie di linee (stampate dalla riga 160); ognuna di esse vi dice a quale numero di percorrenza del loop si riferisce e, di seguito, il valore della variabile di controllo ad ogni tornata.

Il programma riparte sempre da capo (GOTO 100 dell'ultima riga), e per finire bisogna fare un BREAK. Provate a dare di volta in volta valori sempre diversi; vi renderete conto, meglio che leggendo tante parole, di come si comporta il computer nei cicli FOR...NEXT.

Ad esempio, dando 4 come valore iniziale e 4 come valore finale, con qualsiasi numero assegnato a STEP si ha sempre una ed una sola percorrenza del loop.

Oppure introducendo 16, 23 e -3 (inizio, fine e passo), vedrete che il ciclo viene percorso una sola volta, contrariamente a quanto accade in altri computer, che non escono più dal loop. Invece è giusto che esso si accorga che con quei dati il valore finale non sarebbe mai raggiunto, ed esca dal ciclo da solo, senza dover interrompere il programma con un break.

Ad ogni tentativo cercate anche di prevedere quale sarà il valore finale del contatore del ciclo.

Sbizzarritevi a dare i valori che più vi piacciono, fino a che non sarete in grado di prevedere i risultati senza fare errori.

Ancora una cosa: dal diagramma di sintassi si vede che dopo il segno =, dopo TO e dopo STEP possono essere messi sia dei numeri, che delle variabili numeriche, che delle espressioni; ecco alcuni esempi:

```
FOR I=K*3.5 TO A-2 STEP -(A-7)/4
FOR RC=B TO 56-B STEP 3
FOR JJ=0 TO P-9+D*2.81 STEP -(W-P)
```

Osservando il diagramma di sintassi ed anche la riga 1120 del nostro programma di PITA 1, vi sarete accorti che non è necessario ripetere la variabile di controllo dopo NEXT (questo se si ha un solo ciclo).

In particolare, poi, dalla riga 1120 si vede che tra FOR e NEXT si può anche non interporre alcuna istruzione; nel nostro caso allora ordiniamo al computer di percorrere 600 volte il loop, e di non fare niente all'interno di esso. In pratica si ottiene solamente un ritardo, equivalente al tempo che è necessario per fare tutti quei giri a vuoto.

Perché allora è stata messa la linea 1120? Semplicemente per fare rimanere la scritta ESATTO o SBAGLIATO per un po' di tempo, affinché si possa leggere con comodo prima che, con la riga 1130, il programma torni alla linea 1000: lì c'è l'istruzione CLS che cancella tutto; senza la 1120 dal monitor la dicitura formata si verrebbe subito cancellata e non si potrebbe leggere il responso. Si dice che la linea 1120 è un LOOP DI RITARDO.

Siamo così arrivati all'ultima riga del nostro programma, la 1130; in essa troviamo GOTO.

GOTO (D/P)

L'istruzione GOTO (= VAI A) fa fare un salto al numero di linea specificato. Il diagramma di sintassi si trova in FIGURA 33.

fig. 33 diagramma di sintassi di GOTO



L'uso di GOTO è molto semplice: basta farlo seguire da un numero di linea del programma. Dato che il passaggio al numero di riga indicato non dipende da particolari condizioni, si dice che l'istruzione genera un SALTO INCONDIZIONATO: esso avviene ogniqualvolta il programma passa per quella istruzione, senza che dipenda da test o confronti logici.

Tale asserzione vale, ovviamente, se GOTO non è preceduto da un'istruzione del tipo IF...THEN.

Qualora il numero di linea che segue GOTO non esista, si ha una segnalazione d'errore, ed il computer torna a livello di comandi diretti, uscendo così dal programma.

GOTO può essere usato anche in modo diretto, come se fosse un comando. Tale possibilità è spesso molto utile, specialmente in fase di debug, per controllare il comportamento di una parte di programma. Succede sovente, per motivi di verifica, di fare dei break o di mettere degli stop in un programma; ad arresto avvenuto, per ripartire si può dare il comando CONT (che vedremo più avanti) se si desidera riprendere dal punto in cui si era arrivati prima dell'interruzione; se invece si ha la necessità di ripartire da un punto diverso, ad esempio dalla riga 4820, si può fare nei due modi seguenti:

```
RUN 4820
GOTO 4820
```

Essi possono sembrare equivalenti, ma in effetti differiscono per un particolare molto importante: RUN azzerava tutte le variabili (numeriche e di stringa), mentre ciò non accade con GOTO. Una partenza fatta con questa istruzione mantiene in memoria i valori precedentemente assegnati alle diverse variabili. A tale riguardo, ricordare che se si arresta lo svolgimento di un programma con un break o uno stop, e se si fanno poi delle modifiche alle linee del programma stesso, le variabili vengono azzerate automaticamente in ogni caso; un'eventuale partenza data con GOTO, in queste condizioni, sarebbe equivalente ad una fatta con RUN.

LINEA 1130

Siamo così giunti alla fine del nostro programma: l'istruzione GOTO 1000 rimanda incondizionatamente alla riga 1000, ossia all'inizio. In tal modo tutto il ciclo fin qui visto si ripete all'infinito; per arrestare lo svolgimento del programma bisogna fare un break (oppure spegnere la macchina. A proposito, è sempre consigliabile estrarre i dischi dai drive prima di togliere o dare corrente al computer, al fine di evitare possibili danneggiamenti al contenuto dei dischi medesimi).

Arrivati a questo punto esaminiamo velocemente il contenuto delle varie righe:

- 1000 - cancella il monitor
- 1010 - incrementa la variabile ES, che tiene il conto del numero degli esercizi
- 1020 - stampa la scritta ESERCIZIO # , seguita dal valore di ES
- 1030 - genera un numero A intero casuale compreso tra 1 e 10 (estremi compresi)
- 1040 - genera un numero casuale B
- 1050 - scrive all'inizio della quinta riga QUANTO FA
- 1060 - scrive, di seguito alla frase precedente, il valore di A
- 1070 - aggiunge, sempre di seguito, la dicitura PER
- 1080 - aggiunge il valore della variabile B
- 1090 - chiede di introdurre da tastiera il valore R del prodotto di A per B
- 1100 - lascia una riga
- 1110 - stampa ESATTO oppure SBAGLIATO a seconda del valore di R
- 1120 - loop di ritardo
- 1130 - rimanda all'inizio.

Potete riguardare anche il diagramma di flusso: vi renderete conto che si tratta di un ottimo ausilio per schematizzare in modo semplice ed immediato il funzionamento del programma.

Anche l'uso dei diagrammi di sintassi dovrebbe esservi chiaro, dopo i numerosi esempi visti.

Non vi resta che digitare il programma e memorizzarlo su disco, assegnandogli un nome; dato che dopo questa vedremo altre due versioni del problema, meglio riservare il nome PITAGORA al programma definitivo, ed i nomi PITA1 e PITA2 alle due versioni ridotte.

Dopo aver scritto il programma, fatene un listato alla stampante e confrontatelo con quello di FIGURA 19. Se tutto combina, lanciate il programma con RUN e verificate che il funzionamento sia corretto.

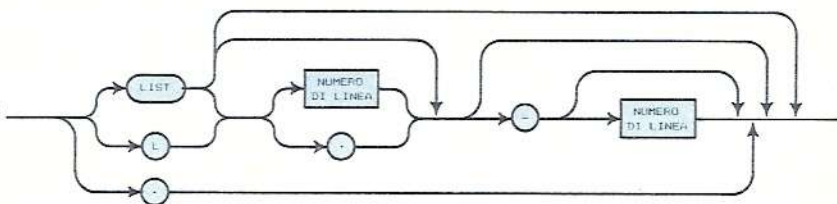
Attenti a non trovarvi nella condizione di aver capito tutto del programma, ma di non sapere le tabelline! Ad ogni buon conto è meglio che vi alleniate a rispondere bene ed in fretta, perché la modifica sostanziale che faremo nella seconda versione di questo programma consiste nell'introduzione del conteggio del tempo impiegato a dare il risultato. Non vorrete fare una figuraccia con vostro figlio o fratellino!

Vediamo ora in modo particolareggiato l'uso di alcuni comandi che vi saranno molto utili per apportare alcune modifiche al programma appena visto. Parleremo infatti di LIST, LLIST, AUTO, EDIT, RENUM, REF e DELETE.

LIST (D/P)

Il comando LIST si usa per conoscere il contenuto delle linee di programma: esse vengono presentate sul monitor. Il diagramma di sintassi è in FIGURA 34.

fig. 34 diagramma di sintassi di LIST



Seguiamo per ora il ramo superiore; digitando LIST da solo e premendo RETURN si ha il listato di tutto il programma, a cominciare dalla prima riga. Esso scorre molto in fretta e sarebbe impossibile riuscire a leggere qualcosa; si può però fermare lo scorrimento premendo contemporaneamente i tasti SHIFT e @. La pressione di un tasto qualunque fa ripartire il listato; per uscire da questo funzionamento senza dovere arrivare alla fine si può fare un break. L'altra strada possibile, quella di scrivere LIST-, è perfettamente equivalente.

Un'altra alternativa è quella di far seguire a LIST un numero di linea; se non si aggiunge nient'altro, si

ottiene il contenuto di quella linea, ammesso che ci sia; se non esiste, non si ha errore e si torna a livello comandi.

Se invece si mette il segno meno dopo il numero di linea, si ha il listato di tutto il programma, a cominciare però dalla riga indicata; per l'arresto e la partenza valgono le considerazioni precedenti.

Se dopo il segno meno si aggiunge un altro numero di linea, si ottiene il contenuto della parte di programma compreso tra le righe specificate. Se una o entrambe non esistono, il listato parte dalla linea immediatamente superiore alla prima indicata e si ferma a quella subito sotto alla seconda. In pratica non si ha mai una segnalazione d'errore.

Esaminiamo ora il ramo inferiore del diagramma di sintassi. Innanzitutto si vede che invece di scrivere LIST ci si può limitare a battere il tasto L; ciò vale però solo se L è seguito da qualcos'altro. Dal diagramma si vede infatti che non è possibile dare solo il comando L; dopo di esso si possono scegliere le alternative già viste per LIST. Di queste, resta da spiegare solo quella di battere il carattere del punto: scrivendo LIST, oppure L, si ottiene il contenuto dell'ultima linea listata o editata. A questo proposito si vede che premendo il carattere del punto da solo, senza farlo precedere da nulla, si ha il medesimo risultato. Questo però avviene solo se la pressione del tasto del punto è la prima fatta dopo l'apparizione della dicitura READY; se questa condizione non è verificata non si ottiene il contenuto dell'ultima riga listata od editata, ma la stampa del punto sul monitor.

Vediamo qualche esempio:

```
LIST
LIST-1200
LIST 230-
LIST 400-430
L20
L.-785
L500-
```

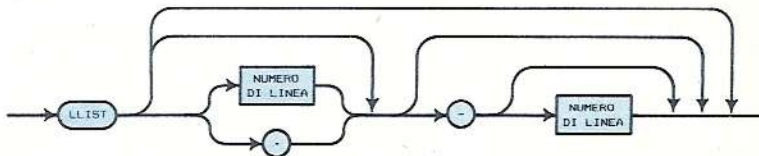
Oltre a tutto questo, è possibile esaminare il listato anche in altri modi. Parliamo innanzitutto del tasto LINE FEED (avanzamento di linea); esso si trova a fianco di RETURN e la sua pressione lista la linea immediatamente successiva all'ultima listata od editata. Anche per questo tasto, come per quelli che esamineremo più sotto, vale la condizione detta poco fa: l'effetto è quello descritto solo se la pressione del tasto avviene per prima dopo la stampa dei READY.

La pressione del tasto ESC fornisce sempre il contenuto della prima linea del programma. Premendo invece il tasto della parentesi quadra aperta ([) si ha la linea precedente all'ultima visualizzata. In pratica, col tasto LINE FEED si va avanti e con quello della parentesi quadra si va indietro lungo il listato. Le possibilità sono quindi molteplici, le più complete presenti sui personal computer.

LLIST (D/P)

Il comando LLIST è molto simile al precedente; esso serve per listare il programma sulla stampante. Il suo diagramma di sintassi è in FIGURA 35.

fig. 35 diagramma di sintassi di LLIST



Esso differisce da quello di LIST solo per il fatto che LLIST non è abbreviabile; per il resto valgono le spiegazioni e la casistica vista per il comando LIST.

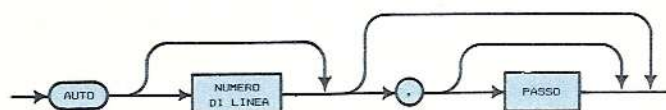
AUTO (D)

Il comando AUTO serve in fase di scrittura del programma, per farsi dare dal computer i numeri delle

nuove linee anziché scriversi da soli.

Il diagramma di sintassi di AUTO è in FIGURA 36.

fig. 36 diagramma di sintassi di AUTO



Il comando AUTO può essere dato in vari modi e cioè: da solo; aggiungendo una virgola; facendolo seguire da un numero, od anche da un numero più una virgola e un altro numero. Piuttosto che perdersi in lunghe spiegazioni, vediamo alcuni esempi:

```
AUTO
AUTO 50
AUTO ,
AUTO 1000, 100
AUTO , 25
```

Il primo caso inizia dalla linea 10 e prosegue con passo 10 tra una riga e la successiva; 10 è quindi il valore di default del numero di linea iniziale e del passo, se essi non sono indicati. Fa eccezione solo il terzo caso, che dà zero come numero di linea iniziale (passo 10). In effetti un programma gira anche se comincia con la riga zero, ma se si prova a rinumerare col comando RENUM, che vedremo tra poco, il computer rifiuta di eseguire l'ordine.

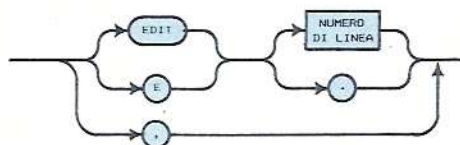
Il secondo esempio dà una numerazione automatica che comincia da 50 e procede di 10 in 10. Il quarto caso parte da 1000 con passo 100; l'ultimo parte da zero con passo 25.

Se durante la numerazione automatica una linea proposta esiste già in memoria, subito dopo il numero viene visualizzato un asterisco, per avvisare della cosa; in tal caso, per non modificare il contenuto di quella riga, occorre uscire dalla numerazione automatica con un break.

EDIT (D)

Il comando EDIT è già stato spiegato in precedenza molto diffusamente. Esso serve per passare a livello di correzione (editing) in una certa linea di programma. Il suo diagramma di sintassi è in FIGURA 27.

fig. 37 diagramma di sintassi di EDIT



RENUM (D)

Nella stesura e nella correzione di programmi anche il comando RENUM torna molto utile; infatti permette di rinumerare, tutte o solo in parte, le linee di programmazione. Il diagramma di sintassi di RENUM è in FIGURA 38.

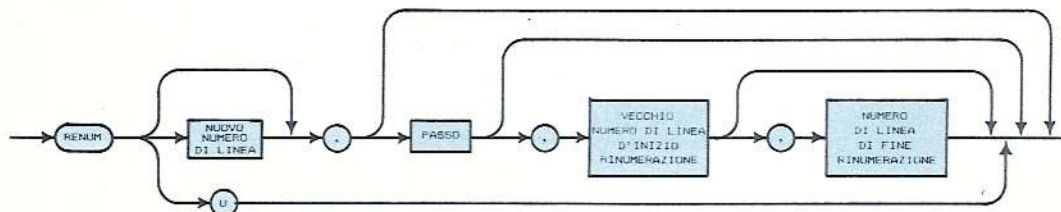


fig. 38 diagramma di sintassi di RENUM

Il modo più semplice per impartire il comando RENUM è il seguente:

RENUM,

Il risultato ottenuto è una rinumerazione di tutto il programma in memoria; la prima linea porta il numero 10, ed il passo tra una riga e la successiva è 10. Ciò significa che i valori di default assunti dal sistema per il nuovo numero di linea e per il passo valgono appunto 10. Da notare che RENUM dato senza la virgola comporta una segnalazione d'errore; la minima configurazione sintatticamente valida è quella data sopra.

Vediamo la prima estensione possibile:

RENUM 100

Questo comando rinumerava il programma assegnando 100 alla prima riga, e procedendo con incremento 10. Ovviamente il numero 100 è solo un esempio; a questo proposito c'è da dire che il massimo numero di linea accettato dal sistema è 65529. Questo è anche il limite massimo accettato per il nuovo numero di linea e per i numeri di riga d'inizio e fine rinumerazione.

D'ora in poi la virgola finale non è più obbligatoria.

Procediamo ancora per gradi; quello successivo è il seguente:

RENUM 100, 25

Il programma viene rinumerato con prima linea 100 e passo 25. Facciamo ancora un'altro passo:

RENUM 1000, 15, 200

Col comando dato in questo modo si fa la rinumerazione di una sola parte del programma: essa parte col numero di riga 1000 e passo 15, ma comincia a rinumerare dalla vecchia linea 200 e non dalla prima. Il risultato è facilmente intuibile: tutta la parte originaria di programma che precedeva la riga 200 conserva la sua vecchia numerazione; dalla 200 in poi assumerà quella nuova, che comincia da 1000 e procede con intervallo 15. Questo è utile per fare molto posto (da 200 a 999), nel caso volessimo inserire parecchie linee in quella posizione del programma.

Siamo così arrivati alla configurazione massima:

RENUM 3200, 100, 1220, 2070

Col comando si rinumerava la zona di programma compresa tra le linee 1220 e 2070, partendo col numero di riga 3200 e procedendo con passo 100. In tal modo si rinumerava solo una posizione intermedia del programma, lasciando inalterati l'inizio e la fine.

La prima regola da osservare è che il numero d'inizio rinumerazione, quello di fine ed il passo devono essere tali da poter contenere in quell'intervallo tutte le linee esistenti; se ad esempio volessimo assegnare nuovi numeri di linea partendo da 1000 ed arrivando a 2000 con passo 100, interessando a questa operazione 18 righe, la cosa sarebbe impossibile. In casi come questo si ha la seguente segnalazione d'errore:

```
SEQ # OVFL          (SEQUENTIAL NUMBER OVERFLOW =  
                     saturazione dei numeri di sequenza)
```

C'è anche un'altra regola da osservare: il nuovo numero di riga deve essere maggiore od uguale al numero di linea d'inizio della rinumerazione.

Ad esempio, scrivendo:

RENUM 500, 5, 730

si ha una segnalazione d'errore ed il comando non viene eseguito. Tale comportamento è estremamente logico, a pensarci bene: se esso venisse accettato tutta la parte originale di programma dalla linea 730 in poi dovrebbe essere rinumerata in modo da trasformare la 730 in 500, e tutte le rimanenti di seguito con passo 5, col pericolo di andare a sovrapporsi alla parte di programma non rinumerata.

Ciò non significa che una linea di programma non possa assumere un numero più basso; anzi, dando il comando

RENUM, 1

ad ogni linea viene assegnato il più basso valore possibile; successivamente si possono fare altre rinumerazioni (il numero di volte non ha limiti) per riordinarle diversamente. Partire da zero con una rinumerazione comporta una segnalazione d'errore; il comando seguente non viene accettato:

RENUM 0, 1

Quando la rinumerazione è terminata appare la scritta DONE (eseguito) e, naturalmente, READY.

Il tempo impiegato per portare a termine il riordino dipende dalla lunghezza del programma e può richiedere anche qualche secondo se esso è molto lungo. A questo proposito è meglio sapere che se

restano disponibili solo pochissime celle di memoria la rinumerazione può andare in crisi; in un caso del genere compare la scritta (tragica):

```
FATAL ERROR; TEXT NOW BAD      (errore fatale ora il testo e' rovinato)
```

Se capita di dover rinumerare un programma lungo che lascia libera poca memoria è consigliabile salvarlo su disco prima di provare a rinumerarlo in modo che se l'operazione non va a buon fine si può recuperare dal disco il lavoro fatto, altrimenti sarebbe perso irrimediabilmente.

Osservando il diagramma di sintassi di RENUM, si vede che non abbiamo ancora trattato la possibilità offerta dal ramo inferiore: se scriviamo:

```
RENUM U
```

ordiniamo al computer di controllare se ci sono, nel programma in memoria, dei rinvii a numeri di linea inesistenti. In caso affermativo compare una scritta di questo tipo:

```
ERROR LINES  
5000/U  DONE
```

altrimenti c'è solo DONE.

Nell'esempio fatto, esiste in qualche linea di programma un GOTO 5000 (o altro richiamo), e la 5000 non esiste. In questo caso, si hanno due possibilità di rinumerazione: o si scrive la linea 5000 o si va a cambiare il rinvio sostituendo a 5000 un numero di riga esistente. Per trovare la posizione o le posizioni in cui compare il rinvio alla linea 5000 si fa uso del comando REF, che vedremo tra poco.

Può anche succedere che in qualche linea di programma venga richiamato un numero di sequenza illecito, ossia superiore a 65529; in questo caso il comando RENUM U genera una scritta del tipo:

```
ERROR LINES  
820/S  DONE
```

Essa ci dice che la linea 820 contiene un numero di riga non ammesso. Col comando RENUM U non si effettua quindi la rinumerazione, ma si controlla solamente se vengono richiamate linee inesistenti o impossibili.

Se viene chiesta una rinumerazione e ci sono errori di rinvio, compare una scritta, di questo tipo:

```
ERROR LINES      (linee d'errore)  
5000/U
```

Si ha cioè una segnalazione analoga alla precedente, ed il comando non viene eseguito (infatti non compare DONE). Resta ancora da esaminare un'ultima possibilità. Supponiamo che il programma da rinumerare contenga la seguente linea:

```
350 GOTO GOSUB 1100
```

Applicando ad esso uno qualsiasi dei comandi RENUM visti, si ottiene la seguente segnalazione d'errore:

```
ERROR LINES  
350/X
```

Si vede quindi che il computer è in grado di individuare, in sede di rinumerazione, quelle linee che contengono errori di sintassi assieme ai rinvii.

Riassumendo, le regole da rispettare con RENUM sono le seguenti: i vari numeri di linea non devono essere superiori a 65529; il vecchio numero di linea d'inizio della rinumerazione non può essere maggiore di quello di fine rinumerazione. Si ha inoltre una segnalazione d'errore anche se si ordina di rinumerare un certo tratto intermedio di un programma, quando, coi valori assegnati all'inizio e al passo, si va oltre la fine specificata.

Resta da dire che si possono omettere alcuni parametri; in tal caso il computer assegna ad essi i valori di default. Dei primi due abbiamo già parlato (valgono 10); il terzo ha come valore di default il numero di linea 1; il quarto assume 65529. Ciò significa che, se non è indicato in modo diverso, la rinumerazione interessa tutto il programma (ossia parte dall'eventuale linea 1 per arrivare all'ultima possibile, la 65529).

Per finire, ecco qualche esempio di richiesta corretta di rinumerazione:

```
RENUM , , 4000 -  
RENUM 120 , 50  
RENUM , ,  
RENUM , 20 , 2750
```

Per familiarizzarsi con questo potente comando del BASIC è meglio caricare in memoria un programma un po' lungo ed esercitarsi a fare rinumerazioni di diverso tipo; una successiva LIST farà vedere gli effetti ottenuti.

REF (D)

Anche il comando REF è di grande utilità, in quanto fornisce tutti i riferimenti del programma: dice quali sono le variabili usate, i numeri di linea richiamati, le costanti presenti.

Il diagramma di sintassi di REF è in figura 39.

fig. 39 diagramma di sintassi di REF

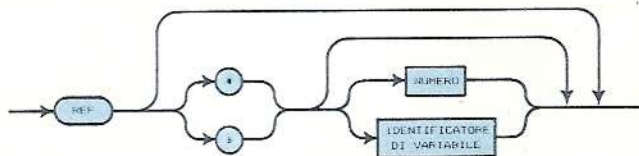


fig. 40 riferimenti del programma di PITA 1

Per spiegare REF facciamo riferimento alla FIGURA 40. Innanzitutto diciamo che essa dà i riferimenti del programma PITA1 appena visto, ed è stata ottenuta impartendo il comando:

REF\$

Questo è uno dei modi possibili di dare il comando REF, e fornisce appunto l'elenco completo dei riferimenti del programma che si trova in memoria. Analizziamo la FIGURA 40 in dettaglio. Si vede che i dati sono divisi in senso verticale in due parti: a sinistra ci sono numeri e, più sotto, lettere; a destra solo numeri. Nella prima riga il numero 1 porta alla destra i numeri 1010 e 1120; se andate a vedere il listato di PITA1 (figura 19), vi rendete conto che alle due linee indicate compare appunto il numero 1. La prima riga ci fornisce quindi i riferimenti del numero 1, ossia dove si trova.

La seconda riga dà i riferimenti del numero 10, e così via per le seguenti. Viene poi la lettera A; al suo fianco sono riportati i numeri di linea dove danno i riferimenti delle variabili B, ES, I ed R.

Come vedete, questo comando è molto potente e ne farete sicuramente un uso sempre più frequente, mano a mano che diverrete esperti e smaliziati in programmazione.

Il comando precedente dà i riferimenti completi sulla stampante, ma li vedrete comparire contemporaneamente anche sul monitor. Per averli solo sul video si deve impartire il comando in questo modo:

REF*

L'elenco parte dall'inizio e non si ferma che alla fine; se fosse lungo non starebbe tutto in una pagina video, quindi ci serve un modo per fermarlo e riprenderlo dal punto d'arresto. Per fare questo basta premere un tasto qualsiasi e lasciarlo per proseguire; la pressione di RETURN interrompe l'elenco e fa tornare a livello di comandi diretti.

Un'altra cosa da dire sul comando REF è che esso non va ad analizzare il contenuto all'interno delle stringhe; se, ad esempio, una riga di programma avesse il seguente contenuto:

```
PRINT "VIA CRACOVIA 19 - BOLOGNA" : GOTO 6240
```

il comando REF riuscirebbe ad estrarre i riferimenti di 6240, ma non darebbe mai quelli del numero 19, poiché esso si trova all'interno di una stringa.

Torniamo alla FIGURA 40, e precisamente alla riga dove sono elencati i riferimenti della variabile numerica ES. Si vede che il numero 1010 è seguito da una barra e dal numero 2: significa che nella linea di programma 1010 la variabile ES compare due volte.

Per continuare le spiegazioni di REF, prendiamo ora la FIGURA 44: vi troviamo i riferimenti del programma PITA2 che vedremo tra breve, il cui listato è in FIGURA 42. Andando a vedere i riferimenti della variabile R1, si vede che il numero 1180 è seguito dalla barra, dal simbolo della stringa e dal numero 2: ciò significa che la variabile R1 compare due volte nella linea 1180 ed è di tipo stringa. Continuando a leggere i riferimenti di R1, si trova poi che essa compare come stringa anche alla riga 1190, poi si trova 2 volte alla linea 1200; in questo caso però R1 è una variabile numerica, in quanto non c'è il simbolo del dollaro.

Andiamo ora a vedere i riferimenti della variabile N. Troviamo dapprima il numero 1070 seguito da una parentesi e dal numero 2: significa che quella variabile è di tipo numerico indicizzato (vettore o matrice, come vedremo), e che compare due volte nella linea 1070.

Qualora la variabile N fosse stata di tipo stringa ed anche indicizzata, avremmo avuto un riferimento di questo tipo:

```
N 310($2
```

Esistono poi anche altri tipi di indicazione dei riferimenti, quando essi si riferiscono a variabili numeriche di tipo intero e in doppia precisione: se avessimo ad esempio una linea di programma come la seguente:

```
25 A=12 : A!=C : A%=5 : A#=D/5 : A#(8)=2
```

I riferimenti della variabile A sarebbero questi:

```
A 25 25/! 25/% 25/# 25(8
```

Ciò significa che REF specifica anche il tipo della variabile numerica (in singola o doppia precisione, intera).

Tornando ai riferimenti di PITA2, alla riga della variabile N si trova anche il numero 1090: tale linea contiene solo REVON, quindi in quella variabile vanno a finire tutti i numeri di linea che contengono l'istruzione REVON. Analogamente per REVOFF, i cui riferimenti compaiono sotto la variabile FF.

Non è ancora finita. Supponiamo di avere in memoria il programma PITA1 e di chiedere i riferimenti della variabile A; il modo per farlo è:

```
REF A
```

(Lo spazio, come sempre, è facoltativo). Si ottiene:

```
A.1030 1060 1110
```

Se, a questo punto, introduciamo il comando REF senza farlo seguire da niente, sul monitor viene visualizzata la linea di programma 1030; digitando nuovamente REF abbiamo la riga 1060, poi avremo la 1110. Provando ancora con REF, compare la scritta:

```
TEXT END
```

ossia il computer, col comando REF, dà in sequenza le linee in cui compare la variabile o il numero relativo al precedente comando REF.

Un successivo REF visualizza TEXT END, poi se si desse ancora REF si ripartirebbe da capo.

La prima volta che si chiedono i riferimenti di un programma, se si digita solo REF si ha una segnalazione d'errore; esso si può dare solo dopo aver ordinato i riferimenti di un numero o di una variabile.

C'è ancora un'altra possibilità. Supponiamo di avere in memoria il programma PITA1 e di scrivere il seguente comando:

```
REF*ES
```

In questo caso il monitor visualizza i riferimenti di tutte le variabili, a cominciare da ES e proseguendo con quelle che vengono dopo (seguendo l'ordine alfabetico). Scrivendo invece:

```
REF*G
```

si avrebbero i riferimenti delle variabili che cominciano da G; dato che questa nel programma non esiste, si hanno solo quelli delle variabili successive, cioè si ottengono i riferimenti di I e di R.

Se si indica la variabile richiesta con più di due caratteri si ha errore.

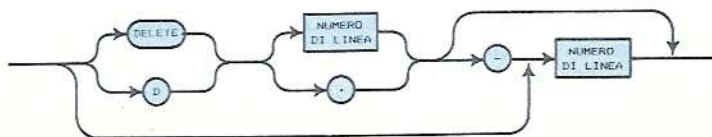
Considerazioni analoghe si possono fare se il comando REF viene richiesto con uscita anche sulla stampante, ossia mettendo il segno del dollaro al posto dell'asterisco.

Si vede quindi che il comando REF è effettivamente molto potente; esso permette di fare ricerche rapide e complete del parametro che interessa.

DELETE (D)

Il comando DELETE (= cancella) serve per cancellare linee di programma. Il suo diagramma di sintassi è in FIGURA 41.

fig. 41 diagramma di sintassi di DELETE



Il modo più semplice per cancellare una linea di programma è quello di scrivere il numero di quella riga e premere RETURN; è la possibilità offerta percorrendo il ramo inferiore del diagramma. Lo stesso risultato si può ottenere facendo precedere il numero dalla parola chiave DELETE o semplicemente da D (che può sempre sostituire DELETE); quindi le tre possibilità seguenti si equivalgono:

```
DELETE 120
D 120
120
```

Esiste poi il seguente modo di impartire il comando:

D.

Esso cancella l'ultima linea editata o listata.

Un'ultimo modo di dare DELETE è il seguente:

```
D 45-80
```

tutte le linee di programma comprese tra 45 e 80 vengono cancellate, estremi compresi. Qualora una o entrambe le righe indicate non esistessero, si ha una segnalazione d'errore ed il comando non sortisce alcun effetto.

```
1000 CLS
1010 S=0 : I1=0 : ES=ES+1
1020 PRINT "ESERCIZIO #";ES : PRINT @18,"(F PER FINIRE)"
1030 PRINT @128,"STO PENSANDO!"
1040 RANDOM
1050 A=RND(10)
1060 B=RND(10)
1070 IF N(A,B)>0 THEN 1050 ELSE N(A,B)=1
1080 PRINT @128,"QUANTO FA ";
1090 REVON
1100 PRINT A;
1110 REVOFF
1120 PRINT " PER ";
1130 REVON
1140 PRINT B;
1150 REVOFF
1160 PRINT " ? ";
1170 FOR I=1 TO 100000
1180 R1$=INKEY$ : IF R1$="" THEN NEXT I
1190 IF R1$="f" THEN ES=ES-1 : GOTO 1350
1200 R1=VAL(R1$) : IF R1=0 THEN I1=I1+I : GOTO 1170
1210 PRINT R1$;
1220 LINEINPUT R2$
1230 R=VAL(R1$+R2$)
1240 PRINT
1250 IF R=A*B THEN PRINT "ESATTO"
1260 IF R<>A*B THEN TE=TE+1 : S=1 : PRINT "SBAGLIATO: ";A;"x";B;"=";A*B
1270 T=INT((I+I1)/120)
1280 P=3-T
1290 IF S=1 THEN P=P-10
1300 TP=TP+P
1310 PRINT "PUNTI :";P
1320 PRINT "TOTALE :";TP
1330 FOR I=1 TO 1000 : NEXT
1340 IF ES<100 THEN 1000
1350 CLS
1360 REVON
1370 PRINT " SITUAZIONE "
1380 REVOFF
1390 PRINT : PRINT
1400 PRINT "TOTALE ESERCIZI :";ES
1410 PRINT "TOTALE ERRORI :";TE
1420 PRINT "TOTALE PUNTEGGIO :";TP
1430 PRINT : PRINT
1440 PRINT "CIAO !!"
1450 END
```

fig. 42 listato di PITA2

SECONDA VERSIONE DEL PROGRAMMA PITAGORA

Dopo aver esaminato il funzionamento di alcuni comandi BASIC che possono esservi utili per proseguire, riprendiamo il programma PITA1 con l'intenzione di apportarvi alcune modifiche. Elenchiamole:

- 1 - desideriamo che il computer scelga le tabelline da proporre sul monitor a caso, ma scartando quelle già chieste; deve quindi presentare tutte le 100 combinazioni possibili, poi fermarsi ed illustrare la situazione delle risposte date;
- 2 - vogliamo che il tempo impiegato per dare la risposta sia conteggiato, per dare, assieme al fatto che la risposta sia giusta o sbagliata, un punteggio ad ogni tabellina chiesta;
- 3 - deve essere possibile interrompere l'esecuzione del programma, senza perdere la visualizzazione della situazione.

Il listato del nuovo programma, che chiameremo PITA2, è in FIGURA 42; il diagramma di flusso in FIGURA 43 e l'elenco dei riferimenti in FIGURA 44.

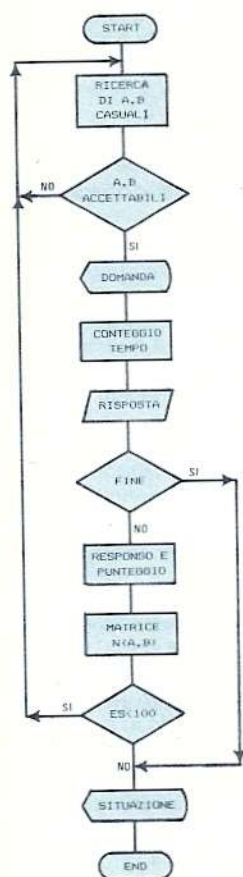


fig. 43 diagramma di flusso di PITA2

```

0 1010/2 1070 1200
1 1010 1070 1170 1190
  1260/2 1290 1330
3 1280
10 1050 1060 1290
18 1020
100 1340
120 1270
128 1030 1080
1000 1330 1340
1050 1070
1170 1200
1350 1190
A 1050 1070/2 1100
  1250 1260/3
B 1060 1070/2 1140
  1250 1260/3
ES 1010/2 1020 1190/2
  1340 1400
FF 1110 1150 1380
I 1170 1180 1200 1270
  1330
I1 1010 1200/2 1270
N 1070/2 1090 1130
  1360
P 1280 1290/2 1300
  1310
R 1230 1250 1260
R1 1180/$ 1190/$ 1200/2
  1200/$ 1210/$ 1230/$
R2 1220/$ 1230/$
S 1010 1260 1290
T 1270 1280
TE 1260/2 1410
TP 1300/2 1320 1420
  
```

fig. 44 riferimenti di PITA2

Vi facciamo notare che quasi tutte le linee del programma precedente PITA1 compaiono immutate anche in PITA2, magari sotto numeri di linea diversi; si tratta quindi di una elaborazione di PITA1, fatta cercando di utilizzare al massimo le linee già digitate. Tale procedura rispecchia quello che si fa normalmente: il programma che abbiamo in testa cresce a poco a poco, e se l'impostazione è buona, partendo da uno schema di base ridotto all'osso si può costruire poco a poco il programma definitivo.

La cosa che aiuta a partire nel modo migliore è la padronanza dei diagrammi di flusso: specialmente il principiante può trovare in essi un valido aiuto per schematizzare il problema e ordinare così le proprie idee. Vi accorgete che, mano a mano che diventerete più esperti in programmazione, farete sempre più di rado i diagrammi di flusso; se però vi capiterà un problema difficile, che non riuscite a risolvere speditamente, sentirete l'esigenza di fare un diagramma almeno parziale. Esso potrà aiutare a schiarirvi le idee. Inoltre l'utilità di un diagramma di flusso si fa sentire molto quando si esamina un programma fatto da altri:

senza il suo ausilio, spesso non si riesce a capire la logica del listato. Anche i propri programmi, a distanza di tempo, risultano poco chiari se sprovvisti del diagramma di flusso. È meglio quindi abituarsi a farli e conservarli, come documentazione del proprio lavoro. Tra analizzare solo un listato ed avere invece a disposizione anche i suoi riferimenti ed il suo diagramma di flusso, ci corre parecchio.

Partiamo pure, allora, con l'analizzare il diagramma di flusso di PITA2, confrontandolo con quello di PITA1.

La prima differenza che si incontra consiste nell'aggiunta di un test fatto subito dopo la scelta dei due numeri casuali A e B: si verifica se essi sono accettabili, in base alla condizione 1 posta prima. Vedremo tra poco come possiamo condurre in porto una tale verifica.

La seconda differenza si trova dopo la visualizzazione della domanda: in questo caso dobbiamo introdurre un meccanismo che tenga il conteggio del tempo impiegato a rispondere. La terza aggiunta è dopo la risposta, e consiste nel verificare se vogliamo interrompere la sequenza delle domande e porre fine al programma. In caso affermativo, dopo la visualizzazione della situazione, si termina; se invece non è stato chiesto di finire, oltre al solito responso ESATTO o SBAGLIATO il programma calcola il punteggio da assegnare alla risposta, e memorizza il fatto che la coppia di numeri A e B appena visualizzata è già uscita; questo per poter poi controllare se una nuova coppia di valori casuali è accettabile o meno.

Segue poi un test per verificare se si è arrivati a 100 esercizi; in caso affermativo il programma fa vedere la situazione delle risposte e va alla fine, altrimenti riparte con una nuova domanda.

Il limite 100 è ovvio: in totale le tabelline da 1 a 10 sono 100.

Oramai dovrete aver preso una sufficiente confidenza coi diagrammi di flusso, in modo da non trovare troppe difficoltà a seguirne le spiegazioni.

La cosa che dobbiamo decidere subito, per poter apportare la prima modifica, è questa: come fare per memorizzare le tabelline già chieste, in modo da poterle scartare se si ripresentano.

Il problema, come accade spesso, ha parecchie soluzioni; quella che vi presentiamo è relativamente semplice, porta via poche linee di programma e non richiede tempi di elaborazione troppo lunghi. Tutti questi fattori sono della massima importanza in ogni programma che si rispetti e dovrete tenerli sempre ben presenti. Probabilmente esistono anche altre soluzioni più brillanti, ma in ogni caso quella che proponiamo ha il grosso pregio di mostrare un utilizzo funzionale e pratico delle matrici a due dimensioni (dette anche, impropriamente, vettori a due dimensioni).

Cominciamo col dire cos'è un VETTORE ad una dimensione: si tratta di un insieme di dati che vengono memorizzati facendo uso di un'unica variabile indicizzata. Il primo dato si trova, ad esempio, depositato nella variabile D(1), il secondo in D(2), e così via (il limite alla dimensione di un vettore è imposto solo dalla quantità di memoria centrale disponibile).

L'esempio fatto si riferisce ad una variabile indicizzata di tipo numerico; si possono fare vettori anche per stringhe. Ad esempio, NS(I) potrebbe indicare un insieme di dati alfanumerici (una raccolta di nomi, un insieme di indirizzi, ecc.). Abbiamo messo la lettera I tra parentesi al posto di un numero, proprio per indicare che al variare del numero I si hanno via via le varie stringhe corrispondenti.

Come sempre, sarà la pratica a far capire bene il senso di tutto questo. Per MATRICE si intende qualcosa che è concettualmente simile al vettore, ma che risulta più complesso: mentre il vettore ha una sola dimensione, la matrice ne ha due o più.

In questo discorso non ci interessa il significato matematico di matrice, ma quello pratico di organizzazione strutturata dei dati. Sotto questo aspetto si può dire che una matrice a due dimensioni non è altro che una tabella a due entrate (righe e colonne).

La Tavola Pitagorica è un esempio tipico di dati organizzati in tabelle: si entra in una colonna scegliendo un numero, si entra in una riga scegliendone un'altro, e si va a leggere cosa c'è scritto nell'incrocio di quella riga con quella colonna. Così, ad esempio ricaviamo che 4 per 8 fa 32. Potremmo pensare di mettere i valori dei prodotti in una matrice a due dimensioni; in tal caso, se V(N,M) è l'identificatore della variabile indicizzata che intendiamo usare (dove M ed N sono due numeri che vanno da 1 a 10), potremmo esprimere l'esempio appena fatto dicendo: $V(4,8) = 32$.

Continuando con gli esempi riferiti alla Tavola Pitagorica, si può scrivere $V(7,3) = 21$ oppure $V(9,5) = 45$ oppure $V(5,9) = 45$ e così via.

Se supponiamo di avere già memorizzato tutti questi valori, e se chiediamo di vederne uno specificando tra parentesi i due numeri che lo contraddistinguono, il computer se lo va a cercare e ce lo propone velocemente.

DIM (D/P)

Come tutte le altre variabili, anche quelle indicizzate sono depositate in locazioni di memoria; mentre però quelle normali non devono rispettare alcun ordine preciso, quelle con indice devono essere contigue

È per questo motivo che per usare variabili indicizzate (ossia vettori o matrici) bisogna DIMENSIONARLE, assegnando loro i valori massimi degli indici. Quando il computer incontra questo dimensionamento, va nella zona di memoria riservata alle variabili e se ne riserva una porzione sufficiente a contenere il numero di dati dichiarati; in tal modo essi saranno contigui.

```

graph LR
    Start(( )) --> DIM{DIM}
    DIM --> ID[IDENTIFICATORE DI VARIABILE]
    ID --> C((C))
    C --> NUMERICA[VARIABILE NUMERICA]
    C --> NUMERO[NUMERO]
    NUMERICA --> S1((1))
    NUMERO --> S1
    S1 --> A((+))
    A --> DIM
    A --> S2((2))
    S2 --> End(( ))

```

Abbiamo già detto che non ci sono limiti al numero di dimensioni e al numero massimo di elementi per ogni dimensione; l'unico fattore che pone dei problemi al riguardo è la memoria. Per darvi un'idea, il penultimo esempio occupa 6926 locazioni: si fa quindi presto a superare la capienza della memoria centrale, considerando che in essa deve trovare posto anche il programma.

Resta da dire che non c'è bisogno di fare la DIM se il numero massimo degli elementi per ogni indice non supera 10: in questo caso il computer si riserva da solo il posto necessario. Da notare, infine, che scrivendo, ad esempio, DIM J(23) si predispongono 24 celle di memoria per i dati: infatti il primo di essi è J(0), il secondo J(1), e così via fino a J(23): il computer comincia a contare sempre da zero.

Alla riga 1010 sono state aggiunte le inizializzazioni a zero di due nuove variabili, S e I1; vedremo più avanti che ciò è necessario per un corretto funzionamento del programma, quando il medesimo riparte da capo con una nuova domanda.

LINEA 1020

È stata aggiunta la scritta (F PER FINIRE), stampata a cominciare dal diciannovesimo carattere della prima riga video. Tale dicitura spiega che per porre termine al programma è sufficiente premere il tasto della lettera F anziché rispondere alla nuova domanda.

LINEA 1030

Questa riga non esisteva nel programma PITA1. È stata messa per il seguente motivo: dato che il computer presenta sullo schermo solo combinazioni non ancora uscite, quando si è vicini al limite delle 100 domande esso può impiegare anche un tempo relativamente lungo per trovare, tra quelli scelti a caso, i valori di A e di B accettabili; ad esempio, potrebbe impiegare 5 secondi, spesi in una ricerca frenetica di nuovi valori che vengono poi scartati. L'entità di questo ritardo è dovuta solo al caso.

Per non mettere in imbarazzo chi usa il programma (non capirebbe il motivo di questo arresto), tra l'inizio della ricerca dei numeri A e B ed il momento in cui viene trovata una coppia di valori accettabili, sul monitor compare la dicitura STO PENSANDO!, stampata all'inizio della quinta riga.

Vi renderete conto, cominciando ad usare il programma, che questa frase resta visibile solo per una frazione di secondo, poi rimane sul monitor per intervalli di tempo via via più lunghi mano a mano che ci si avvicina al limite finale delle 100 domande: all'inizio è estremamente probabile che la prima coppia di valori estratta a caso sia accettabile, mentre andando verso la fine questa eventualità è sempre più rara.

Alla riga successiva troviamo una nuova istruzione: RANDOM.

RANDOM (D/P)

Per poter spiegare RANDOM bisogna completare quanto detto in precedenza sull'istruzione RND. Normalmente, nei personal computer, nella variabile RND vanno a finire dei valori che non sono affatto casuali, ma che vengono scelti entro un set vastissimo di numeri già disponibili all'interno dell'unità centrale. Questi numeri quindi sono sempre residenti nella CPU e sono stati scelti dai progettisti in modo da rispettare una distribuzione casuale.

In realtà, allora, RND non fornisce dei numeri casuali veri e propri, sempre diversi tra loro: se si andasse avanti per moltissime estrazioni, da un certo punto in poi la sequenza dei valori si ripeterebbe.

Per questi motivi si suol dire che l'istruzione RND restituisce numeri PSEUDOCASUALI, ossia quasi casuali.

Normalmente, inoltre, se si mette una RND all'inizio di un programma, ogni volta che si fa un RUN si ottiene sempre lo stesso numero, il primo del set menzionato. Per ottenere valori diversi ad ogni lancio di programma, l'istruzione RND deve essere preceduta da RANDOM, la quale ha la funzione di far cominciare l'estrazione dei numeri pseudocasuali da un qualunque punto intermedio del set a disposizione. Detto in parole povere, è un po' come se si mescolassero le carte prima di distribuirle.

Il computer di NUOVA ELETTRONICA non ha un comportamento come quello ora descritto: infatti, se provate a controllare i valori forniti da RND lanciando più volte un programma, otterrete valori sempre diversi: con RND, nel nostro caso, si comincia sempre a caso nel set a disposizione. Resta comunque il fatto che RANDOM dà un'ulteriore «mescolata»; a rigor di logica, però, RND non richiederebbe più l'uso di RANDOM; è buona norma, in tutti i casi, abituarsi ad usarlo, anche per familiarizzarsi con tutte le istruzioni del BASIC; in tal modo, inoltre, i programmi fatti per il computer NE-Z80 girano anche su modelli diversi, senza doverli adattare aggiungendo RANDOM.

Il diagramma di sintassi di RANDOM è in FIGURA 46.

fig. 46 diagramma di sintassi di RANDOM



LINEE 1040-1050-1060

Dopo quello che abbiamo appena detto, il contenuto della linea 1040 dovrebbe essere chiaro; le due righe seguenti sono rimaste invariate rispetto a PITA1.

LINEA 1070

La linea 1070 è quella che fa accettare o scartare la coppia di numeri casuali A e B appena estratti.

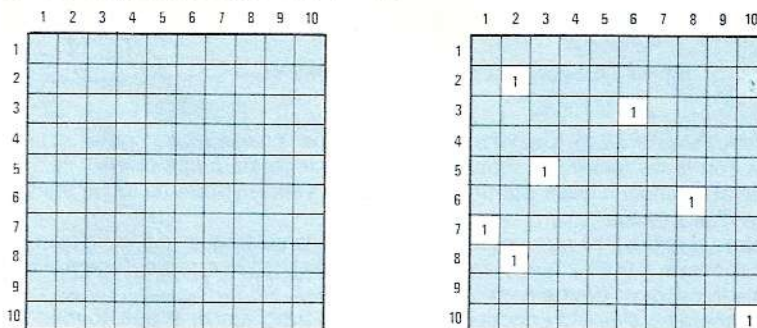
Immaginate di avere una matrice $N(A,B)$ a due dimensioni, dove A e B al massimo possono assumere il valore 10. Si tratta quindi di una tabella quadrata, con 10 righe e 10 colonne. In effetti ne avremmo a disposizione 11, ma quelle con indice zero non le utilizziamo. Dato che il valore massimo degli indici non supera 10, non è necessario fare una dichiarazione con l'istruzione DIM. Quando si lancia il programma, sappiamo che tutte le variabili vengono azzerate; perciò i 100 valori di $N(A,B)$ sono tutti nulli. Alla prima tornata, evidentemente, ogni valore di A e B va bene; infatti la riga 1070, non essendo verificato che $N(A,B)$ sia maggiore di zero, manda in esecuzione l'istruzione che segue ELSE, ossia pone la variabile indicizzata $N(A,B)$ uguale ad uno. Tanto per fissare le idee, se i numeri casuali usciti sono $A = 8$ e $B = 3$, essi sono accettati perché $N(8,3)$ vale zero, e questa stessa variabile viene posta uguale a 1.

Il meccanismo si ripete ad ogni ciclo; pertanto poco per volta le variabili $N(A,B)$ vengono poste uguali a 1.

Quando succede che ai numeri estratti A e B fa riscontro il valore 1 di $N(A,B)$, allora siamo nel caso in cui è verificata la prima parte della riga 1070 ($IF N(A,B) > 0$) ed il programma riparte dalla linea 1050, ossia con una nuova estrazione di A e B. Tale comportamento si ripete fino a che non si hanno dei valori non ancora usciti.

Per chiarire definitivamente il concetto usato, osservate la FIGURA 47.

fig. 47 due quadrati di $N(A,B)$



Il primo quadrato rappresenta la matrice così com'è subito dopo aver dato il RUN: ogni casella contiene il numero zero (noi le abbiamo lasciate in bianco per rendere più evidente la differenza col secondo quadrato). Dopo un certo numero di cicli la situazione potrebbe essere quella raffigurata nel secondo quadrato: in certe caselle, quelle corrispondenti ai valori di A e di B già usciti, compare il numero 1.

Le linee da 1080 a 1160, rispetto all'equivalente gruppo 1050-1080 di PITA1, contengono due nuove istruzioni: REVON e REVOFF.

REVON e REVOFF

Trattiamo assieme le istruzioni REVON e REVOFF perché sono concettualmente molto simili ed hanno lo stesso diagramma di sintassi, come si vede dalla FIGURA 48.

fig. 48 diagramma di sintassi di REVON e REVOFF



Quando un'istruzione di stampa su monitor è preceduta da REVON, i caratteri appariranno in negativo (REVON = REVERSE ON, ossia negativo inserito). Tutte le stampe seguenti continueranno ad essere in negativo, fino a che non si mette l'istruzione REVOFF (REVERSE OFF ossia negativo disinserito). Il meccanismo è molto semplice e non pone problemi di sorta.

(Abbiamo già fatto notare, in altra occasione, che se REVON e REVOFF vengono usate in linee con diverse istruzioni, qualora fossero seguite dai due punti non bisogna lasciare uno spazio prima di essi, altrimenti si ha un errore di sintassi. Questo vale solo per la prima versione del BASIC; in quelle successive il difetto è stato eliminato).

LINEE DA 1080 A 1160

La linea 1080 è uguale alla 1050 del programma PITA1; le istruzioni REVON e REVOFF sono usate in modo da presentare i fattori A e B del prodotto in reverse (negativo). Prima della stampa di ogni numero si dà REVON, subito dopo c'è REVOFF, ed il risultato è quello anzidetto.

Usando il programma noterete che un numero stampato in reverse è lungo due caratteri in più del numero delle cifre che lo compongono. Un carattere è prima del numero ed è riservato all'eventuale segno meno (mai visualizzato nel caso presente perché i numeri sono sempre e solo positivi); un altro carattere è in coda al numero e resta sempre vuoto (serve da separazione tra due numeri stampati consecutivamente, come abbiamo già spiegato).

La presenza del punto e virgola in tutte queste istruzioni di stampa serve per avere i vari dati uno di seguito all'altro, nella quinta riga video.

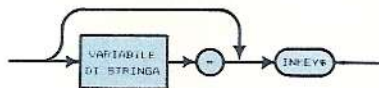
La linea 1160 stampa un punto interrogativo; in PITA1 non c'era perché l'istruzione INPUT R della linea 1090 ne causava la presenza automatica. In questo programma, invece, non si fa uso di INPUT, quindi dobbiamo provvedere noi alla stampa del punto interrogativo.

Tra poco troveremo l'istruzione INKEY\$; vediamola in dettaglio.

INKEY\$ (P)

Il diagramma di sintassi di INKEY\$ è in FIGURA 49.

fig. 49 diagramma di sintassi di INKEY\$



L'istruzione INKEY\$ funziona in questo modo: quando il computer la incontra, si mette in comunicazione con la tastiera ed assegna alla variabile indicata prima di INKEY\$ il carattere che si sta eventualmente premendo in quell'istante. Nel caso non venisse premuto alcun tasto, nella variabile va a finire una stringa nulla (ossia "").

Notare che INKEY\$ può essere usata anche da sola; vedremo vari esempi in futuro.

Così com'è INKEY\$ non è facilmente utilizzabile, in quanto l'operatore non può conoscere l'esatto istante in cui deve premere un tasto.

Supponiamo allora di concepire una parte di programma in questo modo:

```
130 A$=INKEY$ : IF A$="" THEN 130
140 PRINT A$
```

Passando per la linea 130 il calcolatore introita un carattere da tastiera; se, come accade sempre, non si sta già premendo un tasto, il carattere assunto è una stringa nulla. Allora il programma segue l'istruzione che c'è dopo IF, cioè ritorna alla linea 130. In pratica l'elaboratore resta in attesa di una battuta alla tastiera fino a che questa non viene realmente effettuata; in tal caso la condizione IF non è più verificata, ed il programma passa alla linea 140, dove si ordina la stampa di A\$: vedremo apparire il carattere premuto.

È importante notare che INKEY\$ non effettua la visualizzazione del carattere, ma si limita ad assumerlo.

Andando avanti vedremo parecchie applicazioni di questa istruzione.

LINEE 1170-1180

Alla riga 1170 inizia un loop che parte da 1 e può arrivare fino a 100000; le istruzioni eseguite all'interno di esso sono quelle comprese prima di NEXT. In questo caso, osservando la linea 1180, si vede che il NEXT si ha se è verificata la condizione:

```
IF R1$=""
```

Ciò significa che fino a che non si preme un tasto il computer continua a percorrere il loop. Quando invece viene premuto un tasto qualsiasi, la IF non è più verificata ed il programma passa alla linea successiva.

A cosa serve questo meccanismo? Non si tratta che di un contatore nella variabile I, la quale cresce in continuazione fino alla pressione di un tasto. È evidente allora che il valore di I all'uscita del loop è direttamente proporzionale al tempo impiegato a rispondere alla domanda posta dal calcolatore e ci servirà appunto per il computo del punteggio da assegnare alla risposta; rispondendo più in fretta si avrà diritto ad un numero di punti più alto e viceversa.

Il carattere alfanumerico introdotto andrà a finire nella variabile di stringa R1\$, come è specificato nella riga 1180.

LINEA 1190

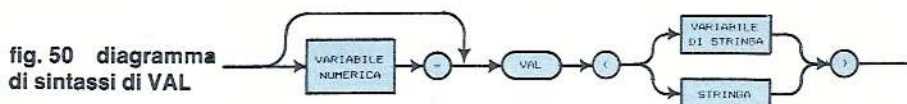
La riga 1190 è quella incaricata di controllare se si vuole porre termine al programma. Infatti se il tasto premuto è quello della lettera F la IF è soddisfatta e vengono eseguite le istruzioni che seguono la IF stessa: la variabile ES, che tiene il conto degli esercizi fatti, viene diminuita di uno, in quanto quello in corso non è stato fatto, poi si ha il rinvio alla linea 1350 con il salto incondizionato GOTO. In quella posizione inizia la visualizzazione della situazione, finita la quale il programma ha termine, come era stato chiesto premendo il tasto della lettera F.

A questo proposito facciamo notare che la lettera posta tra virgolette nella linea 1190 è una F minuscola; mettendo al suo posto la maiuscola, per porre termine al programma bisognerebbe premere contemporaneamente i tasti SHIFT e F (ricordare che le lettere sul monitor appaiono sempre maiuscole, ma che il computer sa distinguere le une dalle altre, a seconda di come sono state digitate. Con la scheda grafica questi problemi non esistono, in quanto le minuscole appaiono tali).

Vediamo, prima di proseguire, un'altra istruzione: VAL.

VAL (D/P)

L'istruzione VAL serve per trasformare una stringa in un numero. Il suo diagramma di sintassi è in FIGURA 50.



VAL è una variabile speciale riservata al sistema; deve essere seguita da una stringa posta tra parentesi. Essa può essere data sia sotto forma di variabile che di stringa posta tra virgolette.

Facciamo alcuni esempi, supponendo di eseguirli a livello di comandi diretti:

```
PRINT VAL("39")
A$="5432.08" : PRINT VAL(A$)
PRINT VAL("22 PERE")
PRINT VAL("PERE 22")
PRINT VAL("11-3")
```

Si richiede sempre la stampa in modo da poter vedere subito qual'è l'effetto di VAL. La prima istruzione fa stampare il numero 39, che da stringa è così passato a variabile numerica; la seconda riga mostra che VAL lavora anche sulle variabili di stringa: vedremo stampato il numero 5432.08. Fate in questo caso un esperimento significativo: chiedete ora la stampa di A\$ (PRINT A\$); vedrete ancora il numero precedente, ma spostato di un carattere a sinistra, all'inizio della riga. Ciò segnala che si tratta di una stringa; infatti il numero ad esso corrispondente inizia al secondo carattere della riga, per il posto che compete al segno.

La terza riga fornisce il numero 22, quella successiva il numero zero e l'ultima il numero 11. Da tutto questo si deduce che se una stringa contiene sia lettere che numeri, il suo valore numerico è zero se il primo carattere non è una cifra, mentre vale il numero che esiste prima di incontrare un qualsiasi carattere che non sia una cifra. Per questo motivo nell'ultimo caso non si ottiene 8.

LINEA 1200

Questa riga di programma contiene tre istruzioni, separate tra di loro dal carattere dei due punti. La prima di esse è la definizione della variabile numerica R1: ad essa va il valore della stringa R1\$, ossia il valore del primo carattere battuto in risposta alla domanda QUANTO FA...?

Segue poi un test di confronto che recita così: se R1 vale zero allora I1 è uguale al valore precedente più quello di I e poi va alla linea 1170.

Al solito è più facile da capire che da spiegare; diciamo subito che lo scopo di questa riga di programma è quello di rifiutare come primo carattere una battuta che non sia stata fatta sui tasti numerici da 1 a 9 (il tasto della lettera F minuscola è già stato riconosciuto dalla linea 1190). Ciò vuol dire che solo digitando una cifra tra 1 e 9 il programma prosegue, altrimenti pone nella variabile I1 il valore raggiunto da I (contatempo) e ritorna all'inizio del loop di conteggio e di attesa di un carattere da tastiera. In pratica,

allora, se il primo carattere digitato non è uno di quelli detti, esso viene rifiutato ed il conteggio del tempo continua. Il fatto che venga respinto anche il numero zero non crea problemi, in quanto non esiste una tabellina con quel risultato.

Vi facciamo notare che GOTO 1170 fa parte delle istruzioni eseguite solo se è verificata la condizione IF precedente.

LINEA 1210

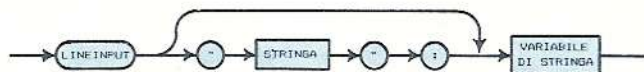
Il programma arriva a questa riga solo se il primo carattere premuto è un numero compreso tra 1 e 9, come abbiamo detto; in tal caso si ottiene sul video la visualizzazione di quel numero. Il punto e virgola serve per avere le stampe successive di seguito a questa: se battiamo un'altra cifra essa verrà scritta accanto alla prima.

Vediamo ora l'istruzione LINEINPUT.

LINEINPUT (P)

L'istruzione LINEINPUT funziona in modo analogo ad INPUT, che abbiamo già visto. Il diagramma di sintassi è in FIGURA 51.

fig. 51 diagramma di sintassi di LINEINPUT



Da esso si vede che è necessario aggiungere una variabile di stringa e che può essere messa una frase da visualizzare sul monitor esattamente come per INPUT.

Le differenze rispetto a INPUT sono: con LINEINPUT si possono introdurre solo variabili di stringa e non numeriche, altrimenti si ha errore; il punto interrogativo non viene stampato; la stringa introdotta può contenere anche virgole; non si può richiedere più di una stringa per volta.

LINEA 1220

Con la linea 1220 il computer aspetta l'introduzione di altri caratteri; se si preme RETURN la stringa R2\$ è nulla; ogni carattere viene accettato ed un eventuale errore di battitura può essere corretto (prima di premere RETURN) premendo il tasto backspace (= spazio all'indietro), ossia quello a destra della fila in alto.

In precedenza non lo avevamo detto, ma vi sarete resi conto che il primo numero introdotto non può più essere cambiato: ciò torna vantaggioso per evitare ... imbrogli nelle risposte: dato che il tempo è conteggiato fino al momento in cui si preme la prima cifra, si potrebbe pensare di digitare subito un numero per fermare il conteggio del tempo, e di modificare in un secondo momento questo valore. Ebbene, per come è fatto il programma, ciò non è possibile.

Il numero delle cifre introdotte con la riga 1220 può essere qualsiasi; in tal modo si arriva fino al 100, che compare nella Tavola Pitagorica. Anzi, modificando opportunamente le linee 1050 e 1060 si può usare il programma anche per chiedere prodotti i cui fattori sono maggiori di 10. In questo caso, però bisogna ricordare di inserire all'inizio del programma una riga contenente la dichiarazione DIM per la matrice N(A,B).

LINEA 1230

Con la riga 1230 si deposita nella variabile numerica R il valore della stringa somma di R1\$ ed R2\$. Così, se R1\$ vale 6 ed R2\$ vale 3, la stringa somma è 63 (non 9, in quanto non si tratta di numeri, ma di caratteri alfanumerici). Quindi VAL("63") diventa il numero 63 e va nella variabile numerica R.

LINEA 1240

Con la linea di programmazione 1240 si ordina al computer di lasciare una riga vuota nella visualizzazione dei dati sul monitor.

LINEE 1250-1260

Per mezzo delle righe 1250 e 1260 si analizza la risposta data. Infatti la 1250 fa scrivere sul monitor la dicitura ESATTO solo se il valore della variabile numerica R è uguale al prodotto di A per B. In caso contrario il programma passa alla linea seguente dove, se R è diverso dal prodotto medesimo, vengono eseguite tre istruzioni. Prima di tutto si incrementa di uno il valore della variabile TE: essa tiene il conto del totale degli errori commessi, e viene quindi incrementata solo se si passa per la linea 1060. Poi si pone la variabile S uguale a 1: essa serve per controllare, successivamente, se è stato commesso un errore oppure no, ai fini dell'assegnazione del punteggio. Per ultimo, viene visualizzato sul monitor il responso SBAGLIATO, seguito dal risultato esatto del prodotto di A per B: chi non lo sa ha modo di impararlo! Infatti l'ultima istruzione della linea 1060 ordina di stampare SBAGLIATO seguito dai due punti; segue poi il valore di A, quindi la lettera X minuscola, il valore di B, il segno uguale e, per finire, il valore del prodotto di A per B. Ogni dato è di seguito al precedente per l'interposizione dei punti e virgola.

Le due linee in questione potrebbero anche essere scritte nel modo seguente:

```
1250 IF R=A*B THEN PRINT "ESATTO" : GOTO 1270
1260 TE=TE+1 : S=1 : PRINT "SBAGLIATO: ";A;"X";B;"=";A*B
```

I risultati ottenuti sarebbero identici. Come vedete, alla linea 1250 è stata aggiunta l'istruzione di salto incondizionato alla riga 1270, per non fare percorrere (nel caso che R sia esatto) anche la 1260. Infatti in quest'ultima è stata tolta la condizione

```
IF R<>A*B
```

poiché è ovvio che se non è vera l'ipotesi della linea 1250 è senz'altro verificata quella scritta ora.

Fare in un modo o nell'altro è solo una questione di gusti; c'è da dire però che se i test da fare fossero parecchi, ad esempio 25, il programma scorrerebbe più veloce con una soluzione analoga alla seconda: infatti, trovata la condizione verificata, verrebbero saltate tutte le altre con GOTO, anziché scartarle una per una.

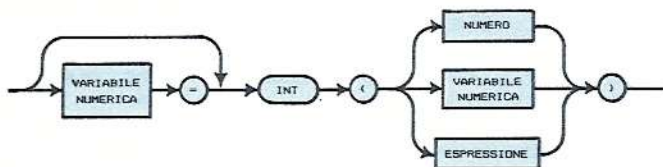
Tra poco troveremo una nuova istruzione: INT, quindi passiamo alla sua spiegazione in dettaglio.

INT (D/P)

L'istruzione INT appartiene, concettualmente, alle funzioni aritmetiche. Essa opera sui numeri e serve per ricavare, da un numero qualsiasi, la sola parte intera.

Il diagramma di sintassi di INT è in FIGURA 52.

fig. 52 diagramma di sintassi di INT



INT è una variabile speciale riservata al sistema; va seguita da un numero tra parentesi e restituisce il numero intero immediatamente inferiore a quello dato. Vediamo qualche esempio:

```
INT (67.99)
INT (67.03)
INT (-29.7291)
```

La prima espressione fornisce il numero 67, come quella dopo; l'ultima dà -30, in ossequio al comportamento descritto di fornire il numero intero inferiore al valore dato.

Dal diagramma di sintassi si vede che l'istruzione INT può essere preceduta da una variabile numerica seguita dal segno =; in questo modo il valore dato da INT viene depositato in quella variabile, per essere poi usato al momento opportuno.

Si vede anche che al posto del numero, tra le parentesi, può essere messa una variabile numerica oppure un'espressione algebrica. Vedremo tra poco un esempio di queste possibilità.

LINEE DA 1270 A 1340

Le righe da 1270 a 1340 calcolano il punteggio, lo visualizzano per un certo lasso di tempo, poi rimandano il programma all'inizio. Andiamo con ordine. Nella 1270 c'è la definizione della variabile T; essa vale la parte intera del numero che si ottiene sommando I con I1 e dividendo il risultato per 120. Occorre prestare attenzione all'uso delle parentesi; se la somma I + I1 non fosse stata racchiusa tra di esse, soltanto I1 verrebbe diviso per 120: il risultato sarebbe diverso da quello che si intendeva calcolare, ma non si avrebbe alcuna segnalazione d'errore essendo l'espressione formalmente corretta.

Inoltre il numero delle parentesi aperte deve essere sempre uguale a quello delle chiuse; in caso contrario si avrebbe una segnalazione d'errore.

Il motivo per cui si è diviso per 120 il conteggio totale del tempo è di avere una T all'incirca uguale al numero di secondi impiegati a rispondere, ed è stato ottenuto per tentativi. Infatti tale conteggio è legato al tipo di operazioni che sono fatte all'interno del loop FOR...NEXT delle linee 1170-1180, quindi varia da programma a programma.

Nella linea 1280 si introduce la variabile P, definendola come la differenza tra il numero 3 ed il valore appena calcolato di T. Questa scelta è giustificata dal fatto che una risposta può essere considerata accettabile se data entro 3 secondi; in quel caso P vale zero, ossia non si assegnano punti.

Al contrario, se il tempo impiegato è minore o maggiore di 3 secondi, si dà un punteggio positivo (3 al massimo) o negativo (senza limiti). Da notare che se non si preme un tasto numerico entro circa 13 minuti (corrispondente al tempo impiegato per percorrere il loop FOR...NEXT della linea 1170 fino al limite superiore di 100000), si va alla riga 1200 e, dato che R1 varrà zero, si ricomincerà un nuovo loop, e così di seguito fino alla prima cifra battuta.

La linea 1290 dice: se S = 1 allora il punteggio viene diminuito di 10. Ossia si analizza il valore della variabile che controlla se si è passati dalla linea 1260 (percorsa se si dà una risposta errata); in caso affermativo si ha una penalizzazione di 10 punti. L'istruzione non viene eseguita se S vale zero. Ora dovrebbe essere chiaro il motivo dell'azzeramento di S e di I1 alla linea 1010; se non lo facessimo il programma darebbe risultati errati: i valori di quelle due variabili devono essere decisi all'interno del ciclo ripetitivo delle domande, volta per volta.

La riga 1300 tiene aggiornata la variabile TP che accumula il punteggio assegnato ad ogni tornata; il suo valore esprime sempre il totale dei punti fino a quel momento.

Le linee 1310 e 1320 servono per visualizzare rispettivamente le scritte PUNTI e TOTALE, seguite ciascuna dal valore che le compete.

Con la linea 1330 si realizza un loop di ritardo, per dare il tempo di leggere il responso ed il punteggio: infatti tra FOR e NEXT non ci sono istruzioni.

Con la linea 1340 si torna all'inizio, ma solo se il numero dell'esercizio appena fatto è minore di 100. Quando invece si è arrivati a quel valore limite, l'istruzione viene ignorata e si prosegue.

LINEE DA 1350 A 1450

Si arriva al blocco di istruzioni compreso tra le linee 1350 e la fine, solo in due casi: se si preme il tasto della lettera F per terminare l'esercizio, o se sono state date tutte e 100 le combinazioni possibili della Tavola Pitagorica. Si comincia col cancellare lo schermo, poi si scrive in reverse la parola SITUAZIONE. Si toglie quindi la scrittura in negativo e si lasciano, con la linea 1390, due righe video.

Vengono poi stampati il numero totale degli esercizi fatti, il numero di errori commessi ed il punteggio conseguito. Dopo aver lasciato altre due righe vuote, compare la scritta CIAO ed il computer torna a livello di comandi diretti perché incontra la parola chiave END. Vediamola.

END (P)

L'istruzione END serve per porre fine al programma. Il suo diagramma di sintassi è riportato in FIGURA 53.

fig. 53 diagramma di sintassi di END



Quando incontra END il computer arresta l'esecuzione del programma e torna allo stato di comandi diretti, segnalato dalla scritta READY.

In molti elaboratori tale istruzione è necessaria per non avere segnalazioni d'errore. Nel calcolatore di

NUOVA ELETTRONICA si usa, in pratica, solo per separare eventuali subroutines poste in coda al programma dalla fine logica del medesimo; in caso contrario, in chiusura delle operazioni, si entrerebbe nella prima subroutine e si avrebbe una segnalazione d'errore incontrando RETURN. Riprenderemo l'argomento quando se ne presenterà l'occasione.

Siamo così arrivati alla fine del programma PITA2. Se provate a rileggere lentamente il listato, tutto dovrebbe esservi chiaro: almeno noi lo speriamo.

TERZA VERSIONE DEL PROGRAMMA PITAGORA

Ora ci proponiamo di rendere abbastanza elaborato il nostro programma delle tabelline, trasformandolo ulteriormente in uno più complesso. Faremo uso anche di un archivio su disco, quindi le modifiche torneranno utili per imparare la gestione dei file. Cominceremo dal tipo più semplice, quello sequenziale.

Per fissare le idee, elenchiamo le modifiche che intendiamo realizzare:

- 1 - creazione di un file per registrarvi gli esercizi di ogni alunno
- 2 - ogni volta che un alunno si accinge a cominciare l'esercitazione, il programma deve andare a trovare i suoi dati (se ci sono) e basarsi su di essi per porre le nuove domande
- 3 - possibilità di proseguire oltre i 100 esercizi
- 4 - superato tale limite, gli esercizi richiesti da quel momento in poi devono essere solo quelli con punteggio più basso, ossia quelli sbagliati un maggior numero di volte o nei quali si è impiegato un tempo elevato per rispondere
- 5 - possibilità di avere su stampante l'elenco delle tabelline sbagliate, oppure di tutte quelle richieste fino a quel momento
- 6 - andare ad aggiornare il file dell'alunno quando decide di terminare gli esercizi.

Il listato della versione definitiva del programma della Tavola Pitagorica è in FIGURA 54; la FIGURA 55 riporta il diagramma di flusso, mentre in FIGURA 56 è rappresentato l'elenco completo dei riferimenti.

fig. 54 listato di PITAGORA

```

1000 ' =====
1010 '          TAVOLA PITAGORICA                      1982 NE-R.C.
1020 ' =====
1030 CLS
1040 ONERROR GOTO 2280
1050 PRINT @192,"";
1060 INPUT "COME TI CHIAMI";NOME$
1070 A$="ASPETTA UN PO", "+NOME$
1080 PRINT @192,A$
1090 ' ----- RICERCA DEI DATI PRECEDENTI -----
1100 OPEN "I",1,NOME$
1110   FOR J=1 TO 10
1120     FOR K=1 TO 10
1130       INPUT #1,N(J,K),P(J,K),E(J,K)
1140     NEXT K
1150   NEXT J
1160 INPUT #1,ES,MIN,TE,TP
1170 CLOSE
1180 ' ----- FORMULAZIONE DEL NUOVO ESERCIZIO -----
1190 CLS
1200 S=0 : I1=0 : ES=ES+1
1210 PRINT "ESERCIZIO #";ES : PRINT @18,"(F PER FINIRE)"
1220 PRINT @128,"STO PENSANDO!"
1230 RANDOM
1240 A=RND(10)
1250 B=RND(10)
1260 IF ES<101 AND N(A,B)>0 THEN 1240
1270 IF ES>100 AND P(A,B)-MIN>5 THEN 1240
1280 PRINT @128,"QUANTO FA ";
1290 REVON
1300 PRINT A;
1310 REVOFF
1320 PRINT " PER ";
1330 REVON
1340 PRINT B;
1350 REVOFF
1360 PRINT " ? ";
1370   FOR I=1 TO 100000
1380     R1$=INKEY$ : IF R1$="" THEN NEXT I
1390   IF R1$="f" THEN ES=ES-1 : CLS : GOTO 1710
1400   R1=VAL(R1$) : IF R1=0 THEN I1=I1+1 : GOTO 1380
1410 PRINT R1$;
1420 LINEINPUT R2$
1430 R=VAL(R1$+R2$)
1440 PRINT
1450 ' ----- VALUTAZIONE DELLA RISPOSTA -----
1460 IF R=A*B THEN PRINT "ESATTO"
1470 IF R<>A*B THEN TE=TE+1 : S=1 : E(A,B)=E(A,B)+1 : PRINT "SBAGLIATO: ";A;"x";
1480   B;"=";A*B
1490   T=INT((I+I1)/120)
1500   P=3-T
1510 IF S=1 AND ES<101 THEN P=P-10
1520 IF S=1 AND ES>100 THEN P=P-5
1530 IF S=0 AND ES>100 THEN P=P+5
1540 TP=TP+P
1550 PRINT "PUNTI :";P
1560 PRINT "TOTALE :";TP
1570 GOSUB 2240
1580 N(A,B)=N(A,B)+1
1590 P(A,B)=P(A,B)+P
1590 IF ES<100 THEN 1190

```

```

1600 ' ----- RICERCA DEL PUNTEGGIO MINIMO -----
1610 MIN=P(1,1)
1620   FOR J=1 TO 10
1630     FOR K=1 TO 10
1640       IF MIN>P(J,K) THEN MIN=P(J,K)
1650     NEXT K
1660   NEXT J
1670 IF ES>100 GOTO 1190
1680 CLS
1690 PRINT "REGISTRAZIONE E STAMPA DEI PRIMI 100 ESERCIZI." : PRINT
1700 ' ----- VISUALIZZAZIONE DEI DATI -----
1710 REVON
1720 PRINT " SITUAZIONE DI ";NOME$;" "
1730 REVOFF
1740 PRINT : PRINT
1750 PRINT "TOTALE ESERCIZI   :";ES
1760 PRINT "TOTALE ERRORI    :";TE
1770 PRINT "TOTALE PUNTEGGIO :";TP
1780 ' ----- REGISTRAZIONE DEI DATI -----
1790 OPEN "O",1,NOME$
1800   FOR J=1 TO 10
1810     FOR K=1 TO 10
1820       PRINT #1,N(J,K);P(J,K);E(J,K);
1830     NEXT K
1840   NEXT J
1850 PRINT #1,ES;MIN;TE;TP
1860 CLOSE
1870 ' ----- MENU' -----
1880 CLS
1890 REVON: PRINT @13," MENU' " : PRINT : PRINT
1900 PRINT " F "; : REVOFF: PRINT " > PER FINIRE"
1910 PRINT : REVON: PRINT 1; : REVOFF: PRINT " > STAMPA DEI SOLI ERRORI"
1920 PRINT : REVON: PRINT 2; : REVOFF: PRINT " > STAMPA COMPLETA"
1930 PRINT : REVON: PRINT 3; : REVOFF: PRINT " > PER CONTINUARE"
1940 PRINT : PRINT : PRINT @461,"SCEGLI"
1950 M$=INKEY$ : IF M$="" THEN 1950
1960 IF M$="f" THEN 2200
1970 M=VAL(M$)
1980 IF M<1 OR M>3 THEN 1950
1990 IF M=3 THEN 1190
2000 ' ----- STAMPA DEI RISULTATI -----
2010 CLS
2020 PRINT "STAMPA DEI RISULTATI";
2030 LPRINT CHR$(27);CHR$(69);"RISPOSTE "; : IF M=1 THEN LPRINT "SBAGLIATE "; :
PRINT " SBAGLIATI"
2040 LPRINT "DI ";CHR$(14);NOME$
2050 LPRINT CHR$(27);CHR$(70)
2060 LPRINT TAB(20);"NUM.USCITE";TAB(39);"NUM.ERRORI";TAB(59);"PUNTEGGIO"
2070   FOR J=1 TO 10
2080     LPRINT : LPRINT
2090     FOR K=1 TO 10
2100       IF M=1 AND N(J,K)>0 AND E(J,K)>0 THEN 2120
2110       IF M=2 AND N(J,K)>0 THEN 2120 ELSE 2130
2120       LPRINT TAB(3);J;"x";K;TAB(24);N(J,K);TAB(43);E(J,K);TAB(62);P(J,K)
2130     NEXT K
2140   NEXT J
2150 LPRINT : LPRINT
2160 LPRINT "TOTALE ESERCIZI   :";ES
2170 LPRINT "TOTALE ERRORI    :";TE
2180 LPRINT "TOTALE PUNTEGGIO :";TP
2190 GOTO 1880
2200 CLS
2210 PRINT "CIAO, ";NOME$;" !"
2220 GOTO 2220
2230 ' ----- ROUTINES DI RITARDO -----
2240   IF S=1 THEN FOR I=1 TO 1200 : NEXT
2250   FOR I=1 TO 1200 : NEXT
2260 RETURN
2270 ' ----- ROUTINE CORREZIONE ERRORE RICERCA FILE DATI INESISTENTE -----
2280 RESUME 1170

```

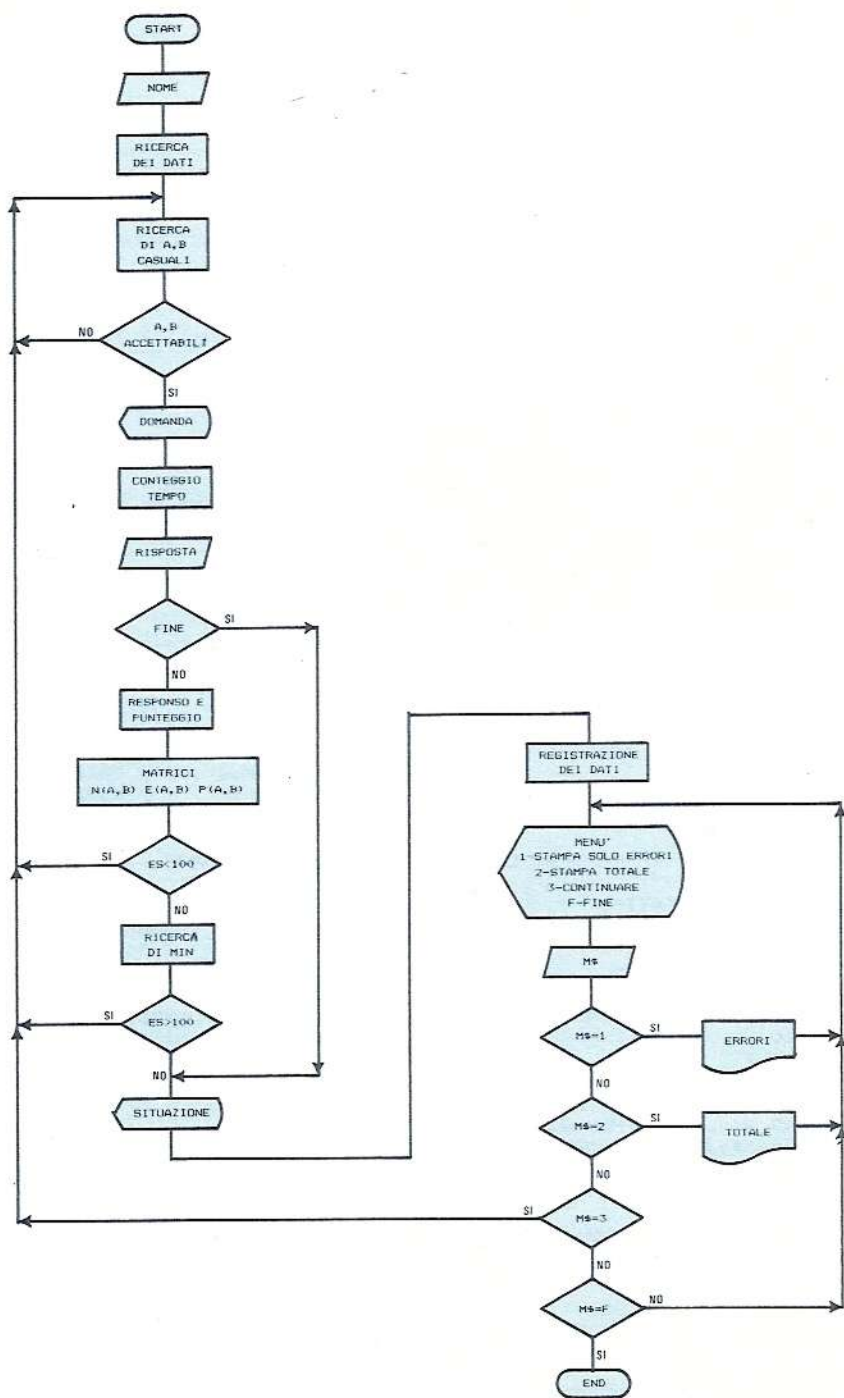



fig. 55 diagramma di flusso di PITAGORA

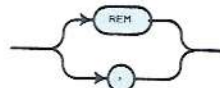
0	1200/2 1260 1400	B	1250 1260 1270 1340
	1520 2100/2 2110		1460 1470/5 1570/2
1	1100 1110 1120 1130		1580/2
	1160 1200 1370 1390	E	1130(1470(2 1820(
	1470/3 1500 1510		2100(2120(
	1570 1610/2 1620	ES	1160 1200/2 1210
	1630 1790 1800 1810		1260 1270 1390/2
	1820 1850 1910 1980		1500 1510 1520 1590
	2030 2070 2090 2100		1670 1750 1850 2160
	2240/2 2250	FF	1310 1350 1730 1900
2	1920 2110		1910 1920 1930
3	1490 1930 1980 1990	I	1370 1380 1400 1480
	2120		2240 2250
5	1270 1510 1520	I1	1200 1400/2 1480
10	1110 1120 1240 1250	J	1110 1130/3 1150
	1500 1620 1630 1800		1620 1640/2 1660
	1810 2070 2090		1800 1820/3 1840
13	1890		2070 2100/2 2110
14	2040		2120/4 2140
18	1210	K	1120 1130/3 1140
20	2060		1630 1640/2 1650
24	2120		1810 1820/3 1830
27	2030 2050		2090 2100/2 2110
39	2060		2120/4 2130
43	2120	M	1950/\$2 1960/\$ 1970
59	2060		1970/\$ 1980/2 1990
62	2120		2030 2100 2110
69	2030	MI	1160 1270 1610 1640/2
70	2050		1850
100	1270 1510 1520 1590	N	1130(1260(1290
	1670		1330 1570(2 1710
101	1260 1500		1820(1890 1910 1920
120	1480		1930 2100(2110(
128	1220 1280		2120(
192	1050 1080	ND	1060/\$ 1070/\$ 1100/\$
461	1940		1720/\$ 1790/\$ 2040/\$
1170	2280		2210/\$
1190	1590 1670 1990	P	1130(1270(1490
1200	2240 2250		1500/2 1510/2 1520/2
1240	1260 1270		1530 1540 1580 1580(2
1380	1400		1610(1640(2 1820(
1710	1390		2120(
1880	2190	R	1430 1460 1470
1950	1950 1980	R1	1380/\$2 1390/\$ 1400/2
2120	2100 2110		1400/\$ 1410/\$ 1430/\$
2130	2110	R2	1420/\$ 1430/\$
2200	1960	S	1200 1470 1500 1510
2220	2220		1520 2240
2240	1560	T	1480 1490
2280	1040	TE	1160 1470/2 1760
A	1070/\$ 1080/\$ 1240		1850 2170
	1260 1270 1300 1460	TP	1160 1530/2 1550
	1470/5 1570/2 1580/2		1770 1850 2180

fig. 56 riferimenti di PITAGORA

REM (P)

L'istruzione REM è molto usata; con essa si possono inserire dei commenti (REMARKS) nel punto desiderato di un programma; il loro scopo è quello di fare da promemoria e da spiegazione a righe o porzioni del programma stesso. Il diagramma di sintassi di REM è in FIGURA 57.

fig. 57 diagramma di sintassi di REM



Come si vede, REM può essere sempre abbreviato col carattere dell'apostrofo. L'unica differenza tra i

due modi è che in linee di istruzioni multiple REM deve essere preceduto dai due punti, mentre la sua abbreviazione ne può fare a meno. Da tenere presente che non bisogna mai aggiungere, nella stessa riga, istruzioni in coda a REM (o alla sua abbreviazione); esse verrebbero sempre ignorate perché il programma, appena trova l'annotazione, salta immediatamente alla linea successiva.

Dopo REM si può mettere tutto quello che si vuole; i caratteri saranno maiuscoli o minuscoli a seconda di come sono stati digitati, esattamente come accade all'interno delle stringhe.

ONERRORGOTO (P)

L'istruzione ONERRORGOTO torna estremamente utile per fare in modo che un programma non termini a causa di un errore; se esso avviene, infatti, si va alla linea indicata dall'istruzione. Da quel punto deve iniziare la routine di correzione dell'errore; caso per caso occorre analizzare il programma 12. Queste, in ogni caso, devono sempre terminare con una istruzione RESUME (che vedremo tra poco).

Il diagramma di sintassi di ONERRORGOTO è in FIGURA 58.

fig. 58 diagramma di sintassi di ONERRORGOTO

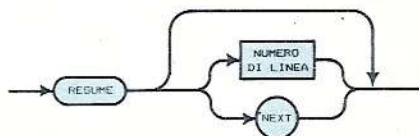


Normalmente la routine di correzione dell'errore viene messa in fondo al programma, ma ciò non è obbligatorio.

RESUME (P)

L'istruzione RESUME deve viaggiare sempre in coppia con ONERRORGOTO; se il programma la incontra senza aver ricevuto ONERRORGOTO, segnala errore. Il diagramma di sintassi di RESUME è in FIGURA 59.

fig. 59 diagramma di sintassi di RESUME



RESUME (= riprendi da) può essere dato senza specificare nient'altro: in tal caso il programma ritorna alla linea in cui si è verificato l'errore.

Se invece RESUME è seguito da un numero di riga, il programma riprende da quel punto.

RESUME NEXT ordina al computer che il programma deve ripartire dalla riga successiva a quella d'errore.

Il programmatore, per fare un uso appropriato delle istruzioni ONERRORGOTO e RESUME, deve prevedere in anticipo quali potranno essere gli errori possibili in un certo programma; solo così potrà effettuare i ritorni in modo adatto. Vedremo diversi esempi, man mano che se ne presenterà l'occasione.

ERROR (P)

Dato che siamo in argomento, esaminiamo anche la funzione dell'istruzione ERROR, il cui diagramma di sintassi è in FIGURA 60.

fig. 60 diagramma di sintassi di ERROR



ERROR si usa per provare il perfetto funzionamento della routine di correzione degli errori. Infatti inserendo nelle posizioni opportune del programma le istruzioni ERROR seguite da un numero, il computer si comporta esattamente come se avvenisse proprio quel tipo di errore; quindi va alla riga specificata

con ONERRORGOTO e rientra in quella indicata da RESUME, permettendo così di verificare se la routine prevista per quel tipo di errore dà i risultati sperati.

Ad esempio, osserviamo la seguente ipotesi di programma:

```
100 ON ERRORGOTO 3000
110 .....
120 .....
320 .....
330 INPUT B
340 C=A/B
341 ERROR 11
350 .....
360 .....
3000 IF ERR/2+1=11 THEN PRINT "DIVISIONE PER ZERO" : RESUME 330
3010 IF ...
```

Nella linea 330 si deve introdurre il valore di B; nella 340 si assegna alla variabile C il valore della divisione di A per B. È allora evidente che se si attribuisce a B il valore zero, il programma va in errore alla riga 340 perché non è possibile dividere un numero per zero. Senza la routine di correzione degli errori si uscirebbe dal programma; nel nostro esempio, invece, in un caso del genere si passa alla linea 3000 dove se B vale zero, il programma stampa la dicitura DIVISIONE PER ZERO e riprende dalla riga 330, ossia viene richiesto il valore di B.

Dopo la linea 3000 ve ne saranno altre adibite alla correzione di errori diversi.

Esaminiamo ora la funzione della riga 341: essa serve per verificare la routine d'errore anche se B non vale zero. Col programma così fatto si passa sempre per la 3000. È evidente che la riga 341 si scrive in fase di messa a punto del programma, poi bisogna toglierla; in pratica serve nel DEBUG.

La linea 3000 dell'esempio contiene la parola chiave ERR; essa, assieme a ERL, serve per la gestione degli errori. Vediamole.

ERR (D/P)

ERR è una variabile speciale riservata al sistema; in essa, in caso d'errore, il computer deposita un numero che è legato al codice del tipo d'errore per mezzo della formula:

$ERR/2+1$

Abbiamo visto, ad esempio, che l'errore DIVISION BY ZERO ha codice 11; tale valore si ottiene appunto applicando la formuletta precedente.

Per l'elenco completo dei tipi d'errore e dei loro codici vedere la tabella apposta.

ERL (D/P)

Anche ERL (da ERROR LINE linea d'errore) è una variabile speciale riservata al sistema; in essa è contenuto il numero della linea di programma che ha causato l'errore.

Usata in congiunzione con ERR, nelle routine di correzione degli errori, è estremamente utile per gestire i vari tipi di errore.

Se riprendiamo l'esempio appena visto, possiamo concepire la riga 3000 in questo modo:

```
3000 IF ERL=340 AND ERR/2+1=11 THEN RESUME 330
```

Dopo aver fatto un programma, in sede di debug si constaterà, molto probabilmente, che esso non gira se si verificano certe circostanze. In questi casi il computer esce dal programma e visualizza la dicitura del tipo d'errore avvenuto. Per conoscere il suo codice e la linea che lo ha causato basta richiedere (a livello di comandi diretti):

```
PRINT ERR/2+1, ERL
```

I due numeri ottenuti sono quelli da usare nella routine di correzione degli errori, con sistemi analoghi a quelli già visti.

Passiamo ora ad esaminare riga per riga il contenuto del programma PITAGORA.

LINEE DA 1000 A 1020

Ogni programmatore dovrebbe prendere l'abitudine di documentare il suo lavoro, inserendo righe di commento nei punti più significativi del programma. Tale procedura porta via un po' di tempo ed occupa

una porzione di memoria centrale; però risulta non solo di grande utilità per chiunque legga il listato del programma, ma costituisce anche una documentazione permanente che lui stesso avrà modo di apprezzare.

Col passare del tempo, infatti, nuovi programmi vanno ad aggiungersi a quelli precedenti e dopo un po', oltretutto, la memoria non è certo sufficiente per ricordare le varie fasi di ogni programma.

Capita spesso di dover apportare delle modifiche ad un lavoro fatto magari qualche mese o, peggio ancora, qualche anno prima; ebbene, la maggior parte del tempo viene persa per cercare di capire una parte di programma che, riletta a distanza di tempo, non ci risulta affatto chiara.

La cosa migliore sarebbe quella di rispettare la seguente procedura:

- A - inserire linee di commento nel punto più significativi del programma;
- B - allegare ad esso il relativo diagramma di flusso; può essere fatto anche a posteriori (in tal caso si scopre quasi sempre che è opportuno fare alcune modifiche che semplificano e rendono più lineare il programma già fatto);
- C - allegare la stampa dei riferimenti, che tornerà molto utile per successive elaborazioni;
- D - accompagnare il tutto con qualche riga di spiegazione (ad esempio l'uso cui sono destinate le variabili usate, oppure la struttura che abbiamo assegnato ai file random o sequenziali utilizzati).

Tutto questo, che può sembrare pura pignoleria, in effetti parte come ordine formale e poi si trasforma a poco a poco in ordine mentale: ci si accorge di riuscire a schematizzare meglio i problemi, di semplificarli e di ordinarli, con grande vantaggio per la struttura ed il buon funzionamento finale di ogni programma.

Le prime tre linee di PITAGORA sono quelle che gli danno l'intestazione: già dall'inizio dobbiamo sapere l'argomento trattato.

Abbiamo detto che dopo REM (l'apostrofo è la sua abbreviazione) si possono usare tutti i caratteri desiderati, come all'interno di una stringa tra virgolette. Ognuno si creerà uno stile personale; quello suggerito è molto semplificato ma, nel contempo, efficace.

LINEE 1030-1040

Dopo la cancellazione dello schermo, si dice al computer di andare alla linea 2280 in caso di errori. Vedremo tra poco il motivo che ha imposto una routine di correzione d'errore.

LINEE DA 1050 a 1080

La linea 1050 può sembrare strana, ma non lo è; essa dice al calcolatore di andare all'inizio della settima riga video (il carattere numero 192 è appunto il primo di quella riga, come si vede dalla mappa video), e di stampare una stringa nulla. Evidentemente, fino a questo punto, sul monitor non si vede l'effetto di questa istruzione; tuttavia, notando che essa termina con un punto e virgola, si può immaginare il motivo per cui è stata messa: la stampa seguente avverrà a partire da quel punto.

Infatti alla linea 1060 troviamo un'istruzione di INPUT, utilizzata facendola precedere dalla frase esplicativa COME TI CHIAMI. Senza di essa si vedrebbe sul monitor un punto interrogativo (indice che bisogna introdurre un dato da tastiera), ma l'operatore non potrebbe sapere di quale dato si tratta. L'uso dell'istruzione INPUT dovrebbe essere ormai chiaro; facciamo solo notare che la variabile richiesta è di tipo stringa e che il suo identificatore è NOME. Tale scelta è di tipo mnemonico, per riconoscere, alla sola lettura, l'uso cui è destinata. Ricordate comunque che il computer ignora i caratteri oltre il secondo; la variabile in questione equivale in tutto e per tutto a NOS.

Il punto interrogativo non compare nella frase tra virgolette perché viene visualizzato automaticamente dall'istruzione INPUT.

Nella linea 1070 si trova la definizione della variabile di stringa A\$, che vale la dicitura tra virgolette più il nome che è stato introdotto con la riga precedente. Si può notare che le stringhe si possono sommare le une alle altre; non è invece possibile sottrarle né, tanto meno, moltiplicarle o dividerle. È invece lecito usare con esse operatori di confronto (ossia = < > e le loro varie combinazioni); ne vedremo alcune applicazioni in altri programmi.

Se supponiamo che il nome assegnato a NOME\$ sia ELISA, la variabile A\$ contiene la frase ASPETTA UN PO', ELISA. Lo spazio dopo la virgola è contenuto nella stringa tra virgolette.

Fino a questo momento abbiamo definito A\$, ma non è stato ancora utilizzato. Ciò avviene alla linea 1080; A\$ viene visualizzato, sempre all'inizio della settima riga video. Questo significa che la frase precedente COME TI CHIAMI, seguita dal punto interrogativo e dal nome introdotto, viene ricoperta dalla nuova dicitura, senza dover cancellare lo schermo. Nell'applicare tale procedimento bisogna solo fare attenzione alla lunghezza delle due scritte: se la seconda fosse più corta della prima, rimarrebbe visibile una parte di quest'ultima. L'inconveniente però può essere risolto aggiungendo in coda ad A\$ un numero di spazi sufficienti.

Nelle righe seguenti troveremo le istruzioni che servono per gestire un file di tipo sequenziale; vediamole prima in dettaglio, facendole precedere da una ampia spiegazione del comando BASIC del DOS.

BASIC

Descrivendo i comandi DOS non abbiamo approfondito molto il discorso riguardante BASIC, ovvero il comando DOS che permette di passare in ambiente BASIC. Lo facciamo ora, cominciando dal diagramma di sintassi di FIGURA 61.

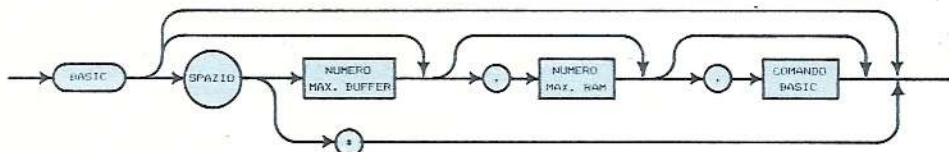


fig. 61 diagramma di sintassi di BASIC

Ovviamente tutto quello che diremo vale partendo dal livello DOS.

Il comando BASIC può essere dato da solo; questa è la possibilità rappresentata dal ramo più alto del diagramma ed è quella normalmente usata. È possibile però dare anche alcune opzioni, quattro per la precisione. In questo caso è indispensabile fare seguire la parola BASIC da uno spazio, e questa particolarità è ben visibile nel diagramma di sintassi.

Vediamo innanzitutto il seguente comando:

BASIC *

Esso serve per passare da DOS a BASIC ripristinando il programma precedente; questo però è possibile solo se fossimo partiti dal livello BASIC per passare al DOS col seguente comando:

CMD "S"

Arrivati al DOS per questa via è possibile tornare al BASIC recuperando il programma che era presente in memoria, impartendo appunto il comando visto. Appena passato in BASIC, il computer esegue automaticamente il listato del programma, proprio per mostrare di averlo recuperato. Questa procedura salva il programma precedente solo se non si è ancora eseguito, a livello DOS, alcun comando.

Vediamo ora un esempio che mostra per esteso l'insieme delle altre tre alternative:

BASIC 7, 53000, PRINT MEM

Ripetiamo ancora che in questo caso lo spazio dopo BASIC è obbligatorio. Il numero 7 ordina al computer di riservare 7 buffer per le comunicazioni tra calcolatore e dischi.

Cos'è un BUFFER? È una porzione di memoria che viene riservata per il deposito temporaneo di dati; nel caso della comunicazione tra dischi e computer, il buffer fa da transito per i dati scambiati da questi due elementi. BUFFER in inglese significa CUSCINETTO: infatti rappresenta un'area di memoria che fa da cuscinetto tra elaboratore e periferiche. Ad esempio esiste anche un buffer di tastiera, dove vengono temporaneamente parcheggiati i dati da essa provenienti.

Il sistema riserva in modo del tutto automatico un buffer per lo scambio di programmi col disco: ogni registrazione, lettura o fusione di programmi (col comando MERGE) avviene utilizzando questo buffer; esso esiste sempre ed è indipendente dai buffer creati dall'utente col comando BASIC del DOS, quindi è sempre disponibile per l'emissione o l'immissione di programmi.

Allora, il primo numero del comando precedente serve per indicare quanti buffer vogliamo creare per gestire i file-archivio. Queste aree di memoria sono prese da quella a disposizione dell'utente, quindi la creazione di ogni buffer comporta un restringimento della memoria utente. Per la precisione, ogni buffer occupa 290 byte di memoria RAM (256 per il buffer vero e proprio più 34 per il blocco di controllo). Meglio quindi non crearne in numero troppo elevato; il sistema, se viene inizializzato caricando il BASIC senza le opzioni, riserva automaticamente 3 buffer per i file. Questo quindi è il valore di default per quel parametro. Al massimo è possibile creare 15 buffer contemporanei per lo scambio dei dati coi dischi; il valore minimo è zero.

Se si prova a caricare il BASIC in questo modo:

BASIC 0

non esiste nessun buffer per i file; se si prova ad aprirne uno si ha una segnalazione d'errore. (Vedere più avanti, all'istruzione OPEN).

Torniamo all'esempio di comando BASIC esteso; dopo il numero 7 c'è una virgola, poi il numero 53000: esso rappresenta, in forma decimale, l'indirizzo massimo che il BASIC può utilizzare per la gestione dei programmi: le celle di memoria superiori sono così protette da un loro uso da parte del BASIC. Questa possibilità può tornare utile per riservare quella porzione di memoria a programmi utente scritti in

linguaggio macchina. Se non viene usata questa opzione, il BASIC si troverà a disposizione tutta la memoria RAM.

Per finire, dopo un'altra virgola troviamo PRINT MEM; anche questo, come i parametri precedenti, è fornito solo a titolo d'esempio. Caricando il BASIC in quel modo si fa eseguire anche quel comando: si visualizza

la disponibilità di memoria. Al suo posto si può mettere un qualsiasi altro comando del BASIC che verrà messo in esecuzione dal computer appena arrivato a livello BASIC.

Come risulta anche dal diagramma di sintassi, nel comando BASIC le varie opzioni si possono dare anche solo in parte; quelle omesse assumeranno i valori di default.

Ecco qualche esempio di utilizzo del comando BASIC:

```
BASIC 5, LOAD"PIPP0"  
BASIC  
BASIC, REVON  
BASIC ,1, 43298
```

OPEN (D/P)

Su disco è possibile registrare programmi, leggerli, cancellarli, assegnare loro un nuovo nome, e così via; si tratta di un uso importante dei dischetti, ma la loro utilità è evidenziata in misura molto maggiore dall'uso dei file. La parola FILE, in inglese, significa FILA, e sta ad indicare appunto una fila più o meno lunga di dati registrati su disco per formare un archivio. Si suole anche dire che i dati registrati su disco costituiscono la MEMORIA DI MASSA del computer: globalmente essi rappresentano grandi quantità di informazioni che vengono elaborate dai programmi.

Così possiamo archiviare in file i nomi dei clienti di uno studio dentistico, oppure la descrizione ed i prezzi degli articoli trattati in una determinata attività commerciale, oppure i testi standard di circolari da inviare ad una certa categoria di fornitori di una ditta, e così via. Non esistono limiti di sorta al tipo di dati da immagazzinare; essi poi, oltre ad essere registrati, vanno anche aggiornati, oppure cancellati, a seconda dei casi. C'è quindi la necessità di istruzioni che permettano di gestire questa massa di informazioni; per questo si parla di GESTIONE DEI FILE.

Ogni operazione fatta su di un file deve innanzitutto individuarlo in modo inequivocabile; questo si ottiene assegnando ad ogni file un nome, che da quel momento in poi lo caratterizza e lo distingue da tutti gli altri. Ovviamente i nomi dati ai vari file presenti su di un disco devono essere diversi.

Il sistema accetta nomi lunghi non più di 8 caratteri, ed il primo di essi deve essere obbligatoriamente una lettera; si possono poi usare lettere oppure numeri. Altri caratteri non vanno messi; in caso contrario il computer tronca il nome a cominciare dal carattere non ammesso, o dal nono in poi.

Iniziamo col dire quali sono le differenze tra file sequenziali e random. Abbiamo già parlato altre volte dell'argomento, ma una ripetizione non fa mai male.

Immaginate allora un'insieme di dati registrati uno di seguito all'altro, dove ognuno di essi può essere di tipo (numero o stringa) e di lunghezza diversi; questo è, in parole povere, un file sequenziale. Caratteristica di questi file è che possono essere letti o scritti solo cominciando dall'inizio, dal primo dato, e spostandosi di uno per volta in sequenza. Così, per leggere ad esempio il tredicesimo dato occorre leggere anche i dodici che lo precedono.

I file random (= casuali) sono invece ad accesso casuale: volendo leggere o registrare un elemento qualsiasi, lo si può fare direttamente senza dover percorrere tutta la fila.

Ogni tipo di file ha caratteristiche sue proprie, per cui non si può dire che uno sia migliore dell'altro: caso per caso si sceglie la soluzione più conveniente. Il tipo random potrebbe sembrare, dalle poche note fatte, preferibile all'altro; in effetti occorre sapere anche che nei file random si possono registrare solo stringhe (vedremo che con opportune istruzioni si possono trasformare i numeri in stringhe, proprio per superare questa limitazione); inoltre i vari dati vanno raggruppati in elementi (detti RECORD) di lunghezza fissa: 255 caratteri.

Si intuisce quindi che la maggior comodità d'accesso dei file random si paga sotto forma di una maggiore complicazione nella loro gestione. Comunque avremo modo di vedere diverse applicazioni per ogni tipo di file; poco per volta vi potrete familiarizzare anche con l'uso degli archivi.

I file random, una volta aperti, possono essere usati sia in lettura che in scrittura; con un solo tipo di apertura è quindi possibile mettersi in comunicazione con un archivio random ed effettuare prelievi di dati oppure operazioni di aggiornamento dei medesimi, andando ad incidere nuovi valori. Per gli archivi sequenziali, invece, il discorso è completamente diverso. Infatti essi possono essere aperti in due modi differenti: uno per leggere i dati e un'altro per registrarli; queste due operazioni non possono essere fatte contemporaneamente proprio perché necessitano di tipi di aperture diverse. Resta da dire che il computer

NE-Z80 è tra i più aggiornati per quanto riguarda il sistema operativo ed il linguaggio BASIC: infatti permette di aprire un file sequenziale in registrazione, andando a scrivere nuovi dati in coda ai precedenti. Molto più spesso, in altri personal, anche per aggiungere dati ad un archivio sequenziale occorre riscriverlo tutto; la differenza non è da poco, in quanto fa risparmiare righe di programmazione e tempi di elaborazione dei dati, riducendo nel contempo le possibilità di commettere errori.

Come abbiamo già fatto notare per la registrazione dei programmi, anche per quella degli archivi bisogna tenere presente che l'incisione di un file con un certo nome cancella i dati in un file, eventualmente presente sul disco, che abbia lo stesso nome. Questa procedura è normale per aggiornare i dati di un archivio, ma occorre fare attenzione nell'assegnazione dei nomi: se per un nuovo programma se ne utilizza uno già presente su disco perché sfruttato in occasioni precedenti, il contenuto del vecchio file verrà perduto o alterato in modo irrimediabile. L'abitudine di usare nomi diversi fa evitare una eventualità simile; bisogna, in ogni caso, fare sempre una DIR per conoscere il contenuto di un dischetto prima di usarlo. Questo, tra l'altro, è uno dei motivi che consigliano di tenere sempre una copia dei dischi.

La figura 62 fornisce il diagramma di sintassi di OPEN.

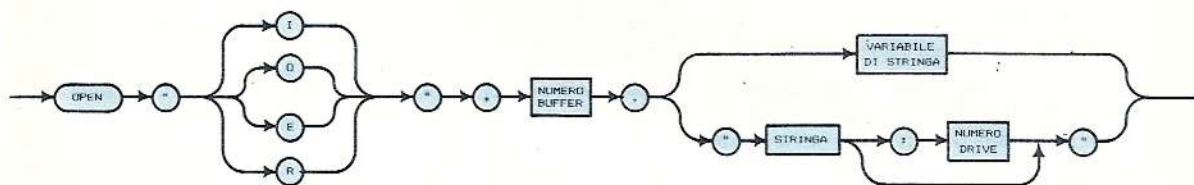


fig. 62 diagramma di sintassi di OPEN

Il modo di aprire un file è esattamente lo stesso, sia che si tratti del tipo sequenziale che random.

Dopo la parola chiave OPEN bisogna mettere le virgolette, poi una delle quattro lettere I O E R, poi ancora virgolette. La lettera R serve per aprire file di tipo random, mentre le altre si usano per quelli sequenziali, e precisamente:

- I - serve per prelevare dati da un archivio sequenziale già esistente
- O - si usa per registrare dati in un file; la scrittura comincia sempre dal primo elemento dell'archivio, quindi cancella tutti gli eventuali dati preesistenti
- E - serve per registrare nuovi dati, ma l'incisione avviene in coda a quelli già presenti; se si tratta della prima registrazione nel file che porta quel nome, essa avviene a cominciare dal primo elemento. Con questo tipo di apertura, un archivio sequenziale è in continua espansione.

Dopo aver specificato il tipo di apertura, si devono aggiungere una virgola, un numero ed un'altra virgola; il numero crea il buffer corrispondente, necessario per il transito dei dati. A questo riguardo occorre tenere presenti le considerazioni fatte sul numero dei buffer che si hanno a disposizione. Se non si è indicato diversamente, il loro numero massimo è tre; quindi se il valore indicato è superiore si ha una segnalazione d'errore.

Bisogna infine specificare il nome del file, e questo si può fare nei due modi soliti: o si mette una variabile di stringa, già definita in precedenza, oppure si mette il nome tra virgolette. Solo usando questo secondo modo è possibile aggiungere (sempre dentro le virgolette) il numero del drive da usare, proprio come si può fare anche nei casi di LOAD, SAVE, KILL e RUN.

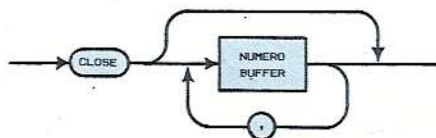
Sul diagramma di sintassi, per non complicarlo eccessivamente, non compaiono le possibilità di dare variabili al posto dei tre parametri (modo d'accesso, numero del buffer, nome del file). Negli esempi che seguono, alcuni sono di questo tipo.

```
OPEN "R", 2, "VITI"
OPEN "O", 12, "ELENCO"
OPEN "E", 1, CL$
OPEN MOD0$, NUMBUFFER, NOMEFILE$
OPEN "I", 3, FX$
```


CLOSE (D/P)

L'istruzione CLOSE effettua la chiusura di un buffer precedentemente aperto. Il diagramma di sintassi è in FIGURA 63.

fig. 63 diagramma di sintassi di CLOSE



Una linea di programma che contenga l'istruzione:

CLOSE

effettua la chiusura di tutti i file aperti fino a quel momento; ogni accesso agli archivi è interdetto, a meno che non venga fatta una nuova apertura. Specificando invece uno o più numeri, si ha la chiusura dei soli buffer corrispondenti; ad esempio:

CLOSE 1,3

chiude solo i file cui erano stati assegnati i buffer 1 e 3; se il buffer 2 è aperto, resta in tale condizione.

Si deve evitare di togliere un disco dal drive se contiene un file che non è stato chiuso; infatti può darsi che gli ultimi dati da registrare siano ancora depositati nel buffer. In tal caso essi vengono scritti sul disco quando si effettua la chiusura del file. Quindi occorre fare sempre la chiusura degli archivi aperti in scrittura, prima di estrarre il dischetto.

C'è poi da dire che la chiusura di un file può essere effettuata in modo automatico dal sistema operativo; questo avviene nei seguenti casi:

- 1 - si cancella il programma in memoria con il comando NEW
- 2 - si dà un RUN
- 3 - si effettua una operazione di fusione di programmi col comando MERGE
- 4 - si fanno delle modifiche o delle aggiunte al programma
- 5 - si azzerano le variabili con l'istruzione CLEAR
- 6 - si verifica un errore nel trasferimento dei dati col disco.

La chiusura di un file già chiuso non provoca errore.

PRINT# (D/P)

Con l'istruzione PRINT# si fanno registrazioni su file sequenziali.

Il diagramma di sintassi di PRINT# è in FIGURA 64.

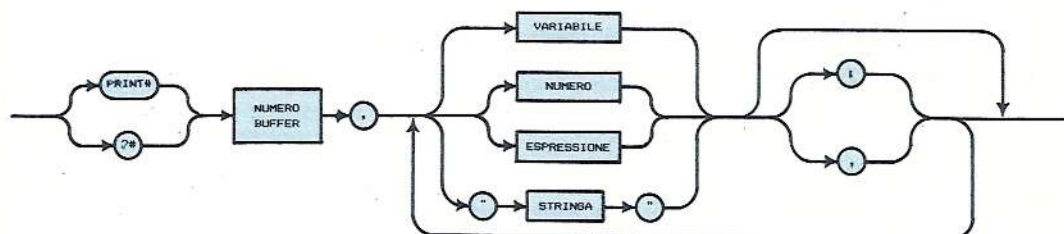


fig. 64 diagramma di sintassi di PRINT#

La complessità è solo apparente, e casomai risiede in considerazioni che non compaiono nel diagramma.

Innanzitutto facciamo notare la grande somiglianza del diagramma con quello di PRINT (FIGURA 22); l'unica differenza, a parte la mancanza del ramo più alto, è data dal numero del buffer.

Apriamo una breve parentesi per ricordare che con la formattazione il disco viene suddiviso in tracce e settori; ogni settore-traccia, della lunghezza di 255 caratteri, è chiamato RECORD.

Ebbene, col l'istruzione PRINT # si fanno scritture su disco che seguono le stesse regole valide per la

PRINT su video, con la differenza che la riga video è lunga 32 caratteri, mentre quella del disco è di 255.
Facciamo un'esempio:

```
PRINT #2, A3;D$
```

Con questa istruzione si registra su disco il valore numerico della variabile A3 e, di seguito (a causa della presenza del punto e virgola), il contenuto della variabile di stringa D\$. Scrivendo invece:

```
PRINT #2, A3, D$
```

i dati anzidetti vengono registrati non adiacenti tra di loro, ma con uno spazio vuoto in mezzo, proprio come accade per le scritture su video. È evidente che questo secondo modo è da evitare, in quanto fa sprecare un certo numero di caratteri del dischetto, che invece deve essere sfruttato il più possibile: sarebbe come avere un quaderno ed utilizzarlo scrivendo una riga ogni tanto.

Le cose si possono complicare notevolmente con l'uso delle variabili di stringa o delle stringhe; parleremo di questo alla prossima istruzione.

Da notare che ordinando di registrare un'espressione aritmetica, sul disco si trova il suo valore; ad esempio:

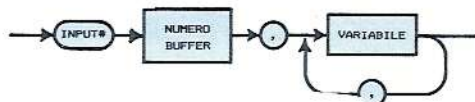
```
PRINT #3, A3*(23-Y)
```

non registra l'espressione così come è scritta, ma il suo valore; supponendo che sia $A3 = 8$ e $Y = 12$, nel file viene scritto il numero 88.

INPUT# (D/P)

L'istruzione INPUT # serve per leggere dati prelevandoli da un file sequenziale. Il suo diagramma di sintassi è in FIGURA 65.

fig. 65 diagramma di sintassi di INPUT#



La variabile può essere sia di tipo numerico che di stringa.
Vediamo qualche esempio:

```
INPUT #5, FQ  
INPUT #2, E$, G(3, 8), B9$, X$  
INPUT #13, A, GH, K, J$, W
```

Può succedere di richiedere la lettura di un dato numerico, mentre il primo elemento che INPUT fornisce è di tipo stringa: evidentemente si ottiene sempre il valore zero. Analogamente, se si legge un dato come stringa anche se esso era stato registrato come numero, si ottiene una stringa che contiene quel numero, ma non un numero (scusate il giro di parole inevitabile).

Ciò significa che per leggere dati da un file sequenziale occorre conoscere il modo in cui i dati numerici e di stringa si susseguono.

Se si cerca di prelevare dati da un file che sia già stato letto fino in fondo si ha una segnalazione d'errore (INPUT PAST END). Vedremo più avanti che esiste un'istruzione di controllo per evitare che questo possa accadere.

Ora ci preme andare a fondo nella questione della registrazione e lettura di stringhe. Infatti nascono delle complicazioni con l'uso delle virgole e delle virgolette. Vediamole.

Tra poco parleremo del carattere corrispondente a RETURN (codice ASCII = 13) e del carattere corrispondente al tasto LINE FEED (codice ASCII = 10); nelle nostre considerazioni li indicheremo rispettivamente nel modo che segue:

<CR> e <LF>

Tali simboli indicheranno quindi i codici ASCII corrispondenti. CR è una terminologia che deriva dalle macchine da scrivere (iniziali di CARRIAGE RETURN, ossia ritorno carrello).

Vediamo ora uno per uno i vari casi che si possono presentare nella registrazione dei dati in un file sequenziale.

Un'istruzione di scrittura di questo tipo:

```
OPEN"O", 1, "PROVA" : PRINT #1, A : CLOSE
```

scrive su disco il valore numerico della variabile A, aggiungendo <CR>.

Fermiamoci a questo punto per fare qualche considerazione; infatti per rendersi conto esattamente di quello che vedremo tra poco, occorre conoscere un po' ciò che avviene sul dischetto.

Quando si registrano file, il loro nome compare nella traccia 17 (a meno che essa non sia stata inibita in sede di formattazione del disco), detta DIRECTORY (=indice). Il contenuto di un disco può essere visualizzato, come già sapete, col comando DIR del DOS.

Assieme al nome del file, vengono anche registrati altri dati, come la posizione nel disco dove il file stesso comincia e quella dove finisce; la prima informazione serve per leggerlo dall'inizio, la seconda per eventuali operazioni di registrazioni in coda, con l'istruzione OPEN"E". In pratica questo secondo dato è quello che ci interessa, e lo chiameremo puntatore di fine file. Quando ordiniamo al computer di registrare in coda ad un archivio sequenziale preesistente, esso va a leggerselo cominciando dall'inizio e contando un numero di byte uguale al valore del puntatore; in tal modo si è posizionato in coda.

Torniamo al nostro esempio di poco fa e facciamo una nuova registrazione:

```
OPEN"E",1,"PROVA" : PRINT #1,55;
```

Il computer va a leggersi il valore del puntatore e registra il numero 55 in coda al valore precedente; TRA I DUE NUMERI RESTA REGISTRATO IL CARATTERE <CR>. Questa volta però dopo il numero 55 non viene registrato <CR>.

Ricordiamo che ogni numero è formato, oltre che dalle sue cifre, anche da un carattere iniziale riservato al segno e da uno terminale che resta sempre vuoto. Se supponiamo, per fissare le idee, che il valore della variabile numerica A registrata la prima volta sia 4963, sul disco troviamo la seguente situazione:

```
*4963* <CR> *55*
```

CON L'ASTERISCO ABBIAMO INDICATO GLI SPAZI (codice esadecimale 20 sul disco). Se ora scriviamo:

```
PRINT #1,-28.07,
```

il computer va a registrare in coda ai dati precedenti il numero indicato; ora la situazione sul disco è la seguente:

```
*4963* <CR> *55*-28.07*****
```

Si può vedere che dopo il numero 55 parte subito l'ultimo valore registrato; in coda a questo ci sono 10 spazi perché con la virgola messa nell'istruzione di scrittura il computer effettua una tabulazione di 16 caratteri, esattamente come per istruzioni di stampa sul video; facendo i conti, si vede che l'ultimo numero occupa appunto 16 byte tra segno, cifre e spazi vuoti alla fine.

In realtà sul disco non vengono registrati i valori decimali, ma il loro equivalente in esadecimale; ad esempio, al posto di <CR> si ha 0D, al posto del segno meno si ha 2D, per gli spazi si ha 20, per il numero 1 c'è 31 e così via.

Esiste poi anche un altro modo per registrare numeri e rileggerli; si può infatti registrare dei numeri trattandoli come stringhe, separandoli col carattere della virgola (codice ASCII 44) e poi rileggerli normalmente come se si trattasse di valori numerici. Facciamo un esempio:

```
OPEN"O",2,"PROVA2" : PRINT #2,"11";",","22";CHR$(44);"33" : CLOSE
```

Con questa istruzione abbiamo aperto il file PROVA2 in registrazione, vi abbiamo registrato tre stringhe (11 22 33) separandole tra di loro con virgole, poi abbiamo chiuso il file. La virgola è stata messa in due modi diversi: sia dandola tra virgolette che per mezzo del codice ASCII corrispondente.

Esiste anche un altro modo, ossia quello di depositare il carattere della virgola in una variabile di stringa e richiamare poi quella variabile.

Proviamo ora a leggere il file:

```
OPEN"I",2,"PROVA2" : INPUT#2,A,B,C : CLOSE
```

Con questa istruzione abbiamo detto al computer di leggere tre valori numerici dal file PROVA2 e di depositarli nelle variabili A, B e C. Chiedendo i valori di queste variabili si ottengono i numeri 11 22 e 33. Si noti che essi erano stati registrati sotto forma di stringhe. La cosa è possibile perché nei file sequenziali i dati vengono registrati in formato ASCII e non in quello compattato, come succede per i file random.

Sul disco, in questo ultimo esempio, troveremmo questa configurazione dei dati:

```
11, 22, 33 <CR>
```

Si può dire, per concludere, che nel processo di registrazione-lettura dei valori numerici in un file sequenziale, i caratteri separatori tra due dati sono tre: <CR>, spazio e virgola, cui vanno aggiunti il carattere di fine del record e quello di fine del file, ovviamente.

Le cose sono più complesse quando si trattano le stringhe.

Se venissero registrate una di seguito all'altra con l'uso del punto e virgola, non sarebbe poi più possibile rileggerle separate, in quanto diverrebbero una stringa unica. Uno dei modi per rimediare a questo inconveniente è quello appena visto: si scrivono le stringhe su disco separandole con virgole. Ciò significa che esse sono uno di quei caratteri che separano le stringhe le une dalle altre.

Esistono anche altri modi di separare le stringhe tra di loro; ciò è necessario nei casi in cui la stringa trattata contenga una o più virgole. Infatti in questo caso esse verrebbero interpretate come caratteri separatori di stringhe: non solo non entrerebbero a far parte della stringa letta, ma questa sarebbe ben diversa da quella registrata perché arriverebbe solo alla prima virgola esclusa; la parte restante fino alla virgola successiva formerebbe il contenuto di un'altra stringa, e così via. Facciamo un esempio:

```
OPEN"O",1,"PROVA3"  
PRINT #3,"NUOVA ELETTRONICA";"VIA CRACOVIA, 19";"BOLOGNA"  
CLOSE
```

Proviamo ora a leggere il file PROVA3:

```
OPEN"I",1,"PROVA3" : INPUT #1,A$,B$,C$ : CLOSE
```

Il computer manda la segnalazione d'errore INPUT PAST END, ossia ci segnala innanzitutto che vogliamo leggere un numero di dati superiore a quello contenuto dal file; vedremo tra poco il perché. Proviamo ora a visualizzare il contenuto delle tre variabili di stringa richieste:

```
PRINT A$,B$,C$
```

Otteniamo per A\$ e B\$ le seguenti scritte:

```
NUOVA ELETTRONICAVIA CRACOVIA  
19BOLOGNA
```

Se chiedessimo anche il contenuto di C\$ otterremmo una stringa nulla.

La spiegazione di tutto questo è abbastanza semplice: interponendo il punto e virgola, le tre frasi sono state registrate attaccate le une alle altre, e sul disco si ha la seguente configurazione:

```
NUOVA*ELETTRONICAVIA*CRACOVIA,*19BOLOGNA<CR>
```

Dato che la virgola, quando si leggono dati da disco, è uno dei caratteri separatori di stringa, la prima istruzione di lettura deposita in A\$ i caratteri compresi tra l'inizio e la virgola; la seconda deposita in B\$ il resto e la terza fa andare in errore; in C\$ pertanto non viene depositato nulla.

Vi facciamo notare che B\$ non comincia con uno spazio e che in lettura abbiamo usato un numero di buffer diverso da quello della registrazione; la cosa è lecita, a patto che il buffer 1 non sia già impegnato.

Facendo il punto della situazione fino ad ora, si può dire che per poter separare in lettura le stringhe occorre inciderle separandole con virgole.

Evidentemente le cose cambiano se si debbono trattare stringhe contenenti virgole; in questo caso la stringa deve essere preceduta, sul disco, da virgolette: il computer, incontrandole, assegna alla stringa tutti i caratteri che seguono, fino a che non incontra altre virgolette. In tal modo le stringhe possono contenere di tutto, virgole e <CR> compresi. Evidentemente, però, non devono essere presenti al suo interno delle virgolette, perché esse diventerebbero automaticamente il carattere terminale della stringa. Facciamo notare che le virgolette non fanno parte della stringa, ma ne sono solo un carattere delimitatore; analogamente, nel caso precedente, per le virgole.

Ricordare, comunque, che una stringa la cui lettura sia iniziata con delle virgolette, termina solo quando il calcolatore trova le virgolette successive; le uniche eccezioni si hanno se si incontra la fine del record (ossia se si arriva al 255-esimo carattere) o se si va fino alla fine del file.

In pratica, se si devono registrare stringhe che debbono contenere anche le virgolette, si usa come carattere delimitatore la virgola; in tal caso però questo carattere non deve essere compreso all'interno della stringa. Se invece essa contiene delle virgole, come caratteri delimitatori si devono usare le virgolette, che però non devono fare parte della stringa.

Evidentemente il caso più sfortunato è quello in cui si debbano trattare stringhe contenenti entrambi i caratteri delimitatori; la cosa non è fattibile ed occorre spezzare la stringa in due o più parti, ognuna delle quali non contenga più di un carattere delimitatore per volta.

Facciamo un esempio, per ricapitolare quanto detto:

```
OPEN"O",1,"PROVA4"  
PRINT #1,"      TRANSISTOR, TRASFORMATORE";",",";"DIODO";"DIODO";  
      CHR$(34);"ZENER";CHR$(34);CHR$(34);  
      "INTEGRATO, RESISTENZA";CHR$(34);",",";"CONDENSATORE"  
PRINT #1,"TRIMMER" : PRINT #1,CHR$(34);"  QUARZO 4 MHZ";CHR$(34)  
CLOSE
```


Su disco si ottiene una configurazione del genere:

```
*****TRANSISTOR,*TRASFORMATORE,DIODO,DIODO"ZENER""INTEGRATO,  
*RESISTENZA",CONDENSATORE<CR>TRIMMER<CR>****QUARZO*4MHZ"<CR>
```

Scriviamo ora un programmino per leggere i dati; non sapendo esattamente quanti ne troveremo, facciamo in questo modo:

```
OPEN"1",1,"PROVA4"  
FOR I=1 TO 15 : PRINT I; : INPUT #1,A$ : PRINT A$ : NEXT
```

Non abbiamo messo la CLOSE perché il computer andrà in errore prima di essere giunti alla fine del loop, in quanto nel file non abbiamo 15 dati: arrivando alla fine dell'archivio si ottiene, alla lettura successiva, INPUT PAST END. Il sistema operativo (DOS), in caso di errore, effettua automaticamente la chiusura dei file aperti.

Il loop è fatto in modo da stampare il numero progressivo del dato letto ed il suo contenuto; notare che ad ogni lettura la stringa viene messa sempre in A\$, che così perde il valore precedente già stampato; per conservare i vari dati sarebbe stato necessario depositarli in un vettore opportunamente dimensionato.

Torniamo al nostro esempio: le righe precedenti forniscono questi dati:

```
1 TRANSISTOR  
2 TRASFORMATORE ---  
3 DIODO  
4 DIODO"ZENER""INTEGRATO  
5 RESISTENZA"  
6 CONDENSATORE  
7 TRIMMER -----  
8   QUARZO 4 MHZ
```

Facciamo notare ancora che gli spazi iniziali prima di TRANSISTOR si sono perduti: questo avviene sempre per le stringhe che hanno la virgola come delimitatore; per conservarli occorre usare le virgolette.

Ci preme far notare che le virgolette che racchiudono le varie stringhe in sede di registrazione non hanno niente a che fare con quelle che vogliamo registrare su disco come delimitatori di stringhe in lettura: queste vanno incise a parte, usando il relativo codice ASCII oppure una variabile di stringa che le contenga.

Per capire il risultato fornito dalla lettura dell'archivio PROVA4 bisogna riferirsi alla struttura dei dati sul disco; andiamo con ordine:

- 1 - la prima stringa non comincia con virgolette; quindi gli spazi iniziali vengono ignorati ed il primo carattere significativo messo in A\$ è la lettera T di TRANSISTOR. Vengono poi letti tutti i caratteri fino ad arrivare alla virgola successiva, ossia viene immesso TRANSISTOR
- 2 - la seconda lettura parte dalla virgola appena vista, ignora lo spazio e legge fino alla virgola seguente, introitando TRASFORMATORE
- 3 - analogamente la quarta volta: si legge DIODO
- 4 - lo stesso procedimento si ripete alla quinta lettura: si va da virgola a virgola, leggendo tutto quello che è in mezzo (spazi, <CR>, <LF> e virgolette eventuali compresi); si ottiene così la stringa numero 4.
- 5 - la quinta lettura ha per delimitatori ancora virgole, quindi immette la parola RESISTENZA seguita da virgolette
- 6 - la sesta lettura termina al carattere <CR>, che è un delimitatore quando la lettura è cominciata con la virgola.
- 7 - la settima lettura fornisce la scritta TRIMMER, compresa tra i delimitatori <CR>
- 8 - l'ultima lettura prima dell'errore fornisce la stringa numero 8: essa contiene anche i tre spazi iniziali perché il primo delimitatore era dato dalle virgolette.

Potrà sembrare astruso, ma il procedimento diventa presto familiare, dopo qualche prova di rodaggio.

Resta ancora una cosa da dire: quando <CR> è preceduto da <LF>, esso non viene considerato come carattere terminale, ma va a far parte del dato, se si tratta di una stringa, mentre viene ignorato se si tratta di una variabile numerica. Nella tabella 10 si trova un riassunto di quanto detto sui delimitatori di dati nei file sequenziali.

Torniamo finalmente ad esaminare il listato del programma PITAGORA.

LINEE DA 1090 A 1170

La riga 1090 è di commento: con essa si dice a cosa serve la porzione di programma che segue (ricerca dei dati precedenti).

Queste linee sono state messe in ossequio agli obiettivi 1 e 2 che ci eravamo prefissi all'inizio del

programma: ogni esercitazione sulle tabelline deve andare a prelevare eventuali dati precedenti per quell'alunno, e fare riferimento ad essi per le domande successive. Esaminiamo con ordine il contenuto di questo gruppo di linee.

Alla riga 1100 il computer va ad aprire in lettura il file NOME, ossia quello che porta i dati precedenti dell'alunno. Se un file con quel nome non esiste, si otterrebbe una segnalazione d'errore ed il calcolatore uscirebbe dal programma. Abbiamo usato il condizionale poiché, avendo previsto una routine di correzione per questa evenienza, in effetti il controllo del programma passa alla linea 2280 (come era indicato nella riga 1040). In quella sede si trova l'istruzione RESUME 1170: il programma prosegue rientrando alla linea 1170, dove il buffer aperto viene chiuso.

Se invece esiste il file NOME, per mezzo delle linee 1110-1150 si formano tre matrici bidimensionali N, P ed E; inoltre vengono letti i valori delle variabili ES, MIN, TE e TP; vedremo più avanti il significato di tutti questi dati. A lettura terminata, il file viene chiuso (linea 1170).

LINEE DA 1180 a 1440

Dopo la linea di annotazioni (1180) si passa alla cancellazione dello schermo (1190) e quindi si azzerano le due variabili S ed I1; la prima resta zero se la risposta sarà esatta, mentre diventa 1 in caso contrario (come vedremo alla 1470). La variabile ES viene poi incrementata di uno; essa ci serve per contare progressivamente gli esercizi fatti; il valore di partenza è quello appena letto da disco (vale ovviamente zero se non esiste il file NOME).

Fin qui nulla di nuovo rispetto a PITA2, come pure per le linee dalla 1210 alla 1250.

La matrice bidimensionale N è stata usata anche in PITA2 e serve, come ricorderete, per controllare se la coppia di valori casuali A e B è già uscita. La linea 1260 stabilisce che se il numero di esercizi è minore di 101 e se la coppia di valori A-B è già stata estratta, il programma torna alla riga 1240 per fare una nuova estrazione. Da notare che se si sono fatti più di 100 esercizi, il programma prosegue in ogni caso, in quanto le due condizioni devono essere verificate contemporaneamente (AND).

Alla riga 1270 troviamo un'altra condizione logica con AND: se gli esercizi sono in numero superiore a 100 e se il punteggio assegnato in precedenza alla risposta data per quella coppia A-B differisce per più di 5 punti dal valore della variabile MIN, allora il programma torna a generare nuovi valori per A e B. MIN è uno dei valori letti dal disco; in quella variabile, ad ogni giro di risposte il programma deposita il valore più basso del punteggio. La sua formazione è identica al programma precedente, quindi non la ripetiamo.

Il programma, poi, arriva alla linea 1280 soltanto in due casi: se il numero di esercizi è inferiore a 101 e la coppia A-B è nuova, oppure se gli esercizi sono più di cento ed il punteggio (memorizzato nella matrice P) della coppia A-B in esame differisce dal valore più basso per meno di cinque punti. Questa seconda condizione è molto importante, perché realizza il quarto degli obiettivi che ci eravamo proposti: vengono richiesti solo gli esercizi che sono stati sbagliati più volte o che hanno richiesto un tempo più lungo di risposta.

Il gruppo di linee da 1280 a 1440 è identico a quello 1080-1240 di PITA2, tranne che nella riga 1390, dove si ha un rinvio alla linea 1710, primo passo da fare quando viene richiesto di por fine agli esercizi.

Esaminiamo ora una nuova istruzione BASIC che incontreremo tra poco: GOSUB.

GOSUB E RETURN (P)

Sono chiamate SUBROUTINE quelle parti di programma che vengono utilizzate più volte; anziché ripeterle nei punti dove servono, vengono scritte una sola volta (generalmente alla fine del programma, ma ciò non è obbligatorio).

Quando arriva il momento di utilizzare quella subroutine, si mette l'istruzione GOSUB seguita dal numero di linea in cui essa comincia. Le cose funzionano in modo che al termine della subroutine, incontrando la parola chiave RETURN, il programma ritorna all'istruzione successiva alla GOSUB di partenza. Ogni subroutine deve quindi terminare con RETURN, e può essere utilizzata tutte le volte che serve, da punti diversi del programma; esso rientrerà sempre all'istruzione successiva all'ultima GOSUB impartita.

Se il computer incontra un RETURN non preceduto da GOSUB, si ha una segnalazione d'errore.

I diagrammi di sintassi di GOSUB e RETURN sono alla FIGURA 66.

fig. 66 diagrammi di sintassi di GOSUB e RETURN



Dall'interno di una subroutine se ne può richiamare un'altra, e così via.

LINEE DA 1450 A 1590 E DA 2230 A 2260

Dopo la linea di annotazioni troviamo la 1460 che è uguale alla 1250 di PITA2. La 1470 differisce dall'analoga 1260 del programma precedente per la presenza della matrice E: in caso di errore il valore corrispondente alla coppia A-B in esame viene aumentato di uno; in tal modo è possibile memorizzare per ogni tabellina il numero di errori che le compete; tale valore andrà sul disco e potrà essere utilizzato anche per la stampa dei risultati.

In pratica, per ognuna delle 100 possibili tabelline noi teniamo memorizzati, per mezzo di tre matrici bidimensionali, tre parametri: il numero di volte che essa è uscita (N), il numero di volte che è stata sbagliata (E) ed il punteggio complessivo che le compete (P). I singoli elementi di queste matrici (300 in tutto) sono quelli registrati e letti nel file sequenziale NOME.

Le linee 1480 e 1490 sono rimaste immutate dal programma precedente.

Seguono poi tre righe che realizzano quanto segue: se la risposta è errata ed il numero di esercizio è inferiore a 101 il punteggio P (quello determinato finora dal tempo impiegato a rispondere) viene diminuito di 10; se la risposta è sbagliata ed il numero di esercizio supera 100, P viene calato solo di 5; se la risposta è esatta e si è oltre i 100 esercizi, P viene aumentato di 5. Evidentemente si tratta di scelte del tutto arbitrarie, ma che risultano adeguate all'obiettivo prefissato di far richiedere solo le tabelline che hanno il punteggio più basso. Col sistema adottato, ad ogni richiesta il punteggio aumenta se la risposta è esatta e se viene data in poco tempo, mentre diminuisce in caso contrario. Può succedere che la stessa tabellina venga proposta consecutivamente per un certo numero di volte: significa che il suo punteggio è talmente disastroso rispetto agli altri che occorrono diverse risposte esatte per riportarlo a valori superiori al più basso delle rimanenti. Variando le condizioni di queste tre righe cambia ovviamente il modo di funzionare del programma.

Seguono poi tre linee dal contenuto identico a quelle corrispondenti di PITA2. Alla riga 1560 troviamo l'istruzione di rinvio alla subroutine che comincia alla linea 2240: si tratta di un ritardo, che nel programma precedente era realizzato alla linea 1330; ora però il tempo è doppio nel caso di risposta errata, rispetto a quello in cui sia esatta, per dare modo di leggere comodamente il valore corretto della tabellina richiesta. Ciò avviene perché il loop di ritardo della riga 2240 viene eseguito solo se S vale 1, ossia solo in caso di errore; il ritardo della linea 2250 viene invece realizzato in ogni caso. La linea 2260 contiene RETURN, che fa tornare il programma all'istruzione seguente all'ultima GOSUB incontrata: si passa così alla riga 1570.

Qui l'elemento della matrice N corrispondente alla coppia A-B viene incrementato di uno; alla linea successiva il punteggio della coppia viene aumentato del valore ultimo conseguito.

Con la riga 1590 il programma torna a proporre un nuovo esercizio, ma solo nel caso che l'ultimo effettuato sia inferiore a 100; quando si è arrivati a quel limite il computer manda in esecuzione le linee che seguono.

LINEE DA 1600 A 1690

Come indicato dalla REM della riga 1600, questa parte del programma serve per trovare il più basso dei punteggi ottenuti per le 100 tabelline chieste, tutte diverse tra di loro.

Il valore cercato viene messo nella variabile MIN; per cominciare la si pone uguale al punteggio della tabellina 1x1, ossia a P(1,1); poi, coi due loop nidificati nelle variabili J e K, si passano in rassegna i restanti 99 valori. Ogni volta che uno di essi risulta inferiore a MIN, tale variabile assume quel valore (riga 1640).

La linea 1670 rimanda alla formulazione di un nuovo esercizio solo nel caso che quello appena fatto superi 100, altrimenti si prosegue con la cancellazione del monitor e la visualizzazione della frase contenuta nella linea 1690.

LINEE DA 1700 A 1770

Con queste righe si passa prima alla scrittura in reverse, poi viene visualizzata la dicitura SITUAZIONE DI seguita dal nome dell'alunno che sta lavorando al computer. Si torna poi alla scrittura normale, si lasciano due righe e si ottengono i valori indicati alle righe 1750, 1760 e 1770.

LINEE DA 1780 A 1860

Questa parte del programma registra il file NOME\$ (ossia quello che porta il nome dell'alunno), mettendovi tutti i nuovi valori; se il file non esisteva, come accade ogniqualvolta un alunno esegue per la prima volta gli esercizi, il computer va sul primo disco non protetto contro le registrazioni ed incide i dati. Se

invece il file esiste già, lo riscrive da capo completamente, inserendovi i valori aggiornati.

La registrazione dei dati avviene con due loop nidificati; quando si arriva alla riga 1820 si scrivono su disco i valori di N, P ed E che competono ad ogni coppia A-B. In coda a questi dati (300 in tutto, come abbiamo già detto) si registrano i valori delle variabili ES, MIN, TE e TP. Ogni volta che si pone termine agli esercizi si scrivono quindi sul disco, nel file appropriato, 304 valori numerici, gli stessi che vengono letti alle linee 1100-1170 quando lo stesso alunno riprende gli esercizi. Ovviamente due individui che abbiano il medesimo nome lavorerebbero sullo stesso file: occorre allora cambiarne uno. Quando l'aggiornamento del file è giunto a termine, il buffer corrispondente viene chiuso.

LINEE DA 1870 A 1990

Questo gruppo di linee visualizza un menù, ossia un quadro in cui è possibile effettuare scelte diverse. Nel nostro caso sono state poste quattro alternative: finire, stampare su carta i soli errori, stampare la situazione completa oppure continuare con gli esercizi.

Il gruppo di righe 1880-1940 serve appunto alla scrittura del menù.

Con la 1950 l'elaboratore resta in attesa di un carattere da tastiera e lo deposita nella variabile di stringa M\$. Se non viene battuto alcun tasto si torna alla condizione d'attesa; se il tasto è quello della lettera F si passa alla linea 2200, che vedremo.

Successivamente (riga 1970) nella variabile numerica M viene messo il valore della stringa M\$, che è formata sempre da un solo carattere. Se esso è diverso da un numero, il suo valore è zero, e va rifiutato in quanto (ad eccezione della lettera F già trattata) deve essere premuto solo uno dei tasti numerici da 1 a 3. Per questo la riga 1980, nel caso che M sia nullo o minore di zero oppure sia maggiore di 3, rimanda alla linea 1950. In tal modo vengono accettate battute effettuate solo sui tasti F 1 2 3 e tutte le altre vengono rifiutate. Se M vale 3, infine, il programma riprende la formulazione delle domande (1990). Se il tasto premuto era quello dei numeri 1 o 2 si prosegue.

Vediamo ora una nuova istruzione: LPRINT.

LPRINT (D/P)

L'istruzione LPRINT serve per effettuare scritture con la stampante. Il suo funzionamento è molto simile a quello di PRINT, come si vede anche dal diagramma di sintassi della FIGURA 67.

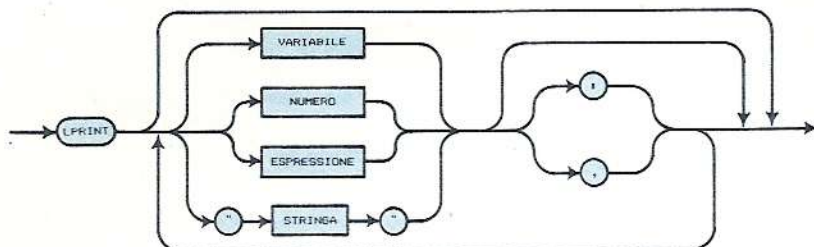


fig. 67 diagramma di sintassi di LPRINT

In pratica l'unica differenza consiste nel fatto che LPRINT non è abbreviabile con l'uso del punto interrogativo. Occorre poi tenere presente che mentre una riga video è di 32 caratteri, quella della stampante può valere 40, 64, 80 o 132 a seconda del tipo di macchina che si ha a disposizione.

Ogni stampante, generalmente, può scrivere in modi diversi, richiamabili con l'uso di caratteri di controllo.

PRINTTAB E LPRINTTAB (D/P)

Trattiamo assieme queste due istruzioni perché sono concettualmente uguali. Con esse si ottengono scritture su video o su stampante effettuando delle tabulazioni; se ad esempio facciamo:

```
PRINTTAB (21) "MARIA"
```

vedremo la parola MARIA scritta al ventiduesimo carattere della riga video (non al 21-esimo, perché il conteggio, come sempre, comincia dal numero zero).

Se il numero messo tra parentesi è maggiore di 255 o minore di zero si ottiene una segnalazione d'errore. Al posto del numero si può mettere una variabile numerica o un'espressione:

```
PRINTTAB(D2)
PRINT TAB(Y-CR*3)
? TAB(H-3)
```

Come si vede tra PRINT e TAB si può mettere uno spazio, e PRINT è abbreviabile col punto interrogativo.

Se il contenuto della parentesi è decimale, il computer effettua la tabulazione arrotondando all'interno inferiore.

Per LPRINTTAB valgono considerazioni analoghe, tranne che essa non può essere abbreviata facendo uso del punto interrogativo.

LINEE DA 2000 A 2190

Questo blocco di linee serve per effettuare le stampe richieste al menù.

Le istruzioni di stampa sono riferite alla stampante EPSON; per macchine diverse occorre adattare in alcune parti.

La linea 2030, ad esempio, contiene:

```
LPRINT CHR$(27);CHR$(69)
```

L'istruzione, usando la EPSON, fa stampare quello che segue in neretto.

Si tratta quindi di due caratteri di controllo dati in codice ASCII.

La stampante MICROLINE, al contrario, non ha questa possibilità, quindi con essa è inutile mettere quei caratteri. In sostituzione si potrebbe usare una stampa con caratteri allargati (codice ASCII 31 per MICROLINE, 14 per EPSON).

In ogni caso, la dicitura RISPOSTE viene stampata seguendo le istruzioni appena dette; se M vale 1, poi, viene aggiunta anche la parola SBAGLIATE (come richiesto al menù scegliendo la possibilità 1).

Da notare che entrambe le diciture sono seguite dal punto e virgola, in modo da avere una stampa dopo l'altra, analogamente a quanto accade per scritture sul monitor. Sempre alla linea 2030 si ha la visualizzazione della parola SBAGLIATI in coda alla frase precedentemente scritta alla riga 2020 (anche questo solo nel caso di M = 1).

La riga 2040 fa stampare in coda a quanto già visto il nome dell'alunno che ha terminato gli esercizi (o che ne ha fatti 100). Il codice ASCII 14, per la EPSON, fa stampare in caratteri allargati. La linea seguente fa tornare ad una stampa in caratteri normali (per la MICROLINE il codice è solo uno: 30).

Con la linea seguente si fanno stampe tabulate ai valori messi tra parentesi: quelle scritte sono le intestazioni dei dati che seguiranno.

I due loop nidificati che seguono servono per stampare solo quello che è stato chiesto al menù. Se la scelta era stata il numero 2 (stampa completa), si ottengono 100 righe, ognuna delle quali comincia con la scrittura della tabellina corrispondente (ad esempio 2x7); seguono poi tre valori numerici, quelli delle variabili-matrice N, E e P: per ogni tabellina compare il numero di uscite, il numero di errori ed il relativo punteggio.

Da un tabellone del genere è quindi agevole controllare il grado di preparazione dell'alunno cui si riferisce.

Se la scelta effettuata al menù era la 1 (stampa dei soli errori), la riga di programma 2100 fa passare alla stampa (linea 2120) solo se N ed E sono maggiori di zero contemporaneamente. Da notare che la stampa completa riporta in ogni caso solo le tabelline uscite, ignorando le altre: questo controllo viene esercitato dalla riga 2110.

Quando tutte le tabelline richieste sono state stampate, vengono lasciate due righe vuote, poi si hanno tre righe contenenti i valori delle variabili ES, TE e TP (linee 2160-2180). Quando anche questi dati sono stati stampati, il programma ritorna al menù.

LINEE DA 2200 A 2220

Si arriva alla linea 2200 solo dal menù con la scelta associata alla pressione del tasto F. Dopo la cancellazione dello schermo si visualizza la scritta CIAO seguita dal nome dell'alunno e da tre punti esclamativi.

Per non fare apparire sul monitor la scritta READY (come accadrebbe se la linea 2220 contenesse END), si è preferito mettere GOTO 2220: il computer va in continuazione alla stessa linea, quindi resta fermo in quel punto fino a che non si fa un break.

A questo punto termina la descrizione del funzionamento del programma PITAGORA; chi è alle prime esperienze di programmazione dovrebbe aver tratto buoni spunti da un'attenta lettura di queste note.

Andando avanti vedremo altri programmi che descriveremo in modo più sommario, ma pur sempre abbastanza approfondito, per vedere di aiutare i meno esperti senza annoiare troppo coloro che sono più avanti nell'apprendimento della programmazione in BASIC.

Prima di passare ad un altro programma vediamo alcune istruzioni non ancora descritte.

NEW (D/P)

Il comando NEW (= nuovo) cancella il programma presente in memoria centrale. Il suo diagramma di sintassi è in FIGURA 68.

fig. 68 diagramma di sintassi di NEW



Il comando va usato con attenzione: se il programma da annullare è importante e non esiste su disco, è meglio effettuarne una registrazione prima di perderlo.

NEW può essere introdotto anche in un programma, ma non ha molto senso in quanto arrivati a quell'istruzione il computer cancella tutto il programma e torna allo stato comandi; tutt'al più potrebbe fare le veci di END.

MEM E FRE (D/P)

L'uso di MEM e di FRE permette di conoscere la quantità di memoria centrale ancora disponibile. Entrambe sono delle variabili speciali riservate al sistema, pertanto assegnando ad identificatori di variabile uno di quei due nomi si ha una segnalazione d'errore.

I diagrammi di sintassi di MEM e di FRE sono riportati nelle FIGURE 69 e 70.

fig. 69 diagramma di sintassi di MEM



fig. 70 diagramma di sintassi di FRE

L'uso che si fa normalmente dei due comandi è per conoscere, in sede di stesura di un programma, la memoria ancora libera:

```
PRINT MEM
PRINT FRE (0)
```

I due comandi così espressi sono equivalenti e forniscono lo stesso numero. Da notare che FRE deve essere seguito da un parametro numerico posto tra parentesi; in caso contrario si ha errore. Normalmente quindi si usa MEM, che è più breve da digitare; esiste però un'applicazione di FRE molto interessante. Scrivendo:

```
PRINT FRE (A$)
```

si ottiene un numero che non rappresenta più la memoria ancora libera, ma il numero di caratteri ancora disponibili per le stringhe. Tra parentesi può essere messo un qualsiasi identificatore di stringa, od anche una stringa tra virgolette:

```
PRINT FRE (B5$)
? FRE (K$)
PRINT FRE ("1")
PRINT FRE (" ")
```

Tutti gli esempi forniscono il medesimo risultato.

A questo proposito vi rimandiamo alla parola chiave CLEAR, per ulteriori considerazioni.

MEM e FRE possono essere usate anche all'interno dei programmi; in tal caso si impiegano generalmente in espressioni di questo tipo:

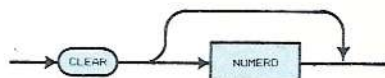
```
IF MEM(2700 THEN ...
```

CLEAR (D/P)

L'istruzione CLEAR azzerà tutte le variabili, sia numeriche che di stringa, e riserva a queste ultime il numero di caratteri indicato dalle cifre che seguono la parola chiave.

Il diagramma di sintassi di CLEAR è in figura 71.

fig. 71 diagramma di sintassi di CLEAR



Si nota che il numero dopo CLEAR può essere omissso; in tal caso il computer assume come numero di default 50: ciò significa che per le stringhe il sistema riserva 50 caratteri. Se, durante lo svolgimento del programma, l'uso delle stringhe comporta il superamento di questo limite, si ottiene una segnalazione d'errore (OUT OF STRING SPACE).

Caso per caso quindi occorre valutare il numero di caratteri impegnati per le stringhe, e riservare ad esse un numero appropriato di byte.

All'avviamento del computer, il sistema riserva automaticamente 50 caratteri, che in molti casi sono sufficienti; se in un programma non si fa uso di stringhe si può annullare lo spazio ad esse riservato:

```
CLEAR 0
```

In tal modo si liberano 50 celle di memoria, che possono essere sfruttate in altri modi (programma più lungo oppure un maggior numero di elementi nelle matrici).

Dato che CLEAR azzerà anche le variabili, occorre metterlo all'inizio del programma.

STOP E CONT (D/P)

L'istruzione STOP viene inserita in un programma generalmente in sede di debug, per controllarne il funzionamento: con esso lo svolgimento del programma viene interrotto ed il calcolatore si porta allo stato di comandi diretti, proprio come se fosse stato fatto un break. Il vantaggio rispetto a quest'ultimo è che lo si può mettere esattamente nella posizione desiderata.

CONT serve invece per fare riprendere il programma dal punto in cui si era fermato. Da notare che STOP è un'istruzione, mentre CONT è un comando. I diagrammi di sintassi di STOP e CONT sono in FIGURA 72.

fig. 72 diagrammi di sintassi di STOP e CONT



Nell'uso di CONT bisogna sapere che esso non può diventare operante se dopo l'arresto del programma sono state editate una o più linee; si possono quindi chiedere valori di variabili, modificarli a livello di comandi diretti, listare il programma (tutto o in parte), mandare in esecuzione comandi DOS, ecc.: dopo ognuna di queste cose è possibile riprendere l'esecuzione dal punto interrotto di STOP (o da un break).

LET (D/P)

L'istruzione LET serve per assegnare il valore ad una variabile.

Il suo diagramma di sintassi è in FIGURA 73.

fig. 73 diagramma di sintassi di LET



Come si vede, l'uso di LET è facoltativo e generalmente non viene applicato; esiste in molti BASIC per renderli compatibili con quelli di altri computer. Per ESPRESSIONE, nel diagramma, è da intendersi o un valore numerico, o un'espressione algebrica, o una variabile, o una stringa (nel caso di tratti di una definizione di variabile alfanumerica).

Ecco alcuni esempi leciti:

```
LET GN=.067
GN=0.067
F$=A1$+"FERMO POSTA"
DD#=E#/23.886
I%=INT(C-3*H(7))
LET NM$=LEFT$(MM$, LEN(MM$)-5)+Q$
```

I primi due casi sono coincidenti: LET e lo zero prima del punto decimale sono facoltativi. Col terzo esempio si definisce la variabile di stringa F\$ come la somma di A1\$ e della frase tra virgolette. Segue poi la definizione della variabile numerica DD in doppia precisione e di quella intera I, quindi ancora una definizione di variabile alfanumerica.

Non sono possibili invece inizializzazioni di questo tipo:

```
5.82=ER
BT=C(2)=3.71
```

Col primo caso non si è definita la variabile ER, ma si è introdotta la linea numero 5 di contenuto .82 = ER (che è un non senso e causa una segnalazione d'errore se il programma con quella riga viene messo in esecuzione).

Il secondo caso non manda in errore, ma azzerà BT e lascia C(2) al suo valore.

UN GIOCO ROMPICAPO: MASTERMIND

Vediamo ora un altro programma che interesserà coloro che sono appassionati di giochi d'abilità e d'intuito; si tratta del MASTERMIND, probabilmente noto alla maggior parte dei lettori.

Esso non è eccessivamente lungo da digitare considerando che la metà circa delle linee è impegnata all'assegnazione del punteggio e alle informazioni, parti che possono, in un primo momento, anche essere omesse; se il gioco vi piace, potete sempre aggiungerle in seguito.

Le istruzioni per il gioco possono essere lette direttamente sul listato del programma; è un gioco ormai classico, per cui non ci dilunghiamo troppo su questo punto. Oltretutto esso è tornato prepotentemente di moda in questi ultimi tempi, dedicato anche ai bambini più piccoli perché basato sull'uso di segni + e segni — per giudicare il numero proposto. Il computer svolge due mansioni: determina in modo casuale un numero di 5 cifre non ripetute e mostra sul video la situazione dei segni + e — corrispondente al numero proposto dal giocatore. Se il numero pensato (si fa per dire) dal computer fosse questo:

8 0 3 1 9

si avrebbero le seguenti situazioni (i numeri sono quelli proposti dal giocatore ed i segni sono il responso del computer):

```
22444 (NULLA)
26078 {-- }
50139 {+--- }
```

Il meccanismo per la sua semplicità e intuizione non dovrebbe dar adito a dubbi. Il giocatore può proporre anche un numero contenente due o più cifre doppie (al limite potrebbe dare anche 5 cifre tutte uguali tra di loro, ottenendo in risposta o un solo segno + o nulla). Niente vieta di modificare il programma in modo che vengano accettati solo numeri con 5 cifre diverse; in tal modo il gioco risulta un po' più difficile.

La FIGURA 74 contiene il listato e i riferimenti del programma; il suo diagramma di flusso si trova in FIGURA 75.

Nel programma, che su disco chiameremo MASTMIND (in quanto la lunghezza dei nomi non deve superare gli 8 caratteri), sono presenti molti loop; si tratta quindi di un ottimo esercizio di applicazione del ciclo FOR...NEXT e la comprensione del suo funzionamento dovrebbe essere di valido aiuto per i principianti.

fig. 74 listato di Mastmind e riferimenti

```

1000 '=====
1010 '          MASTER MIND          1982 NE-R.C.
1020 '=====
1030 CLS
1040 CLEAR 60
1050 C$=STRING$(32,CHR$(32)) : E$="Il numero deve avere 5 cifre"
1060 REVON
1070 PRINT @192,"    M A S T E R    M I N D    "
1080 REVOFF
1090 PRINT @480," PREMI UN TASTO (I=ISTRUZIONI)";
1100 GOSUB 2030
1110 CLS
1120 IF X$<>"i" AND X$<>"I" THEN 1320
1130 '+++++ istruzioni +++++
1140 PRINT "Il computer pensa un numero di 5 cifre e tu devi indovinarlo.
1150 PRINT "Il numero non ha cifre doppie e puo' iniziare con zero.
1160 PRINT
1170 PRINT "Ad ogni tentativo il computer risponde con un + per ogni
1180 PRINT "cifra esatta sia per posizione che per valore, e con un - se
1190 PRINT "hai indovinato il valore di una cifra ma non la sua posizione
1200 PRINT "esatta.
1210 GOSUB 2000
1220 PRINT "Ad ogni tentativo devi scrivere un numero di 5 cifre e premere
1230 PRINT "RETURN.
1240 PRINT
1250 PRINT "Il computer ti da' 20 tentativi poi interrompe il gioco e dice
1260 PRINT "il numero che aveva pensato. Se vuoi fermarti prima, devi
1270 PRINT "digitare il punto interrogativo e premere RETURN.
1280 PRINT : PRINT
1290 PRINT " ..... A U G U R I .....
1300 GOSUB 2000
1310 '+++++ formazione numero casuale senza ripetizioni +++++
1320 RANDOM
1330 A$="" : N$=""
1340   FOR I%=1 TO 5
1350     A(I%)=RND(10)-1
1360     IF I%=1 THEN 1400
1370     FOR J%=1 TO I%-1
1380       IF A(I%)=A(J%) THEN 1350
1390     NEXT J%
1400     A$=A$+STR$(A(I%))
1410   NEXT I%
1420 '----- N$=numero senza spazi -----
1430   FOR I%=2 TO 10 STEP 2
1440     N$=N$+MID$(A$,I%,1)
1450   NEXT I%
1460 CLS
1470 '+++++ tentativi +++++
1480   FOR I=1 TO 20
1490     REVON
1500     PRINT " TENTATIVO #I;
1510     REVOFF
1520     PRINT TAB(19)"";
1530     FOR T%=0 TO 9 : SP(T%)=0 : SM(T%)=0 : NEXT T%
1540     LINEINPUT T$
1550     IF T$="" THEN CLS : I=I-1 : GOTO 1830
1560     IF LEN(T$)<>5 THEN PRINT E$ : FOR J=0 TO 800 : NEXT J : PRINT CHR$(27)
CHR$(27)CHR$(31); : GOTO 1490
1570     IF T$=N$ THEN 1750
1580     PRINT CHR$(27)" TENTATIVO #I;TAB(19)T$;TAB(26)"";
1590     FOR J%=1 TO 5
1600       T(J%)=VAL(MID$(T$,J%,1))
1610     NEXT J%
1620 '----- vettore SP() e segni PIU' -----
1630     FOR K%=1 TO 5
1640       IF T(K%)=A(K%) THEN PRINT "+"; : SP(T(K%))=1
1650     NEXT K%

```

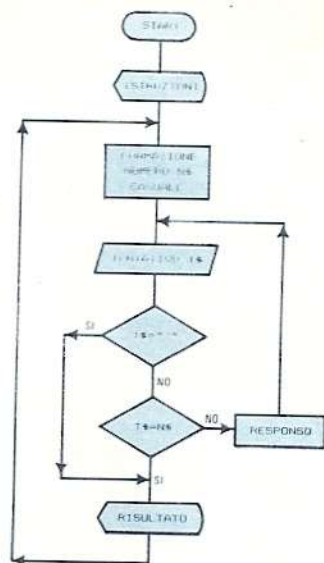
```

1660 '----- vettore SM() e segni MENO -----
1670     FOR K%=1 TO 5
1680         FOR LZ=1 TO 5
1690             IF T(K%)=A(LZ) AND SP(T(K%))=0 AND SM(T(K%))=0 THEN SM(T(K%))=1
1700                 : PRINT "-";
1710             NEXT LZ
1720         NEXT K%
1730     PRINT
1740     NEXT I : I=I-1
1750 '+++++ risultato +++++
1760 CLS
1770 IF I=1 THEN Z$=" HAI UNA FORTUNA SFACCIATA !! " : GOTO 1840
1780 ON I/4+1 GOTO 1780,1790,1800,1810,1820,1830
1790 Z$=" HAI FORTUNA !! " : GOTO 1840
1800 Z$=" BENISSIMO !! " : GOTO 1840
1810 Z$=" BENE !! " : GOTO 1840
1820 Z$=" MALE !! " : GOTO 1840
1830 Z$=" MALISSIMO !! " : GOTO 1840
1840 Z$=" MEGLIO SE CAMBI GIOCO !! "
1850     FOR Z=1 TO 3
1860         PRINT @96,C$
1870     NEXT Z
1880     FOR X=0 TO 99 : NEXT X
1890     REVON
1900     PRINT @96,Z$
1910     REVOFF
1920     FOR X=0 TO 99 : NEXT X
1930     NEXT Z
1940     PRINT : PRINT
1950     REVON
1960     PRINT " NUMERO : "A$ "
1970     REVON
1980     PRINT "# dei tentativi effettuati : "I
1990     PRINT : PRINT
2000     GOTO 1090
2010 '+++++ subroutines +++++
2020     REVON
2030     PRINT @480," per continuare premi un tasto ";
2040     REVON
2050     IF X$=INKEY$ : IF X$="" THEN 2030
2060     CLS
2070     RETURN

```

0	1530/3 1560 1690/2	1090 1980	1450/% 1480 1500
	1860 1900	1320 1120	1550/2 1580 1730/3
1	1340 1350 1360 1370/2	1350 1380	1760 1770 1960
	1440 1480 1550 1590	1400 1360	J 1370/% 1380/% 1390/%
	1600 1630 1640 1670	1490 1560	1560/2 1590/% 1600/%2
	1680 1690 1730 1760	1750 1570	1610/%
	1770 1840	1780 1770	K 1630/% 1640/%3 1650/%
2	1430/2	1790 1770	1670/% 1690/%4 1710/%
3	1840	1800 1770	L 1680/% 1690/% 1700/%
4	1770	1810 1770	N 1060 1330/\$ 1440/\$2
5	1340 1560 1590 1630	1820 1770	1490 1570/\$ 1870
	1670 1680	1830 1550 1770	1930 2000
9	1530	1840 1760 1780 1790 1800	SM 1530(1690(2
10	1350 1430	1810 1820	SP 1530(1640(1690(
19	1520 1580	2000 1210 1300	T 1530/%4 1540/\$ 1550/\$
20	1480	2030 1100 2030	1560/\$ 1570/\$ 1580/\$
26	1580	A 1330/\$ 1350(1380(2	1600/\$ 1600(1640(2
27	1560/2 1580	1400/%2 1400(1440/\$	1690(4
31	1560	1640(1690(1740/\$	X 1120/%2 1860/2 1900/2
32	1050/2	C 1050/\$ 1850/\$	2030/\$2
60	1040	E 1050/\$ 1560/\$	Z 1760/\$ 1780/\$ 1790/\$
96	1850 1880	FF 1080 1510 1890 1950	1800/\$ 1810/\$ 1820/\$
99	1860 1900	2020	1830/\$ 1840 1880/\$
192	1070	I 1340/% 1350/% 1360/%	1910
480	1090 2010	1370/% 1380/% 1400/%	
800	1560	1410/% 1430/% 1440/%	

fig. 75 diagramma di flusso di mastmind



Come abbiamo fatto altre volte, nel listato ci sono diverse linee le cui istruzioni sono marginate a sinistra di 3 oppure 6 caratteri: ciò semplifica di molto l'individuazione dei vari loop a colpo d'occhio, soprattutto quando si è in presenza di loop nidificati, ossia uno dentro l'altro. Quelli più interni sono spostati più a destra, quindi sono facilmente separabili per poter meglio comprendere la loro funzione. Resta comunque sottinteso che l'eliminazione delle marginature non pregiudica il funzionamento del programma: anzi, consigliamo di non metterle sia per fare prima a digitare il tutto, sia per risparmiare memoria centrale (in questo caso comunque l'ingombro non è tale da costringere a fare economia di caratteri).

Analizziamo dapprima velocemente il diagramma di flusso. All'inizio troviamo le istruzioni del gioco (vedremo che si possono saltare), poi il computer genera un numero casuale di cinque cifre, depositandolo nella variabile di stringa `NS`. Non stupisca il fatto di usare stringhe per trattare numeri: è un procedimento molto frequente nei programmi. Vedremo, nel nostro caso, il motivo per in quale si adotta una tale soluzione.

A questo punto il giocatore deve proporre un numero (sotto forma di stringa `TS`); se è digitato un punto interrogativo si passa alla parte finale, dove viene visualizzato il numero generato dal calcolatore assieme ad altri dati. Se invece si scrive un numero di 5 cifre, l'elaboratore ne analizza il contenuto raffrontandolo col numero generato, ed assegna il responso basato sui simboli `+` e `-`; si torna poi all'introduzione di un nuovo numero. Il processo ha termine quando `TS` risulta uguale a `NS` (in effetti il programma è fatto in modo da finire dopo 20 tentativi sbagliati). Dopo la visualizzazione dei dati si torna all'inizio, e si può ripetere il ciclo; per uscire dal programma occorre fare un `break`.

Ci preme far notare che un diagramma di flusso può essere estremamente dettagliato oppure molto succinto; generalmente si preferisce l'ultima soluzione per esprimere meglio il funzionamento del programma; in questo compariranno molte cose che non sono nel diagramma.

Prima di analizzare il contenuto delle varie linee, vediamo un'istruzione nuova: `STRING$`.

STRING\$ (D/P)

L'istruzione `STRING$` è molto utile quando si devono trattare stringhe lunghe formate da caratteri tutti uguali tra di loro: anziché scriverle per intero si possono definire velocemente facendo uso, appunto, di `STRING$`.

Il diagramma di sintassi di quest'istruzione si trova in FIGURA 76.

fig. 76 diagramma di sintassi di `STRING$`



STRING\$ è una variabile speciale riservata al sistema; dopo di essa si deve mettere una parentesi, poi un'espressione numerica, poi una virgola seguita da una stringa, e per finire si chiude la parentesi.

Vediamo un esempio:

```
A$=STRING$(16, "*")
```

Nella variabile alfanumerica A\$ viene depositata una stringa di 16 caratteri tutti uguali all'asterisco; è evidente il vantaggio di definire A\$ in questo modo, anziché dover battere 16 volte quel carattere. Ovviamente la convenienza si fa sentire in misura sempre maggiore all'allungarsi della stringa definita: se i caratteri fossero 208, ad esempio, non esisterebbero dubbi in proposito.

Per espressione numerica, al solito, si intende una delle seguenti possibilità: numero, variabile numerica o espressione algebrica. Come stringa si può introdurre una variabile alfanumerica, oppure una stringa tra virgolette, od anche definire un carattere mediante l'uso di CHR\$.

Ecco qualche esempio d'uso:

```
PRINT STRING$(13, "M")
PRINT STRING$(13, "MAMMA")
D$=STRING$(J1, CHR$(58))
D$=STRING$(J1, ":")
AZ$=STRING$(3*C, H$)
LPRINT STRING$(137, RIGHT$(NE$, 1))
```

I primi due casi forniscono un risultato identico: la stampa di 13 lettere M. Ciò significa che solo il primo carattere della stringa da ripetere viene preso in considerazione, ed i rimanenti vengono ignorati.

Il terzo esempio deposita nella variabile alfanumerica D\$ una stringa di caratteri tutti uguali al codice ASCII 58 (corrispondente ai due punti), in numero uguale al valore della variabile numerica J1; se essa supera 255 si ha una segnalazione d'errore, in quanto quello è il limite massimo alla lunghezza di qualsiasi stringa. Il caso che segue è equivalente, in quanto al posto del codice 58 si sono messi direttamente i due punti.

Considerazioni simili a quelle già svolte valgono per gli ultimi due esempi.

CHR\$ E ASC (D/P)

In FIGURA 77 è riportato il diagramma di sintassi di CHR\$.

fig. 77 diagramma di sintassi di CHR\$



CHR\$ è un'altra variabile speciale riservata al sistema; in essa viene depositato il carattere corrispondente al codice ASCII specificato entro parentesi; esso viene normalmente dato direttamente sotto forma di numero, ma potrebbe anche essere il risultato di un'espressione numerica. In ogni caso il valore di questo parametro non deve essere maggiore di 255, altrimenti si incorre in una segnalazione d'errore.

Passiamo ora alla parola chiave ASC, il cui diagramma di sintassi è in FIGURA 78.

fig. 78 diagramma di sintassi di ASC



Il compito svolto dalla funzione ASC è esattamente l'inverso di quello di CHR\$: fornisce il valore decimale in codice ASCII del primo carattere della stringa posta tra parentesi. Ecco qualche esempio:

```
AS=ASC("MICA")
PRINT ASC(Y$)
IF ASC(AA$)=80 THEN ...
```


Nel primo caso si deposita nella variabile AS il numero 77 (codice ASCII decimale della lettera M maiuscola; se fosse minuscola si avrebbe 109). Il secondo esempio stampa il codice ASCII del primo carattere contenuto in Y\$; in quello successivo l'istruzione che segue THEN viene eseguita se il codice ASCII del primo carattere della stringa AAS vale 80 (ossia se AAS comincia con la lettera P maiuscola).

Rammentiamo che i codici ASCII cambiano a seconda che ci si trovi in stampa normale o in reverse, come appare anche dalla figura 1 del fascicolo precedente. Per mostrare tutti i codici ASCII sia in stampa normale che negativa, provate a far girare il seguente programma:

```
10 CLS:R=ABS(R)-1:IFRREVDFELSEREVON:PRINT"REVON":PRINT
20 FORI=1TO255:PRINTI;TAB(6);:IFNOTRPRINTCHR$(192);
30 PRINTCHR$(I):FORJ=1TO200:NEXTJ:PRINT:NEXTI:FORJ=1TO1000:NEXTJ:GOTO10
```

Esso è stato scritto eliminando tutti gli spazi e concentrandolo in tre righe: è un ottimo esempio di quello che non si dovrebbe mai fare, in quanto si ottiene un listato difficilmente leggibile ed un programma poco comprensibile. La sua esecuzione mostra, in una prima tornata, tutti i caratteri dei codici ASCII in stampa normale, preceduti dal loro numero di codice; arrivati a 255 si riparte da capo, ma in reverse. Si possono così notare le differenze dal caso precedente.

I codici dall'uno al 31 sono in parte occupati da caratteri di controllo:

- 8 - spazio indietro
- 10 - <LF> «line feed»
- 13 - <CR> (ritorno carrello)
- 14 - cursore acceso
- 15 - cursore spento
- 23 - converte da 32 a 16 caratteri per riga video
- 24 - spazio indietro del cursore
- 25 - spazio avanti del cursore
- 26 - una linea in giù
- 27 - una linea in su
- 28 - HOME (cursore in alto a sinistra)
- 29 - cursore all'inizio della riga
- 30 - cancella fino alla fine della riga
- 31 - cancella lo schermo dalla posizione del cursore fino in fondo.

I codici dal 224 al 255 effettuano tabulazioni, rispettivamente, da zero a 31 spazi.

Provate ora quest'altro programma:

```
10 CLS : PRINT " CODICI ASCII NORMALI E REVERSE"
20 REVON : PRINT @128," codice" : REVOFF
30 PRINT @450,"premere un tasto per fermare"
40 FOR I=32 TO 255
50 REVON : PRINT @135,I : REVOFF : PRINT @192,"-CHR$(I)- "
60 PRINT @288,"-"; : REVON : PRINT CHR$(I); : REVOFF : PRINT "- "
70 FOR J=1 TO 400 : NEXT J,I
80 PRINT : PRINT "da capo !" : FOR J=1 TO 1500 : NEXT : GOTO 10
```

Ricordare le considerazioni fatte in precedenza per REVON e REVOFF: mettendo uno spazio tra di essi e gli eventuali due punti seguenti si ha una segnalazione d'errore. Questo però vale solo per la prima versione del BASIC; con quelle seguenti l'imperfezione è stata corretta. Per questo motivo d'ora in poi non terremo più conto di questo fattore.

Facciamo qualche considerazione sul programma appena visto.

In questo caso i caratteri corrispondenti ai vari codici ASCII compaiono sullo schermo uno per volta, nelle due versioni normale e reverse. Inoltre ogni carattere è preceduto e seguito da un trattino (segno meno): questo torna utile dal codice 225 in poi, quando ogni carattere corrispondente fornisce un certo numero di spazi; in stampa normale non si capirebbe quanti sono, mentre i trattini non lasciano adito a dubbi.

Con entrambi i programmi ci si può rendere conto che fino al codice 128 si ottengono gli stessi caratteri sia in scrittura normale che reverse; da quel punto in poi, invece, cominciano le differenze.

Esaminiamo ora, finché siamo in argomento, altre funzioni di stringa.

LEFT\$ E RIGHT\$ (D/P)

Queste due istruzioni svolgono operazioni simili, quindi le trattiamo assieme. I loro diagrammi di sintassi sono nelle FIGURE 79 e 80.

fig. 79 diagramma di sintassi di LEFT\$

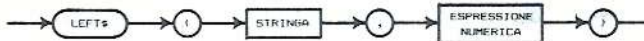


fig. 80 diagramma di sintassi di RIGHT\$



Vediamo un esempio per LEFT\$:

```
SL$=LEFT$("CARLOTTA", 5)
```

Con questa istruzione diciamo al computer di prendere i primi cinque caratteri della stringa tra virgolette, e di depositarli nella variabile alfanumerica SL\$; si avrà quindi la parola CARLO.

Vediamo un esempio per RIGHT\$:

```
SRS=RIGHT$("CARLOTTA", 5)
```

In questo caso nella variabile SRS va a finire la parola LOTTA, ossia i cinque caratteri a destra della parola di partenza.

Ecco qualche applicazione delle due istruzioni:

```
LPRINT RIGHT$(G$, 8)
IF LEFT$(RC$, 4) = "VITE" THEN ...
E$=LEFT$(NN$, 3*2)+RIGHT$(D$, 1)
K$=RIGHT$(LEFT$("NUOVA ELETTRONICA", 5), 4)
```

Con l'ultimo caso si ottiene in K\$ la parola UOVA.

L'espressione numerica dopo la virgola deve essere minore di 255 e positiva, altrimenti il calcolatore va in errore; può invece essere decimale (in tal caso il computer considera solo la parte intera) od anche zero (nessun carattere viene prelevato dalla stringa di partenza).

MID\$(D/P)

La funzione di stringa MID\$ è concettualmente simile alle due precedenti, con la differenza che serve per estrarre una stringa dal mezzo di un'altra.

Il diagramma di sintassi di MID\$ è in FIGURA 81.

fig. 81 diagramma di sintassi di MID\$



Il primo numero dopo la virgola indica la posizione, all'interno della stringa di partenza, da cui si inizia il prelievo; il secondo numero dice quanti caratteri vanno presi.

Vediamo un esempio:

```
NE$="NUOVA ELETTRONICA" : PRINT MID$(NE$, 2, 4)
```

Si ottiene la parola UOVA, come in uno dei casi visti prima.

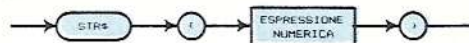
Per le due espressioni numeriche tra parentesi valgono le stesse considerazioni fatte poc'anzi. Da notare che se la prima di esse vale 1 si ottiene una funzione uguale, come effetto, a LEFT\$; analogamente, variando opportunamente i due valori, si può avere un funzionamento uguale a RIGHT\$.

STR\$(D/P)

La funzione di stringa STR\$ ha un comportamento inverso a quello già visto per VAL: trasforma un numero in una stringa.

Il diagramma di sintassi di STR\$ è in FIGURA 82.

fig. 82 diagramma di sintassi di STR\$



Si tratta di un'altra variabile speciale riservata al sistema. Ecco qualche esempio:

```
PRINT STR$(503.98)
V$=STR$(A*2.01-D/C→ 2)
IF STR$(A*B)=C$ THEN ...
```

Facciamo notare che un numero trasformato in stringa conserva uno spazio iniziale (quello riservato al segno) anche se il valore è positivo; perde invece lo spazio finale. Ad esempio, nel primo caso la stringa ottenuta ha una lunghezza di 7 caratteri.

Occorre fare molta attenzione nell'uso di STR\$, per non incorrere in sorprese; ad esempio, scrivendo:

```
PRINT STR$(1234567)
```

si ottiene la seguente visualizzazione:

```
1.23457E+06
```

La cosa si spiega pensando a tutte le considerazioni fatte sui numeri in singola e doppia precisione; il numero assegnato è di sette cifre, quindi viene rappresentato in virgola mobile con solo sei cifre (l'ultima delle quali è arrotondata), il tutto seguito dal valore dell'esponente (+ 6 in questo caso; la lettera E cambia in D se si tratta di una rappresentazione in virgola mobile di un numero in doppia precisione).

Per terminare l'esame delle varie funzioni di stringa, escluse quelle usate nella gestione dei file random, dobbiamo vederne ancora due: LEN e INSTR.

LEN (D/P)

La funzione di stringa LEN determina la lunghezza di una stringa. Il suo diagramma di sintassi è in FIGURA 83.

fig. 83 diagramma di sintassi di LEN



Si tratta di un'altra variabile numerica riservata al sistema (rammentiamo che questo comporta una segnalazione d'errore se si cerca di usare un identificatore uguale). Vediamo qualche esempio:

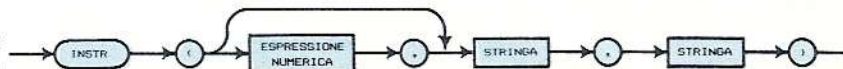
```
H=LEN("NUOVA ELETTRONICA")
TY=5*A-LEN(W$)
PRINT RIGHT$(STR$(55), LEN(STR$(55))-1)
```

Il primo caso restituisce il numero 17; l'ultimo fornisce il modo per ottenere una stringa da un numero, eliminando il carattere vuoto iniziale quando il numero stesso è positivo. Se tale procedimento viene applicato ad un numero negativo, il segno meno non entra nella stringa finale.

INSTR (D/P)

La funzione INSTR serve per scoprire se una stringa esiste all'interno di un'altra. Il suo diagramma di sintassi è riportato in FIGURA 84.

fig. 84 diagramma di sintassi di INSTR



Per spiegare il funzionamento di INSTR, definiamo prima una stringa nel modo che segue (operando a livello di comandi diretti):

```
N$="CALCOLATORE ELETTRONICO"
```

Qui di seguito riportiamo alcuni esempi fatti sulla stringa appena definita; tra parentesi compaiono i risultati di ogni caso:

```
PRINT INSTR(N$, "E")      (11)
PRINT INSTR(N$, "e")      (0)
PRINT INSTR(3, N$, "E")    (0)
PRINT INSTR(11, N$, "E")   (11)
PRINT INSTR(13, N$, "E")   (13)
PRINT INSTR(1, N$, "E")    (11)
PRINT INSTR(N$, CHR$(69))  (11)
PRINT INSTR(N$, "ELET")    (13)
PRINT INSTR(N$, "Elet")    (0)
```

Innanzitutto c'è da dire che, come si vede anche dal diagramma di sintassi, il primo numero tra le parentesi può essere omissivo; in questo caso il suo valore di default è 1, come si verifica confrontando il primo col sesto esempio. In pratica, il primo numero indica da quale carattere, all'interno della stringa data, deve iniziare la ricerca della seconda stringa. Se questa esiste, ma si trova in un altro punto, si ottiene il numero zero; se invece è al punto indicato dal primo numero, si ottiene il numero stesso.

In ogni caso, il numero restituito dal computer prende sempre come riferimento l'inizio della stringa data.

Il settimo esempio è equivalente al primo, in quanto il codice ASCII della lettera E maiuscola è 69. Gli ultimi due casi indicano che all'interno di una stringa può essere ricercato non solo un carattere, ma anche una stringa più lunga; il numero che si ottiene fornisce sempre la posizione d'inizio della seconda stringa all'interno della prima.

Per le diciture ESPRESSIONE NUMERICA e STRINGA del diagramma di sintassi sono valide le osservazioni più volte fatte; lo stesso dicasi per i valori che possono assumere i numeri all'interno delle parentesi.

Cominciamo ora l'analisi del programma MASTMIND; pensiamo di poter andare più in fretta di quanto non abbiamo fatta finora, poiché spesso si tratterebbe di ripetere le stesse considerazioni già viste.

LINEE DA 1000 A 1120

Poche sono le cose da annotare in questo tratto di programma; una è alla linea 1040, dove la normale disponibilità di caratteri per le stringhe (50) viene ampliata fino a 60. Senza questa istruzione il programma andrebbe in errore. Alla riga 1050 si definisce C\$ come stringa formata da 32 spazi; servirà alla fine, per cancellare un'intera riga video. Questo è uno dei tanti modi di risolvere il problema.

Nella linea 1100 c'è il rinvio alla subroutine che ferma il programma fino a che non viene premuto un tasto qualsiasi; in effetti quella subroutine inizia alla riga 2000, ma nulla vieta di entrare in essa anche più avanti. Il motivo per cui si è agito in questo modo è che, all'inizio del programma, serve una scritta diversa da quella usata successivamente, a causa della possibilità di scegliere la lettura delle istruzioni premendo il tasto della lettera I.

Al ritorno dalla subroutine, dopo la cancellazione dello schermo, c'è la linea 1120 che si incarica di gestire la scelta fatta: se è stato premuto il tasto della lettera I (maiuscola o minuscola) si ottengono le istruzioni del gioco, altrimenti si passa alla riga 1320. Sono state messe le due alternative per la lettera I perché nei BASIC successivi al primo esiste la possibilità di configurare la tastiera in modi diversi (maiuscole col tasto SHIFT premuto oppure no).

LINEE DA 1130 A 1300

Questa parte contiene le istruzioni del gioco, in due pagine video; per passare da una all'altra si deve premere un tasto qualunque (subroutine 2000).

LINEE DA 1310 A 1460

Qui il computer forma un numero casuale di cinque cifre, senza che ne esistano delle doppie.

Nella riga 1360 comincia un ciclo FOR...NEXT nella variabile I%, che viene percorso 5 volte: ad ogni giro

viene generato un numero pseudocasuale A(I%).

Ciò significa innanzitutto che tali numeri sono depositati in un vettore, per essere poi trattati in un secondo tempo senza perdere la loro individualità.

Si è fatto uso di una variabile intera I% per rendere più veloce l'esecuzione del ciclo. Questa è una caratteristica importante del computer ed è bene conoscerla. Proviamo a fare i seguenti esercizi, a livello di comandi diretti:

```
FOR I=1 TO 10000 : NEXT  
FOR I=1 TO 10000 : NEXT I  
FOR I%=1 TO 10000 : NEXT  
FOR I%=1 TO 10000 : NEXT I%
```

Sono tutti loop di ritardo, ma differiscono tra di loro per i tempi di esecuzione. Infatti i tempi impiegati sono rispettivamente di circa 19, 25, 11 e 14 secondi. Le differenze sono notevoli, come vedete. Se si inserisce in un programma un loop di ritardo, ciò non ha un'importanza eccessiva, tranne nel fatto che la variabile I occupa 2 byte in più di I%. Se invece tra FOR e NEXT si eseguono delle istruzioni, il tempo globale al termine del ciclo è minore se si è usata una variabile intera anziché in singola precisione (usando una variabile di controllo del loop in doppia precisione, si ottiene una segnalazione d'errore). Questo programma fa un uso ampio delle variabili intere proprio per rendere il più veloci possibili i molti cicli FOR...NEXT presenti.

La linea 1350 è quella che genera il numero casuale compreso tra zero e nove. La 1360 manda alla riga 1400, ma solo se I vale 1, ossia al primo giro: la prima cifra estratta è sempre accettata e va a far parte della stringa A\$, nel modo che vedremo.

Il loop nella variabile J% che va dalla linea 1370 alla 1390 è quello che evita le ripetizioni: infatti la nuova cifra estratta A(I%) viene confrontata con tutte le precedenti (infatti J% varia da 1 al valore precedente di I%) e se risulta uguale ad una di esse il programma torna alla riga 1350, ossia genera un nuovo numero casuale.

Questo processo si ripete per ogni cifra estratta, tranne che per la prima, che è ovviamente sempre accettabile.

Vediamo come si forma il numero di 5 cifre, quello che deve essere indovinato. Inizialmente la variabile A\$ viene azzerata, assieme a N\$ (linea 1330). Al primo giro del loop la prima cifra, dopo essere stata trasformata in stringa con l'istruzione STR\$, viene sommata ad A\$.

Ad ogni tornata successiva si accumulano in A\$ le cifre estratte, dopo essere state trasformate in stringhe. Se supponiamo che le cifre siano 3 9 0 1 7, A\$ alla fine è lungo 10 caratteri poiché STR\$ ne lascia uno prima di ogni cifra: è come se A\$ fosse stato definito nel modo seguente:

```
A$=" 3 9 0 1 7"
```

Ciò spiega la presenza del loop che parte dalla linea 1430: con esso, dalla stringa anzidetta, si estrarrebbero solo le cifre tralasciando gli spazi e il risultato viene depositato in N\$. Al primo giro I% vale 2 e in N\$ è posto il secondo carattere di A\$, estraendolo con l'istruzione MID\$; al secondo giro I% vale 4 (ossia il valore iniziale più il passo 2), quindi viene estratta solo la seconda cifra, e così via fino alla quinta.

Alla fine è come se avessimo definito N\$ in questo modo:

```
N$="39017"
```

LINEE DA 1470 A 1610

Alla linea 1480 inizia l'introduzione dei tentativi da parte del giocatore; si tratta di un loop percorso al massimo 20 volte. Da notare che la variabile di controllo di questo ciclo può essere in singola precisione, senza comportare allungamenti apprezzabili nei tempi di esecuzione.

Innanzitutto viene scritto in reverse TENTATIVO #, seguito dal numero progressivo I; con la 1540 si porta il cursore alla tabulazione 19; notare l'uso dei punti e virgola.

La linea 1530 azzerata il contenuto dei vettori SP(T%) e SM(T%), formati ciascuno da 10 elementi (da zero a nove); per ognuna delle possibili cifre estratte (appunto dieci, da zero a nove), SP assumerà il valore 1 se a quella cifra va assegnato il segno + (cifra esatta per valore e posizione). Analogamente SM passerà al valore 1 se la cifra interessata è giusta di valore ma al posto sbagliato (segno — sul monitor). Si usano spesso, come in questo caso, identificatori di variabili che abbiano un aggancio mnemonico alla loro funzione: SP sta per Segni Più, SM per Segni Meno; analogamente per altre variabili del programma.

Con la linea 1540 il computer resta in attesa di un numero di cinque cifre. Esso viene chiesto sotto forma di stringa (T\$), perché torna più comodo poi separare ogni cifra dalle altre; inoltre ciò permette anche di interrompere i tentativi prima di aver indovinato il numero. Il modo per farlo è alla riga seguente: se T\$ è un

punto interrogativo, si va alla linea 1830, ossia alla fine del gioco, dopo aver cancellato lo schermo e diminuito di 1 il valore di I. La linea 1560 rifiuta il numero introdotto se è formato da un numero di cifre diverso da cinque. La stampa dei caratteri CHR\$(27) causa uno spostamento verso l'alto di due righe e quella di CHR\$(31) cancella lo schermo dalla posizione del cursore fino alla fine della pagina video.

La riga 1570 manda alla fine del gioco se il numero scritto dal giocatore è uguale a quello generato dal computer.

Con la linea 1580 si riscrive poi la dicitura TENTATIVO # (in positivo, questa volta); ci si porta alla tabulazione 19, si riscrive T\$ (ossia il numero scritto dal giocatore) ed il cursore viene fermato alla tabulazione 29, per mezzo del punto e virgola: da quel punto partiranno i segni più e meno (se ci sono).

Prima di passare al responso coi segni, bisogna estrarre dalla stringa digitata le varie cifre; si ottiene questo col loop che va dalla linea 1590 alla 1610: nei singoli elementi del vettore T(J%) vengono depositi i valori delle cifre contenute nella stringa T\$, passandole in rassegna una per volta e trasformandole in numeri con l'istruzione VAL.

LINEE DA 1620 A 1730

Ora troviamo, come è indicato anche dalle annotazioni, la parte di programma che stabilisce i segni più o meno. Si comincia dai segni più, ossia dalle cifre esatte per posizione e valore. Il loop che parte alla linea 1630 per terminare alla 1650 confronta appunto la prima cifra del tentativo con la prima del numero casuale generato dal computer; se esse sono uguali si ha la stampa di un segno + ed il corrispondente elemento del vettore SP passa da zero a uno. Il procedimento si ripete 5 volte, passando in rassegna tutte le cifre.

Se supponiamo che il numero introdotto dal giocatore sia 25037 e che quello del computer sia sempre 39017, alla fine del loop avremo ottenuto 2 segni + (corrispondenti alle cifre zero e 7), e la situazione degli elementi del vettore SP sarà la seguente (cominciando dalla cifra zero, poi 1 e così via fino al nove):

1 0 0 0 0 0 0 1 0 0

Gli elementi sono tutti nulli tranne quelli corrispondenti allo zero e al 7. Questo sistema ci tornerà utile alle righe successive, per evitare di dare un segno meno ad una cifra che si trovi al posto giusto; i singoli elementi del vettore SP hanno la funzione di FLAG, ossia di segnalatori di segno + per la cifra cui sono associati.

L'assegnazione dei segni meno è più difficoltosa. Infatti occorre non solo non darli a cifre che hanno già avuto un più, ma bisogna assegnarne non più di uno nel caso che il numero del giocatore contenga delle cifre che compaiono più di una volta. Il risultato si ottiene facilmente usando un flag anche per i segni meno, esattamente come si è fatto per i più.

Alla linea 1670 comincia un loop percorso 5 volte, che ha al suo interno un secondo ciclo anch'esso effettuato 5 volte. Quello esterno (lettera K%) prende le varie cifre del numero tentativo, cominciando dalla prima. Ogni cifra viene confrontata con tutte quelle del numero generato dal calcolatore. Quando risulta che due cifre sono uguali ($T(K\%) = A(L\%)$), il computer va a vedere se quella cifra ha già avuto un segno + (SP) oppure un segno - (SM). Se la risposta è no per entrambi i casi, stampa un segno meno, mette ad uno l'elemento SM corrispondente e procede ad altri confronti (25 in tutto).

Facciamo presente che finora siamo sempre all'interno del loop iniziato alla riga 1480; gli altri visti sono nidificati con questo. Dal loop iniziale si esce in tre modi diversi: se si introduce il punto interrogativo, se il tentativo è uguale al numero del computer, se si sono date 20 risposte senza indovinare. In ogni caso si arriva alla parte finale.

Esaminiamo ora in dettaglio una nuova istruzione: ON...GOTO.

ON...GOTO (P)

ON...GOTO serve per effettuare rinvii a numeri di linea diversi, a seconda del valore assunto dalla variabile numerica indicata nell'istruzione.

Il diagramma di sintassi è in FIGURA 85.

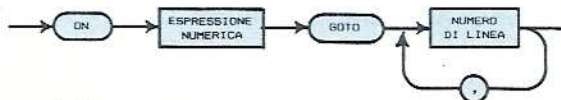


fig. 85 diagramma di sintassi di ON...GOTO

Al solito, ESPRESSIONE NUMERICA indica una variabile oppure un'espressione algebrica. Vediamo

qualche applicazione:

```
ON C% GOTO 1200, 3050, 700
ON AF-5.71*S3 GOTO 300, 400, 500, 600, 700
```

Il primo esempio significa che a seconda (ON) del valore assunto da C% deve andare alle linee (GOTO) 1200, 3050, 700. Il funzionamento avviene in questo modo: se C vale 1 il programma prosegue alla riga 1200, se vale 2 si passa alla 3050, se vale 3 si va alla 700.

Dal modo stesso che è stato usato per la spiegazione, si vede che ON...GOTO si usa al posto di una serie più o meno lunga di IF...THEN; con una sola linea di programma si risolve in modo semplice e breve una situazione che altrimenti richiederebbe più spazio e non godrebbe di altrettanta chiarezza.

Restano da esaminare altri casi possibili; se C% vale più di tre, il controllo del programma passa alla linea seguente a quella in cui si trova ON...GOTO; se C% assume valori decimali, questi vengono arrotondati all'intero immediatamente inferiore. Se C% vale zero si passa ugualmente alla linea successiva; se C% è negativo si ottiene una segnalazione d'errore.

LINEE DA 1740 A 2050

Si arriva alla riga 1750 da quelle immediatamente precedenti (più di 20 risposte errate) o dalla 1570 (risposta esatta).

In questa seconda ipotesi, supponendo che si sia verificata al primo tentativo, il computer scrive la frase della riga 1760 e manda alla riga 1840, che vedremo poco più avanti; diversamente si prosegue.

La linea 1770, a seconda del valore dell'espressione $I/4 + 1$, rimanda alle righe indicate, dove si trovano vari giudizi basati sul numero di tentativi fatti dal giocatore prima di indovinare il numero. Il giudizio più drastico, quello della linea 1830, si ha se il numero non è stato indovinato entro 20 tentativi, oppure se si è rinunciato prima premendo il punto interrogativo (linea 1550).

Le righe da 1840 a 1910 servono per fare lampeggiare per 3 volte il responso ricevuto dal computer, in reverse.

Con la 1940 si stampa poi il numero casuale generato dal calcolatore; notare che a questo scopo è stata usata la stringa A\$, quella lunga 10 caratteri.

Viene poi visualizzato il numero dei tentativi fatti, e c'è infine il rinvio alla linea 1070, all'inizio del programma.

Le linee dalla 2000 alla fine contengono la subroutine per la richiesta di un carattere da tastiera; di essa si è già parlato.

Completa la tua biblioteca tecnica con il **15° volume** di **NUOVA ELETTRONICA**



Ogni volume, di oltre 510 pagine, completo di copertina brossurata e plastificata **L. 12.500**

In ogni volume sono raccolte le seguenti riviste:

Volume 1 riviste dal n. 1 al n. 6
Volume 2 riviste dal n. 7 al n. 12
Volume 3 riviste dal n. 13 al n. 18
Volume 4 riviste dal n. 19 al n. 24
Volume 5 riviste dal n. 25 al n. 30
Volume 6 riviste dal n. 31 al n. 36
Volume 7 riviste dal n. 37 al n. 43

Volume 8 riviste dal n. 44 al n. 48
Volume 9 riviste dal n. 49 al n. 55
Volume 10 riviste dal n. 56 al n. 62
Volume 11 riviste dal n. 63 al n. 66
Volume 12 riviste dal n. 67 al n. 70
Volume 13 riviste dal n. 71 al n. 74
Volume 14 riviste dal n. 75 al n. 78
Volume 15 riviste dal n. 79 al n. 82

POSSIBILI CONFIGURAZIONI PER IL COMPUTER Z80

Le schede disponibili per il computer Z80 sono ormai molto numerose ed offrono vaste possibilità di uso e di specializzazione al sistema. Riteniamo quindi utile a questo punto fare un «quadro riassuntivo» di tutte le svariate possibilità di lavoro che di volta in volta si vengono ad avere.

Iniziamo perciò con una tabellina di promemoria per avere sotto mano comodamente le sigle ed il riferimento di ogni scheda:

SIGLA	DESCRIZIONE	RIVISTA
LX 380	Alimentatore in grado di fornire tutte le tensioni stabilizzate per il computer	n. 68/69
LX 381	Scheda BUS su cui vanno inserite tutte le altre schede	n. 68/69
LX 382	Scheda CPU	n. 68/69
LX 383	Interfaccia per tastiera esadecimale	n. 68/69
LX 384	Tastiera esadecimale con display	n. 68/69
LX 385	Interfaccia per registratore a cassette	n. 70
LX 386	Espansione di memoria RAM, da 8k (statica)	n. 70
LX 387	Tastiera alfanumerica	n. 72
LX 388	Interfaccia video (32 x 16 caratt. alfanum.)	n. 73
LX 389	Interfaccia parallela per stampante	n. 73
LX 390	Interfaccia per floppy disk	n. 75
LX 391	Alimentatore per floppy disk	n. 76
LX 392	Espansione di memoria RAM da 32k (dinamica)	n. 76
LX 394	Interfaccia per programmatore di eeprom	n. 75
LX 395	Programmatore di eeprom	n. 75
LX 529	Interfaccia video-grafica	n. 85/85
LX 547	Scheda interrupt	n. 89
LX 548	Basic residente su eeprom	

Le schede LX 382, LX 390 e LX 392 hanno inoltre, singolarmente, diversi modi di funzionamento e quindi anche predisposizioni di componenti e di collegamenti differenti a seconda dei casi. Più precisamente si ha:

- LX 382/A** = È la stessa scheda CPU in cui però è stato sollevato il piedino 6 dell'integrato 74L200. In questo modo vengono escluse sia le due RAM 2114 che la EPROM 2708 che quindi possono anche essere tolte dai rispettivi zoccoli. Questa modifica è necessaria quando al computer viene aggiunta la scheda di interfaccia per il floppy disk in quanto, in questo caso, la gestione del computer è affidata al monitor presente sulla scheda di interfaccia stessa.
- LX 390/A** = È la scheda di interfaccia per floppy disk con la EPROM programmata per lavorare assieme all'interfaccia video LX 388 e siglata 1.390. Su tale schema, infine, dovranno essere effettuati i ponticelli:
- P1 su A-B-C
P2 su A-C
P3 su A-C
P4 su A-B
- LX 390/B** = È ancora l'interfaccia per floppy disk adattata però al funzionamento con l'interfaccia video grafica LX 529. Le connessioni dei ponticelli rimangono invariate rispetto alla versione precedente mentre la EPROM inserita nello zoccolo è siglata 2.390.
- LX 392/A** = È la scheda di espansione di memoria da 32k dinamica, indirizzata da 0000 a 7FFF. Su questa scheda vanno effettuati i ponticelli P1, P2 e P3.
- LX 392/B** = Con questa scheda si ottengono solo 16k di memoria RAM dinamica e va effettuato il solo ponticello P4 in modo da ottenere un indirizzamento da 8000 a BFFF.

Possiamo quindi iniziare la descrizione vera e propria delle varie configurazioni ottenibili, usando le schede descritte secondo diverse combinazioni. I sistemi che si otterranno, andranno per via crescente di complessità partendo da un sistema di tipo «didattico» fino ad arrivare ad una configurazione tipo «personal computer». Seguendo perciò questa nostra traccia, potrete preventivamente decidere quali schede acquistare a seconda delle vostre specifiche esigenze, senza perdite di tempo e di denaro.

CONFIGURAZIONE 1

Le schede necessarie sono:

Alimentatore (LX 380)
BUS (LX 381)
CPU (LX 382)
Interfaccia esadecimale (LX 383)
Tastiera esadecimale (LX 384)

Questo è l'insieme minimo possibile da realizzare con le schede descritte. È la tipica configurazione a livello «didattico» con la quale si possono scrivere e provare dei programmi inseriti nel computer direttamente in linguaggio macchina, cioè con notazioni esadecimali. La memoria che il computer ha a disposizione in questa configurazione, è la sola memoria presente sulla scheda stessa della CPU e cioè non più di 1 Kbyte. È possibile comunque aggiungere una o più schede di espansione di memoria RAM statica da 8 Kbyte (LX 386) oppure una scheda di memoria dinamica da 32 Kbyte. Un'altra soluzione molto usata in questo caso per ampliare la memoria «economicamente», è quella di collegare al sistema un registratore con la relativa interfaccia (LX 385) con il quale memorizzare tutti i vari programmi scritti. In questo modo si ha sempre a disposizione tutto lo spazio di memoria presente nel sistema senza però dover perdere di volta in volta i programmi scritti in precedenza.

La limitazione maggiore di questa configurazione è che non è possibile operare in alcun modo con linguaggi evoluti, avendo la possibilità di «colloquiare» con la macchina solamente con notazioni esadecimali.

CONFIGURAZIONE 2

Le schede necessarie sono:

Alimentatore (LX 380)
BUS (LX 381)
CPU (LX 382)
Interfaccia registratore (LX 385)
Tastiera alfanumerica (LX 387)
Interfaccia video (LX 388)
Espansione di memoria almeno 8K (LX 386)

Questa è la versione massima di un sistema ancora di tipo «didattico» ed ha in più, rispetto alla versione precedente, la possibilità di lavorare con linguaggi evoluti. Infatti è ora possibile caricare in memoria, attraverso il registratore, il linguaggio BASIC da 5.5 Kbyte e fare così i primi programmi di apprendimento usando il linguaggio BASIC.

È necessario, e lo sarà sempre d'ora in poi, un televisore o un monitor video (tipo LX 599) da collegare all'interfaccia video.

CONFIGURAZIONE 3

Le schede necessarie sono:

Alimentatore (LX 380)
BUS (LX 381)
CPU (LX 382 A)
Interfaccia video (LX 388)
Tastiera alfanumerica (LX 387)
Interfaccia Floppy Disk (LX 390)

Espansione di memoria almeno 8K (LX 386) fino ad un massimo di 40K (LX 386 + LX 392).

—Dischetto con BASIC 5.5

Questa configurazione è la base minima di un sistema tipo «personal computer» ed offre già caratteristiche di programmazione e di lavoro decisamente più ampie dei casi precedenti. Dispone infatti di un monitor con funzioni ampliate rispetto a quello presente nella scheda CPU ed è possibile perciò svolgere programmi notevolmente più complessi sia in BASIC che in linguaggio macchina, avendo a disposizione, come memoria di massa, una unità floppy.

CONFIGURAZIONE 4

Le schede necessarie sono:

Alimentatore (LX 380)

BUS (LX 381)

CPU (LX 382/A)

Interfaccia video (LX 388)

Tastiera alfanumerica (LX 387)

Interfaccia Floppy Disk (LX 390/A)

Espansione di memoria almeno da 32K (LX 392/A) fino ad un massimo di 56k (LX 392 + LX 392/B + LX 386)

— Dischetto con DOS 2.1.

Il sistema così ottenuto è una configurazione notevolmente potenziata di un «personal computer» e si basa su di un Sistema Operativo residente su Disco, (DOS 2.1) che permette il caricamento di un BASIC in forma estesa, l'uso di un EDITOR ASSEMBLER e di un DISASSEMBLER e di un evoluto programma di DEBUGGING. Il sistema, che può essere espanso fino ad avere 4 drive collegati all'interfaccia per floppy disk per un totale di 400 Kbyte, è in grado di operare con programmi sia gestionali che scientifici e, con l'aggiunta dell'interfaccia per stampante (LX 389), può fornire tabulati ed impaginazioni complesse per qualunque tipo di lavorazione adattabile ad una stampante per computer.

CONFIGURAZIONE 5

Sono necessarie le schede:

Alimentatore (LX 380)

BUS (LX 381)

CPU (LX 382/A)

Interfaccia floppy Disk (LX 390/B)

Tastiera Alfanumerica (LX 387)

Interfaccia Video Grafica (LX 529)

Espansione di memoria almeno da 48K (LX 392 + LX 392/B) fino ad un massimo di 60K.

— Dischetto DOS Grafico

Quest'ultima configurazione è, per ora, l'espansione massima ottenibile dal sistema con le schede fino ad ora pubblicate. Usando l'interfaccia video grafica LX 529, è possibile collegare al computer una stampante senza dover necessariamente collegare al BUS l'interfaccia per stampante LX 389 ed inoltre, in questo modo, si ottengono dei formati di stampa corrispondenti al formato della pagina video. Le stesse capacità operative della configurazione precedente perciò, vengono ora ampliate da una grafica video nettamente superiore, tale da permettere elaborazioni molto più complesse portando direttamente sul video i tabulati ed i vari formati per la stampa.

È indispensabile, con questa interfaccia video, l'uso di un monitor tipo LX 599 in quanto un normale televisore non ha la banda passante necessaria alla corretta visualizzazione del segnale stesso.

CONFIGURAZIONE 6 — PROGRAMMATORE DI EPROM

Sono necessarie le schede:

Alimentatore (LX 380)

BUS (LX 381)

CPU (LX 382)

Interfaccia esadecimale (LX 383)

Tastiera esadecimale (LX 384)

Interfaccia per programmatore di eprom (LX 395)

Programmatore di eprom (LX 395)

Espansione di memoria almeno da 8K (LX 386) per un massimo di 24k.

Con questo tipo di configurazione si possono programmare, copiare o leggere tutti i più diffusi tipi di memorie programmabili esistenti attualmente in commercio (2708, 2716, 2516, 2732 ecc.). Volendo memorizzare i listati dei programmi da inserire nelle eprom o letti dalle eprom stesse, si può aggiungere al sistema un registratore con ovviamente la relativa interfaccia per registratore LX 385.

CONFIGURAZIONE 7 — BASIC RESIDENTE

Sono necessarie le schede:

Alimentatore (LX 380)

BUS (LX 381)

CPU (LX 382)

Basic residente (LX 548)

Espansione di memoria:

8 K Statica (LX 386)

L'ultima configurazione ora descritta è prevista per espandere e completare il computer nella sua configurazione "minima", cioè senza scheda video grafica (in questa configurazione tale scheda non può essere utilizzata) e senza dover utilizzare dei floppy disk come memoria di massa in quanto tale funzione viene svolta da un normale registratore a cassette.

La sintassi del Basic implementato su tale scheda, è descritta da pag. 92 a pag. 109 della rivista n. 96.

CONTABILITÀ SEMPLIFICATA

Questo programma è stato interamente rivisto e corretto al fine di renderne l'uso più semplice ed immediato rispetto alla versione precedente.

Quanto esposto nelle pagine seguenti metterà chiunque nelle condizioni di poter utilizzare immediatamente il programma che consente, nonostante l'estrema semplicità d'impiego, l'esecuzione di tutte le operazioni contabili relative a contabilità acquisti e vendite — (fatturazione, intestazione dei registri I.V.A., stampe dei totali delle aliquote I.V.A., stampa del modulo 13/A allegato alla dichiarazione, stampa dei codici di raggruppamento, lista dei clienti e dei fornitori in ordine alfabetico e numerico).

Il programma necessita di uno spazio di memoria di 56K, è quindi indispensabile avere collegato al computer le espansioni di memoria LX 392 da 32K, LX 392 da 16K ed LX 386 da 8K. Oltre a questo, la configurazione base del computer necessaria al corretto funzionamento del programma, prevede almeno un drive, una stampante (Microline o Epson indifferentemente) e l'interfaccia video LX 382.

Premesso questo, passiamo alla descrizione vera e propria del programma e del suo impiego.

Dopo aver acceso il computer e pigiato il tasto di RESET, sul monitor appare:

**BOOTSTRAPPING VERSION 1.0
INSERT NE DOS DISK AT DRIVER 0
THEN TYPE SPACE -**

Inserite ora il disco contabilità nel drive 0 e quindi pigiate il tasto di interlinea. Il computer così caricherà la lista del contenuto del disco e, a caricamento avvenuto, apparirà sul monitor la scritta:

**PER TORNARE IN DOS BATTI (A)
PER TORNARE IN BASIC BATTI (B)
PER IL DIRECTORY BATTI (C)
PER LANCIARE UN PROGRAMMA BATTI
IL NUMERO CORRISPONDENTE
1) CONTABILITÀ SEMPLIFICATA
COMANDO?**

Ora digitate 1, e a questo punto, viene caricato il programma della contabilità semplificata e durante questa fase sul monitor apparirà la scritta **STO LANCIANDO IL PROGRAMMA CONTABILITÀ SEMPLIFICATA**.

Dopo un breve istante, il sistema è pronto ad operare ed inizia il suo lavoro inizializzando le periferiche necessarie al programma. Sul monitor infatti appare:

**(M)ICROLINE
(E)PSON
SCEGLIERE LA STAMPANTE?***

Digitate la lettera M e il tasto RETURN se avete collegato una stampante MICROLINE oppure la lettera E ed il tasto RETURN se avete una EPSON.

Questa operazione è necessaria, in quanto le due stampanti hanno un set di istruzioni diverso, soprattutto per quanto riguarda il movimento del carrello e la generazione di caratteri speciali che vengono usati nella stampa dei vari tabulati generati dal programma stesso.

Questa operazione viene fatta solo all'inizio del programma ed è un passaggio che non verrà più ripetuto durante lo svolgersi delle operazioni a meno che non si esca dal programma principale.

Fatto questo, sul video appare la tabella dei menù principale al quale si dovrà sempre far riferimento all'inizio di tutte le operazioni. Sul monitor perciò avremo:

**CONTABILITÀ SEMPLIFICATA
MENÙ PRINCIPALE**

**1 - STAMPA PAG. NUM. REGISTRI
2 - CREAZIONE ARCHIVI DATI
3 - CONTABILITÀ ACQUISTI
4 - CONTABILITÀ VENDITE
0 - STOP**

DIGITA NUMERO DI SELEZIONE

Prima di proseguire nella descrizione, apriamo una breve parentesi per chiarire un particolare che ha creato non poche perplessità fra i lettori che hanno già lavorato con questo programma. A questo punto, infatti, è necessaria anche un'altra procedura preliminare che, analogamente al caso della stampante, non verrà poi più ripetuta durante il normale svolgersi delle operazioni; si deve cioè informare il sistema sul numero di drive con cui si vuole lavorare ed inoltre, si dovrà creare sul disco uno spazio in cui andranno scritti gli elenchi dei clienti e dei fornitori. All'inizio perciò, quando ancora non è stato inserito alcun fornitore né alcun cliente, dovrete obbligatoriamente selezionare il numero 2 del menù principale. Così facendo avrete sul video la scritta:

— **QUALE DRIVE VUOI USARE?** —

e dovreste rispondere 0 se avete collegato al sistema un solo drive oppure 1 se ne avete collegati due. Sul video ora viene visualizzata la tabella:

CONTABILITÀ SEMPLIFICATA

2 INIZIALIZZAZIONE ARCHIVI

1 - CREA. ARCH. CONTAB. ACQUISTI
2 - CREA. ARCH. CONTAB. VENDITE
0 - STOP

DIGITA NUMERO DI SELEZIONE

Il programma entra quindi nella routine di inizializzazione del disco e le operazioni da eseguire a questo punto sono differenti a seconda del numero di drive collegati al computer. Esaminiamo perciò, separatamente questi due possibili modi di operare.

Nel caso in cui si dispone di un solo drive, digitate semplicemente il numero 1 e successivamente, alla comparsa della scritta:

INSERISCI ARCHIVIO
ANAGRAFICO FORNITORI
NEL DRIVE: 0
DIGITA 'A'

scrivete A ed attendete la ricomparsa della tabellina precedente.

In questo modo, il programma apre tutti i file necessari all'archivio e alla contabilità fornitori e sulla traccia DIRECTORY del disco, saranno presenti, infatti, i file:

MASTAC/DAT
CONTAC/DAT
ANAFOR/DAT

Digitate ora il numero 2 e, alla comparsa della scritta:

INSERISCI ARCHIVIO
ANAGRAFICO CLIENTI
NEL DRIVE: 0
DIGITA 'A'

seguite la stessa procedura del caso precedente, cioè digitando A. Analogamente a prima, quindi, il programma apre ora tutti i file necessari all'archivio e alla contabilità clienti e sulla traccia DIRECTORY del disco vengono aggiunti ai precedenti, i file:

MASTVE/DAT
CONTVE/DAT
ANACLI/DAT

Terminata questa fase potrete tornare al menù principale digitando 0. Rimane perciò da analizzare la seconda possibilità di lavoro e cioè il caso in cui siano collegati due drive al computer. Per prima cosa, dovreste ora inserire nel drive 1 un disco VERGINE FORMATTATO e quindi digitare 1. Il programma, a questo punto, prosegue in modo del tutto analogo al caso precedente, con la sola differenza che, nel video, la scritta relativa al drive sarà 1 e non 0. Perciò sarà visualizzato:

INSERISCE ARCHIVIO
ANAGRAFICO FORNITORI
NEL DRIVE: 1
DIGITA 'A'

Senza ripetere ulteriormente tutto quanto, è ormai chiaro come procede il programma in questa fase, in quanto segue una procedura del tutto identica alla precedente.

Terminata perciò l'inizializzazione dei dischi, tornate al menù principale digitando 0 e iniziate lo svolgimento delle operazioni di contabilità vera e propria.

Qui di seguito forniamo la spiegazione dettagliata del significato di ciascuna possibilità di lavoro descritta nel menù principale e selezionabile digitando il numero corrispondente.

MENÙ PRINCIPALE — NUMERO SELEZIONATO: 1

Con questa scelta potrete intestare i fogli della contabilità per la vidimazione all'ufficio I.V.A.. Sul video viene visualizzato:

CONTABILITÀ SEMPLIFICATA
1 INTESTAZIONE GIORNALI IVA
DAI 'A' PER REGISTRO ACQUISTI
'V' PER REGISTRO VENDITE

La sola differenza nelle due possibili opzioni, sta nell'intestazione di inizio del blocco di fogli da vidimare. Nel caso A, l'intestazione del primo foglio stampato è REGISTRO ACQUISTI mentre nel caso V, è REGISTRO VENDITE. Per il resto, la sequenza delle operazioni svolte è del tutto identica in entrambi i casi. È sufficiente quindi, analizzare solo una delle due possibilità, essendo l'altra una banale ripetizione. Digitate quindi 'A' oppure 'V' e sul video appare la scritta:

DAI NOME E RAG. SOC. DITTA
?

Scrivete quindi il nome e la ragione sociale della vostra Ditta, (nell'esempio riportato in fig. 91 abbiamo scritto COMPUTER s.n.c.) e quindi pigiare il tasto RETURN. Sul video ora compare la scritta:

DAI NUMERO DI PAGINE ??

Scrivete ora il numero delle pagine da intestare per la vidimazione e quindi pigiate di nuovo il tasto RETURN. A questo punto la stampante inizia a scrivere in alto a sinistra in ogni pagina, il numero progressivo corrispondente ad ogni foglio, così come deve essere per la vidimazione dei registri all'ufficio IVA. Terminata questa operazione il programma torna nuovamente al menù principale.

MENÙ PRINCIPALE — NUMERO SELEZIONATO: 2

Lo scopo di questa parte di programma è stato già ampiamente trattato nell'introduzione di questo articolo, ma possiamo ora aggiungere alcune note importanti riguardanti il numero massimo di clienti e fornitori che il programma può gestire a seconda del numero di drive che sono stati selezionati precedentemente con questa routine.

Nel caso vi sia collegato un solo drive, infatti, una volta che il disco è stato completamente riempito, non è più possibile aggiungere altri nominativi a quelli già inseriti. Fra l'altro, il numero dei nominativi inseribili in questo caso, è abbastanza piccolo (50 circa in totale fra clienti, fornitori e relative fatture) perché, in questo stesso disco, è memorizzato anche tutto il programma. Quindi, questo tipo di configurazione di lavoro, con un solo drive collegato, è adatto soprattutto a scopi didattici o ad imprese con un limitato numero di clienti e fornitori. Se invece il sistema è stato inizializzato per l'uso di due drive, non ci sono limitazioni al numero di clienti e fornitori inseribili; l'unica operazione da fare, una volta che lo spazio disponibile nel disco è stato completato, è l'inizializzazione di un nuovo disco eseguendo le operazioni già descritte nell'esecuzione delle fasi preliminari di questo programma. Continuiamo perciò la descrizione del menù principale.

MENÙ PRINCIPALE — NUMERO SELEZIONATO: 3

Questa parte di programma contiene la gestione vera e propria della contabilità acquisti con tutte le operazioni di calcolo e stampa ad essa inerenti.

All'inizio di questa routine, cioè subito dopo aver digitato 3 nel menù principale, appare sul video la scritta:

**INSERISCI ARCHIVIO
ANAGRAFICO FORNITORI
INSERISCI ARCHIVIO
CONTABILE ACQUISTI
BATTI 'RETURN'**

Questo messaggio serve a ricordare all'utente di inserire nel drive 1 il disco inizializzato per l'inserzione degli archivi clienti e fornitori. Infatti, quando il programma è stato già usato almeno una volta, avrete ovviamente già disponibile un disco inizializzato e contenente una certa quantità di nominativi e fatture relative a clienti e fornitori.

Quindi, non avendo eseguito la procedura 2 del menù principale, non è stata ancora richiesta l'inserzione del disco nel drive e, il programma ve lo rammenta ora, essendo indispensabile a questo punto per il proseguimento del programma stesso. Se non fosse già presente, perciò, inserite fisicamente nel drive il disco e successivamente pigiate RETURN per proseguire.

Quanto detto, vale ovviamente nel caso in cui siano stati selezionati inizialmente due drive. Contrariamente infatti, operando con un solo disco, basta semplicemente pigiare RETURN per proseguire.

Il programma ora si svolge allo stesso modo per entrambi i casi, perciò, dopo aver pigiato il tasto RETURN, sul video appare il menù completo della contabilità acquisti così composto:

TAB. 2 CONTABILITÀ ACQUISTI GUIDA OPERATIVA

- 1 - ACQUISIZIONE FATT. ACQUISTI**
 - 2 - CORREZIONE FATTURE ACQUISTI**
 - 3 - STAMPA REGISTRO ACQUISTI**
 - 4 - STAMPA CONTAB. FORNITORI**
 - 5 - STAMPA LISTA FORNITORI**
 - 6 - AGG. ANAGRAFICO FORNITORI**
 - 7 - STAMPA TOTALI ALIQUOTE IVA**
 - 8 - INTERROGAZ. ARCH. FATTURE**
 - 9 - ALLEG. 13/A DICHIARAZIONE**
 - 10 - LISTA CODICE RAGGRUPPAMENTO**
 - 11 - STAMP. LISTA FORN. ORD. ALFAB.**
 - 0 - STOP**
- DIGITA NUMERO DI SELEZIONE?**

Digitando uno dei numeri del menù e, successivamente, pigiando RETURN, si entra nella fase operativa corrispondente indicata dal menù stesso.

A questo punto, per chiarire al meglio tutte le possibilità di lavoro presenti in questa parte di programma, faremo un esempio pratico iniziando con l'inserimento nell'archivio dei dati anagrafici dei clienti. Per fare questo, occorre digitare il numero 6 ed il tasto RETURN e sul video apparirà la scritta:

**AGGIORNAMENTO ANAGRAFICO FORNITORI
DAI 'I' PER INSER. NUOVO CODICE
'A' PER CORREZIONE
'X' PER USCIRE**

Digitate perciò la lettera I e RETURN per inserire il nominativo del cliente da memorizzare. Il sistema si predispone quindi per questa operazione ed essendo ora il primo nominativo da inserire, risponderà:

INSERZIONE FORNITORE COD. N. 0
DENOMINAZIONE
?

Proseguendo nell'esempio, scrivete perciò il nome della Ditta da inserire accertandovi che tale nominativo in totale non sia più lungo di 25 caratteri. Contrariamente, infatti, sarebbe necessario usare delle abbreviazioni per riportare entro i limiti concessi, la lunghezza del nominativo stesso. Scrivendo quindi, ad esempio, ELETTRONICA S.P.A. e non ELETTRONICA — SOCIETÀ PER AZIONI —, è possibile inserire tranquillamente questo nominativo nel programma e proseguire pigiando il tasto RETURN. Viene chiesto ora il codice fiscale, e sul video apparirà la scritta:

COD. FISCALE
?

Il numero massimo dei caratteri che vi è consentito scrivere è limitato a 16 ed è un numero sempre più che sufficiente per qualunque tipo di codice fiscale o partita IVA. La lunghezza massima di questo dato è infatti di 11 numeri per la partita IVA e di 15 per il codice fiscale. Scrivete perciò 0234579 e quindi pigiate RETURN per inserire anche questa informazione. Per ultimo, a questo punto, viene richiesto l'indirizzo del fornitore e quindi analogamente appare sul video:

INDIRIZZO
?

In questo caso, il numero massimo di caratteri inseribili è il più alto possibile e cioè 35 in totale, per permettere di scrivere indirizzi più complessi possibile senza dover ricorrere a troppe abbreviazioni. Scrivete quindi VIA VITTORIO, 43, BOLOGNA e pigiate ancora il tasto RETURN. Avete così terminato l'inserzione del nominativo del primo fornitore e perciò il programma inserisce ora nel disco i dati che voi avete scritto e successivamente torna all'inizio della routine per l'aggiornamento anagrafico dei fornitori, riportando sul video la tabellina di partenza.

Allo stesso modo, quindi, potrete inserire tutti i nominativi degli altri fornitori digitando ancora la lettera I e proseguendo in modo identico a quanto appena fatto. L'aggiornamento fornitori prevede anche un'altra possibilità di lavoro infatti, digitando la lettera 'A', si ha la possibilità di correggere eventuali errori commessi nell'inserzione dei nominativi. Così facendo, sul video appare la scritta:

COD. FORNITORE?

Scrivete perciò il codice numerico corrispondente al nominativo del fornitore da correggere e quindi, per continuare l'esempio, digitate il numero 1. Pigate quindi il tasto di RETURN ed il programma risponderà:

CORREZ. FORNITORE COD. N. 1
ELETTRONICA S.P.A.
0234579
VIA VITTORIO, 43 BOLOGNA
DENOMINAZIONE

fig. 87

LISTA CODICI FORNITORI

COD.	DENOMINAZIONE	COD. FISCALE	INDIRIZZO
1	ELETTRONICA S.P.A.	0234579	VIA VITTORIO 45, BOLOGNA
2	PROGETTI S.N.C.	PRG56M78	VIA GRANDE, 5/8 PARMA
3	STUDIO S.R.L.	STD55HB90	P. ZZA LARGA, 3 FORLÌ

Supponiamo di dover cambiare solo il numero della via del fornitore, correggendo 43 con 45. Alla domanda DENOMINAZIONE ed alla successiva domanda COD FISCALE, ricopiate dunque, esattamente, quanto scritto in precedenza mentre, alla voce INDIRIZZO, scrivete VIA VITTORIO, 45 BOLOGNA. Il programma cambia ora automaticamente nell'archivio i dati relativi a questo fornitore e, terminata questa operazione, torna nuovamente all'inizio della routine di aggiornamento anagrafico fornitori. Avendo terminato tutte le operazioni relative a questa parte di programma, potete ora uscirne digitando 'X'. Così facendo avrete nuovamente sul video la guida operativa iniziale (TAB. 2) e potrete ora inserire anche le fatture di acquisto. Prima di effettuare questa operazione però, sarà molto comodo avere una stampa di tutti i vari fornitori inseriti nella memoria fino ad ora per poter ricordare il relativo codice numerico attribuito a ciascuno di essi dal programma. Digitate perciò il numero 5 e RETURN. La stampante in questo modo fornirà il tabulato riportato in fig. 87 e quindi tornerà nuovamente alla guida operativa iniziale (TAB. 2). A questo punto, quindi, proseguite inserendo nella memoria del sistema le fatture di acquisto da registrare. Per far questo digitate il numero 1 e RETURN e sul video apparirà la scritta:

INSERIZIONE FATTURA ACQUISTI N. 1
DATA REGISTRAZIONE (GG/MM)
?

Scrivete quindi per esempio 3/1 e pigiate RETURN. Analogamente viene ora richiesta la:

DATA FATTURAZIONE (GG/MM/AA)
?

A questa voce rispondete ad esempio 01/01/84. Digitando di nuovo RETURN, il programma richiede il codice numerico del fornitore a cui si riferisce la fattura. Facendo perciò riferimento alla fig. 87 dove è riportata appunto la stampa dei codici numerici dei fornitori inseriti, si legge tale dato e lo si scrive. Per proseguire nel nostro esempio, perciò, supponiamo che la fattura da registrare si riferisca alla ditta ELETTRONICA S.P.A.. Digitate quindi il numero 1 e RETURN ed il programma risponderà scrivendo sul video:

DENOMINAZIONE
ELETTRONICA S.P.A.
0234579
VIA VITTORIO, 45 BOLOGNA
IMPONIBILE?

Iniziano, ora, da parte del programma, una serie di richieste per l'introduzione degli importi veri e propri della fattura da registrare. Inserendo le cifre relative ad ogni richiesta, seguite da RETURN si scrivono così sequenzialmente nella memoria: L'IMPONIBILE, l'eventuale importo NON IMPONIBILE ed il relativo ARTICOLO DI ESENZIONE per tale importo, l'ALiquota IVA da applicare al costo ed il relativo importo derivante dalla voce IMPOSTA IVA.

A questo punto è prevista, appositamente per la contabilità semplificata, l'introduzione di un numero che tiene conto del tipo di spesa descritta dalla fattura. La richiesta viene formulata con la scritta:

CONTRO RAGGRUPPAMENTO
?

Il significato del numero da scrivere è riportato in fig. 88. Fatto ciò, quindi, abbiamo già inserito tutti i dati necessari all'operazione di registrazione della fattura e perciò ci viene richiesto se, in questa stessa fattura, esistono o no altri articoli da aggiungere separatamente come ad esempio avviene nel caso di acquisto di materiali con aliquote IVA differenti fra loro. Il programma dunque richiede:

CONTINUA FATTURAZIONE (S/N)
?

In caso di risposta affermativa (Digitando 'S') si torna automaticamente all'inizio di questa fase per aggiungere i rimanenti dati alla fattura mentre al contrario, se avete effettivamente terminato la scrittura di questi dati, digitate 'N' per concludere la fattura stessa. Così facendo, sul video appare la scritta:

TOTALE FATTURA?

fig. 88

SITUAZIONE COSTI AL 10/1/84

(1)	INVENTARIO INIZIO ANNO	0	
	T O T A L E		0
(2)	PAGHE	0	
(3)	CONTRIBUTI	0	
(4)	INDENNITA' LICENZIAMENTO	0	
	T O T A L E		0
(5)	MATERIE PRIME	100,000	
	T O T A L E		100,000
(6)	AMMORTAM.A DETRAZ. TOTALE	0	
	T O T A L E		0
(7)	AMMORTAM.A DETRAZ. PLURIEN.	0	
	T O T A L E		0
(8)	COMPENSI A TERZI	510,000	
	T O T A L E		510,000
(9)	CANONI LOCAZIONE	0	
	T O T A L E		0
(10)	INTERESSI PASSIVI (ES.ART.15)	0	
	T O T A L E		0
(11)	ASSICURAZIONI ART.10	0	
	T O T A L E		0
(12)	ENEL	0	
	T O T A L E		0
(13)	CARBURANTI A DETRAZIONE TOTALE	0	
(14)	CARBURANTI NON DETRAIBILI	0	
	T O T A L E		0
(15)	SIP	0	
(16)	TRASPORTI	200,000	
(17)	PRANZI	52,000	
(18)	ALBERGHI	0	
(19)	VARIE	205,000	
	T O T A L E		457,000

T O T A L E G E N E R A L E 1,067,000

Inserita quindi anche quest'ultima cifra, pigiate il tasto RETURN e sul video viene visualizzato:

ALTRA FATTURA (S/N)?

Potrete perciò inserire i dati di una nuova fattura digitando 'S' oppure, una volta inserite tutte le fatture da registrare, ripetendo per ciascuna di esse le semplici operazioni descritte finora tornate alla guida operativa di inizio (TAB. 2) digitando 'N'.

Abbiamo così esaurito la spiegazione delle operazioni più importanti contenute in questa parte di programma e, come avete visto, le operazioni descritte fino ad ora sono molto semplici e si svolgono in un modo del tutto analogo a quello usato comunemente in un qualsiasi ufficio amministrativo. Nella guida operativa comunque, sono presenti altri numeri selezionabili oltre a quelli esposti fino ad ora ma il lavoro svolto dalle routine richiamabili in questo modo, è un banale ciclo di stampa e non presenta quindi alcuna difficoltà operativa da parte dell'utente. Diamo perciò un rapido resoconto di queste operazioni di stampa per completezza di esposizione.

Ripartiamo perciò dalla guida operativa riportata nella TAB. 2 e riferendoci a questa procediamo punto per punto:

NUMERO DIGITATO 3 E RETURN

Il programma richiede da che numero di fattura deve iniziare la stampa richiesta. Infatti sul monitor appare la scritta:

DAI IL PRIMO NUMERO DA STAMPARE. ?

Se volete stampare tutte le fatture registrate, dovete digitare il numero 1 e RETURN altrimenti, se volete ad esempio la stampa dalla decima fattura in poi, digitate 10 e RETURN. Fatto questo il computer inizia la scrittura richiesta ed il formato di stampa è riportato in fig. 89. Terminato questo lavoro, il programma torna automaticamente alla TAB. 2 di partenza.

fig. 89

DATA REG.	NUM. PROGR.	DATA FATTURA2.	DENOMINAZIONE	TOTALE IMPONIBILE	NON IMPON/ESEN	COD. ES.	ALIBQUOTA IVA	IMPOSTA IVA	TOTALE FATTURA
1/1	1	1/1/84	ELETTRONICA S.	100,000	0		18	18,000	
				200,000	5,000	B/2	15	30,000	353,000
3/1	2	3/1/84	PROGETTI S.N.C	500,000	10,000	XYZ	30	150,000	660,000
7/1	3	7/1/84	PROGETTI S.N.C	200,000	0		18	36,000	236,000
10/1	4	10/1/84	STUDIO S.R.L.	50,000	2,000	B/2	15	7,500	59,500

NUMERO DIGITATO 4 E RETURN

Procedura di stampa della contabilità generale dei fornitori. Il formato di stampa è riportato in fig. 90. Terminato il lavoro di scrittura, il programma torna automaticamente alla TAB. 2.

NUMERO DIGITATO 5 E RETURN

Procedura per la stampa della lista anagrafica e dei codici numerici dei fornitori. Il formato di stampa è riportato in fig. 87. Terminato il lavoro di scrittura, il programma torna automaticamente alla TAB. 2.

NUMERO DIGITATO 7 E RETURN

Procedura per la stampa del riassuntivo dell'IVA raggruppato per aliquote. Il formato di stampa è riportato in fig. 92. Terminato il lavoro di scrittura, il programma torna automaticamente alla TAB. 2.

NUMERO DIGITATO 9 E RETURN

Procedura per la stampa dell'elenco dei fornitori fatta con le modalità necessarie alla dichiarazione IVA per la compilazione dell'allegato 13/A alla dichiarazione stessa. Il formato di stampa è riportato in fig. 93. Terminato il lavoro di scrittura, il programma torna automaticamente alla TAB. 2.

fig. 90

C O N T A B I L I T A' F O R N I T O R I

DENOMINAZIONE: ELETTRONICA S.P.A.
COD. FISCALE: 0234579
INDIRIZZO: VIA VITTORIO 45, BOLOGNA

CODICE IND.: 1

DATA	IMPONIBILE	ALiquOTA	IMPOSTA
1/1/84	100,000	18	18,000
1/1/84	200,000	15	30,000

RIEPILOGO :
IMPONIBILE 300,000 IMPOSTA 48,000

DENOMINAZIONE: PROGETTI S.N.C.
COD. FISCALE: PRG56M78
INDIRIZZO: VIA GRANDE, 5/B PARMA

CODICE IND.: 2

DATA	IMPONIBILE	ALiquOTA	IMPOSTA
3/1/84	500,000	30	150,000
7/1/84	200,000	18	36,000

RIEPILOGO :
IMPONIBILE 1,000,000 IMPOSTA 234,000

DENOMINAZIONE: STUDIO S.R.L.
COD. FISCALE: STD55H890
INDIRIZZO: P.ZZA LARGA, 3 FORLI'

CODICE IND.: 3

DATA	IMPONIBILE	ALiquOTA	IMPOSTA
10/1/84	50,000	15	7,500

RIEPILOGO :
IMPONIBILE 1,050,000 IMPOSTA 241,500

fig. 91

R E G I S T R O V E N D I T E
COMPUTER S.N.C.

R E G I S T R O A C Q U I S T I
COMPUTER S.N.C.

NUMERO DIGITATO 10 E RETURN

Procedura per la stampa del riassuntivo della situazione dei costi fatti per raggruppamenti di spese. Il formato di stampa è riportato in fig. 88. Terminato il lavoro di scrittura, il programma torna automaticamente alla TAB. 2.

NUMERO DIGITATO 11 E RETURN

Procedura per la stampa dei fornitori riportati in ordine alfabetico. Il programma torna automaticamente alla tab. 2 una volta terminate le funzioni di stampa il cui formato è riportato in fig. 94.

fig. 92

TOTALI PER ALIQUOTA IVA

ALIQUOTA	IMPONIBILE	IMPOSTA
15	250,000	37,500
18	300,000	54,000
30	500,000	150,000
TOTALE IMPONIBILE	1,050,000	
TOTALE IMPOSTA	241,500	

fig. 93

1	ELETTRONICA S.P.A.	0234579	VIA VITTORIO 45, BOLOGNA		
IMPONIBILE	300,000	IMPOSTA	48,000	TOTALE FATTURE	353,000
NUM.FATTURE	2	NON IMPONIBILE ART. 8	5,000	NON SOGGETTO / ESENTE	0
2	PROGETTI S.N.C.	PR656M78	VIA GRANDE, 5/B PARMA		
IMPONIBILE	700,000	IMPOSTA	186,000	TOTALE FATTURE	896,000
NUM.FATTURE	2	NON IMPONIBILE ART. 8	0	NON SOGGETTO / ESENTE	10,000
3	STUDIO S.R.L.	STD55H890	P.ZZA LARGA, 3 FORLI'		
IMPONIBILE	50,000	IMPOSTA	7,500	TOTALE FATTURE	59,500
NUM.FATTURE	1	NON IMPONIBILE ART. 8	2,000	NON SOGGETTO / ESENTE	0

fig. 94

*****	A	A	A	A	A	*****
*****	B	B	B	B	B	*****
*****	C	C	C	C	C	*****
*****	D	D	D	D	D	*****
*****	E	E	E	E	E	*****
1	ELETTRONICA S.P.A.	0234579	VIA VITTORIO 45, BOLOGNA			
*****	F	F	F	F	F	*****
*****	G	G	G	G	G	*****
*****	H	H	H	H	H	*****
*****	I	I	I	I	I	*****
*****	J	J	J	J	J	*****
*****	K	K	K	K	K	*****
*****	L	L	L	L	L	*****
*****	M	M	M	M	M	*****
*****	N	N	N	N	N	*****
*****	O	O	O	O	O	*****
*****	P	P	P	P	P	*****
2	PROGETTI S.N.C.	PRG56M78	VIA GRANDE, 5/B PARMA			

*****	Q	Q	Q	Q	Q	*****
*****	R	R	R	R	R	*****
*****	S	S	S	S	S	*****
3	STUDIO S.R.L.	STD55H890			P.ZZA LARGA,3 FORLI'	
*****	T	T	T	T	T	*****
*****	U	U	U	U	U	*****
*****	V	V	V	V	V	*****
*****	W	W	W	W	W	*****
*****	X	X	X	X	X	*****
*****	Y	Y	Y	Y	Y	*****
*****	Z	Z	Z	Z	Z	*****

Abbiamo così concluso la spiegazione di tutta la parte di programma inerente alla contabilità fornitori. Possiamo perciò tornare al menù principale (TAB. 1) digitando 0 e da questo, digitando 4 entrare nella parte di programma relativa alla contabilità vendite.

Così facendo sul monitor compare la tabella:

CONTABILITÀ VENDITE

GUIDA OPERATIVA

- 1 - ACQUISIZ. FATTURE VENDITE
- 2 - CORREZIONE FATTURE VENDITE
- 3 - STAMPA REGISTRO VENDITE
- 4 - STAMPA CONTABILITÀ CLIENTI
- 5 - STAMPA LISTA CLIENTI
- 6 - AGGIORNAMENTO ANAGRAFICO CLIENTI
- 7 - STAMPA TOT. ALIQUOTE IVA
- 8 - INTERROG. ARCHIVIO FATTURE
- 9 - STAMP. LISTA CLIE. ORD. ALFAB.
- 0 - STOP

DIGITA NUMERO DI SELEZIONE?

Come potete vedere, la guida operativa per la contabilità clienti è perfettamente uguale a quella per la contabilità fornitori ed inoltre, operativamente, le istruzioni necessarie al funzionamento di questa parte di programma sono la copia esatta delle istruzioni date precedentemente. Le differenze fra queste due parti di programma sono perciò nelle sole intestazioni delle operazioni nelle quali è stata sostituita la parola «VENDITE» alla parola «ACQUISTI» e alla parola «FORNITORE» la parola «CLIENTE». I formati di stampa perciò sono stati modificati di conseguenza e sono riportati di seguito rispettivamente:

GUIDA OPERATIVA CONTABILITÀ VENDITE

- NUMERO DIGITATO 3 = Formato di stampa di fig. 95
- NUMERO DIGITATO 4 = Formato di stampa di fig. 96
- NUMERO DIGITATO 5 = Formato di stampa di fig. 97
- NUMERO DIGITATO 7 = Formato di stampa di fig. 98
- NUMERO DIGITATO 9 = Formato di stampa di fig. 99

Ora siete in grado di gestire comodamente, ed in modo assolutamente sicuro, la contabilità semplificata di una qualsiasi attività commerciale e di ottenere inoltre, velocemente, tutti i quadri riassuntivi del vostro bilancio anche quotidianamente. Fra l'altro, lavorando con due drive, avete la possibilità di tenere un archivio clienti e fornitori di capacità praticamente illimitata ma di dimensioni fisiche decisamente ridotte ed inoltre, le consultazioni di questi stessi archivi sono estremamente rapide e semplificate. Imparerete comunque a conoscere i pregi di questo modo di lavorare soprattutto usando il programma quotidianamente, toccando così con la mano la potenza operativa di tutto il vostro sistema.

fig. 95

DATA REG.	NUM. PROGR.	DATA FATTURAZ.	DENOMINAZIONE	TOTALE IMPONIBILE	NON IMPON/ESEN	COD. ES.	ALIQUOTA IVA	IMPOSTA IVA	TOTALE FATTURA
1/1	1	1/1/84	FABBRICA S.A.S	300,000	0		18	54,000	354,000
7/1	2	7/1/84	GIOVANNI BIANC	100,000	2,500	YYY	15	15,000	
				50,000	0		18	9,000	176,500
10/2	3	10/2/84	COSTRUZIONI S.	300,000	20,000	8/2	3	9,000	329,000

fig. 96

CONTABILITA' CLIENTI

DENOMINAZIONE: COSTRUZIONI S.N.C.

COD. FISCALE: 0057890453

CODICE IND.: 1

INDIRIZZO: VIA DEI LAVORI, 34 MILANO

DATA	IMPONIBILE	ALiquOTA	IMPOSTA
10/2/84	300,000	3	9,000

RIEPILOGO :
IMPONIBILE 300,000 IMPOSTA 9,000

DENOMINAZIONE: FABBRICA S.A.S.

COD. FISCALE: FRCJ90

CODICE IND.: 2

INDIRIZZO: L.GO SEMPIONE, 2 GENOVA

DATA	IMPONIBILE	ALiquOTA	IMPOSTA
1/1/84	300,000	18	54,000

RIEPILOGO :
IMPONIBILE 600,000 IMPOSTA 63,000

DENOMINAZIONE: GIOVANNI BIANCHI

COD. FISCALE: GVN BNC 21E56 44

CODICE IND.: 3

INDIRIZZO: P.ZZA DEI FRATI, 3 ROMA

DATA	IMPONIBILE	ALiquOTA	IMPOSTA
7/1/84	100,000	15	15,000
7/1/84	50,000	18	9,000

RIEPILOGO :
IMPONIBILE 750,000 IMPOSTA 87,000

fig. 97

LISTA CODICI CLIENTI

COD.	DENOMINAZIONE	COD. FISCALE	INDIRIZZO
1	COSTRUZIONI S.N.C.	0057890453	VIA DEI LAVORI, 34 MILANO
2	FABBRICA S.A.S.	FRCJ90	L.GO SEMPIONE, 2 GENOVA
3	GIOVANNI BIANCHI	GVN BNC 21E56 44	P.ZZA DEI FRATI, 3 ROMA

fig. 98

TOTALI PER ALIQUOTA IVA		
ALIQUOTA	IMPONIBILE	IMPOSTA
3	300,000	9,000
15	100,000	15,000
18	350,000	63,000
TOTALE IMPONIBILE	750,000	
TOTALE IMPOSTA	87,000	

fig. 99

*****	A	A	A	A	A	*****
*****	B	B	B	B	B	*****
*****	C	C	C	C	C	*****
1	COSTRUZIONI S.N.C.	0057890453			VIA DEI LAVORI ,34 MILANO	
*****	D	D	D	D	D	*****
*****	E	E	E	E	E	*****
*****	F	F	F	F	F	*****
2	FABBRICA S.A.S.	FRCJ90			L.GO SEMPIONE ,2 GENOVA	
*****	G	G	G	G	G	*****
3	GIOVANNI BIANCHI	GVN BNC 21E56 44			P.ZZA DEI FRATI ,3 ROMA	
*****	H	H	H	H	H	*****
*****	I	I	I	I	I	*****
*****	J	J	J	J	J	*****

*****	K	K	K	K	K	*****
*****	L	L	L	L	L	*****
*****	M	M	M	M	M	*****
*****	N	N	N	N	N	*****
*****	O	O	O	O	O	*****
*****	P	P	P	P	P	*****
*****	Q	Q	Q	Q	Q	*****
*****	R	R	R	R	R	*****
*****	S	S	S	S	S	*****
*****	T	T	T	T	T	*****
*****	U	U	U	U	U	*****
*****	V	V	V	V	V	*****
*****	W	W	W	W	W	*****
*****	X	X	X	X	X	*****
*****	Y	Y	Y	Y	Y	*****
*****	Z	Z	Z	Z	Z	*****

LISTATO ESATTO DEL PROGRAMMA GESTIONE AZIENDALE

```

1000 '=====
1010 '          GESTIONE AZIENDALE 1                      1982 NE-R.C.
1020 '=====
1030 CLEAR 1000 : Y$=CHR$(34)
1040 ONERROR GOTO 10220
1050 CLS
1060 REVDN: PRINT@166," gestione aziendale " : REVOFF
1070 PRINT : PRINT : PRINT
1080 GOSUB 10040
1090 OPEN"r",1,"dati"
1100 FIELD1, 8 AS S$, 2 AS N$, 2 AS M$, 243 AS A$
1110 GET1,1
1120 SC%=CVD(S$) : NC%=CVI(N$) : MC%=CVI(M$)
1130 GET1,2
1140 SF%=CVD(S$) : NF%=CVI(N$) : MF%=CVI(M$)
1150 CLOSE
1160 '+++++ MENU' PRINCIPALE +++++
1170 RM=0 : CR=0
1180 CLS : REVDN: PRINT@11," MENU' "
1190 PRINT : PRINT : PRINT"1"; : REVOFF: PRINT" > registrazioni clienti"
1200 PRINT : REVDN: PRINT"2"; : REVOFF: PRINT" > registrazioni fornitori"
1210 PRINT : REVDN: PRINT"3"; : REVOFF: PRINT" > letture clienti"
1220 PRINT : REVDN: PRINT"4"; : REVOFF: PRINT" > letture fornitori"
1230 PRINT : REVDN: PRINT"5"; : REVOFF: PRINT" > situazione pagamenti"
1240 M=5
1250 GOSUB 10010
1260 IF X=5 GOTO 3150
1270 IF X=1 OR X=3 THEN C=1 : NO$="cliente" : NN$="movcl"
1280 IF X=2 OR X=4 THEN C=0 : NO$="fornitore" : NN$="movfo"
1290 ON X GOTO 1310,1310,2620,2620
1300 '+++++ MENU' REGISTRAZIONI +++++
1310 CLS
1320 REVDN: PRINT"1"; : REVOFF: PRINT" > per registrare movimenti"
1330 PRINT : REVDN: PRINT"2"; : REVOFF: PRINT" > per aggiungere un ";NO$
1340 PRINT : REVDN: PRINT"3"; : REVOFF: PRINT" > per correggere un ";NO$
1350 PRINT : REVDN: PRINT"4"; : REVOFF: PRINT" > per tornare al menu'"
1360 M=4
1370 GOSUB 10010
1380 ON X GOTO 2220,1400,1980,1170
1390 '----- AGGIUNTA NOMINATIVI -----
1400 TX$=" nome " : Q=30
1410 GOSUB 10110
1420 LINEINPUT N$
1430 IF LEN(N$)>Q GOTO 1400
1440 IF LEN(N$)<Q THEN N$=N$+" " : GOTO 1440
1450 TX$=" via "
1460 GOSUB 10110
1470 LINEINPUT V$
1480 IF LEN(V$)>Q GOTO 1450
1490 IF LEN(V$)<Q THEN V$=V$+" " : GOTO 1490
1500 TX$=" citta' "
1510 GOSUB 10110
1520 LINEINPUT C$
1530 IF LEN(C$)>Q GOTO 1500
1540 IF LEN(C$)<Q THEN C$=C$+" " : GOTO 1540
1550 TX$=" telefono " : Q=20
1560 GOSUB 10110
1570 LINEINPUT T$
1580 IF LEN(T$)>Q GOTO 1550
1590 IF LEN(T$)<Q THEN T$=T$+" " : GOTO 1590
1600 TX$=" annotazioni " : Q=129
1610 GOSUB 10110
1620 LINEINPUT R$
1630 IF LEN(R$)>Q GOTO 1600
1640 I$=0 : P$=0
1650 TX$=" tot.fattura " : Q=16
1660 GOSUB 10110
1670 INPUT I$
1680 TX$=" acconto
1690 GOSUB 10110
1700 INPUT P$

```

```

1710 Z$=N$+V$+C$+T$+R$
1720 CLS
1730 GOSUB 10040
1740 IF C=0 THEN NF%=NF%+1 : PU=NF% ELSE NC%=NC%+1 : PU=NC%
1750 IF CR=1 OR RM=1 THEN PU=N
1760 OPEN"r",1,NO$
1770 FIELD1, 8 AS I$, 8 AS P$, 239 AS W$
1780 LSET I$=MKD$(I#) : LSET P$=MKD$(P#) : LSET W$=Z$
1790 PUT1, PU
1800 OPEN"r",2,"dati"
1810 FIELD2, 8 AS S$, 2 AS N$, 2 AS M$, 243 AS A$
1820 IF C=0 GOTO 1890
1830 IF RM=0 THEN SC%=SC%+I$-P$ ELSE SC%=SC%+IM%-PM#
1840 LSET S$=MKD$(SC%)
1850 LSET N$=MKI$(NC%)
1860 LSET M$=MKI$(MC%)
1870 LSET A$=""
1880 PUT2,1 : GOTO 1950
1890 IF RM=0 THEN SF%=SF%+I$-P$ ELSE SF%=SF%+IM%-PM#
1900 LSET S$=MKD$(SF%)
1910 LSET N$=MKI$(NF%)
1920 LSET M$=MKI$(MF%)
1930 LSET A$=""
1940 PUT2,2
1950 CLOSE
1960 IF RM=1 GOTO 2220 ELSE GOTO 1170
1970 '----- CORREZIONE NOMINATIVI -----
1980 CR=1
1990 CLS
2000 PRINT"quale numero di ";NO$
2010 PRINT"vuoi correggere ?" : PRINT : PRINT "(0 per tornare al menu)" : PRINT
2020 INPUT N
2030 IF N=0 GOTO 1170
2040 IF C=0 AND N>NF% GOTO 1980
2050 IF C=1 AND N>NC% GOTO 1980
2060 CLS
2070 GOSUB 10040
2080 OPEN"r",1,NO$
2090 FIELD1, 8 AS I$, 8 AS P$, 239 AS Z$
2100 GET1,N
2110 CLOSE
2120 CLS
2130 PRINT NO$ : PRINT
2140 REVON: PRINT LEFT$(Z$,30) : REVOFF
2150 GOSUB 10050
2160 IF X$="0" GOTO 1170
2170 CLS
2180 IF C=0 THEN NF%=NF%-1
2190 IF C=1 THEN NC%=NC%-1
2200 GOTO 1400
2210 '----- REGISTRAZIONE MOVIMENTI -----
2220 RM=1 : IM#=0 : PM#=0
2230 CLS
2240 PRINT"numero del ";NO$ : PRINT"(0 per finire)" : PRINT
2250 INPUT N
2260 IF N=0 GOTO 1170
2270 IF C=1 AND N>NC% GOTO 2220
2280 IF C=0 AND N>NF% GOTO 2220
2290 CLS
2300 GOSUB 10040
2310 OPEN"r",1,NO$
2320 FIELD1, 8 AS I$, 8 AS P$, 239 AS Z$
2330 GET1,N
2340 I#=CVD(I$) : P#=CVD(P$)
2350 CLS
2360 PRINT NO$;" n. ";N : PRINT
2370 REVON: PRINT LEFT$(Z$,30) : REVOFF
2380 GOSUB 10050
2390 IF X$="0" THEN CLOSE : GOTO 1170
2400 PRINT@128,CHR$(31);
2410 LINEINPUT "data (gg.mm.aa): ";D$

```



```

2420 PRINT"merce : (max. 95 caratteri)" : LINEINPUTM$
2430 IF LEN(M$)>95 PRINT@160,CHR$(31); : GOTO 2420
2440 PRINT@288,""; : INPUT"importo :";IM$
2450 INPUT"acconto :";PM$
2460 PRINT"note : (max. 63 caratteri)" : LINEINPUT A$
2470 IF LEN(A$)>63 PRINT@352,CHR$(31); : GOTO 2460
2480 REVDN: PRINT@480," confermi ? (s/n) "; : REVOFF
2490 X$=INKEY$
2500 IF X$="s" OR X$="n" GOTO 2520
2510 GOTO 2490
2520 IF X$="n" THEN CLOSE : GOTO 2220
2530 CLS
2540 GOSUB 10040
2550 OPEN"e",2,NN$
2560 PRINT#2, N; Y$; D$; Y$; Y$; M$; Y$; IM$; PM$; Y$; A$; Y$;
2570 CLOSE
2580 I#=I#+IM$ : P#=P#+PM$
2590 IF C=1 THEN MCZ=MCZ+1 ELSE MFZ=MFZ+1
2600 GOTO 1750
2610 '+++++++ MENU' LETTURE ++++++
2620 CLS
2630 REVDN: PRINT"1"; : REVOFF: PRINT" > per leggere movimenti"
2640 PRINT : REVDN: PRINT"2"; : REVOFF: PRINT" > per leggere nomi"
2650 PRINT : REVDN: PRINT"3"; : REVOFF: PRINT" > per tornare al menu'"
2660 M=3
2670 GOSUB 10010
2680 ON X GOTO 2930,2700,1170
2690 '----- LETTURA NOMINATIVI -----
2700 CLS
2710 IF C=1 AND MCZ=0 GOTO 1170
2720 IF C=0 AND MFZ=0 GOTO 1170
2730 GOSUB 10040
2740 IF C=1 THEN H=MCZ ELSE H=MFZ
2750 OPEN"r",1,NO$
2760 FIELD1, 8 AS I$, 8 AS P$, 30 AS N$, 30 AS V$, 30 AS C$, 20 AS T$, 129 AS A$
2770 FOR I=1 TO H
2780 GET1,I
2790 CLS
2800 REVDN: PRINT NO$;" numero";I : REVOFF
2810 I#=CVD(I$) : P#=CVD(P$)
2820 PRINT : PRINTN$ : PRINTV$ : PRINTC$
2830 PRINT : PRINT"telefono : ";T$
2840 REVDN: PRINT" note " : REVOFF: PRINTA$
2850 PRINT"tot.fatt.:";I#
2860 PRINT"acconto :";P#
2870 GOSUB 10050
2880 IF X$="0" GOTO 2900
2890 NEXT
2900 CLOSE
2910 GOTO 1170
2920 '----- LETTURA MOVIMENTI -----
2930 CLS
2940 IF C=1 AND MCZ=0 GOTO 1170
2950 IF C=0 AND MFZ=0 GOTO 1170
2960 GOSUB 10040
2970 IF C=1 THEN H=MCZ ELSE H=MFZ
2980 OPEN"i",1,NN$
2990 FOR I=1 TO H
3000 INPUT#1, N, D$, M$, I$, P$, A$
3010 CLS : REVDN
3020 PRINT NO$; N;
3030 PRINT" $ movim.:";I
3040 PRINT : PRINT" data " : REVOFF: PRINT " " : PRINT D$
3050 PRINT : REVDN: PRINT" merce " : REVOFF: PRINT M$
3060 REVDN: PRINT" tot.fatt. " : REVOFF: PRINT I#
3070 PRINT : REVDN: PRINT" acconto " : REVOFF: PRINT P#
3080 PRINT : REVDN: PRINT" note " : REVOFF: PRINT A$
3090 GOSUB 10050
3100 IF X$="0" GOTO 3120
3110 NEXT

```

```

3120 CLOSE
3130 GOTO 1170
3140 '----- SITUAZIONE DEI PAGAMENTI -----
3150 CLS
3160 REVON: PRINT " situazione fornitori: " : REVOFF
3170 PRINT : PRINT
3180 IF SF#>0 PRINT "    da pagare l.";SF#
3190 IF SF#<0 PRINT "    pagate in pu' l.";ABS(SF#)
3200 IF SF#=0 PRINT "    fatture tutte saldate"
3210 PRINT : REVON: PRINT " situazione clienti  : ":REVOFF
3220 PRINT : PRINT
3230 IF SC#>0 PRINT "    da incassare l.";SC#
3240 IF SC#<0 PRINT "    incassate in piu' l.";ABS(SC#)
3250 IF SC#=0 PRINT "    nessun cliente insoluto"
3260 PRINT : REVON: PRINT " > ";
3270 IF SF#>SC# PRINT "dare l.";SF#-SC#
3280 IF SC#>SF# PRINT "avere l.";SC#-SF#
3290 IF SC#=SF# PRINT "    pareggio  "
3300 REVOFF
3310 GOSUB 10070
3320 GOTO 1170
10000 '----- SUBROUTINES -----
10010 PRINT@448,"scegli"
10020 X=VAL(INKEY$)
10030 IF X<1 OR X>M GOTO 10020 ELSE RETURN
10040 PRINT"programma in corso - attendere" : RETURN
10050 REVON: PRINT@480," premi un tasto (0 per finire) "; : REVOFF
10060 GOTO 10080
10070 REVON: PRINT@480," premi un tasto "; : REVOFF
10080 X$=INKEY$
10090 IF X$="" GOTO 10080
10100 RETURN
10110 CLS
10120 REVON: PRINT@192, TX$ : REVOFF
10130   FOR I=1 TO 5
10140     PRINT@207,"      "
10150     FOR J=1 TO 15
10160       NEXT J
10170     PRINT@207,Q;"caratteri"
10180     FOR J=1 TO 15
10190       NEXT J
10200   NEXT I
10210 PRINT : RETURN
10220 PRINT ERR/2+1 : STOP

```


RIFERIMENTI DEL PROGRAMMA GESTIONE AZIENDALE

0	1170/2 1280 1640/2	2420 2430	1430/\$ 1440/\$3 1710/\$
	1740 1620 1830 1890	2460 2470	1750 1810/\$ 1850/\$
	2030 2040 2180 2220/2	2490 2510	1910/\$ 2020 2030
	2260 2280 2710 2720/2	2520 2500	2040 2050 2100 2140
	2940 2950/2 3180	2620 1290/2	2250 2260 2270 2280
	3190 3200 3230 3240	2700 2680	2330 2360 2370 2480
	3250	2900 2880	2560 2630 2640 2650
1	1090 1100 1110/2	2930 2680	2760/\$ 2800 2820/\$
	1130 1270/2 1740/2	3120 3100	2840 3000 3010 3020
	1750/2 1760 1770	3150 1260	3050 3060 3070 3080
	1790 1880 1960 1980	10010 1250 1370 2670	3160 3210 3260 10050
	2050 2080 2090 2100	10020 10030	10070 10120
	2180 2190/2 2220	10040 1080 1730 2070 2300	NC 1120/% 1740/%3 1850/%
	2270 2310 2320 2330	2540 2730 2960	2050/% 2190/%2 2270/%
	2590/3 2710 2740	10050 2150 2380 2870 3090	2710/% 2740/%
	2750 2760 2770 2780	10070 3310	NF 1140/% 1740/%3 1910/%
	2940 2970 2980 2990	10080 10060 10090	2040/% 2180/%2 2280/%
	3000 10030 10130	10110 1410 1460 1510 1560	2720/% 2740/%
	10150 10180 10220	1610 1660 1690	NN 1270/\$ 1280/\$ 2550/\$
2	1100/2 1130 1280	10220 1040	2980/\$
	1800 1810/3 1880	A 1100/\$ 1810/\$ 1870/\$	ND 1270/\$ 1280/\$ 1330/\$
	1940/2 2550 2560	1930/\$ 2460/\$ 2470/\$	1340/\$ 1760/\$ 2000/\$
	10220	2560/\$ 2760/\$ 2840/\$	2080/\$ 2130/\$ 2240/\$
3	1270 2660	3000/\$ 3080/\$	2310/\$ 2360/\$ 2750/\$
4	1280 1360	C 1270 1280 1520/\$	2800/\$ 3020/\$
5	1240 1260 10130	1530/\$ 1540/\$3 1710/\$	P 1640/\$ 1700/\$ 1770/\$
8	1100 1770/2 1810	1740 1820 2040 2050	1780/\$ 1780/\$ 1830/\$
	2090/2 2320/2 2760/2	2180 2190 2270 2280	1890/\$ 2090/\$ 2320/\$
11	1180	2590 2710 2720 2740	2340/\$ 2340/\$ 2580/%2
15	10150 10180	2760/\$ 2820/\$ 2940	2760/\$ 2810/\$ 2810/\$
16	1650	2950 2970	2860/\$ 3000/\$ 3070/\$
20	1550 2760	CR 1170 1750 1980	PM 1830/\$ 1890/\$ 2220/\$
30	1400 2140 2370 2760/3	D 2410/\$ 2560/\$ 3000/\$	2450/\$ 2560/\$ 2580/\$
31	2400 2430 2470	3040/\$	PU 1740/2 1750 1790
34	1030	FF 1060 1190 1200 1210	Q 1400 1430 1440 1480
63	2470	1220 1230 1320 1330	1490 1530 1540 1550
95	2430	1340 1350 2140 2370	1580 1590 1600 1630
128	2400	2480 2630 2640 2650	1650 10170
129	1600 2760	2800 2840 3040 3050	R 1620/\$ 1630/\$ 1710/\$
160	2430	3060 3070 3080 3160	RM 1170 1750 1830 1890
166	1060	3210 3300 10050 10070	1960 2220
192	10120	10120	S 1100/\$ 1120/\$ 1140/\$
207	10140 10170	H 2740/2 2770 2970/2	1810/\$ 1840/\$ 1900/\$
239	1770 2090 2320	2990	SC 1120/\$ 1830/\$4 1840/\$
243	1100 1810	I 1640/\$ 1670/\$ 1770/\$	3230/%2 3240/%2 3250/\$
288	2440	1780/\$ 1780/\$ 1830/\$	3270/%2 3280/%2 3290/\$
352	2470	1890/\$ 2090/\$ 2320/\$	SF 1140/\$ 1890/\$4 1900/\$
448	10010	2340/\$ 2340/\$ 2580/%2	3180/%2 3190/%2 3200/\$
480	2480 10050 10070	2760/\$ 2770 2780	3270/%2 3280/%2 3290/\$
1000	1030	2800 2810/\$ 2810/\$	T 1570/\$ 1580/\$ 1590/\$3
1170	1380 1960 2030 2160	2850/\$ 2990 3000/\$	1710/\$ 2760/\$ 2830/\$
	2260 2390 2680 2710	3030 3060/\$ 10130	TX 1400/\$ 1450/\$ 1500/\$
	2720 2910 2940 2950	10200	1550/\$ 1600/\$ 1650/\$
	3130 3320	IM 1830/\$ 1890/\$ 2220/\$	1680/\$ 10120/\$
1310	1290/2	2440/\$ 2560/\$ 2580/\$	V 1470/\$ 1480/\$ 1490/\$3
1400	1380 1430 2200	J 10150 10160 10180	1710/\$ 2760/\$ 2820/\$
1440	1440	10190	W 1770/\$ 1780/\$
1450	1480	M 1100/\$ 1120/\$ 1140/\$	X 1260 1270/2 1280/2
1490	1490	1240 1360 1810/\$	1290 1380 2160/\$
1500	1530	1860/\$ 1920/\$ 2420/\$	2390/\$ 2490/\$ 2500/%2
1540	1540	2430/\$ 2560/\$ 2660	2520/\$ 2680 2880/\$
1550	1580	3000/\$ 3050/\$ 10030	3100/\$ 10020 10030/2
1590	1590	MC 1120/% 1860/% 2590/%2	10080/\$ 10090/\$
1600	1630	2940/% 2970/%	Y 1030/\$ 2560/\$6
1750	2600	MF 1140/% 1920/% 2590/%2	Z 1710/\$ 1780/\$ 2090/\$
1890	1820	2950/% 2970/%	2140/\$ 2320/\$ 2370/\$
1950	1880	N 1060 1100/\$ 1120/\$	
1980	1380 2040 2050	1140/\$ 1180 1200	
2220	1380 1960 2270 2280	1210 1220 1230 1320	
	2520	1330 1340 1350 1420/\$	

Panasonic KX-P1090

Caratteristiche: 80 col./80 carat. sec.
Bidirezionale ottimizzata
FRIZIONE/TRATTORE

Interfaccia: PARALLELA
CENTRONICS

8 formati: Pica—Elite
Compresso-Espanso

Grafica: BIT IMAGE
(960-1920 punti per linea)

Costo: L. 822.000 + I.V.A.

**Tutti i modelli sono reperibili presso
NUOVA ELETTRONICA e presso i
nostri più forniti CONCESSIONARI.**

Panasonic KX-P1091

Caratteristiche: 80 col./120 carat. sec.
Bidirezionale ottimizzata
FRIZIONE/TRATTORE

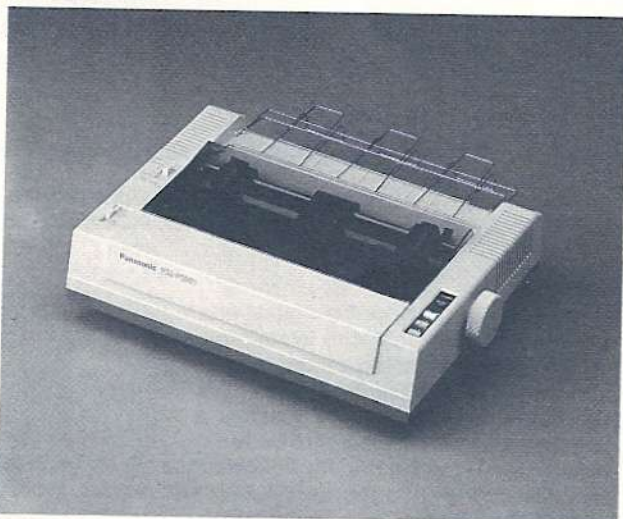
Interfaccia: PARALLELA
CENTRONICS
PC.IBM COMPATIBILE

32 formati: Pica-Elite-Italico
Compresso-Espanso
N.L.Q. e proporzionale.

Grafica: BIT IMAGE
(480/960/1920
punti per linea)

Costo: L. 1.050.000 + I.V.A.

Sconto speciale a tutti gli Abbonati.



Panasonic KX-P1092

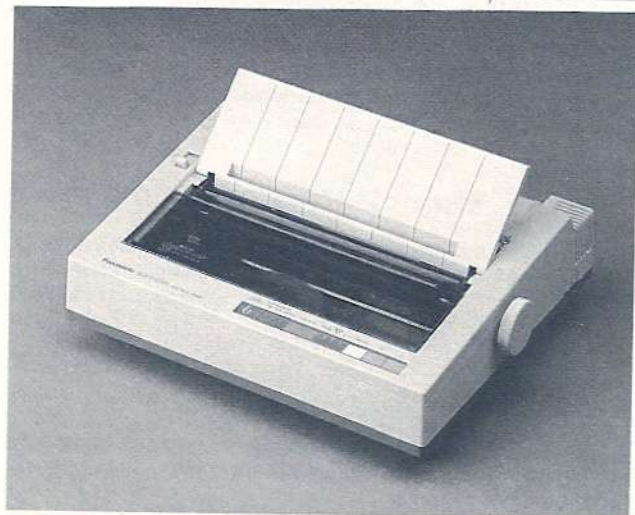
Caratteristiche: 80 col./180 carat. sec.
Bidirezionale ottimizzata
FRIZIONE/TRATTORE/
PLOTTER

Interfaccia: PARALLELA
CENTRONICS
PC.IBM COMPATIBILE

64 formati: Pica-Elite-Italico
Compresso-Espanso
N.L.Q. e proporzionale
(totalmente miscelabili)

Grafica: BIT IMAGE
(480/960/1920
punti per linea)

Costo: L. 1.250.000 + I.V.A.



Il programma simula una battaglia spaziale in difesa della «GALASSIA» fra una astronave comandata dal giocatore, l'«ENTERPRISE», ed un gruppo di vascelli nemici, i famigerati «KLINGON».

Il gioco, molto elaborato ed avvincente, è ricco di varianti e di possibilità di strategia ed è aiutato in questo da una grafica video molto curata. Proprio per queste esigenze, il programma è stato dedicato specificatamente per l'uso con l'interfaccia video **GRAFICA** ed inoltre necessita di uno spazio di memoria di 56 Kbyte per poter girare correttamente. Lo svolgimento del gioco, proprio per le sue molteplici possibilità nella tattica di conduzione, difficilmente presenta condizioni di «battaglia» analoghe fra loro salvo, ovviamente, nella parte iniziale di introduzione al gioco stesso.

La nostra descrizione sarà, perciò, inerente alle operazioni di inizio necessarie per lanciare il programma e alle nozioni sulla strategia del gioco per spiegarne le possibilità di uso in modo da offrire ai nostri lettori un facile riferimento per poter capire rapidamente lo svolgersi del gioco. Iniziamo quindi dall'accensione stessa del computer e, pigiato il tasto di RESET, attendete che sul monitor appaia la dicitura per il consenso all'inserimento del disco nel drive, cioè l'ormai nota scritta:

GRAFIC — MONITOR 1.0

Type

«ESC» for BOOTSTRAPPING

«RETURN» for testig VIDEO

>_<

pigiare perciò il tasto «ESC» e, successivamente alla comparsa della scritta:

**Insert NE-DOS-DISK Version Graftic in drive 0
then type a key**

>_<

inserire il disco nel drive 0 e digitare una lettera qualsiasi sulla vostra tastiera, ad esempio A. Il drive ora si accenderà e dopo qualche istante apparirà sul monitor la sagoma ingrandita dell'«ENTERPRISE», la vostra sofisticata nave spaziale.

Trascorsi circa 10 secondi, il programma inizia il suo svolgimento normale e vi informa sul numero delle navi nemiche presenti nella galassia e perciò da distruggere, sull'anno astrale di inizio della vostra battaglia assieme a quanti anni avete a disposizione per portare a termine la battaglia ed infine sul numero di basi stellari che serviranno all'Enterprise per rifornirsi di energia e per riparare i danni subiti durante lo svolgersi delle azioni.

Una volta lette queste informazioni preliminari, potete iniziare il gioco vero e proprio digitando RETURN. Avrete così riportata sul video la condizione di partenza e cioè la mappa del settore della zona di galassia in cui ora vi trovate e tutti i dati relativi allo stato della vostra astronave. Inoltre vi viene indicato se nel vostro settore sono presenti o no delle unità nemiche con la scritta:

«CONDIZIONE ROSSA» = Nemici nel settore

«CONDIZIONE VERDE» = Nessun nemico nel settore

La mappa della galassia infatti, è suddivisa in 64 settori ognuno dei quali è riconoscibile da un nome e da un numero romano che ne identificano la posizione nella scacchiera della galassia stessa. La configurazione del vostro «campo di battaglia astrale» perciò è così disposta:

	1	2	3	4	5	6	7	8
1			ANTARES				SIRIUS	
2			RIGEL				DENEK	
3			PROCYON				CAPELLA	
4			VEGA				BELTELGENSE	
5			CANOPUS				ALDEBARANI	
6			ALTAIR				REGULUS	
7			SAGITTARIUS				ARCTURUS	
8			POLLUX				SPICA	
	I	II	III	IV	I	II	III	IV

fig. 100

Il programma quindi comunica la vostra posizione scrivendo sul video il nome del settore a voi corrispondente. Questo significa che la scritta:

VEGA II corrisponde al quadrante 4,2

REGULUS IV corrisponde al quadrante 6,8 e così via.

Premendo ora il tasto di RETURN, comparirà una tabella accanto alla riproduzione del vostro settore, nelle quali sono elencate tutte le possibili operazioni che la vostra astronave può compiere:

- 0 - per rassegnare il comando
- 1 - per manovrare la nave
- 2 - per radar a corto raggio
- 3 - per radar a lungo raggio
- 4 - per sparare phaser
- 5 - per sparare razzi fotonici
- 6 - per manovrare gli schermi
- 7 - per rapporto completo
- 8 - per attivare il computer

Digitando il numero corrispondente potrete perciò passare all'azione diretta per cercare di portare a termine con successo la vostra missione. La maggior parte di queste operazioni sono di comprensione immediata e non necessitano perciò di ulteriori spiegazioni; ci soffermiamo quindi su quei soli punti, per così dire, «degni di nota», cercando di dare più che altro dei suggerimenti «strategici» non essendoci, fra l'altro, vere e proprie difficoltà d'uso di questo programma. Il comando 0 (per rassegnare il comando), serve unicamente ad uscire dal programma stesso e tornare quindi al linguaggio basic e perciò è una istruzione a sé che userete solo quando, prima della fine del gioco, decidete di «deporre le armi»!

NUMERO DIGITATO: 3 (Radar a lungo raggio)

Con questo comando potrete avere la situazione della galassia negli otto quadranti adiacenti a quello in cui voi siete. Questo messaggio vi viene riferito attraverso una tabellina, visualizzata in basso a destra dello schermo, così formata:

```

+---+---+---+
1001103011001
+---+---+---+
1002100311131
+---+---+---+
1101111011031
+---+---+---+

```

fig. 101

La vostra astronave si trova al centro di questa zona, cioè nel quadrante in cui è scritto 003; queste cifre descrivono, settore per settore, la situazione del quadrante corrispondente e precisamente:

- 103 -** 1ª cifra: numero di KLINGON nemici (= 1)
2ª cifra: numero di basi stellari (= 0)
3ª cifra: numero di stelle (= 3)

L'ultimo dato, quello cioè relativo alle stelle, è molto importante in quanto, essendo gli unici oggetti a non poter essere distrutti dalle vostre armi, oltre ad ostacolarvi i movimenti, riflettono su di voi i colpi che vorreste invece indirizzare alle astronavi nemiche. Cercate perciò di evitare i settori in cui siano presenti nemici e stelle in numero elevato.

NUMERO DIGITATO: 6 (Manovrare gli schermi)

Gli «schermi» sono la protezione più importante della «Enterprise» e sono l'unica difesa a disposizione dell'astronave contro gli attacchi nemici e contro eventuali ritorni di esplosioni dovuti a colpi sparati nella direzione delle stelle. Inizialmente questi schermi sono a 0, sono cioè privi di energia e lasciano perciò sguarnita da ogni difesa la vostra astronave. Dovrete perciò utilizzare una parte dell'energia a vostra disposizione per attivare questi schermi selezionando appunto il numero 6 nella tabellina iniziale. Scrivendo ora la cifra corrispondente alle «unità di energia» che vorrete destinare a questo scopo (ad esempio 500) attiverete automaticamente questa difesa.

NUMERO DIGITATO: 8 (Attivare il computer)

Con questa selezione si attiva il computer di bordo della Enterprise il quale è in grado di fornirvi indicazioni importantissime per la buona riuscita delle operazioni in corso. Le mansioni che il computer è in grado di svolgere sono:

- 1 - registrazione cumulativa**
- 2 - traiettoria razzi fotonici**
- 3 - dati di atterraggio**
- 4 - computo direzione/distanza**
- 5 - mappa costellazione**

Digitando il numero corrispondente alla funzione da svolgere, il computer verrà attivato e messo a disposizione del «capitano dell'astronave» per fornirgli le indicazioni volute.

Digitando 1 viene riportata la situazione di tutta la galassia così come il computer riesce a riceverla tramite i suoi radar e la mappa della galassia così fornita è del tutto simile a quella riportata nella TAB. 1. I rimanenti comandi sono forniti di calcolo che il computer di bordo possiede e che servono a fornire i dati per manovrare l'astronave o le armi dell'astronave stessa.

Tutte le informazioni assunte in questo modo sono molto importanti nel vivo dello svolgersi della battaglia e assicurano il successo delle vostre operazioni a meno che l'astronave non abbia subito qualche danno durante questo periodo. L'attraversamento della galassia è infatti pieno di rischi e di imprevisti e tutto può comportare dei danni, per altro prontamente segnalati durante il gioco, a qualche apparecchiatura di bordo mettendo fuori uso i razzi fotonici o il radar a lungo raggio o qualche altra delicata parte della vostra astronave. Questi ed altri imprevisti movimenteranno continuamente lo svolgersi di questo gioco e lasciamo quindi a voi il piacere ed il «RISCHIO» di scoprirli da soli.

Quando ci si pone di fronte ad un computer, il primo problema che occorre risolvere è quello di capirne il funzionamento. In tutti i manuali che si possono reperire e che dovrebbero servire a risolvere questo problema, troviamo sempre le solite spiegazioni, cioè come accenderlo, quali tasti digitare per ottenere una stampa, per scrivere una semplice istruzione ecc. ma nessuno spiega che il computer è una macchina in grado di eseguire velocemente delle operazioni, ma solo se queste sono scritte in un linguaggio a lui comprensibile, e cioè in linguaggio macchina.

Per comprendere che cos'è questo linguaggio macchina, occorre subito precisare che quando noi scriviamo sulla tastiera delle lettere o parole, come PRINT, GOSUB, RUN ecc., queste non possono entrare nella CPU, perché il computer non è un registratore a nastro sul quale possiamo memorizzare delle frasi, ma solo un integrato che dispone di ingressi atti a riconoscere esclusivamente dei livelli logici 1 o 0, cioè presenza di una tensione positiva o tensione uguale a zero.

Perciò la CPU, disponendo di 8 piedini di ingresso, riconoscerà la lettera A solo se su questi ingressi è presente 00000001, la B solo se appare la condizione 0000010, la lettera C se appare invece un 00000100 (questo è solo un esempio teorico, per far capire che occorre semplicemente modificare il livello logico su questi ingressi per ottenere i vari dati).

Pertanto, per far eseguire i vari comandi al computer, dovremmo, per ognuno di questi, scrivere una infinità di numeri 0001000 — 1100000 — 1001000 — 11110001 — ecc., e questo, come comprenderete, risulta molto scomodo e laborioso. Per rendere più semplice l'uso del computer si è pensato di utilizzare dei linguaggi interpreti, come ad esempio il Basic.

Questi linguaggi, se permettono all'operatore un più facile uso del computer, hanno lo svantaggio di rallentarne il funzionamento.

Per poter meglio comprendere questo concetto, supponiamo che un operatore di nazionalità ITALIANA voglia dialogare con il computer, in grado di capire solo il CINESE. Non potendo studiare questa incomprensibile lingua, si è pensato di preparare un dizionario con tutte le parole di comando del computer, e cercare un traduttore che sia in grado di capire il nostro linguaggio, che sappia riconoscere le istruzioni dalle semplici parole e sappia riscrivere il tutto nel solo LINGUAGGIO conosciuto dal computer.

Così, quando scriviamo STAMPA (cioè l'istruzione PRINT del Basic), il traduttore deve sfogliare questo dizionario fino a trovare questa parola poi, trovata, la dovrà riscrivere in cinese. Se poi scriviamo «BARI», dovrà nuovamente sfogliare questo dizionario e non trovando la parola come una istruzione, dovrà riconvertire ogni lettera in livelli logici equivalenti e memorizzarli, per poi riconvertire la stessa parola o in stampa o sul video e, per far tutto questo, gli occorre del tempo.

Per rendere più veloci queste operazioni, sarebbe necessario eliminare questo traduttore (linguaggio Basic) e cercare un linguaggio, un po' più complesso per l'operatore, ma che venga subito capito dal CINESE. Ad esempio, anziché scrivere FERROVIE DELLO STATO ITALIANO, che dovremmo poi tradurre in tale lingua, se noi insegniamo al cinese che FFSS equivale a quanto scritto sopra, non avrà più bisogno di consultare nel dizionario ben quattro parole. Lo stesso dicasi per STATI UNITI D'AMERICA: se gli abbiamo precedentemente spiegato che USA ha lo stesso significato, la traduzione sarà molto più veloce, oppure, al posto di IMPEDENZA DI ALTA FREQUENZA, semplifichiamo con JAF.

Sapendo il significato di ognuna di queste sigle, il nostro CINESE riuscirà velocemente a tradurle, come potremmo fare noi se un cinese, anziché scrivere IUNCINCANLI FUN LA HOMI GIUN MON fun ci scrive 10 JAF: noi capiremmo subito che sono DIECI IMPEDENZE DI ALTA FREQUENZA senza l'aiuto di nessun traduttore.

Non dovendo quindi eseguire un complicato lavoro di traduzione, l'esecuzione di un qualsiasi programma diventa velocissima.

Passando da questo esempio astratto all'uso pratico del computer, gli esempi fatti ricalcano fedelmente il funzionamento di un linguaggio evoluto (il traduttore con il vocabolario) o del programma scritto in linguaggio macchina (una lunga serie di codici convenzionali).

Vi sarete sicuramente accorti, nei programmi molto lunghi scritti in Basic, che intercorre un certo lasso di tempo fra il comando di RUN e la risposta che il computer fornisce e questo tempo viene solitamente giustificato dicendo che, in quel periodo, la macchina «sta pensando». In realtà, come ormai avrete intuito, il tempo che scorre è impiegato per tradurre, comandare ed eseguire le varie operazioni presenti nel programma. Niente di «pensante» dunque, né tantomeno di «misterioso»!

Per rendere più veloce la macchina, anziché scrivere i programmi in basic e poi lasciare all'interprete il compito di comunicare con il computer, sarebbe necessario scrivere direttamente i programmi in linguaggio macchina.

In effetti questa seconda ipotesi è possibile ma, logicamente, dovendo utilizzare il linguaggio macchina per «farci capire» dal computer, il compito del programmatore diventa molto più difficile e complesso.

Tale linguaggio non è costituito, come si è abituati a vedere, da istruzioni scritte nella forma PRINT, GOTO ecc., bensì solo ed esclusivamente da numeri, scritti in esadecimale, che corrispondono ognuno alle singole operazioni che il computer è in grado di eseguire.

I vantaggi offerti da questo tipo di programmazione sono essenzialmente la maggior velocità di esecuzione ed inoltre la possibilità di eseguire operazioni che, utilizzando unicamente l'interprete per comunicare operativamente con il computer, risulterebbero molto più complicate o addirittura impossibili da eseguire.

Un tipico esempio di questo, come vedremo nel seguito di questo paragrafo, è l'utilizzo al meglio delle caratteristiche grafiche dell'interfaccia video grafica LX.529, richiamandola in modo diretto anziché da basic.

Il linguaggio macchina però presenta lo svantaggio di essere molto complicato da usare in quanto è facile «perdersi» fra una lista uniforme di numeri ed è per questo che, volendo ottenere le stesse caratteristiche di una programmazione in linguaggio macchina ed avere anche un modo di scrittura dei programmi più facile e comprensibile, si utilizza ancora una volta un interprete, che è appunto l'EDITOR-ASSEMBLER.

EDITOR-ASSEMBLER PER COMPUTER NE-Z80

La caratteristica essenziale di questo linguaggio è proprio quella di essere strettamente legato al tipo di CPU utilizzata nel computer ed infatti l'interprete EDITOR/ASSEMBLER per un computer come quello di NUOVA ELETTRONICA, che utilizza una CPU tipo Z80, è totalmente differente da quello, ad esempio, per l'APPLE, che utilizza una CPU tipo 6502.

Come tutti i «linguaggi evoluti», anche l'editor/assembler ha le sue precise regole di sintassi ma la complessità di questo linguaggio non è data tanto da queste regole o dal numero delle istruzioni (anche se, come vedremo tra breve, ne esistono alcune abbastanza complesse) quanto dal fatto che, per comprenderle, occorre prima aver chiare alcune nozioni che sono strettamente legate alla «struttura fisica» della macchina e che appartengono perciò all'«hard-ware» del computer.

Nelle varie istruzioni che riporteremo infatti, troverete sempre parole ricorrenti come ad esempio «l'ACCUMULATORE», «il REGISTRO HL», «lo STACK-POINTER» o «i FLAG» e quindi, per dare a tutti coloro che non conoscono il significato di tali termini, la possibilità di seguirci ugualmente, apriamo una breve parentesi necessaria a spiegare la struttura interna della CPU Z80, a cui così strettamente il linguaggio EDITOR/ASSEMBLER si collega.

La struttura interna della CPU Z80

La CPU è il corpo centrale del computer ed è il nucleo che governa la totalità delle operazioni svolte dalla macchina. Il modo di operare delle diverse CPU esistenti, è molto simile fra loro e la distinzione fra i diversi tipi di microprocessore (tipo, ad esempio, Z80, 8080, 8085, 6500, 6800 ecc.) è fatta non tanto sulla loro struttura interna quanto invece sulla complessità delle singole operazioni che possono svolgere e sul numero di registri interni che ognuno possiede. Entriamo perciò «nel vivo» del problema e vediamo come avviene lo svolgersi delle operazioni all'interno di un microprocessore.

Come la maggior parte dei circuiti digitali (ed il microprocessore è un circuito digitale), tutte le operazioni svolte dal computer sono governate da un CLOCK, cioè da un segnale a frequenza fissa, generato da un oscillatore a quarzo: più alta è la frequenza di clock e maggiore sarà la velocità di esecuzione di ogni singola istruzione.

Le frequenze comunemente utilizzate in questi circuiti sono: 1 MHz, 2 MHz e 4 MHz.

Occorre ricordarsi che il microprocessore comunica con tutti i circuiti esterni tramite due vie principali. La prima è una porta a 8 bit (cioè in pratica 8 fili paralleli) detta **BUS dei DATI** (DATA BUS), su cui vengono trasmessi e ricevuti i DATI.

La seconda è una porta a 16 bit (cioè 16 fili paralleli) detta **BUS degli INDIRIZZI** (ADDRESS BUS), sulla quale è presente, istante per istante, l'indirizzo relativo al dato presente sul BUS dei DATI.

Esistono infine i terminali di controllo, che servono a predisporre i circuiti esterni a seconda delle esigenze della comunicazione e perciò avremo un terminale di READ (che significa LEGGERE), che predispone i circuiti esterni alla lettura di un dato, un terminale di WRITE (che significa SCRIVERE) che li predispone invece per la scrittura in memoria dei dati, ed altri, di minor importanza, di cui ci occuperemo quando parleremo di operazioni ad essi inerenti.

Riprendiamo perciò il tema iniziale e vediamo subito un primo esempio pratico di come si svolge una semplice istruzione all'interno del computer.

Supponiamo, ad esempio, di voler trasferire il contenuto della cella di memoria di locazione 1000 (in esadecimale), nella cella di memoria di locazione 20A7 (sempre in esadecimale).

Questa operazione, come del resto la maggior parte delle funzioni svolte dal microprocessore, fa uso dei **registri interni** della CPU e perciò è necessario innanzitutto spiegare cosa sono questi «registri interni» già più volte menzionati.

I registri interni della CPU.

I registri interni del microprocessore non sono altro che delle semplici celle di memoria a 8 o a 16 bit, poste all'interno della CPU stessa, e sono le sole celle di memoria sulle quali il computer può operare DIRETTAMENTE.

I registri interni sono così raggruppati:

- A = Accumulatore (8 bit)
- F = Registro dei Flag (8 bit)
- B = Memoria a 8 bit
- C = Memoria a 8 bit
- D = Memoria a 8 bit
- E = Memoria a 8 bit
- H = Memoria a 8 bit
- L = Memoria a 8 bit
- IX = Registro INDICE X (16 bit)
- IY = Registro INDICE Y (16 bit)
- SP = Stack Pointer (16 bit)
- PC = Program Counter (16 bit)
- I = Vettore di interruzione (8 bit)
- R = Rinfresco memorie
- A' = Accumulatore (registro alternativo a 8 bit)
- F' = Registro dei Flag (alternativo a 8 bit)
- B' = Memoria a 8 bit
- C' = Memoria a 8 bit
- D' = Memoria a 8 bit
- E' = Memoria a 8 bit
- H' = Memoria a 8 bit
- L' = Memoria a 8 bit

Quindi, qualsiasi operazione noi vogliamo eseguire, dovremo sempre caricare all'interno della CPU, in uno dei registri a disposizione, il dato che ci interessa e quindi, su di esso, eseguire le operazioni volute.

Torniamo ora al nostro esempio di partenza e vediamo come avviene lo scambio del contenuto di una cella di memoria, da una locazione (nell'esempio la 1000 in esadecimale) all'altra (la 20A7 in esadecimale).

Per prima cosa il computer indirizza la cella di memoria di locazione 1000 e quindi ne legge il contenuto. Fatto questo, il dato così ricavato viene posto nel registro interno chiamato ACCUMULATORE e successivamente viene indirizzata la cella di memoria di locazione 20A7. La cella viene ora messa in scrittura ed il computer ricopia il contenuto dell'ACCUMULATORE, che ora contiene quanto risultava presente nella locazione 1000, e lo passa nella cella di memoria 20A7.

Analogamente a questo primo esempio, la CPU può compiere molte altre operazioni elementari, come ad esempio sommare o sottrarre fra loro il contenuto di due celle di memoria (ovviamente sempre caricando dapprima il dato nell'ACCUMULATORE o in qualche altro registro interno e poi sommando o sottraendo a questo il dato successivo), decrementarne o incrementarne il contenuto, comparare il valore di una celle di memoria con l'accumulatore e, in base al risultato ottenuto, compiere altre operazioni, eseguire un AND logico o un OR logico su di un registro e l'accumulatore, ruotare il contenuto di un registro verso destra o verso sinistra e tante altre operazioni simili, più o meno complesse.

Dalla composizione di tutte queste singole operazioni elementari, si possono ottenere poi dei processi molto più complessi e cioè tutti quei risultati di calcolo ed elaborazione dati che abitualmente vediamo eseguire da un computer.

Come abbiamo appena visto nel semplice esempio riportato, esiste una «priorità» nell'utilizzo dei vari registri interni: il registro A, l'ACCUMULATORE, è ad esempio uno dei registri principali della CPU e viene chiamato in causa nella maggioranza delle operazioni che il computer esegue.

Subito di seguito, seguendo un ordine di importanza dei registri interni del microprocessore, vengono:
PC (Program-Counter)
SP (Stack-Pointer).

Questi due registri meritano una trattazione particolare in quanto seguono tutte le operazioni che vengono svolte dal computer. Facciamo al solito un piccolo esempio pratico per illustrare più chiaramente le funzioni svolte dal Program-Counter e dallo Stack-Pointer.

I registri PROGRAM-COUNTER (PC) e STACK-POINTER (SP)

In un programma scritto in linguaggio macchina, tutte le varie istruzioni risiedono ciascuna in celle di memoria, una adiacente all'altra. Il numero di celle occupato da ogni istruzione dipende dal tipo di istruzione, ad esempio l'istruzione **LD A, (8822H)**, che significa carica nell'accumulatore il contenuto della cella di memoria di indirizzo 8822 in esadecimale, occupa tre celle di memoria così suddivise:

Supponendo di avere scritto il programma a partire dalla cella di memoria 1000 (sempre in esadecimale), in questa cella verrà scritto il dato 3A (che è il codice, detto «codice operativo», dell'istruzione **LD A**), nella cella 1001 verrà scritto il dato 22 e nella cella 1002 il dato 88.

Perciò, osservando di seguito le varie celle di memoria che contengono il programma, avremo una cella che contiene il «codice operativo» dell'istruzione da eseguire, seguita da alcune celle di memoria che contengono il dato a cui l'istruzione fa riferimento.

Perché il computer possa interpretare correttamente tutta questa sequenza di numeri, gli occorre un «puntatore» che gli specifichi, di volta in volta, quale è il codice operativo dell'istruzione e quali sono invece i dati ad essa relativi e questa è la funzione del **Program-Counter**.

Questo registro infatti, provvede a mantenere, costantemente aggiornato, l'indirizzo della cella di memoria contenente il codice operativo della prossima istruzione da eseguire.

Lo **Stack-Pointer** invece, è un registro che gestisce i «salti» fra le varie linee di programma che il computer deve eseguire durante lo svolgersi delle operazioni. Per capire meglio il funzionamento e lo scopo di questo registro, prenderemo come esempio un programma complesso, all'interno del quale siano presenti delle **subroutine**, cioè dei sottoprogrammi che contengono delle sequenze di operazioni da ripetere più volte durante l'esecuzione del programma e quindi scritte separatamente, una volta per tutte, senza doverle più riscrivere inutilmente nel programma principale.

Il salto ad una **subroutine**, avviene con l'istruzione di **CALL** ed il ritorno dalla subroutine al programma, avviene invece con l'istruzione di **RET**.

Normalmente, l'esecuzione del programma nel quale siano presenti delle subroutine, avviene nel modo seguente:

Quando il computer incontra l'istruzione di **CALL**, esegue un salto alla locazione di memoria che contiene la subroutine. Ad esempio, scrivendo **CALL 3000**, esegue un salto alla locazione 3000, a partire dalla quale sono presenti tutte le celle di memoria che contengono le istruzioni della subroutine desiderata. Vengono quindi eseguite tutte le operazioni presenti in questa parte di programma ed alla fine, risultando sempre presente un **RET**, il computer torna nuovamente al programma principale e riprende lo svolgimento delle operazioni dall'istruzione subito seguente alla **CALL 3000**.

Esiste tuttavia un altro modo di esecuzione delle subroutine del programma, seguendo una struttura «nidificata».

Vediamo nella pratica in che cosa consiste questa struttura «nidificata».

Quando nel programma principale, il computer incontra una istruzione di **CALL**, passerà, come già vi abbiamo spiegato, alla locazione di memoria che contiene la subroutine desiderata.

All'interno di questa subroutine però, è presente, prima dell'istruzione di **RET**, un'altra istruzione di **CALL** (ad esempio **CALL 2000**) per cui il computer effettua un nuovo salto alla successiva subroutine così indirizzata. Questo potrebbe ripetersi poi per altre due, tre, dieci volte, in quanto, all'interno delle nuove subroutine chiamate, può essere ancora presente, prima dell'istruzione di **RET**, una nuova istruzione di **CALL**.

Alla fine, nell'ultima subroutine chiamata, verrà incontrata l'istruzione **RET** e a questo punto il computer, prima di giungere al programma principale, dovrà eseguire all'indietro, lo stesso percorso fatto con le varie istruzioni di **CALL**.

Ad ogni istruzione di **RET** perciò, tornerà dall'ultima subroutine a quella precedente e così di seguito, fino ad arrivare al programma principale.

Mentre per il primo modo di esecuzione delle subroutine è sufficiente memorizzare il solo indirizzo del programma principale a cui il computer deve tornare dopo l'esecuzione della subroutine, nel secondo, per ottenere il corretto svolgersi delle operazioni, non è più sufficiente un solo indirizzo ma si devono memorizzare tutti gli indirizzi di ritorno dalle varie subroutine.

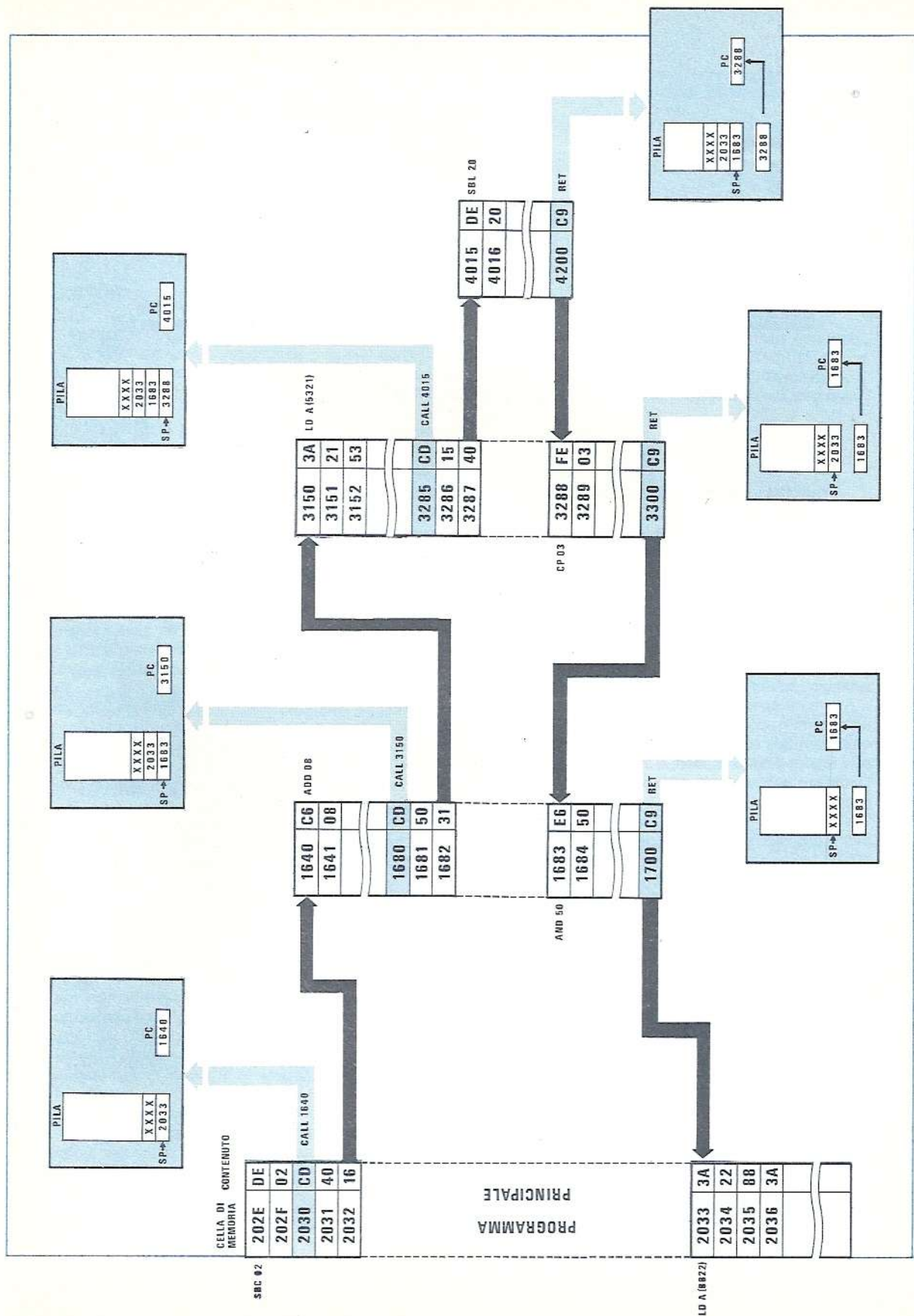


fig. 102 Esempio di programma con struttura nidificata.

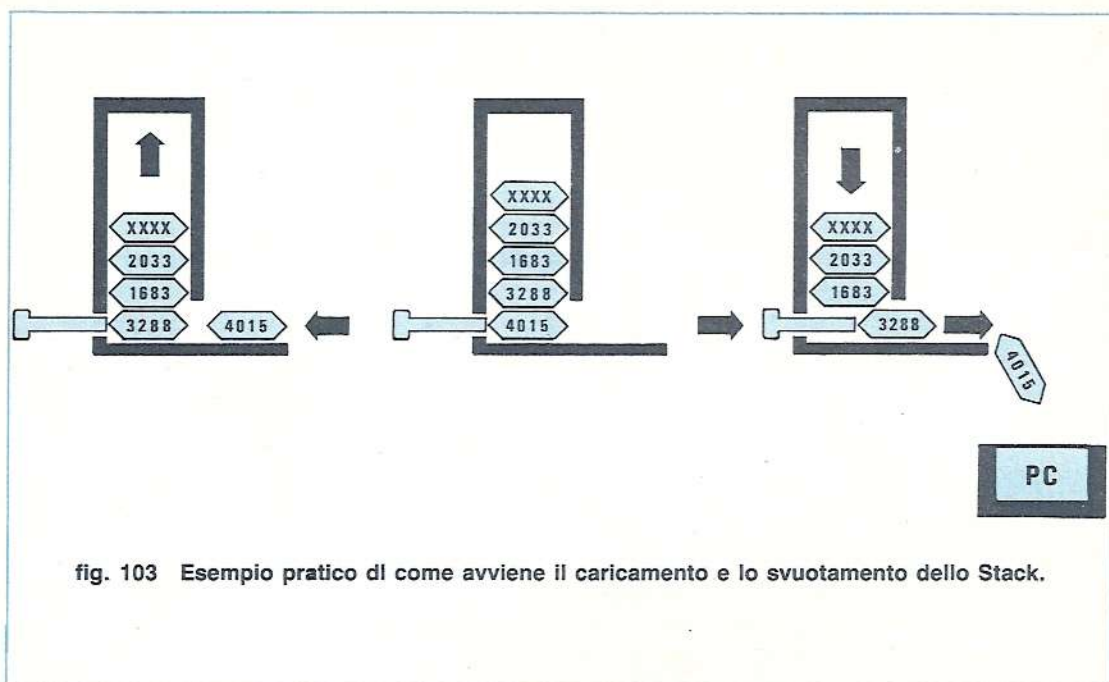


fig. 103 Esempio pratico di come avviene il caricamento e lo svuotamento dello Stack.

Per questo motivo non è più sufficiente una sola cella di memoria in quanto gli indirizzi da memorizzare possono essere più di uno, quindi occorre costruire un'area di memoria esterna, nella quale immagazzinare tutti questi indirizzi ed inoltre si dovrà sapere, di volta in volta, quale cella dovrà essere indirizzata in questa area per poterli poi prelevare in modo corretto.

A queste operazioni provvedono lo **Stack** e lo **Stack-Pointer**. Lo **Stack** (che in inglese significa PILA) è un'area di memoria che il computer riserva automaticamente, nella memoria RAM esterna a disposizione, entro la quale vengono scritti tutti gli indirizzi di ritorno da memorizzare, mentre lo **Stack-Pointer**, come lascia intuire il nome stesso, è invece un registro «puntatore», all'interno del quale è presente, di volta in volta, l'indirizzo della cella di memoria dello Stack, nel quale è contenuto l'indirizzo di ritorno necessario al programma.

Il riempimento dell'area di memoria dello Stack, avviene con una modalità ben precisa e cioè tutti questi indirizzi di ritorno vengono posti in memoria uno seguente all'altro, proprio come si farebbe se si dovessero riporre all'interno di un contenitore, tante monete: si costruirebbe cioè una PILA ed è per questo motivo che tale area di memoria è chiamata **STACK** (cioè PILA).

Sempre rimanendo nell'analogia fatta, una volta costruita questa pila, per estrarre le monete introdotte si inizierà sempre riprendendo l'**ultima moneta**, che si troverà in cima alla pila, e così a ritroso fino alla prima depositata.

L'area di memoria dello Stack viene perciò continuamente riempita e svuotata durante lo svolgersi del programma.

Ogni volta che si deve memorizzare un indirizzo di ritorno vengono occupate due nuove celle di memoria all'interno dello Stack (l'indirizzo infatti è un dato a 16 bit perciò occorrono due celle da 8 bit) e il contenuto dello Stack-Pointer viene decrementato di 2 per «puntare» così alla nuova cella di memoria utilizzata.

Quando invece si deve «ritornare» da una subroutine alla linea di programma da cui è avvenuto il precedente salto, viene letto inizialmente il contenuto dello Stack-Pointer per sapere in quale cella di memoria dello Stack è contenuto l'indirizzo di ritorno. Una volta letto tale indirizzo, questa informazione è posta nel Program-Counter ed il computer perciò, riprenderà ad eseguire le istruzioni dalla nuova cella di memoria così indirizzata, mentre il contenuto dello Stack-Pointer viene incrementato di 2 in modo che, al prossimo comando di ritorno da una subroutine, questo registro «punti» automaticamente alla cella di memoria che contiene l'informazione necessaria al nuovo ritorno da eseguire.

Questo, in pratica, è il funzionamento dello Stack e del registro dello Stack-Pointer.

In figura 102 potrete osservare «graficamente» un esempio di programma con subroutine nidificante ed il relativo funzionamento dello Stack e dello Stack-Pointer in questo caso.

La sintassi delle istruzioni del linguaggio Editor/Assembler.

Questo interprete, si compone di due blocchi distinti, ognuno dei quali ha funzione specifiche ben determinate.

— **EDITOR** — primo blocco che serve per scrivere i programmi veri e propri seguendo le regole di sintassi che vedremo di seguito. (**Programma sorgente**)

— **ASSEMBLATORE** — secondo blocco dell'interprete che serve per costruire, partendo dal programma scritto sotto EDITOR, il programma ad esso equivalente, scritto però in linguaggio macchina. (**Programma oggetto**)

EDITOR

Iniziamo dunque a descrivere le istruzioni vere e proprie, definendo prima di tutto la loro sintassi generale.

Ogni istruzione è composta da 5 nuclei distinti, chiamati comunemente «campi», ad ognuno dei quali è assegnato un compito preciso. Ogni campo, in una singola istruzione, è diviso dall'altro da almeno uno spazio. Vediamo subito un primo esempio pratico. Prendiamo una generica linea di programma, ad esempio:

110 START LD A, (1000H); Carica l'accumulatore

ed analizziamo in dettaglio le varie parti che costituiscono tale linea.

L'istruzione viene così scomposta:

1° Campo Num. Linea	2° Campo Label	3° Campo Istruz.	4° Campo Oprando	5° Campo Commenti
110	START	LD	A, (1000H)	;Carica l'accum.

Numero di linea (nell'esempio è 110)

È il numero di linea corrente, cioè la linea di programma dell'istruzione. Come abbiamo già accennato, tale numero viene fornito automaticamente dall'interprete e perciò è già definito senza alcun intervento da parte del programmatore.

Label o Etichetta (Nell'esempio START)

Questo nucleo dell'istruzione serve ad **identificare** la linea di programma nella quale si trova l'etichetta stessa. Come risulterà più chiaro quando parleremo delle istruzioni di «salto», queste «etichette servono a definire la posizione nella quale il programma deve «saltare», a seconda delle condizioni che si verificano. Ogni «Label» può essere costituita da un massimo di 6 caratteri, sia numerici che alfabetici, con la sola condizione che il **primo carattere sia una lettera**. Così sono ammesse, ad esempio, tutte le seguenti label:

```
START1  
LOOP0  
FINE2  
S23/B
```

È vietato l'uso di caratteri speciali (come ad esempio le parentesi, l'apostrofo, il cancelletto ecc.) e l'uso dello spazio in quanto, quest'ultimo, porta automaticamente l'interprete a leggere il campo successivo.

L'etichetta è un campo che può anche essere omissa dall'istruzione. In questo caso, dopo il numero di linea, si dovrà digitare uno spazio (la barra).

Istruzione (Nell'esempio, LD)

È il nucleo fondamentale dell'istruzione ed è la parte che definisce l'operazione da eseguire. I codici di operazione che si possono scrivere a questo punto, sono tutte le istruzioni tipiche della CPU utilizzata (nel nostro caso quindi quelle dello Z80) e le troverete raggruppate schematicamente nelle tavole seguenti. Ogni istruzione fondamentale verrà descritta ed analizzata separatamente, come vedremo fra breve.

Questo campo dell'istruzione non può essere mai omissa, a meno che non si voglia scrivere una linea di commento al programma. Questo è comunque un caso particolare e ne parleremo al momento opportuno.

Operando (nell'esempio A, (1000H))

In questo campo vengono definiti i termini su cui agisce l'operazione precedentemente definita. Nello esempio fatto, l'istruzione completa era perciò LD = Load (che significa «carica») il contenuto della cella di memoria 1000, in esadecimale (H significa Hexdecimal, cioè esadecimale), nell'accumulatore.

Commenti (Nell'esempio; Carica nell'accumulatore)

Quest'ultimo campo serve per aggiungere dei commenti di spiegazione o di promemoria nelle linee di programma. È un nucleo dell'istruzione che può essere omesso ed in ogni caso non altera minimamente le operazioni svolte dai campi precedenti.

Un'altra precisazione importante, per concludere questo primo argomento, è il fatto che, per ogni linea di programma, si può definire **una ed una sola istruzione**. Questa precisazione, che a prima vista può sembrare banale, è invece molto importante in quanto molto spesso, programmando ad esempio in basic, si è abituati a vedere più istruzioni raggruppate in una stessa linea di programma, mentre in EDITOR/ASSEMBLER questo non è mai possibile.

Ad ogni linea quindi, compete sempre una singola operazione e non esistono eccezioni di sorta.

Vediamo ora dettagliatamente le varie istruzioni che possono essere scritte nel linguaggio EDITOR/ASSEMBLER e che possono essere inserite nel terzo e nel quarto campo di ogni singola istruzione. Per ognuna di queste daremo perciò sia la sintassi dell'istruzione che l'operando su cui questa può agire.

NOTE GENERALI per i simboli utilizzati in tutte le istruzioni:

— Tutti i registri a 8 bit visti precedentemente, possono essere trattati dallo Z80 sia singolarmente, come il registro H, B, A, ecc., oppure in coppia, per formare registri a 16 bit. In quest'ultimo caso le coppie di registri utilizzabili sono: AF, BC, DE, HL ed SP ed i registri alternativi AF', BC', DE', HL', SP'.

— Il simbolo **r** è utilizzato per indicare genericamente uno qualunque dei registri A, B, C, D, E, H ed L e viene sempre specificato, per ogni istruzione, quali sono i registri interessati.

— Il simbolo **rr** è utilizzato per indicare genericamente una qualunque coppia di registri BC, DE, HL, SP, IX o IY e viene sempre specificato, per ogni istruzione, quali sono le coppie di registri interessate.

— Quando un termine è scritto fra parentesi, significa che viene considerato il contenuto della cella di memoria, il cui indirizzo è quello scritto nel termine entro parentesi. In pratica, scrivendo ad esempio (HL), significa che bisogna considerare il contenuto della cella di memoria il cui indirizzo è contenuto nel registro HL.

— Analogamente al caso appena visto, il simbolo (nn), significa il contenuto della cella di memoria di generico indirizzo nn.

— Il simbolo **n** indica un qualunque dato ad 8 bit, scritto in esadecimale.

— Il simbolo **b** indica una generica LABEL a cui il programma dovrà «saltare» se è verificata la condizione espressa nell'istruzione.

— Il simbolo **cc** indica una generica «condizione» che deve essere verificata per eseguire una operazione. Tali condizioni, fanno riferimento allo «stato» dei bit nel registro dei Flag, che troverete descritti subito di seguito e sono:

NZ = Non zero (Zero Flag = 0)

Z = Zero (Zero Flag = 1)

NC = Non Carry (Carry Flag = 0)

C = Carry (Carry Flag = 1)

PO = Parità Pari (Parity Flag = 0)

PE = Parità Dispari (Parity Flag = 1)

P = Segno Positivo (Sign Flag = 0)

M = Segno Negativo (Sign Flag = 1)

— Il registro F, detto registro dei **Flag** è un registro a 8 bit la cui struttura interna è molto particolare. Ogni singolo bit di questo registro infatti, contiene una informazione relativa allo **stato macchina** e cioè al verificarsi o meno, durante l'esecuzione di un programma, di certe condizioni particolari. Visto che queste informazioni sono molto importanti e vengono spesso utilizzate nei programmi, riportiamo per esteso il significato di ciascun bit di questo registro in modo da poter comprendere senza difficoltà il significato delle istruzioni che faranno uso dei Flag.

Il registro F è così costituito:

- Bit 0** = Riporto (Carry) — C
- Bit 1** = Sottrazione (Subtract) — N
- Bit 2** = Parità o Numero in eccesso (Parity/Overflow) — P/V
- Bit 3** = Non assegnato
- Bit 4** = Riporto decimale (Half carry) — H
- Bit 5** = Non assegnato
- Bit 6** = Risultato Zero (Zero) — Z
- Bit 7** = Segno del numero (Sign) — S

— **Riporto** = Quando un'operazione di somma o di sottrazione dà come risultato un numero maggiore della massima capacità del registro interessato (cioè maggiore di 255 o minore di 0 se l'operazione è su 8 bit o maggiore di 65535 o minore di 0 se è su 16 bit). In questo caso il bit del Carry, cioè il primo bit del registro dei flag, si porta automaticamente a livello logico 1, altrimenti si porta a 0.

— **Sottrazione** = Come abbiamo appena specificato, sia la somma che la sottrazione influenzano il bit del Carry, portandolo a 1 o a 0 a seconda che ci sia o meno un riporto. Tale riporto però, per ottenere un risultato corretto dell'operazione in corso, va trattato in modo diverso a seconda che si tratti di una somma o di una sottrazione per cui è necessario riconoscere da che tipo di operazione derivi il riporto ottenuto. Il bit N del registro dei flag indica appunto se l'ultima operazione eseguita è stata una somma o una sottrazione.

— **Parità/Overflow** = Questo bit è utilizzato in due modi differenti a seconda del tipo di operazione in corso. Quando si eseguono delle operazioni logiche (come ad esempio un AND o un OR) allora questo bit tiene conto della parità del risultato cioè:

1 = (Parità dispari) se il numero degli 1 nel risultato è dispari, cioè se sono presenti 1, 3, 5 o 7 bit a 1 nel risultato dell'operazione.

0 = (Parità pari) se il numero degli 1 nel risultato è pari, cioè se sono presenti 0, 2, 4, 6 o 8 bit a 1.

Se invece l'operazione in corso è una somma o una sottrazione di numeri con segno, allora questo bit segnala un errore di superamento della capacità, cioè se il risultato è maggiore di +127 o minore di -128.

— **Riporto decimale** = Questo bit viene utilizzato nelle operazioni decimali e segnala se è presente un riporto nelle somme o nelle sottrazioni. È quindi del tutto analogo al bit del Carry visto precedentemente per le operazioni su numeri binari.

— **Risultato Zero** = Questo bit vale 1 quando il risultato di una operazione logica o aritmetica è zero.

— **Segno del numero** = Nelle operazioni su numeri con segno, questo bit viene portato ad 1 se il risultato è un numero NEGATIVO.

Terminata questa prima parte introduttiva, possiamo iniziare la descrizione vera e propria delle istruzioni del linguaggio, per il computer Z80.

LE ISTRUZIONI DEL COMPUTER Z.80

ADC = (Added with Carry) — Addiziona con riporto

- ADC HL,rr Somma al contenuto del registro HL, il contenuto di una qualunque coppia di registri BC, DE, HL o SP.
- ADC A,r Somma al contenuto dell'accumulatore, il contenuto di uno qualunque dei registri A, B, C, D, E, H o L.
- ADC A,n Somma a 1 il contenuto dell'accumulatore, il dato n.
- ADC A,(rr) Somma all'accumulatore, il contenuto della cella di memoria il cui indirizzo è contenuto in una qualunque coppia di registri (HL), (IX+n) o (IY+n).

ADD = (Added) — Addiziona senza riporto

- ADD A,r Somma al contenuto dell'accumulatore, il contenuto di uno qualunque dei registri A, B, C, D, E, H o L.
- ADD A,n Somma all'accumulatore, il dato n.
- ADD A,(rr) Somma all'accumulatore, il contenuto della cella di memoria di indirizzo specificato in una qualunque coppia di registri (HL), (IX+n) o (IY+n).
- ADD HL,rr Somma al contenuto del registro HL, il contenuto di una qualsiasi coppia di registri BC, DE, HL o SP.
- ADD IX,rr Somma al contenuto del registro IX, il contenuto di una qualsiasi coppia di registri BC, DE, IX o SP.

ADD IY,rr Somma al contenuto del registro IY, il contenuto di una qualsiasi coppia di registri BC, DE, IY o SP.

AND = (Logical "AND") — AND logico

AND r AND logico fra il contenuto dell'accumulatore ed il contenuto di un qualunque registro A, B, C, D, E, H o L.
AND n AND logico fra il contenuto dell'accumulatore e il dato n.
AND (rr) AND logico fra il contenuto dell'accumulatore ed il contenuto della cella il cui indirizzo è contenuto in una qualunque coppia di registri (HL), (IX + n), (IY + n).

BIT = (Bit Test) — Test su di un Bit

BIT b,r Verifica se il bit b di uno qualunque dei registri A, B, C, D, E, H o L è a livello logico 1 o 0. Se è a 1 pone a 0 il bit Z del registro dei Flag, in caso contrario lo pone a 1.
 Esempio: BIT 4,D
 Se il quarto bit del registro D è a livello logico 1, il Flag Z si porta a 0, mentre se è a livello logico 0, il Flag Z si porta ad 1.
BIT b,(rr) Verifica se è a 1 o a 0 il bit b della cella di memoria, il cui indirizzo è specificato nella coppia di registri (HL), (IX + n) o (IY + n).

CALL = (Call Subroutine) — Chiama una subroutine

CALL cc,nn Chiama la subroutine di indirizzo nn, se la condizione cc è verificata.
CALL nn Chiama incondizionatamente la subroutine di indirizzo nn.

CCF = (Complement Carry Flag) — Complementa a 1 il Flag del Carry

CCF Se il bit del Carry nel registro dei Flag è a livello logico 1, con questa istruzione viene portato a zero e viceversa.

CP = (Compare) — Compara

CP n Compara l'accumulatore con il dato n.
CP r Compara l'accumulatore con il contenuto di uno qualunque dei registri A, B, C, D, E, H ed L.
CP (rr) Compara l'accumulatore con il contenuto della cella di memoria il cui indirizzo è specificato in una coppia qualunque di registri (HL), (IX + n) o (IY + n).

CPD = (Compare and Decrement) — Compara e decrementa

CPD Compara l'accumulatore con la cella di memoria il cui indirizzo è specificato nel registro HL, decrementa il valore di HL e di BC.

CPDR = (Compare, Decrement and Repeat) — Compara, decrementa e ripeti

CPDR Compara l'accumulatore con il contenuto della cella di memoria il cui indirizzo è specificato nel registro HL, decrementa il valore del registro HL e del registro BC. Questo ciclo di operazioni viene eseguito ripetitivamente e si arresta quando il contenuto del registro BC è 0 o quando il risultato della comparazione fra l'accumulatore e la cella di memoria non dà esito positivo.

CPI = (Compare and Increment) — Compara ed incrementa

CPI Compara l'accumulatore con il contenuto della cella di memoria il cui indirizzo è specificato nel registro HL, incrementa il valore del registro HL e decrementa il valore del registro BC.

CPIR = (Compare, Increment and Repeat) — Compara, incrementa e ripeti

CPIR Compara l'accumulatore con il contenuto della cella di memoria il cui indirizzo è specificato nel registro HL, incrementa il valore di HL e decrementa il valore del registro BC. Questo ciclo di operazioni viene eseguito ripetitivamente e si interrompe quando il valore del registro BC è 0 o quando 1 a comparazione fra l'accumulatore ed il contenuto della cella di memoria dà esito positivo.

CPL = (Complement) — Complementa a 1 l'accumulatore

CPL Con questa istruzione vengono portati a livello logico 1 tutti i bit dell'accumulatore che risultavano a livello logico 0 e viceversa.

DAA = (Decimal Adjustment Accumulator) — Aggiustamento decimale dell'accumulatore

DAA Quando vengono eseguite operazioni come la somma o la sottrazione, l'incremento o il decremento e la negazione fra numeri DECIMALI (cioè in codice BCD) queste operazioni, in determinate condizioni, possono dare origine ad errori di calcolo in quanto il computer è in grado di eseguire operazioni solo fra numeri BINARI (cioè in codice Binario).
L'istruzione DAA aggiusta automaticamente tali possibili errori, riportando nell'accumulatore l'esatto risultato dell'operazione in corso.

DEC = (Decrement) — Decrementa

DEC r Decrementa il contenuto di uno qualunque dei registri A, B, C, D, E, H o L.
DEC rr Decrementa il contenuto di una coppia qualunque dei registri BC, DE, HL, SP, IX o IY.
DEC (rr) Decrementa il contenuto della cella di memoria il cui indirizzo è specificato in uno dei registri (HL), (IX + n) o (IY + n).

DI = (Disable Interrupts) — Disabilita l'interruzione

DI Quando la CPU incontra questa istruzione, viene disabilitato il ciclo di interruzione ed il computer non risponde più alle richieste di interruzione provenienti dal mondo esterno.

DJNZ = (Decrement and Jump if Non Zero) — Decremento e salta fino a Non Zero

DJNZ e Decrementa il valore del registro B e salta alla LABEL «e» fino a quando il contenuto del registro B non vale 0.

EI = (Enable Interrupts) — Attiva l'interruzione

EI Quando la CPU incontra questa istruzione, viene attivato il funzionamento di interruzione ed il computer risponde ad ogni richiesta di interruzione proveniente dal mondo esterno.

EX = (Exchange) — Scambia

EX (SP),rr Scambia il contenuto di una coppia qualunque di registri HL, IX o IY, con il contenuto delle due celle di memoria, il primo indirizzo delle quali è specificato nel registro SP mentre la seconda è la cella seguente. Facciamo un esempio: se il registro SP contiene il valore 3456, il registro HL contiene il valore 8912, la cella di memoria di indirizzo 3456 contiene il dato 11 e la cella di memoria 3457 il dato 22, dopo l'istruzione EX (SP), HL, si avrà:

prima	dopo
SP = 3456	SP = 3456
HL = 8912	HL = 2211
3456 = 11	3456 = 12
3457 = 22	3457 = 89

EX DE,HL Scambia il contenuto del registro HL con il contenuto del registro DE.

EX AF,AF' Scambia il contenuto del registro AF' con il contenuto del registro AF.

EXX Scambia il contenuto dei registri BC, DE ed HI con il contenuto dei rispettivi registri BC', DE' ed HL'.

HALT = (CPU Halt) — Attesa di interruzione o di reset

HALT Questa istruzione blocca le operazioni del computer e mantiene la CPU in attesa di un successivo comando di interruzione o di un comando di reset.

IM = (Interrupt Mode) — Modo di interruzione

La CPU Z80 possiede 3 diversi tipi di interruzione, richiamabili con le istruzioni IM 0, IM 1 ed IM 2. Le procedure di funzionamento di queste tre istruzioni sono diverse fra loro per cui le spiegheremo separatamente.

IM 0 Con questa istruzione, la periferica che ha inviato al computer la richiesta di interruzione, può scrivere sul BUS il codice di una qualunque istruzione ed il computer, in risposta all'interruzione, eseguirà come prima operazione, tale istruzione.

- IM 1** Quando viene incontrata questa istruzione, la CPU predispone il ciclo di interruzione con un «restart» alla locazione 38 in esadecimale. Vale a dire che quando una periferica richiede una interruzione, il computer riprende il ciclo delle operazioni, eseguendo come prima operazione, l'istruzione contenuta nella cella di memoria di indirizzo 38 in esadecimale.
- IM 2** Questo «modo» di interruzione è il più complesso. Quando la CPU incontra questa istruzione, alla richiesta di interruzione di una periferica il computer costruisce un indirizzo a 16 bit, dei quali i primi 8 bit più significativi sono presi dal registro interno 1 mentre gli 8 bit meno significativi vengono forniti dalla periferica che ha richiesto l'interruzione. Fatto questo, la CPU compie un salto alla locazione di memoria così ottenuta ed inizia ad eseguire le operazioni descritte da questo punto della memoria in poi.

IN = (Input) — Carica un dato proveniente da una periferica esterna

- IN A,(n)** Questa istruzione comanda il caricamento di un dato proveniente da una periferica esterna. La procedura che il computer segue in questo caso è la seguente: il termine *n* entro parentesi, compreso fra 0 ed FF in esadecimale, (comprese quindi fra 0 e 255) viene posto sui primi 8 bit del bus degli indirizzi in modo da selezionare una fra le 256 possibili periferiche. Contemporaneamente il contenuto del registro A viene posto sui rimanenti 8 bit più significativi del bus degli indirizzi ed è perciò possibile espandere la capacità di indirizzamento delle unità periferiche fino a 64K. Viene quindi letto il dato presente nella periferica selezionata e caricato nell'accumulatore.
- IN r,(C)** Questa istruzione è del tutto analoga a quella appena descritta con la sola differenza che sulla parte bassa del bus degli indirizzi viene posto il contenuto del registro C, mentre sugli 8 bit più significativi viene ricopiato il contenuto del registro B. Il dato letto dalla periferica selezionata viene poi caricato in uno qualunque dei registri A, B, C, D, E, H o L.

INC = (Increment) — Incrementa

- INC r** Incrementa il contenuto di uno qualunque dei registri A, B, C, D, E, H, L, IX o IY.
- INC rr** Incrementa il contenuto di una coppia qualunque di registri BC, DE, HL o SP.
- INC (rr)** Incrementa il contenuto della cella di memoria il cui indirizzo è specificato in una coppia qualunque di registri (HL), (IX + n) o (IY + n).

IND = (Input and Decrement) — Carica un dato da una periferica e decrementa

- IND** Questa istruzione comanda il caricamento di un dato da una periferica, analogamente a quanto visto con l'istruzione IN.
- La procedura che il computer segue in questo caso è leggermente più complessa del caso precedente:
- Inizialmente, il contenuto del registro C viene posto sui primi 8 bit del bus degli indirizzi per selezionare una delle 256 possibili unità periferiche mentre il contenuto del registro B è posto sui rimanenti 8 bit più significativi del bus degli indirizzi. Il dato letto dalla porta periferica così selezionata viene scritto sul bus dei dati ed inviato alla CPU. Fatto questo, viene scritto sul bus degli indirizzi il contenuto del registro HL ed il dato precedentemente letto viene scritto nella cella di memoria di indirizzo corrispondente. Infine viene decrementato il contenuto del registro B e del registro HL.

INDR = (Input, Decrement and Repeat) — Carica un dato da una periferica, decrementa e ripeti

- INDR** Questa istruzione è del tutto simile alla IND vista precedentemente con la sola differenza che, al termine del ciclo di caricamento del dato, dopo aver decrementato il contenuto del registro HL e del registro B, se quest'ultimo non è zero, il ciclo di lettura si ripete nuovamente. Durante le operazioni di lettura del dato dalla periferica, è sempre attivo il riconoscimento di richiesta di interruzione per cui è possibile fermare lo svolgersi delle operazioni inviando alla CPU un segnale di richiesta di interruzione.

INI = (Input and Increment) — Carica un dato da una periferica ed incrementa

- INI** È una istruzione di lettura di un dato da una unità periferica ed è del tutto analoga all'istruzione IND. La sola differenza è che, al termine delle operazioni, viene decrementato il contenuto del registro B, mentre il contenuto del registro HL viene INCREMENTATO.

INIR = (Input, Increment and Repeat) — Carica un dato da una periferica

INIR È ancora una istruzione di lettura di un dato da una unità periferica ed è del tutto analoga all'istruzione INDR vista precedentemente. La sola differenza è che, al termine delle operazioni, viene INCREMENTATO il contenuto del registro HL mentre, analogamente a prima, viene decrementato il valore del registro B e ripetuto il ciclo se il suo contenuto non è uguale a zero.

JP = (Jump) — Salto assoluto

JP nn Salta incondizionatamente alla locazione di indirizzo nn.
JP (rr) Salta incondizionatamente alla locazione il cui indirizzo è specificato in una qualunque coppia di registri (HL), (IX) o (IY).
JP cc,nn Se la condizione cc è verificata, salta alla locazione di indirizzo nn.

JR = (Jump Relative) — Salto relativo

JR e Salta incondizionatamente alla linea di programma identificata dalla LABEL e.
JR cc,e Se la condizione cc è verificata, salta alla linea di programma identificata dalla LABEL e. La condizione cc da verificare, per questa istruzione, può essere:
 C — (Salta se Carry = 1)
 NC — (Salta se Carry = 0)
 Z — (Salta se Flag Zero = 1)
 NZ — (Salta se Flag Zero = 0)

LD = (Load) — Carica

LD r,n Carica il dato n, in uno qualunque dei registri A, B, C, D, E, H o L.
LD rr,nn Carica il dato a 16 bit nn, in una coppia qualunque di registri BC, DE, HL, SP, IX o IY.
LD rr,(nn) Carica in una coppia qualunque di registri BC, DE, HL, SP, IX o IY, il contenuto delle due celle di memoria, la prima delle quali ha indirizzo nn e la seconda ha indirizzo nn + 1.
LD (rr),n Carica il dato n, nella cella di memoria il cui indirizzo è specificato in una coppia qualunque di registri (HL), (IX + n) o (IY + n).
LD r,(rr) Carica in uno qualunque dei registri A, B, C, D, E, H o L, il contenuto della cella di memoria, il cui indirizzo è specificato nei registri (HL), (IX + n) o (IY + n).
LD (nn),rr Carica il contenuto di una coppia qualunque di registri BC, DE, HL, SP, IX o IY, nelle due celle di memoria la prima delle quali ha indirizzo nn e la seconda ha indirizzo nn + 1.
LD A,(nn) Carica nell'accumulatore, il dato presente nella cella di memoria di indirizzo nn.
LD A,(rr) Carica nell'accumulatore, il contenuto della cella di memoria il cui indirizzo è specificato in una coppia qualunque di registri (BC), (DE) o (HL).
LD (nn),A Carica il contenuto dell'accumulatore, nella cella di memoria di indirizzo nn.
LD (rr),A Carica l'accumulatore, nella cella di memoria il cui indirizzo è specificato nei registri (BC), (DE), (HL), (IX + n) o (IY + n).
LD A,I Carica nell'accumulatore, il contenuto del registro I.
LD A,R Carica nell'accumulatore, il contenuto del registro R.
LD R,A Carica nel registro R, il contenuto dell'accumulatore.
LD I,A Carica nel registro I, il contenuto dell'accumulatore.
LD r,r' Carica il contenuto di uno qualunque dei registri A', B', C', D', E, H' o L', in uno qualunque dei registri A, B, C, D, E, H o L.
LD SP,rr Carica nel registro SP (Stack-Pointer) il contenuto di una coppia qualunque di registri HL, IX o IY.

LDD = (Load and Decrement) — Carica e decrementa

LDD Carica nella locazione di memoria il cui indirizzo è specificato nel registro DE, il contenuto della cella di memoria il cui indirizzo è specificato nel registro HI, quindi decrementa il contenuto del registro DE, del registro HL e del registro BC.

LDDR = (Load, Decrement and Repeat) = Carica, decrementa e ripeti

LDDR L'istruzione è del tutto analoga alla precedente LDD solo che ora, al termine delle operazioni, viene ripetuto tutto il ciclo fino a quando il registro BC non vale 0.

LDI = (Load and Increment) — Carica ed incrementa

LDI Carica nella locazione di memoria il cui indirizzo è specificato nel registro DE, il contenuto della cella di memoria il cui indirizzo è specificato nel registro HI, quindi incrementa il contenuto del registro DE e del registro HL e decrementa il contenuto del registro BC.

LDIR = (Load, Increment and Repeat) — Carica, incrementa e ripeti

LDIR L'istruzione è del tutto analoga alla precedente LDI solo che ora, al termine delle operazioni, viene ripetuto tutto il ciclo fino a quando il registro BC non vale 0.

NEG = (Negate) — Negazione

NEG Questa istruzione esegue il complemento a 2 del dato contenuto nell'accumulatore e pone nell'accumulatore il risultato ottenuto.

NOP = (No Operation) — Istruzione nulla

NOP Quando la CPU incontra questa istruzione, viene eseguito un semplice ciclo di operazioni che occupa 4 cicli macchina (con un clock a 2 MHz vengono spesi complessivamente 2 microsecondi) e non viene alterato nessun parametro interno od esterno.

OR = (Logical OR) — Operatore logico OR

OR n Esegue una operazione di OR logico fra l'accumulatore ed il dato n. Il risultato viene posto nell'accumulatore.

OR r Esegue una operazione di OR logico fra il dato contenuto nell'accumulatore ed il dato presente in uno qualunque dei registri A, B, C, D, E, H o L. Il risultato viene posto nell'accumulatore.

OR (rr) Esegue una operazione di OR logico fra il dato contenuto nell'accumulatore ed il dato contenuto nella cella di memoria il cui indirizzo è specificato in una coppia qualunque di registri (HL), (IX + n) o (IY + n). Il risultato viene posto nell'accumulatore.

OTDR = (Out, Decrement and Repeat) — Carica un dato su di una porta periferica, decrementa e ripeti

OTDR Con questa istruzione si invia un dato ad una unità periferica esterna. Il ciclo di operazioni per eseguire questa istruzione è il seguente:
Inizialmente viene posto sul bus degli indirizzi, il contenuto del registro HL e viene così selezionata una cella di memoria. Il contenuto di questa cella viene temporaneamente memorizzato all'interno della CPU quindi viene decrementato il valore contenuto nel registro B ed il contenuto del registro C è posto sui primi 8 bit del bus degli indirizzi, per selezionare l'unità periferica di uscita. Contemporaneamente, il contenuto del registro B viene posto sui rimanenti 8 bit più significativi del bus degli indirizzi. Successivamente, sulla porta periferica così selezionata, viene scritto il dato letto precedentemente, il registro HL viene decrementato e, se il valore del registro B non è 0 il ciclo viene interamente ripetuto.
Durante l'esecuzione di tutte le operazioni è sempre attivo il riconoscimento di richiesta di interruzione per cui è possibile bloccare lo svolgersi delle operazioni inviando alla CPU un segnale di richiesta di interruzione.

OTIR = (Out, Increment and Repeat) — Carica un dato su di una porta periferica, incrementa e ripeti

OTIR Questa istruzione è del tutto simile all'istruzione OTDR appena descritta con la sola differenza che ora il contenuto del registro HL non viene decrementato, bensì incrementato.

OUT = (Out) — Carica un dato su di una porta periferica

OUT (C),r Esegue il caricamento di un dato su di una unità periferica esterna. La sequenza delle operazioni che vengono svolte per questa istruzione è la seguente:

Il contenuto del registro C è posto sui primi 8 bit del bus degli indirizzi per selezionare l'unità periferica mentre il contenuto del registro B è posto sui rimanenti 8 bit più significativi. Successivamente viene posto sul bus dei dati il contenuto di uno qualunque dei registri A, B, C, D, E, H o L e questo dato viene scritto nella porta periferica selezionata.

OUT (n),A Esegue il caricamento di un dato, su di una unità periferica esterna. La sequenza delle operazioni che vengono svolte per questa istruzione è la seguente:

Il dato n viene posto sui primi 8 bit del bus degli indirizzi per selezionare la porta periferica mentre il contenuto dell'accumulatore viene posto sui rimanenti 8 bit più significativi del bus

degli indirizzi. Successivamente il dato presente nell'accumulatore viene posto sul bus dei dati e scritto all'interno dell'unità periferica precedentemente selezionata.

OUTD = (Out and Decrement) — Carica un dato su di una porta periferica e decrementa

OUTD È una istruzione di caricamento di un dato su di una unità periferica esterna ed il suo funzionamento è del tutto analogo a quello descritto per l'istruzione OTDR. La sola differenza, in questo caso, è che l'istruzione viene eseguita una sola volta, qualunque sia il contenuto del registro B.

OUTI = (Out and Increment) — Carica un dato su di una porta periferica ed incrementa

OUTI È una istruzione di caricamento di un dato su di una unità periferica esterna ed il suo funzionamento è del tutto analogo a quello descritto per l'istruzione OTDIR. La sola differenza, in questo caso, è che l'istruzione viene eseguita una sola volta, qualunque sia il contenuto del registro B.

POP = (Popped) — Carica l'ultima locazione dello Stack

POP rr Carica in uno qualunque dei registri AF, BC, DE, HL, IX o IY, l'indirizzo contenuto nello Stack, a cui punta il registro dello Stack-Pointer, ed incrementa di 2 il contenuto dello Stack-Pointer.

PUSH = (Pushed) — Carica nello Stack

PUSH rr Carica nello Stack, nella cella di memoria a cui punta il registro dello Stack-Pointer, il contenuto di uno qualunque dei registri AF, BC, DE, HL, IX o IY.

RES = (Reset) — Poni a Zero un Bit

RES b,r Poni a 0 il bit b (da 0 a 7) di uno qualunque dei registri B, C, D, E, H o L.

RES b,(rr) Poni a 0 il bit b (da 0 a 7) della cella di memoria il cui indirizzo è contenuto in uno qualunque dei registri (HL), (IX + n) o (IY + n).

RET = (Return) — Ritorna da una Subroutine

RET Ritorna dalla subroutine incondizionatamente.

RET cc Ritorna dalla subroutine, se la condizione «cc» è verificata.

RETI = (Return from Interrupt) — Ritorna da una interruzione

RETI Questa istruzione si utilizza solo alla fine di una subroutine che gestisce le operazioni per una richiesta di interruzione. Con RETI si tiene conto anche della priorità della richiesta di interruzione e perciò vengono automaticamente «nidificate» le richieste di interruzione incontrate ed eseguite nella loro esatta priorità.

RETN = (Return from Non maskable interrupt) — Ritorna da una interruzione non mascherabile

RETN Questa istruzione si utilizza solo alla fine di una subroutine che gestisce le operazioni per una richiesta di interruzione non mascherabile.

RL = (Rotate Left) — Ruota a sinistra

RL r Ruota verso sinistra con il carry, il contenuto di uno qualunque dei registri A, B, C, D, E, H o L. Il funzionamento di questa istruzione è molto semplice e consiste, in pratica, nello spostare di un posto verso sinistra, il contenuto di ciascun bit. L'ultimo bit è posto nel flag del Carry mentre il flag Carry è posto nel primo bit. Ad esempio, supponendo che il registro D contenga il dato, scritto in binario:

10001111

e che il flag del carry contenga 0. Scrivendo l'istruzione RL D, si otterrà che all'interno del registro D sarà presente il dato:

011110

ed il flag del carry conterrà 1.

RL (rr) Ruota verso sinistra con il carry, il contenuto della cella di memoria, il cui indirizzo è contenuto in uno qualunque dei registri (HL), (IX + n) o (IY + n).

RLA = (Rotate Left Accumulator) — Ruota verso sinistra l'accumulatore

RLA È un'istruzione del tutto analoga alla precedente, ma esegue la rotazione solo sull'accumulatore ed il tempo di esecuzione delle operazioni è più breve.

RLC = (Rotate Left Circular) = Ruota a sinistra circolarmente

RLC r Ruota verso sinistra il contenuto di uno qualunque dei registri A, B, C, D, E, H o L, senza carry. L'ultimo bit viene ricopiato sia nel primo bit che nel carry.

RLC (rr) Ruota verso sinistra il contenuto della cella di memoria il cui indirizzo è contenuto in uno qualunque dei registri (HL), (IX + n) o (IY + n), senza carry. Il contenuto dell'ultimo bit, viene ricopiato sia nel primo bit che nel carry.

RLCA = (Rotate Left Circular Accumulator) — Ruota a sinistra circolarmente l'accumulatore

RLCA È una istruzione del tutto analoga all'istruzione RLC vista precedentemente. La sola differenza è che RLCA agisce solo sull'accumulatore ed il tempo di esecuzione dell'istruzione è più breve.

RLD = (Rotate Left Digit) — Ruota a sinistra 4 bit alla volta

RLD È una istruzione complessa che esegue lo spostamento di quattro bit alla volta fra l'accumulatore e la cella di memoria il cui indirizzo è contenuto nel registro HL. Le operazioni comandate da questa istruzione si svolgono nel modo seguente:

I primi 4 bit dell'accumulatore vengono ricopiati negli ultimi 4 bit dell'accumulatore.

Gli ultimi 4 bit dell'accumulatore vengono ricopiati nei primi 4 bit della cella di memoria il cui indirizzo è contenuto nel registro HL.

Il contenuto dei primi 4 bit della cella di memoria il cui indirizzo è specificato nel registro HL, vengono ricopiati nei primi 4 bit dell'accumulatore.

RR = (Rotate Right) — Ruota verso destra

RR r Ruota verso destra con il carry, il contenuto di uno qualunque di registri A, B, C, D, E, H o L. Il primo bit è ricopiato nel flag del carry mentre il flag del carry è posto nell'ultimo bit del registro.

RR (rr) Ruota verso destra con il carry, il contenuto della cella di memoria, il cui indirizzo è contenuto in uno qualunque dei registri (HL), (IX + n) o (IY + n).

RRA = (Rotate Right Accumulator) — Ruota verso destra

RRA È un'istruzione del tutto analoga alla precedente ma esegue la rotazione solo sull'accumulatore. Il tempo di esecuzione delle operazioni relative, è però più breve.

RRC = (Rotate Right Circular) — Ruota a destra circolarmente

RRC r Ruota verso destra il contenuto di un qualunque registro A, C, D, E, H o L, senza carry. Il contenuto del primo bit è ricopiato sia nell'ultimo bit che nel flag del carry.

RRCA = (Rotate Right Circular Accumulator) — Ruota verso destra circolarmente, l'accumulatore

RRCA È un'istruzione del tutto analoga alla precedente ma esegue la rotazione solo sul registro A. La differenza consiste solo nel minor tempo di esecuzione dell'istruzione.

RRD = (Rotate Right Digit) — Ruota verso destra 4 bit alla volta

RRD r È un'istruzione complessa di rotazione verso destra, di 4 bit alla volta, fra l'accumulatore ed il contenuto della cella di memoria il cui indirizzo è specificato nel registro HL.

Le operazioni comandate da questa istruzione, vengono eseguite nel seguente modo:

I primi 4 bit dell'accumulatore, vengono ricopiati negli ultimi 4 bit della cella di memoria il cui indirizzo è specificato nel registro HL.

Gli ultimi 4 bit della cella di memoria il cui indirizzo è specificato nel registro HL, vengono ricopiati nei primi 4 bit della stessa cella.

I primi quattro bit della cella di memoria il cui indirizzo è specificato nel registro HL, vengono ricopiati nei primi 4 bit dell'accumulatore.

RST = (Restart) — Riparti

RST P È una istruzione che comanda la reesecuzione di una subroutine di partenza e di inizializzazione del computer. Tale subroutine può essere scritta, come prima istruzione di partenza, nella cella di memoria specificata dal termine p e può essere, in esadecimale, uno qualunque dei seguenti numeri:

00 - 08 - 10 - 18 - 20 - 28 - 30 - 38

Nei primi 8 bit del program counter, quando viene incontrata tale istruzione, viene sempre posto il numero 00 in esadecimale.

SBC = (Subtract with Carry) — Sottrai con riporto

SBC A,r Sottrai dall'accumulatore, il contenuto di un qualunque registro A, B, C, D, E, H o L ed il bit del flag Carry.

SBC HL,rr Sottrai dal contenuto del registro HL, il contenuto di uno qualunque dei registri BC, DE, HL o SP ed il bit del flag Carry.

SCF = (Set Carry Flag) — Setta il Carry

SCF Poni ad 1 il bit del Carry, nel registro F dei Flag.

SET = (Set) — Poni a 1

SET b,r Poni a 1 il bit b (da 0 a 7) di uno qualunque dei registri A, B, C, D, E, H o L.

SET b,(rr) Poni ad 1 il bit b (da 0 a 7) della cella di memoria il cui indirizzo è specificato in uno qualunque dei registri (HL), (IX + n) o (IY + n).

SLA = (Shift Left Arithmetic) — Sposta a sinistra di una posizione, con regola aritmetica

SLA r Sposta verso sinistra tutti i bit di un qualunque registro A, B, C, D, E, H o L. Poni poi 0 nel primo bit del registro interessato e ricopia nel carry l'ottavo bit dello stesso registro.

SLA (rr) Sposta verso sinistra tutti i bit della cella di memoria il cui indirizzo è specificato in uno qualunque dei registri (HL), (IX + n) o (IY + n). Poni a 0 il primo bit di tale cella di memoria e ricopia nel carry l'ottavo bit della cella stessa.

SRA = (Shift Right Arithmetic) — Sposta a destra di una posizione, con regola aritmetica

SRA r Sposta verso destra di una posizione, tutti i bit di un qualunque registro A, B, C, D, E, H o L. Ricopia il primo bit del registro interessato nel carry e lascia inalterato il valore dell'ultimo bit dello stesso registro.

SRA (rr) Sposta verso destra di una posizione, tutti i bit della cella di memoria il cui indirizzo è specificato in uno dei registri (HL), (IX + n) o (IY + n). Ricopia il primo bit della cella di memoria nel carry e lascia inalterato il valore dell'ultimo bit della cella stessa.

SRL = (Shift Right Logical) — Sposta verso destra di una posizione, con regola logica

SRL r Sposta verso destra di una posizione, tutti i bit di un qualunque registro A, B, C, D, E, H o L. Poni a 0 l'ultimo bit del registro interessato e ricopia nel carry il primo bit dello stesso registro.

SRL (rr) Sposta verso destra di una posizione, tutti i bit della cella di memoria il cui indirizzo è specificato in uno qualunque dei registri (HL), (IX + n) o (IY + n). Poni a 0 l'ultimo bit di tale cella di memoria e ricopia nel carry, il primo bit della stessa cella.

SUB = (Subtract) — Sottrai senza riporto

SUB r Sottrai dall'accumulatore, il valore contenuto in un qualunque registro A, B, C, D, E, H o L e poni il risultato nell'accumulatore.

SUB (rr) Sottrai dall'accumulatore il contenuto della cella di memoria il cui indirizzo è specificato in uno qualunque dei registri (HL), (IX + n) o (IY + n). Il risultato lo poni nell'accumulatore.

XOR = (Exclusive OR) — OR esclusivo

XOR r Esegui l'operazione logica di OR esclusivo, fra il contenuto dell'accumulatore ed il contenuto di un qualunque registro A, B, C, D, E, H o L. Il risultato è posto nell'accumulatore.

XOR (rr) Esegui l'operazione logica di OR esclusivo fra l'accumulatore ed il contenuto della cella di

memoria il cui indirizzo è specificato in uno qualunque dei registri (HL), (IX + n) o (IY + n). Il risultato è posto nuovamente nell'accumulatore.

Oltre alle istruzioni ora descritte, ne esistono altre (6 in tutto), attraverso le quali vengono definite le più importanti caratteristiche del programma e cioè l'indirizzo della cella di memoria a partire dalla quale verrà inserito in memoria il programma, la locazione di inizio del programma stesso, la fine e la definizione di aree di memoria riservate.

Vediamo quindi, per completare questo paragrafo, la descrizione di queste ultime istruzioni.

ORG (ORIGIN)

Definisce la cella di inizio da cui cominciare ad inserire in memoria il programma. Questa istruzione va sempre scritta all'inizio del programma e perciò, supponendo che la linea 100 contenga la prima istruzione, se scriviamo:

```
100 ORG 7000
```

quando il programma verrà assemblato, il codice oggetto corrispondente verrà posto in memoria a partire dalla cella di memoria 7000 in poi.

Nota bene: è possibile scrivere contemporaneamente più programmi, uno di seguito all'altro, e per ognuno di questi si dovrà specificare, con una istruzione di **ORG**, la cella di inizio. Alla fine di ogni programma si deve sempre scrivere l'istruzione **END** come vedremo di seguito.

END (END OF ASSEMBLY)

È un'istruzione di fine programma e deve essere sempre senta al termine di ogni listato. Assieme a questa istruzione, può essere specificata anche una «label» corrispondente all'istruzione di inizio del programma, ad esempio:

```
650 END START
```

In questo modo, se l'esecuzione del programma deve iniziare ad esempio dalla linea 300 e a tale linea è assegnata l'etichetta (cioè la «label») **START**, quando il computer eseguirà il programma, la prima istruzione che verrà svolta sarà quella scritta alla linea 300.

Se non viene specificata alcuna label di inizio programma, il computer assume automaticamente, come locazione di inizio, la cella 0000.

DEFB (DEFINIT BYTE)

Definisce, all'interno di un programma, una cella di memoria nella quale memorizzare un dato. Tale dato può essere un numero o un carattere in ASCII ed in quest'ultimo caso, il termine andrà scritto fra virgolette. Ad esempio:

```
230 DEFB 0D  
320 DEFB 'F'
```

Così facendo, nella cella di memoria nella quale verrà posta la linea di programma 230, verrà scritto il numero (in esadecimale) 0D e, analogamente per la linea di programma 320, verrà posto in memoria il corrispondente codice ASCII della lettera F, cioè 44.

DEFW (DEFINIT WORD)

Definisce, analogamente all'istruzione precedente, un'area di memoria all'interno della quale memorizzare dati o caratteri necessari alla stesura del programma. In questo caso, vengono dedicati 2 byte per il dato da memorizzare, quindi potremo scrivere:

```
180 DEFW 0DF4  
350 DEFW 'NE'
```

e nelle celle di memoria alle quali corrisponderanno queste linee di programma, troveremo, uno di seguito all'altro, i due numeri 0D ed F4 oppure il corrispondente codice ASCII delle due lettere N ed E, cioè 4E e 45.

DEFM (DEFINIT MEMORY)

Definisce un'area di memoria riservata ai dati da memorizzare. La lunghezza di questa area di memoria è definita automaticamente dal numero dei dati scritti nell'istruzione e cioè:

```
DEFM 510F1ACD32  
DEFM '3FINE9'
```

occuperanno rispettivamente 5 celle di memoria (per i dati numerici 51, 0F, 1A, CD e 32) e 6 celle per il corrispondente codice ASCII dei caratteri 3, F, I, N, E e 9.

EQU (EQUATE)

Con questa istruzione viene definita una periferica o un'area di memoria attraverso una label, anziché attraverso il suo indirizzo assoluto.

In questo modo, ad esempio, volendo utilizzare come memoria temporanea di un dato del programma la cella di indirizzo 40FF, è possibile assegnare a tale cella un'etichetta, ad esempio BUFFER, scrivendo all'inizio del programma la seguente istruzione:

```
200 BUFFER EQU 40FFH
```

NOTA: La lettera H posta in coda all'indirizzo, specifica che tale indirizzo è stato scritto in notazione ESA-DECIMALE. Se il primo termine di tale numero è una lettera (come accadrebbe volendo utilizzare ad esempio, la cella di indirizzo B07E) si deve necessariamente anteporre al numero uno 0, cioè scrivere:

```
200 BUFFER EQU 0B07EH
```

In questo modo, alla label BUFFER, verrà assegnata la cella di memoria di indirizzo esadecimale B07E.

Come scrivere i programmi con l'EDITOR/ASSEMBLER

La nuova versione dell'EDITOR/ASSEMBLER per il computer Z80 N.E., è predisposta per la scheda grafica LX.529 e sulla scheda Floppy-Disk (LX.390) dovrà essere presente la EPROM 2390.

La configurazione di memoria può essere invece indifferentemente da 16K fino a 56K, a seconda delle schede di espansione inserite.

A questo punto possiamo passare definitivamente all'uso pratico del linguaggio e perciò iniziamo, come sempre, accendendo il computer ed inserendo il dischetto contenente il linguaggio nel drive 0 del computer.

Digitando due volte consecutivamente il tasto di ESCAPE, il sistema operativo verrà caricato all'interno del computer e sul video apparirà la scritta:

```
DOS READY
```

Per lanciare il programma dell'interprete EDITOR/ASSEMBLER si dovrà digitare:

```
EDTASM
```

seguito da RETURN e, dopo alcuni secondi, sul video apparirà la scritta:

```
C.H. EDITOR/ASSEMBLER 1.1
```

*

L'asterisco che comparirà nella linea successiva, starà ad indicare che l'interprete è a vostra disposizione ed è in attesa di un comando.

Abbiamo appena visto, dettagliatamente, le varie istruzioni che possono essere inserite nelle linee di programma ed il loro significato per esteso; non rimane quindi che analizzare il «modo» di scrittura, cioè le pseudoistruzioni che il linguaggio possiede, necessarie a scrivere, correggere o verificare un programma.

COMANDI DIRETTI DEL PROGRAMMA EDITOR. Le PSEUDO-ISTRUZIONI

I (Insert)

Inserisce le istruzioni nella linea di programma. Questa prima istruzione consentirà quindi di inserire nella memoria del computer, il programma vero e proprio. Digitando I e Return, sul video apparirà il numero 100 ed il cursore si posizionerà accanto a tale numero.

Questa è perciò la prima linea di programma nella quale sarà possibile inserire le istruzioni. Digitando ancora il tasto di Return, sul video apparirà il numero 110, scritto sotto al 100 ed il cursore si posizionerà accanto a quest'ultimo numero. Così facendo, verrà «saltata» la linea 100 ed il programma inizierà perciò alla linea 110.

L'incremento di 10, fra una linea e l'altra, ed il numero 100 iniziale, sono stabiliti inizialmente dal linguaggio. Per modificarli sarà sufficiente scrivere, insieme alla lettera I, due numeri, separati da una virgola, che corrisponderanno alla prima linea di programma e all'incremento fra le linee.

Quindi, scrivendo ad esempio:

I 50,2

seguito da Return, sul video apparirà il numero 50 come prima linea di programma e, ogni volta che si digiterà Return, appariranno via via le linee 52, 54, 56 ecc.

Inizialmente, quando non è presente alcuna istruzione di programma all'interno della memoria del computer, ogni istruzione verrà scritta di seguito all'altra, separata ciascuna da un Return ed apparirà sempre, automaticamente, il nuovo numero di linea nel quale inserire la nuova istruzione.

Terminata la stesura del programma, per uscire dall'inserzione sarà sufficiente digitare BREAK o CTRL A.

A questo punto saranno presenti, all'interno della memoria del computer, tutte le linee di programma scritte. Digitando nuovamente I e Return per inserire altre istruzioni nel programma già esistente, il computer si posizionerà sulla linea seguente all'ultima scritta e riprenderà l'inserzione delle istruzioni analogamente a prima. Potrà capitare, però, di dovere inserire delle istruzioni all'interno del programma e non solamente alla fine del programma stesso. In questo caso, supponendo di avere scritto il programma a partire dalla linea 100 con incremento di 10, volendo inserire una istruzione fra la linea 130 e la linea 140, si dovrà scrivere, ad esempio:

I 135

e Return. Terminato di scrivere l'istruzione, digitando Return, sul video apparirà la scritta:

NO ROOM BETWEEN LINE

Questo messaggio significa, tradotto dall'inglese, che non c'è più spazio fra linee del programma ed infatti, incrementando di 10 il numero 135, non ci si troverebbe più fra la linea 130 e la 140, bensì fra la 140 e la 150.

Nel caso in cui si vogliano scrivere più istruzioni fra queste due linee, si dovrà dare un incremento inferiore a quello presente fra le istruzioni stesse e cioè digitare, ad esempio:

I 132,2

Così facendo si potranno inserire 4 linee di programma. Vedremo subito di seguito, altri modi di modificare le linee di programma, cancellandole (con il comando R), correggendole (con il comando E) o rinumerando tutto il programma (con il comando N).

E (EDIT)

Richiama una linea di programma per correggerla o modificarla. La funzione di Edit è del tutto analoga, come funzionamento, a quella presente nel Basic perciò rimandiamo la descrizione del funzionamento di questo comando al primo volume del Basic (vedi da pag. 66 a pag. 73), senza dilungarci in inutili ripetizioni.

N (RENUM)

Rinumerava le linee di programma. Questo comando consentirà di riordinare le linee di programma, anche parzialmente, seguendo una numerazione ed un incremento preordinato. Il funzionamento di questa istruzione è molto semplice. Digitando ad esempio:

N 300,20

si otterrà la rinumerazione di tutte le linee di programma, dalla 300 in poi, con incremento, fra linea e linea, di 20.

I due numeri scritti accanto ad N, cioè 300 e 20, potranno anche essere omessi e perciò sarà possibile digitare semplicemente:

N

e Return. In questo caso il computer assumerà, come valore da assegnare alla linea di partenza, la prima linea del programma e, come valore dell'incremento, l'ultimo valore di incremento impostato.

Infine è possibile scrivere:

N 200

ed in questo caso si otterrà la rinumerazione del programma a partire dalla linea 200, assumendo, come incremento fra le linee, l'ultimo valore di incremento impostato.

R (REWRITE)

Riscrivere una linea di programma già esistente. Digitando perciò:

R 250

e Return verrà automaticamente cancellato il precedente contenuto della linea 250 ed il cursore risulterà posizionato accanto a tale linea per inserirvi la nuova istruzione.

Se la linea 250 non era stata ancora scritta, il computer segnalerà l'errore con la scritta:

NO SUCH LINE

D (DELETE)

Cancella le linee di programma. Scrivendo semplicemente:

D 120

e Return, verrà cancellata la sola linea 120. Se si vorranno cancellare più linee contemporaneamente, ad esempio dalla linea 200 alla linea 300, si dovrà scrivere:

D 200:300

Anche in questo caso, se si comanderà la cancellazione di una linea inesistente, il computer segnalerà l'errore con la scritta:

NO SUCH LINE.

P (PRINT)

Stampa sul video le linee di programma. Scrivendo semplicemente l'istruzione:

P

e Return, si otterrà la visualizzazione delle prime 22 linee di programma e ad ogni successivo comando P, verranno visualizzate le 22 linee seguenti e così via fino alla fine del programma.

Una volta digitato P e Return ed ottenuta la stampa delle linee volute, premendo il tasto del LINE FEED verrà visualizzata la riga successiva, mentre con il tasto [verrà visualizzata la linea precedente.

Altre possibilità di scrittura di questo comando sono:

P 100

per ottenere la stampa su video della sola linea 100.

P 150:280

per ottenere la stampa su video delle linee di programma dalla 150 alla 280.

P 100:*

per ottenere la stampa su video delle linee di programma dalla 100 fino alla fine.

P.:500

per ottenere la stampa su video dall'ultima linea di programma editata alla 500.

P.:*

per ottenere la stampa su video dall'ultima linea di programma editata fino alla fine.

H (LPRINT)

Riporta sulla stampante le linee di programma. È una istruzione di stampa e segue le stesse regole di sintassi appena descritte per l'istruzione P.

T (TESTO)

Riporta su stampante il programma senza scrivere, però, i numeri di riga. Le regole di scrittura sono ancora identiche al caso precedente.

F (FIND)

Cerca una qualunque parola all'interno del programma e stampa sul video la linea che la contiene. La parola da ricercare andrà scritta accanto alla lettera F, senza lasciare alcuno spazio, perciò, volendo ad esempio ricercare all'interno di un programma la parola FINE o LD, si scriverà:

FFINE

o

FLD

seguito dal testo di Return. Nel caso non venga trovata la parola, il computer segnalerà:

STRING NOT FOUND

W (WRITE)

Registra un programma su disco. Il nome del programma non potrà essere più lungo di 6 caratteri e non si potranno inserire al suo interno dei segni di punteggiatura. L'unico simbolo che potrà essere utilizzato è «/» dopo il quale si potranno inserire altri 3 caratteri, che vengono utilizzati per specificare il «tipo» di file che verrà registrato (ad esempio /TXT per un file di testo, /ASM per un file in assembler, ecc.). Volendo salvare un programma su disco, si dovrà perciò digitare, ad esempio:

W PROVA/TXT

seguito, al solito, dal comando di Return. Se sul disco non è presente un file con lo stesso nome, sul video apparirà la scritta:

FILE NON-EXISTENT

REPLAY «C» TO CREATE IT

Digitando la lettera C il computer eseguirà l'istruzione di caricamento, creando sul disco un file con il nome PROVA/TXT.

Nel caso in cui invece nel disco sia già presente un file con lo stesso nome, sul video apparirà la scritta:

FILE $\backslash$ ALREADY EXISTS. USE IT???

Rispondendo di sì (digitando cioè la lettera Y), sul disco verrà cancellato il file precedente ed al suo posto verrà scritto il nuovo file, mentre rispondendo di no (cioè digitando il tasto N), il computer rilascerà il comando di scrittura su disco e non eseguirà alcuna operazione.

Chi possiede più di un Drive, potrà poi registrare un programma indifferentemente sul primo o sul secondo Drive, semplicemente aggiungendo, in coda all'istruzione di Write, il simbolo «:» seguito dal numero di Drive su cui si vorrà scrivere il File. Per registrare quindi il programma PROVA/TXT sul secondo Drive (cioè sul drive 1), si dovrà scrivere:

W PROVA/TXT :1

Nota: se non verrà specificato nulla, il computer assumerà sempre come disco corrente il Drive 0.

L (LOAD)

Legge un programma caricato su disco e lo pone in memoria. Ad esempio, avendo salvato su disco un programma con il nome PROVA/TXT, per ricaricarlo in memoria si dovrà digitare:

L PROVA/TXT

seguito da Return. Se in memoria è presente un altro programma, il computer chiederà se il programma da leggere dal disco dovrà essere scritto in memoria di seguito a quello già presente, oppure se dovrà cancellare ciò che è presente in memoria per porvi solamente il nuovo programma. La domanda sarà posta nel modo seguente:

TEXT IN BUFFER
ARE YOU CONCATENATING???

Rispondendo di sì (cioè digitando Y) il nuovo programma verrà scritto in memoria di seguito a quello esistente, mentre rispondendo di no (cioè digitando N) il contenuto corrente della memoria verrà cancellato. Chi possiede 2 Drive ed ha registrato un programma sul secondo Drive, per leggere quest'ultimo file dovrà digitare, analogamente a quanto visto nell'istruzione di Write:

L PROVA/TXT :1

Anche in questo caso, se non verrà specificato nulla, il drive 0 sarà assunto come disco corrente.

B (BACK TO DOS)

Ritorna al DOS, abbandonando il linguaggio EDITOR/ASSEMBLER.

COMANDI DIRETTI DEL PROGRAMMA ASSEMBLER. L'ASSEMBLATORE.

Il programma Assembler è il secondo nucleo dell'interprete Editor/Assembler, ed il suo compito è quello di creare, dal programma scritto precedentemente, (detto programma SORGENTE) un programma ad esso equivalente, scritto direttamente in linguaggio macchina.

Questo programma, chiamato programma OGGETTO, è quello che poi verrà eseguito nella realtà dal computer e può essere salvato su disco o come file tipo /CMD (= Comando) per essere subito eseguito da DOS, oppure come qualunque altro tipo di file ad /CIM (= Core Image), oppure /TXT (= Text) ecc.

In pratica quindi, l'uso dell'Assemblatore è l'ultima operazione che si deve eseguire al termine della stesura di un listato ed è l'operazione che crea l'effettivo programma da eseguire come risultato del nostro lavoro.

Non esiste quindi una analogia diretta con le operazioni compiute da qualunque altro linguaggio, come ad esempio il Basic, infatti per portare un esempio pratico, mentre nel Basic, al termine della stesura del

programa, è sufficiente digitare Run per ottenere l'esecuzione immediata delle operazioni, nel linguaggio Editor/Assembler non esiste questo passaggio diretto.

Vediamo, in pratica, cosa significa tutto ciò.

Supponiamo di essere in Basic e di scrivere un programma per ottenere la stampa su video di una scritta qualunque, ripetuta 10 volte; nel relativo listato saranno perciò presenti una istruzione di FOR-NEXT e una istruzione di PRINT e digitando RUN sul video appariranno le 10 scritte comandate ed il computer, terminate le operazioni, si fermerà.

Passiamo ora al caso dell'assembler e supponiamo di aver scritto tutta la serie di istruzioni necessarie a compiere queste stesse 10 scritture sul video. Terminata la stesura del programma, non potremmo eseguirlo immediatamente, perché così come è scritto, per il computer non significa ancora nulla.

Bisognerà infatti richiamare il programma Assembler, utilizzando i comandi che di seguito vi riporteremo, e con questo «creare» il programma OGGETTO, ad esso corrispondente.

Fatto questo, se al file così costruito è stata assegnata l'estensione /CMD, scrivendo da DOS il nome del file, si otterrà finalmente l'immediata esecuzione del programma stesso.

Come sarà facile constatare, il tempo necessario al computer per poter scrivere sul video le 10 scritte desiderate, sarà sensibilmente inferiore a quello necessario per compiere le stesse operazioni in Basic, come già vi avevamo anticipato all'inizio di questo paragrafo.

Può essere utile, in molti casi, verificare il corretto funzionamento del programma prima di creare il file /CMD ad esso corrispondente. Per far questo potrete assemblare il vostro programma con un'altra estensione qualsiasi, ad esempio /TXT, ed utilizzare poi il programma DEBUG come già vi abbiamo spiegato nel precedente volume del BASIC + DOS.

Vediamo ora le istruzioni del nucleo ASSEMBLATORE.

A (ASSEMBLA)

Esegue l'assemblaggio del testo scritto e ne riporta il codice oggetto corrispondente su disco. Digitando semplicemente A e Return, il computer richiede il NOME FILE da assegnare al programma oggetto e con tale nome verrà registrato su disco.

NOTA: Questa istruzione genera semplicemente il codice oggetto, perciò per salvare anche il programma sorgente, quello cioè scritto precedentemente in EDIT, si dovrà prima usare il comando W, poi il comando A.

Assieme al comando principale A, possono essere forniti anche altri sottocomandi, separati fra loro dal simbolo «/». Questi comandi sono:

WE (WAIT ERROR)

(Abbreviazione dall'inglese WAIT ERROR, che tradotto letteralmente significa «Attendi per un Errore»).

Scrivendo anche questa istruzione assieme al comando A, viene eseguito normalmente l'assemblaggio del testo fino a quando non viene incontrato un errore. In questo caso, vengono sospese le operazioni di assemblaggio sul testo ed è perciò possibile correggere la linea di programma errata uscendo dall'assemblatore con CONTROL C ed entrando in EDIT. Se invece si vuole saltare l'errore e proseguire nell'assemblaggio per verificare altri possibili difetti del programma, è sufficiente digitare RETURN.

Ovviamente, se durante le operazioni di assemblaggio non viene incontrato alcun errore, viene letto tutto il testo fino alla fine senza interruzioni ed il tutto si svolge come se l'istruzione WE non fosse presente.

NL (NO LISTING)

(Abbreviazione dall'inglese NO LISTING, che tradotto letteralmente significa «senza listare»).

Con questa istruzione scritta, al solito, dopo il comando principale A, viene eseguito l'assemblaggio del testo, ma il risultato delle operazioni non viene stampato sul video.

NO (NO OBJECT-CODE)

(Abbreviazione dall'inglese NO OBJECT-CODE, che tradotto letteralmente significa «Senza Codice Oggetto»).

Assembla il testo senza scrivere su disco il codice oggetto corrispondente.

NS (NO SYMBOL TABLE)

(Abbreviazione dall'inglese NO SYMBOL-TABLE, che tradotto letteralmente significa «Senza Tavola dei Riferimenti»).

Al termine delle operazioni di assemblaggio, non viene stampata la tavola dei riferimenti.

LP (LINE PRINTER)

(Abbreviazione dall'inglese LINE-PRINTER, che tradotto letteralmente significa «STAMPANTE»).

Viene assemblato il testo ed il risultato viene riportato sulla stampante anziché sul video.

Sul monitor del computer verranno visualizzati soltanto gli eventuali errori.

Come abbiamo detto, le varie istruzioni supplementari da aggiungere dopo il comando principale A, vanno scritte senza spazi e separate fra loro dal simbolo «/», per cui potrete scrivere:

A/LP/NO

per avere la stampa su carta del programma senza registrarlo su disco, oppure:

A/WE/NO

per verificare il programma e tutte le altre possibili combinazioni di queste singole istruzioni.

Solitamente, quando si scrivono programmi molto lunghi, visto che le operazioni che l'assemblatore deve eseguire lo sono altrettanto, conviene, in prima stesura, eseguire tali operazioni con il comando:

A/WE/NL/NS

In questo modo, non dovendo trasferire alcuna istruzione sul video, la verifica e la traduzione del programma, avvengono in un tempo molto più breve.

Note aggiuntive sulla scheda video grafica LX.529

Come vi avevamo anticipato, riportiamo qui di seguito alcune note sull'utilizzo della scheda video grafica LX.529 ed inoltre, vista la quantità di richieste che ci sono pervenute in tal senso, alcune caratteristiche dell'integrato SY.6545 necessarie per capire meglio le funzioni svolte da questa scheda.

Fondamentalmente, la scheda video-grafica è costituita da due blocchi distinti, che sono:

- un circuito di interfaccia verso il microprocessore, costituito dai tre integrati Z.80 PIO (vedi IC8, IC9 ed IC10).

- il circuito controllore video e le RAM video ad esso inerenti, costituito dal SY.6545 e dalle tre memorie RAM (vedi IC4, IC5, IC6 ed IC7).

I rimanenti integrati presenti nel circuito generano e governano i vari segnali necessari al funzionamento della scheda e cioè il segnale di clock, la decodifica degli indirizzi, il sommatore del segnale video composito ecc., mentre le due EPROM (vedi IC25 ed IC34) più la RAM 3 (vedi IC26) servono per il generatore di caratteri e per la memoria degli attributi grafici della scheda.

Alcune funzioni «grafiche» sono già sviluppate all'interno del sistema operativo, vale a dire che nella EPROM siglata 2390 e nel dischetto BASIC + DOS/GRAFICA, sono già presenti tutte le «routine», in linguaggio macchina, necessarie a gestire alcune funzioni tipicamente grafiche, come ad esempio l'accensione o lo spegnimento di un singolo punto sul video, il disegno di una linea retta fra due punti dello schermo o di una figura geometrica come un rettangolo, un quadrato o una circonferenza; pertanto, sfruttando queste istruzioni, è possibile accedere semplicemente alle caratteristiche «grafiche» senza dover necessariamente rivedere tutto lo schema elettrico ed il funzionamento del circuito, per altro piuttosto complesso.

Infatti, digitando direttamente le istruzioni che ora vi elencheremo, otterrete subito da BASIC, le funzioni grafiche corrispondenti, e perciò la gestione della grafica risulterà molto semplice per chiunque.

Per chi volesse invece approfondire tale argomento, e sappiamo dalle vostre lettere che tale richiesta è molto diffusa, riporteremo di seguito una descrizione più dettagliata delle caratteristiche della scheda, utile a coloro che vorranno utilizzare tutta la potenzialità del circuito, anche con programmi scritti in assembler.

LE FUNZIONI GRAFICHE DIRETTE

— LAMPEGGIO

BLINK 1 = Attiva il lampeggio a circa 1,5 Hz.

BLINK 0 = Disattiva il lampeggio.

BLINK X = Il carattere X è simbolico e sta per un qualsiasi numero compreso fra 0 e 255. Se tale numero è pari viene attivato il lampeggio, se è dispari viene disattivato. Scrivendo semplicemente BLINK, il lampeggio viene disattivato.

— SOTTOLINEATURA

UNDER 1 = Attiva la stampa sottolineata su video.

UNDER 0 = Disattiva la sottolineatura.

UNDER X = Il carattere simbolico X sta per qualsiasi numero compreso fra 0 e 255. Se tale numero è pari viene disattivata la sottolineatura, mentre se è dispari viene attivata. Scrivendo semplicemente UNDER viene disattivata la sottolineatura.

— DOPPIA LUMINOSITÀ

HLIGHT 1 = Attiva la doppia luminosità.

HLIGHT 0 = Disattiva la doppia luminosità.

HLIGHT X = Il carattere simbolico X sta per qualsiasi numero compreso fra 0 e 255. Se tale numero è pari viene disattivata la doppia luminosità, mentre se è dispari viene attivata. Scrivendo semplicemente HLIGHT la doppia luminosità viene disattivata.

— REVERSE

REON = Attiva la scrittura in reverse.

REOFF = Disattiva la scrittura in reverse.

La stessa funzione può essere ottenuta anche con la sola istruzione REV nel modo seguente:

REV 1 = Attiva il reverse.

REV 0 = Disattiva il reverse.

REV X = Il carattere simbolico X sta per qualsiasi numero compreso fra 0 e 255. Se tale numero è pari viene disattivato il reverse, mentre se è dispari viene attivato. Scrivendo semplicemente REV il reverse viene disattivato.

— CURSORE

CURSOR ON = Accende il cursore.

CURSOR OFF = Spegne il cursore.

La stessa funzione può essere ottenuta anche scrivendo l'istruzione CURSOR seguita da un numero, nel modo seguente:

CURSOR 1 = Accende il cursore.

CURSOR 0 = Spegne il cursore.

CURSOR X = Il carattere simbolico X sta per qualsiasi numero compreso fra 0 e 255. Se tale numero è pari viene spento il cursore, mentre se è dispari viene acceso. Scrivendo semplicemente CURSOR il cursore viene acceso.

— FORMATO VIDEO

SIZE 1 = Commuta il video sul formato 40 × 24 caratteri. (cioè un massimo di 40 caratteri su ogni riga, con 24 righe sullo schermo).

SIZE 0 = Commuta il video sul formato 80 × 24 caratteri. (cioè un massimo di 80 caratteri su ogni riga, con 24 righe sullo schermo).

SIZE X = Il carattere simbolico X sta per qualsiasi numero compreso fra 0 e 255. Se tale numero è pari il formato video è commutato su 80×24 , mentre se è dispari su 40×24 . Scrivendo semplicemente SIZE, il formato video viene commutato su 80×24 .

— GRAFICA VIDEO

GRAPH 0 = Commuta la scheda video in semigrafica, 160×72 punti (cioè un massimo di 160 punti su di ogni riga, con 72 righe sullo schermo).

GRAPH 1 = Commuta la scheda video in grafica ad alta risoluzione, 320×144 punti (cioè un massimo di 320 punti su di ogni riga, con 144 righe sullo schermo).

GRAPH X = Il carattere simbolico X sta per qualsiasi numero compreso fra 0 e 255. Se tale numero è pari la scheda rimane in semigrafico normale, mentre se è dispari viene commutata in grafica ad alta risoluzione. Scrivendo semplicemente GRAPH viene automaticamente settato il formato semigrafico normale.

— SCROLLING

SCROLL 1 = Blocca lo «scrolling» sul video.

SCROLL 0 = Attiva lo «scrolling» sul video.

SCROLL X = Il carattere simbolico X sta per qualsiasi numero compreso fra 0 e 255. Se tale numero è pari rimane attivo lo «scrolling», mentre se è dispari viene bloccato. Scrivendo semplicemente SCROLL la funzione rimane attiva.

— MODE

Questa istruzione riassume tutte le istruzioni riportate fin'ora e dà la possibilità di attivare contemporaneamente più funzioni fra quelle descritte. La sintassi dell'istruzione è la seguente:

MODE 0 = Formato video 80×24

MODE 1 = Attiva lampeggio

MODE 2 = Attiva reverse

MODE 4 = Attiva sottolineatura

MODE 8 = Doppia luminosità

MODE 16 = Grafica alta risoluzione

MODE 32 = Disabilita lo «scrolling»

MODE 64 = Formato video 40×24

MODE 128 = Spegne il cursore

Sommando semplicemente i numeri riportati accanto all'istruzione MODE, si possono richiamare contemporaneamente più funzioni, ad esempio, volendo attivare la grafica ad alta risoluzione con doppia luminosità e senza lo scrolling sul video, dovremmo sommare $16 + 32 + 8 = 56$ e perciò scrivere:

MODE 56

Analogamente per qualsiasi altra combinazione di istruzioni.

È ovvio che, per disabilitare le funzioni così richiamate, dovremo scrivere la corrispondente istruzione singolarmente, come abbiamo visto precedentemente.

— BEEP ACUSTICO

BEEP = Provoca l'emissione di una nota acustica.

BEEP X = Il carattere simbolico X sta per un qualsiasi numero compreso fra 1 e 255 e provoca l'emissione di una nota dal beeper, ripetuta tante volte quanto è il valore assegnato ad X.

BEEP \$ = Provoca l'emissione di una nota acustica solo in caso di errore.

BEEP # = Disabilita la funzione BEEP \$.

BEEP & = Provoca l'emissione di una nota acustica ad ogni istruzione di INPUT.

BEEP % = Disabilita la funzione BEEP &.

Nota: Per interrompere una sequenza di note troppo lunga, è sufficiente digitare CONTROL A (cioè dare un BREAK).

— PAUSA

WAIT X = Esegue un ritardo proporzionale al valore della variabile numerica scritta in X. Ad esempio, scrivendo WAIT 100 si ha un ritardo di circa 1 secondo, mentre scrivendo WAIT 1000 si ha un ritardo di circa 10 secondi. Il numero X può assumere qualunque valore compreso fra 0 e 32768.

Se durante l'esecuzione della temporizzazione si vuole interrompere il comando, è sufficiente digitare CONTROL A (cioè dare un BREAK).

— ISTRUZIONI DI PLOTTER

Le funzioni che ora vi descriveremo, possono essere suddivise in quattro gruppi fondamentali, a seconda delle operazioni svolte e più precisamente:

- 1 — Funzioni che agiscono su di un punto.
- 2 — Funzioni che agiscono su di una linea.
- 3 — Funzioni che agiscono su di una figura geometrica regolare.
- 4 — Funzioni che agiscono su di una circonferenza.

1) FUNZIONI DI PUNTO

SET (Y,X) = Accende il punto sullo schermo la cui coordinata è definita dai due caratteri simbolici y ed x (Y definisce la coordinata VERTICALE, mentre X quella ORIZZONTALE).

Se si è in semigrafica (160 x 72 punti), le coordinate possono essere tutte quelle comprese fra 0,0 e 71,159, mentre se si è in alta risoluzione (320 x 144 punti) si può andare da 0,0 fino a 143,319.

RESET (Y,X) = È l'esatto opposto della funzione precedente, cioè spegne il punto di coordinata x,y.

POINT (Y,X) = Esegue un test sul punto di coordinata x,y e torna con 1 se il punto è acceso, oppure con 0 se è spento. Esempio:

A = POINT (20,50).

Se il punto di coordinate x=50 e y=20 è acceso, allora la variabile A assumerà il valore 1, mentre se è spento, in A ritroveremo il valore 0.

2) FUNZIONI DI RIGA

SETLINE (Y,X)—(U,V) = Traccia una linea su video tra i punti definiti dalle coordinate y,x ed u,v. Come per il caso precedente, il valore massimo che può essere inserito nell'istruzione per le due coordinate x ed y, dipende dalla risoluzione grafica prescelta perciò, in semigrafica normale, (160 x 72 punti) il valore delle coordinate può essere compreso fra 0,0 e 71,159, mentre in grafica ad alta risoluzione (320 x 144 punti), tali valori possono essere scelti fra 0,0 e 143,319.

RESETLINE (Y,X)—(U,V) = È esattamente l'opposto della funzione precedente, cioè cancella la linea tracciata fra i due punti di coordinate x,y e v,u.

NOTA: Quando si vogliono tracciare più linee consecutive è possibile seguire una sequenza di operazioni più semplice, che definire ogni volta le coordinate del punto precedente come punto di inizio della linea successiva.

In questo caso infatti, è sufficiente scrivere la coordinate del solo punto di fine della linea (cioè la coordinata del punto U,V) in quanto ad Y ed a X vengono automaticamente assegnati i valori precedenti di U e di V.

Per fare un esempio supponiamo di voler tracciare tre linee consecutive, partendo dal punto di coordinata x = 10 ed y = 30, congiungendo i punti x = 20/y = 40, x = 12/y = 32, x = 24/y = 50. Normalmente si dovrebbe scrivere:

SETLINE (30,10)—(40,20)

SETLINE (40,20)—(32,12)

SETLINE (32,12)—(50,24)

Seguendo invece il secondo metodo, potremo scrivere più semplicemente:

SETLINE (30,10)—(40,20)

SETLINE —(32,12)

SETLINE —(50,24)

ed otterremo lo stesso risultato.

POINTLINE (Y,X)—(U,V) = Esegue un test sulla linea tracciata fra i punti di coordinata x,y ed v,u e torna con 1 se la linea è presente oppure uno 0 se non è presente.

3) FUNZIONI SU FIGURE GEOMETRICHE REGOLARI

SETSQ (Y,X)—(U,V) = Traccia un rettangolo con i lati definiti dalle due coordinate x,y ed v,u. I lati del rettangolo saranno paralleli agli assi x ed y dello schermo video e la loro lunghezza sarà perciò pari a V-X per i lati orizzontali ed U-Y per i lati verticali.

RESETSQR (Y,X) — (U,V) = È l'esatto opposto della funzione precedente in quanto cancella il rettangolo di coordinate x,y e u,v.

NOTA: Analogamente a quanto abbiamo appena visto per la funzione SETLINE, anche in questo caso, volendo disegnare dei rettangoli fra loro adiacenti, cioè che abbiano sempre un tratto di lato in comune, è sufficiente scrivere una sola delle due coordinate, cioè:

SETSQR — (U,V)

in quanto, per Y ed X, vengono assunti i valori precedenti di U e V.

POINTSQR (Y,X) — (U,V) = Esegue un test sul rettangolo definito dalle coordinate x,y e v,u. Dall'operazione torna con un 1 se tutti i punti che formano il quadrato sono accesi, oppure con uno 0 se tale condizione non è verificata.

4) FUNZIONI SU CIRCONFERENZE

SETCIRCLE (Y,X) — R = Traccia una circonferenza con centro nel punto di coordinate x,y e raggio pari a R.

RESETCIRCLE (Y,X) — R = Cancella la circonferenza di centro x,y e di raggio R.

POINTCIRCLE (X,Y) — R = Esegue un test sui punti della circonferenza di centro x,y e raggio R. Se tutti i punti sono accesi, allora torna con 1, se invece questa condizione non si è verificata, torna con 0.

A questo punto, avendo terminato la descrizione di tutte le istruzioni dirette della scheda video grafica, già sviluppate nel sistema operativo del computer, passiamo alla descrizione più dettagliata delle caratteristiche del circuito, per dare la possibilità di utilizzare questo circuito anche per funzioni più complesse o comunque diverse da quelle da noi previste.

IL CIRCUITO ELETTRICO DELLA SCHEDA LX.529

Il nucleo principale del circuito è ovviamente l'integrato SY.6545, che costituisce il vero e proprio controllore video.

Questo integrato ha una struttura interna alquanto complessa, infatti la Casa Costruttrice, per renderlo adattabile a qualunque tipo di circuito e di applicazione, ha predisposto al suo interno una serie di «registri programmabili» tramite i quali vengono stabilite tutte le principali caratteristiche di funzionamento dell'integrato, cioè la frequenza di scansione orizzontale e verticale, il numero di caratteri per riga, il numero totale delle righe da visualizzare, la posizione della pagina sul video ed infine la posizione e la dimensione del cursore sulla pagina video.

Questi registri sono in totale 18 e a questi vanno aggiunti 2 registri ausiliari, previsti per la gestione di una penna ottica.

L'accesso a questi registri interni avviene seguendo una precisa sequenza di comandi, che giungono sui piedini di ingresso 23, 24 e 25 del controllore video.

A questi piedini fanno capo rispettivamente:

Piedino 23 = Fase 2. È un ingresso di sincronizzazione fra il controllore video ed i circuiti esterni ad esso collegati. Infatti, il trasferimento delle informazioni sulle 8 linee dei dati, avviene solo in corrispondenza di un impulso positivo inviato su tale piedino di ingresso.

Piedini 24 e 25 = «Register Select» (RS pied.24) e «Chip Select» (CS pied.25). Questi due ingressi governano l'accesso ai registri interni del controllore video.

Innanzitutto, per accedere ai questi registri è necessario che il piedino del «Chip Select» (che tradotto significa «Selezione del Dispositivo») sia a livello logico 0 e in tale stato deve rimanere durante tutte le operazioni di lettura o di scrittura nei registri del controllore.

Contrariamente infatti, mantenendo il CS a livello logico 1, il controllore video prosegue nelle normali operazioni per la gestione dell'immagine sul video e non risponde alle richieste della CPU.

Inizialmente, anche il piedino di ingresso del «Register Select» deve essere a livello logico 0 perché l'informazione presente sulle 8 linee dei dati venga caricata all'interno del controllore video ed in questo modo viene selezionato uno dei suoi 20 registri interni.

Dopo questa operazione iniziale, sulle otto linee dei dati saranno disponibili gli otto bit del registro selezionato e perciò sarà possibile leggere o scrivere in essi l'informazione voluta.

Questa sequenza di istruzioni è di fondamentale importanza per comprendere il funzionamento «globale» di tutta la scheda video, in quanto ogni accesso ai registri interni del controllore video deve necessariamente seguire questo tipo di procedura.

Nella tabella riportata di seguito, troverete l'elenco completo di tutti i registri interni dell'integrato SY.6545. Come per qualsiasi altro tipo di periferica di un sistema a microprocessore, anche per questa scheda avremo un decodificatore di indirizzo che specificherà in quale locazione di memoria risulta inserita la scheda all'interno dell'area indirizzabile dal computer.

Tabella dei registri interni del SY-6545-1

Num. Reg.	Specifica	Valore	R	W
R0	Num. di caratteri —1 per riga	111	—	x
R1	Num. di caratteri visualizzati per riga	80	—	x
R2	Posizione orizzontale dell'immagine	87	—	x
R3	Larghezza degli impulsi di sincronismo verticale e orizzontale	40	—	x
R4	Num. di righe in totale —1 per quadro	20 26	—	x
R5	Aggiustamento fine della frequenza di scansione verticale	00	—	x
R6	Num. di righe visualizzate per quadro	24	—	x
R7	Posizione verticale della pagina	24	—	x
R8	Registro di controllo	72	—	x
R9	Num. di righe —1 che formano ogni singolo carattere	11	—	x
R10	Riga di inizio del cursore	96 00	—	x
R11	Riga di fine del cursore	11	—	x
R12	Indirizzo alto di memoria della prima riga da visualizzare	prog.	—	x
R13	Indirizzo basso di memoria della prima riga da visualizzare	prog.	—	x
R14	Posizione del cursore (Ind. Alto)	prog.	x	x
R15	Posizione del cursore (Ind. Basso)	prog.	x	x
R16	Penna Ottica (Ingresso ind. Alto)		x	—
R17	Penna Ottica (Ingresso ind. Basso)		x	—
R18	Indirizzo-corrente di scrittura/lettura (Ind. Alto)	prog.	x	x
R19	Indirizzo corrente di scrittura/lettura (Ind. Basso)	prog.	x	x
R31	UPDATE — Aggiornamento		—	—

In questo caso, abbiamo utilizzato l'area di memoria delle periferiche con indirizzi, in esadecimale, 8C e 8D. Non dovete meravigliarvi se con solo 2 indirizzi si possono selezionare tutti i 20 registri dell'integrato, in quanto, come abbiamo appena specificato, la comunicazione con questi registri non avviene in modo diretto, ma tramite una sequenza di operazioni comandate dalle sole linee di RS e CS, mentre l'indirizzamento dei registri ed il trasferimento dei dati in essi presenti, avviene sempre e soltanto attraverso le 8 linee dei dati.

Per maggior chiarezza, facciamo un semplice esempio e supponiamo di voler leggere il contenuto del registro 14. Per prima cosa, il computer selezionerà la scheda video portando a **livello logico 0** il piedino 25 del «Chip Select». Mantenendo a **livello logico 0** anche l'ingresso del «Register Select» (piedino 24), il computer scriverà sulle 8 linee dei dati il numero 14 corrispondente al registro desiderato. A questo punto, portando a **livello logico 1** il piedino del «Register Select», il controllore video ricopierà il contenuto del registro 14 sulle 8 linee dei dati ed il computer potrà così leggerne il valore.

Ovviamente, le operazioni per la scrittura di un dato all'interno di un registro, si svolgeranno con la stessa sequenza di operazioni con la sola differenza che, alla fine, il computer invece di leggere il contenuto del registro, scriverà al suo interno il valore voluto.

Vediamo pertanto dalla tabella di questi registri, il loro significato per esteso.

R0 = Il contenuto di questo registro specifica il numero TOTALE di caratteri —1 che vengono conteggiati dal controllore video per ogni riga del quadro video. Ovviamente, una singola linea del video è composta da un numero maggiore di caratteri di quelli che vengono realmente visualizzati perché, riproducendoli tutti, i caratteri posti alle estremità della pagina video risulterebbero distorti e sui bordi dello schermo, tanto da essere praticamente indecifrabili. Per questo motivo, vengono utilizzati due registri per definire, il primo (R0) il numero totale di caratteri per ogni riga ed il secondo, (vedi R1), per definire invece quanti di questi caratteri dovranno essere visualizzati sulla pagina video.

È importante sottolineare che ad ogni singolo carattere corrisponderà un impulso di clock per cui, definire quanti caratteri dovranno essere presenti su ciascuna riga, equivale anche a stabilire la frequenza di scansione orizzontale.

R1 = Come abbiamo appena anticipato, questo registro contiene il numero di caratteri che verranno effettivamente visualizzati dalla CRT su ogni riga dello schermo del monitor.

R2 = Come conseguenza dei primi due registri, questo terzo registro interno stabilisce da che punto della riga visualizzata iniziare a far apparire i caratteri precedentemente definiti.

R3 = Il contenuto di questo registro è doppio e specifica la larghezza degli impulsi di sincronismo verticale ed orizzontale. In questo modo è possibile adattare il segnale video in uscita alle caratteristiche di qualunque tipo di monitor, senza dover modificare il valore dei componenti per regolare la larghezza degli impulsi di sincronismo.

R4 = Analogamente a quanto detto per il registro 0, il registro 4 contiene il numero TOTALE di righe —1 che formano una intera pagina video e tale numero sarà maggiore delle righe effettivamente visualizzate sul monitor. Come abbiamo già visto a proposito del registro 0, definendo il numero di righe totali che formano un quadro completo, viene automaticamente definita anche la frequenza di scansione verticale del segnale video composito.

R5 = È un registro analogo a quello appena descritto e serve per correggere in modo più preciso, il numero di righe che completano un quadro video, quindi per aggiustare «in fine» la frequenza di scansione verticale.

R6 = Definisce il numero delle righe da visualizzare per ogni quadro.

R7 = Definisce la posizione verticale della pagina video sul monitor.

R8 = È un registro composto e serve per definire le caratteristiche di alcuni segnali di controllo rispetto alla configurazione elettrica della scheda nella quale è inserito il video controller stesso. Ogni singolo bit di questo registro ha quindi un suo preciso significato:

Bit 0 e 1 = Video interlacciato o no.

Bit 2 = Indirizzamento delle memorie video (Binario o Riga-Colonna)

Bit 3 = Accesso alle memorie video (Diretto o Trasparente)

Bit 4 = Ritardo sul segnale di «Display Enable» (SI/NO)

Bit 5 = Ritardo sul segnale di posizionamento del cursore (SI/NO)

Bit 6 = «UPDATE» — Segnalazione di indirizzo di memoria o di dato. (Modo Trasparente)

Bit 7 = «UPDATE» — Segnalazione di accesso libero durante le ritracce verticali o orizzontali o sul segnale di Fase 2. (Modo Diretto).

Il significato per esteso del funzionamento di questi bit è molto complesso ed è strettamente legato alle scelte di progetto della scheda. Essendo parametri di sistema, sono tutti valori definiti e non modificabili, pena il non funzionamento della scheda. Sono quindi informazioni che appesantirebbero enormemente la trattazione di questo argomento senza apportare alcuna utilità pratica all'uso e alla comprensione del funzionamento della scheda.

R9 = Definisce il numero di linee che deve occupare ciascun carattere, comprese le linee di spazio fra una riga e l'altra. In altre parole, questo registro definisce lo «spessore» di ciascuna riga.

R10 = Definisce il punto di inizio del cursore all'interno di ciascun singolo carattere. Questo registro, assieme al registro R11 che definisce la fine del cursore all'interno del carattere, serve in pratica per programmare lo «spessore» del cursore.

R11 = Punto di fine del cursore all'interno del carattere. (Vedi registro precedente).

R12 = Definisce da che indirizzo iniziare a visualizzare il contenuto della memoria video. Essendo un indirizzo a 14 bit, nel registro 12 saranno scritti i 6 bit più significativi di tale indirizzo, mentre nel registro 13 gli ultimi 8 bit meno significativi.

R13 = 8 bit meno significativi dell'indirizzo di memoria video da cui iniziare la visualizzazione. (Vedi registro precedente).

R14 = Definisce la posizione del cursore sulla pagina video. Essendo un indirizzo a 14 bit, il registro 14 contiene i 6 bit più significativi di questo indirizzo, mentre il registro 15 contiene gli 8 bit meno significativi.

R15 = Vedi registro 14.

R16 = Questo registro contiene l'indirizzo della pagina di memoria individuata dalla penna ottica. Come per il caso dei registri precedenti, essendo un dato a 14 bit questo primo registro conterrà i 6 bit più significativi dell'informazione, mentre il registro 17 conterrà gli 8 bit rimanenti.

R17 = Vedi registro 16.

R18 = Questo registro contiene l'indirizzo della memoria video nel quale è stato scritto l'ultimo carattere ed è possibile variare il suo contenuto ed ottenere la scrittura di un carattere in qualsiasi altra posizione della mappa video. Ad ogni scrittura di un carattere, il contenuto di questo registro viene automaticamente incrementato di 1, perciò, volendo scrivere più caratteri uno di seguito all'altro, è sufficiente scrivere in questo registro il solo indirizzo iniziale. Essendo anche questo indirizzo a 14 bit, sono necessari due registri per contenere tutta l'informazione ad esso relativa ed infatti R18 contiene i 6 bit più significativi dell'indirizzo, mentre R19 contiene i rimanenti 8 bit meno significativi.

R19 = Vedi registro precedente.

R31 = UPDATE — È un registro «fantasma» in quanto, pur essendo indirizzabile, non contiene alcuna informazione e non è possibile scrivere alcun dato al suo interno. Questo registro, però, è di fondamentale importanza nella gestione del processore video in quanto ogni aggiornamento nel contenuto delle memorie video, deve essere seguito da un accesso all'«UPDATE». Tale operazione infatti, abilita l'aggiornamento di questo valore, dando il consenso al controllore video per l'esecuzione fisica dell'operazione. Se non si esegue un accesso al registro di «UPDATE», l'operazione di scrittura non ha alcun esito ed il contenuto delle memorie video rimane inalterato.

Al termine di questo paragrafo, troverete due esempi di programmi in ASSEMBLER nei quali si fa uso di questi registri per la gestione della scheda video-grafica con funzioni dirette di scrittura. Potrete utilizzare tali programmi come valido esempio di utilizzo ed anche come termine di confronto per verificare, nella pratica, l'uso di questi nuovi concetti che abbiamo fin'ora esposto solo teoricamente.

Per completare la trattazione del circuito, parleremo ora delle funzioni svolte dai tre integrati Z.80-PIO (vedi IC8, IC9 ed IC10), utilizzati per gestire l'accesso del computer all'area di memoria video.

Ogni singola PIO assolve ad una precisa funzione e risponde ad un preciso indirizzo all'interno dell'area di memoria delle periferiche del computer. La disposizione esatta di questi tre integrati è la seguente:

Integrato	Funzione svolta	Indirizzo (esadecimale)
PIO 0 (IC10) (Porta A)	Modo SEMIGRAFICO = Carat. ASCII. Modo GRAFICO = mappa grafica RAM 0.	DATI = 80 CONTROLLO = 82
PIO 0 (IC10) (Porta B)	Uscita caratteri ASCII per la stampante.	DATI = 81 CONTROLLO = 83
PIO 1 (IC9) (Porta A)	Modo SEMIGRAFICO = non utilizzata. Modo GRAFICO = mappa grafica RAM 1.	DATI = 84 CONTROLLO = 86

Integrato	Funzione svolta	Indirizzo (esadecimale)
PIO 1 (IC9) (Porta B)	Entrata caratteri ASCII dalla tastiera (negati)	DATI = 85 CONTROLLO = 87
PIO 2 (IC8) (Porta A)	Modo SEMIGRAFICO = non utilizzata Modo GRAFICO = mappa garica RAM 2	DATI = 88 CONTROLLO = 8A
PIO 2 (IC8) (Porta B)	Ingresso «BUSY» stampante, controllo 40/80 carat.	DATI = 89 CONTROLLO = 8B

I due indirizzi specificati nell'ultima colonna a destra della tabella, corrispondono, in esadecimale, all'ingresso dei dati nella scheda (DATI) e all'ingresso dei comandi di programmazione della periferica.

Come potete osservare dallo schema elettrico della scheda video-grafica, l'indirizzamento delle RAM video è gestito solo ed esclusivamente dal controllore video, in quanto i piedini di ingresso degli indirizzi di tali memorie, (vedi piedini 1,2,3,4,5,6,7,8,9,19,22,23) risultano tutti collegati alle uscite dell'integrato SY.6545.

Ogni operazione di scrittura di un carattere su tali memoria, dovrà quindi essere direttamente gestita dal controllore video per quanto riguarda l'indirizzamento della cella di memoria corrispondente, mentre il dato da stampare passerà attraverso le tre PIO di interfacciamento.

Nel modo semigrafico, quando cioè si devono stampare caratteri alfanumerici, viene utilizzata la porta A della PIO 0 per il carattere e la RAM 3, posta all'indirizzo esadecimale 8E. Quest'ultimi specificano le caratteristiche di visualizzazione del carattere e sono così raggruppati:

Bit 0 = BLINK (attivo a 0)
 Bit 1 = REVERSE (attivo a 0)
 Bit 2 = SOTTOLINEATO (attivo a 0)
 Bit 3 = DOPPIA LUMINOSITÀ (attivo a 0)
 Bit 4 = ALFANUMERICO/GRAFICO (alfa = 1)
 Bit 5-8 = Non utilizzati.

Nel modo grafico invece, l'informazione relativa al punto da visualizzare viene scritta nelle tre RAM 0, 1 e 2, in modo da ottenere una definizione sul video pari a 320 x 144 punti e l'informazione viene trasmessa alla scheda attraverso le tre PIO 0, 1 e 2. Anche in questo caso, la RAM 3 sovrastende agli attributi dei caratteri, con la sola limitazione che gli attributi non si intendono riferiti ad ogni singolo punto, ma alla dimensione di un intero carattere, costituito da 24 punti elementari. (Matrice 6 x 4).

A questo punto possiamo concludere il paragrafo, riportando il listato in Assembler di due programmi di esempio di gestione della scheda video-grafica e, di seguito, la descrizione delle funzioni svolte da questi.

```

00100 ;*****
00110 ;*** Routine di accesso diretto al video ***
00120 ;*** FILE: VIDEO1/TXT ***
00122 ;*** FRANCESCO SECCHI 03/85 ***
00130 ;*****
00140 ;Questa routine porta direttamente sul video
00150 ;i dati scritti nelle memorie D500H,D501H
00160 ;e D502H, nel punto di coordinate
00170 ;X,definita alla memoria D800H, e
00180 ;Y,definita alla memoria D801H.
00190 ;
00200 ;
D000 00210          ORG      OD000H
00220 ;
00230 ;calcolo delle coordinate del punto
00240 ;
D000 F5 00250          PUSH   AF
D001 C5 00260          PUSH   BC
D002 E5 00270          PUSH   HL
D003 210000 00280          LD     HL,00

```



```

D006 015000 00290 LD BC,50H ;lunghezza di una riga
D009 3A01D8 00300 LD A,(0D801H)
D00C FE00 00310 CP 00
D00E 2804 00320 JR Z,ASSEX
D010 09 00330 LOOP1 ADD HL,BC ;coordinata (Y)
D011 3D 00340 DEC A
D012 20FC 00350 JR NZ,LOOP1
D014 3A00D8 00360 ASSEX LD A,(0D800H);coordinata (X)
D017 4F 00370 LD C,A
D018 0600 00380 LD B,00
D01A 09 00390 ADD HL,BC
00400 ;nel registro (HL) c'e' ora l'indirizzo del carattere
00410 ;imposta l'INDIRIZZO nella CRTC
00420 ;
D01B CD69D0 00430 CALL AZZER
D01E 3E12 00440 LD A,18 ;seleziona il
D020 D38C 00450 OUT (8CH),A ;registro 18.
D022 7C 00460 LD A,H ;Scrive l'indirizzo
D023 D38D 00470 OUT (8DH),A ;ALTO nel registro.
D025 3E13 00480 LD A,19 ;Seleziona il
D027 D38C 00490 OUT (8CH),A ;registro 19.
D029 7D 00500 LD A,L ;Scrive l'indirizzo
D02A D38D 00510 OUT (8DH),A ;BASSO nel registro.
D02C 3E0C 00520 LD A,12 ;azzera i registri di
D02E D38C 00530 OUT (8CH),A ;inizio pagina
D030 3E00 00540 LD A,0 ;R12 e R13
D032 D38D 00550 OUT (8DH),A
D034 3E0D 00560 LD A,13
D036 D38C 00570 OUT (8CH),A
D038 3E00 00580 LD A,0
D03A D38D 00590 OUT (8DH),A
00600 ;
00610 ;imposta la CRTC nel modo grafico
00620 ;ed elimina il cursore
00630 ;
D03C 3E0F 00640 LD A,0FH
D03E D38E 00650 OUT (8EH),A
00660 ;
00670 ;scrive il carattere
00680 ;
D040 3A00D5 00690 LD A,(0D500H)
D043 D380 00700 OUT (80H),A ;primo dato in R0
D045 3E1F 00710 LD A,31 ;UPDATE
D047 CD69D0 00720 CALL AZZER
D04A D38C 00730 OUT (8CH),A
D04C 3A01D5 00740 LD A,(0D501H)
D04F D384 00750 OUT (84H),A ;secondo dato in R1
D051 3E1F 00760 LD A,31
D053 D38C 00770 OUT (8CH),A
D055 3A02D5 00780 LD A,(0D502H)
D058 D388 00790 OUT (88H),A ;terzo dato in R2
D05A 3E1F 00800 LD A,31
D05C D38C 00810 OUT (8CH),A
00820 ;ripristina i registri per tornare al DOS
D05E AF 00830 XOR A
D05F 323232 00840 LD (3232H),A
D062 323332 00850 LD (3233H),A
D065 E1 00860 POP HL
D066 C1 00870 POP BC
D067 F1 00880 POP AF
D068 C9 00890 RET
00900 ;routine di attesa per UPDATE.
D069 DB8C 00910 AZZER IN A,(8CH)
D06B CB7F 00920 BIT 7,A
D06D 28FA 00930 JR Z,AZZER
D06F C9 00950 RET
00960 ;
0000 00970 END
00000 TOTAL ERRORS

```

```

ASSEX D014 00360 00320
AZZER D069 00910 00430 00720 00930
LOOP1 D010 00330 00350

```

COMMENTI:

Questo programma permette di accedere direttamente al video, scrivendo un carattere definito a vostro piacere e di riportarlo in una locazione qualsiasi all'interno della finestra video, «mappata» con due assi cartesiani X ed Y. L'origine degli assi (cioè il punto di coordinata 0,0), è fissato in alto a sinistra della pagina video e la variazione massima consentita è di 360 x 144.

Il carattere viene stampato sul video in grafica ad alta risoluzione e la «mappa» di ogni carattere è definita da una matrice a 4 x 6 elementi.

All'interno di questa matrice i punti vengono descritti in tre insiemi di otto punti, riferiti alle tre RAM 0, RAM 1 e RAM 2, ed ogni singolo elemento della matrice corrisponderà ad un singolo bit. Ovviamente se il bit corrispondente è a 1 il punto risulterà acceso, mentre se il bit è a 0 il punto risulterà spento. La corrispondenza è la seguente:

matrice del carattere

bit 0	bit 1	bit 2	bit 3	<	8 bit della RAM 0
bit 4	bit 5	bit 6	bit 7		
bit 0	bit 1	bit 2	bit 3	<	8 bit della RAM 1
bit 4	bit 5	bit 6	bit 7		
bit 0	bit 1	bit 2	bit 3	<	8 bit della RAM 2
bit 4	bit 5	bit 6	bit 7		

È perciò possibile definire, all'interno di questa matrice, un qualsiasi tipo di carattere, semplicemente accendendo o spegnendo i singoli elementi che la formano. Come sempre, ad ogni bit corrisponde un proprio «peso» e cioè:

BIT 0 = 1 ; BIT 1 = 2 ; BIT 2 = 4 ; BIT 3 = 8 ; BIT 4 = 16 ; BIT 5 = 32 ; BIT 6 = 64 ; BIT 7 = 128

Semplicemente sommando questi numeri, potrete ottenere l'accensione di un qualsiasi punto della matrice, ad esempio, volendo riprodurre una freccetta obliqua così rappresentata, dovrete scrivere:

				<	RAM 0 = 16 + 32 + 64 + 128 = 240
*	*	*	*		
		*	*	<	RAM 1 = 4 + 8 + 32 + 128 = 172
	*		*		
*			*	<	RAM 2 = 1 + 8 = 9

Per utilizzare questo programma scritto in assembler all'interno di programmi in BASIC, dovrete caricare il BASIC con l'istruzione iniziale:

BASIC 49000

in modo da limitare l'area di memoria che il basic occupa e lasciare perciò il posto alla routine in linguaggio macchina presente in memoria.

Fatto questo, una volta che si è in ambiente BASIC, è sufficiente definire una istruzione di DEFUSR con indirizzo corrispondente alla routine in linguaggio macchina per disporre, in pratica, di un nuovo comando operativo.

Nell'esempio riportato di seguito, abbiamo scritto un semplice programma in basic con il quale potrete vedere le funzioni svolte da questi programmi e capirne meglio l'utilizzo.


```

100 '#####
200 '## FRANCESCO SECCHI 03.85 ##
300 '## Caricare il BASIC con: BASIC 49000 ##
310 '## Routine di prova: PRO2 ##
330 '#####
400 CLS:DEFUSRO=&HD000
500 CMD"load video1/asm"
600 IX=&HD800:IY=&HD801
700 RO=&HD500:R1=&HD501:R2=&HD502
800 POKERO,&HOF:POKER1,&H55:POKER2,&HFO
900 FORA=1TO24:K=K+1:POKEIX,A:POKEIY,K:G=USRO(O):X=KEY:NEXT

```

Lanciando il programma, dopo che il sistema lo avrà caricato in memoria in linguaggio macchina, premendo un qualsiasi tasto della tastiera vedrete apparire, uno di seguito all'altro, dei simboli grafici sullo schermo.

Passiamo ora ad un secondo programma di esempio, con il quale potrete ottenere la visualizzazione di maschere video in un tempo molto più breve di quello che sarebbe necessario sfruttando le solite istruzioni di PRINT da BASIC.

Dopo il primo caricamento infatti, la maschera definita apparirà, quasi istantaneamente, dopo la chiamata del programma in linguaggio macchina che gestisce tale funzione.

```

00100 ;*****
00110 ;**** Nome testo : ROUVIDEO/TXT ****
00120 ;**** Nome oggetto : ROUVIDEO/ASM ****
00130 ;**** C.Charitakis P.Poggi, 60 S.LAZZARO - BO - ****
00140 ;**** 05/12/83 ****
00150 ;*****
00160 ;Questa routine permette di portare al video campi di
00170 ;lunghezza desiderata.La fine del campo si segnala con '00'
00180 ;All' inizio del campo si deve dare la posizione con due
00190 ;BYTES che individuano la posizione assoluta al VIDEO.
00200 ;
00210 ;
D000 00220 ORG OD000H ;start del programma
008C 00230 POCP EQU 8CH ;porta controllo SYS6545
008D 00240 PODA EQU 8DH ;porta dati SY6545
0080 00250 POPIO EQU 80H ;porta PIO RAM 0 per dati
008E 00260 DAPIO EQU 8EH ;porta PIO RAM 3 Attributi
3043 00270 ATTR EQU 3043H ;Registro DOS per gli attributi
C000 00280 REGIS EQU 0C000H ;Esempio locaz. ove sono i dati da
00290 ;visualizzare
00300 ;
D000 E5 00310 PUSH HL
D001 C5 00320 PUSH BC
D002 F5 00330 PUSH AF
00340 ;
D003 0E0B 00350 LD C,0BH ;offset per registri 12/13
D005 CD25D0 00360 CALL INIT
D008 0E11 00370 LD C,11H ;offset per registri 18/19
D00A CD25D0 00380 CALL INIT
00390 ;
00400 ;
D00D 2100C0 00410 LD HL,REGIS
D010 4E 00420 LD C,(HL)
D011 23 00430 INC HL
D012 46 00440 LD B,(HL)
D013 E5 00450 PUSH HL
D014 C5 00460 PUSH BC
D015 E1 00470 POP HL
D016 CD31D0 00480 CALL OKPRO ;chiama routine posizionamento
D019 E1 00490 POP HL
D01A 23 00500 LOOP INC HL
D01B 7E 00510 LD A,(HL)
D01C FE00 00520 CP 00 ;Il campo e finito ?
D01E 2837 00530 JR Z,RIPR ;Si vai a ripristinare valori.
D020 CD40D0 00540 CALL PRES ;No mostra il carattere !!!
D023 18F5 00550 JR LOOP
00560 ;

```

00570 ;***** Inizializza i registri di SY6545 a zero ****

00580 ;
00590 INIT LD B,02H
00600 PRREG LD A,B
00610 ADD A,C
00620 OUT (POCP),A;carica registro interessato
00630 XOR A
00640 OUT (PODA),A;e lo azzerà.
00650 DJNZ PRREG
00660 RET

00670 ;
00680 ;***** Definisce la posizione al VIDEO *****
00690 ;

00700 OKPRO LD A,12H
00710 OUT (POCP),A
00720 LD A,H
00730 OUT (PODA),A
00740 LD A,13H
00750 OUT (POCP),A
00760 LD A,L
00770 OUT (PODA),A
00780 RET

00790 ;
00800 ;***** Presenta al VIDEO *****
00810 ;

00820 PRES PUSH AF
00830 AZZER IN A,(POCP);aspetta che lo status
00840 BIT 07H,A ;del SY6545 sia zero.
00850 JR Z,AZZER
00860 POP AF
00870 OUT (POPIO),A;visualizza il carattere
00880 LD A,(ATTR);e gli da' gli attributi.
00890 OUT (DAPIO),A
00900 LD A,31
00910 OUT (POCP),A
00920 XOR A
00930 OUT (PODA),A
00940 RET

00950 ;
00960 ;***** Ripristina valori ai registri *****
00970 ;

00980 RIPR XOR A ;azzerà doppio registro
00990 LD (3232H),A ;della finestra video
01000 LD (3233H),A ;del BASIC-DOS
01010 POP AF
01020 POP BC
01030 POP HL
01040 RET
01050 END
00000 TOTAL ERRORS

ATTR	3043	00270	00880	
AZZER	D041	00830	00850	
DAPIO	008E	00260	00890	
INIT	D025	00590	00360	00380
LOOP	D01A	00500	00550	
OKPRO	D031	00700	00480	
POCP	008C	00230	00620	00710 00750 00830 00910
PODA	008D	00240	00640	00730 00770 00930
POPIO	0080	00250	00870	
PRES	D040	00820	00540	
PRREG	D027	00600	00650	
REGIS	C000	00280	00410	
RIPR	D057	00980	00530	

Questo programma si rivelerà molto comodo ogni qualvolta si vorrà riportare velocemente sul video una maschera definita precedentemente, senza dovere ricorrere ad una lunga sequenza di operazioni tipo PRINT, PRINT TAB, ecc.

Viene infatti definita un'area di memoria nella quale è ricopiata l'immagine video e successivamente, quando deve apparire questa maschera, viene caricata tutta l'area di memoria sullo schermo: un'operazione che oltre ad essere più comoda all'atto della stesura del programma, è anche molto più veloce della corrispondente funzione svolta direttamente da BASIC.

Nel programma riportato qui di seguito, per permettervi di verificare nella pratica il funzionamento del sistema di stampa da noi adottato, abbiamo scritto un programma di esempio in BASIC che utilizza questa routine.

```

100 REM*****
110 REM*** ESEMPIO6 ****
120 REM*** FARE BASIC 49000 ****
130 REM*** ROUTINE PROVA ROUVIDEO/ASM ****
140 REM*** C.CHARITAKIS 08/12/83 ****
150 REM*****
160 CLEAR1300:DEFUSRO=&HD000:P=&HC000
170 CMD"load rouvideo/asm"
180 CLS
190 A=0:A$=&A:REM*** A=POSIZIONE (0) ****
200 POKEP,&H(RIGHT$(A$,2)):POKEP+1,&H(LEFT$(A$,2))
210 READ M$:FORX=1TOLEN(M$):POKEP+1+X,ASC(MID$(M$,X,1)):NEXT K=X-1
220 READM$:FORY=1TO22:FORX=1TOLEN(M$):POKEP+K+1+X,ASC(MID$(M$,X,1)):NEXTX:K=K+X-1:NEXTY
230 RESTORE:READ M$:FORX=1TOLEN(M$):POKEP+K+1+X,ASC(MID$(M$,X,1)):NEXT:POKEP+P+1+X,0
240 PRINT"SONO PRONTO":BEEP5:Y$=KEYIN$
250 A=USRO(0)
260 GOTO260
270 DATA"*****"
280 DATA"***"
    
```

LEZIONI BASIC su FLOPPY

Chi ha realizzato il nostro computer Z80 potrà disporre di tutte le lezioni Basic per un corso completo di 50 lezioni su dischetti floppy.

Per poter utilizzare queste lezioni, il computer deve disporre della stampante per poter scrivere il testo delle lezioni, dell'interfaccia video-grafica LX.529 più ovviamente uno o due drive per floppy-disk.

Per quanto riguarda la **memoria**, la minima configurazione richiesta è di 40K e perciò sul BUS del vostro computer dovrà essere inserita almeno una scheda di espansione da 32 K (LX.392) più una da 8K (LX.386).

Nelle ultime lezioni del corso, precisamente a partire dalla 31/esima in poi, abbiamo modificato le routine che gestiscono la stampa in modo da poter eseguire la lezione stessa anche solo sul video, senza l'uso obbligato della stampante.

In queste lezioni, all'inizio del programma, verrà presentato sul video un «menu» nel quale potrete scegliere una delle seguenti opzioni:

- S — per la lezione Stampata
- V — per la lezione Video
- E — per fare solo Esercizi
- F — per Finire

Cominciamo dal caso in cui si intenda eseguire la lezione solo sul video: sarà sufficiente premere il tasto V e, sul monitor, ad ogni vostro comando appariranno, una di seguito all'altra, tutte le pagine della lezione in corso. Inizialmente apparirà la pagina relativa all'indice degli argomenti trattati e quindi, di seguito, tutte le altre.

Nella parte inferiore dello schermo sarà presente il numero della lezione ed il numero della pagina visualizzata: questa numerazione corrisponde a quella presente nelle pagine della lezione stampate su carta con la sola differenza che, ad ogni pagina «su carta», corrispondono tre pagine video e quindi, ad esempio, alla stampa della pagina 4 corrisponderanno, sul monitor, le pagine 3,A — 3,B — 3,C e così via.

Utilizzando il monitor per visualizzare la lezione, è possibile scorrere il testo sia in avanti, digitando «>», che all'indietro, digitando «<», proprio come se stessimo «sfogliando» le pagine di un libro.

In questo modo, arretrando fino alla prima pagina nella quale è presente il «menu», è possibile modificare la scelta iniziale ed ottenere la lezione su stampante o fare i soli esercizi in essa contenuti.

Vediamo subito quest'ultima opzione: in questo caso, l'esecuzione del programma prevede l'uso della stampante e perciò, inizialmente, bisogna scegliere definire il tipo di stampante usata (Epson o Microline) e, così facendo, le lezioni appariranno contemporaneamente sul monitor e trasferite su carta.

Ovviamente, utilizzando anche la stampante, non si potrà più arretrare nella visualizzazione del testo in quanto la stampa degli esercizi potrà soltanto procedere in avanti.

Vogliamo concludere questa nota ricordando che tutte le lezioni del Basic sono disponibili sia per la configurazione con **1 solo drive** sia per la configurazione con **2 drive** e, vista la differente disposizione delle lezioni sui dischetti a seconda dei due casi, è importante, nell'acquistarli, **specificare** se si desidera la versione da 1 o da 2 drive: per un solo floppy la sigla del disco dovrà essere seguita dalla lettera /D (cioè DISCO 8/D oppure DISCO 23/D) mentre per due floppy non è necessario aggiungere nulla al numero del disco corrispondente (cioè è sufficiente DISCO 5 oppure DISCO 10 ecc.).

Per 1 solo drive:

Chi possiede un solo drive, dovrà inizialmente inserire il dischetto del BASIC + DOS e, dopo aver acceso il computer e premuto il tasto del reset, premere due volte il tasto di ESCAPE.

Come ormai saprete, in questo modo verrà caricato il sistema operativo DOS e successivamente il linguaggio BASIC.

Eseguita questa semplice operazione iniziale, dovrete estrarre dal drive il disco del BASIC + DOS ed inserire al suo posto quello contenente la lezione desiderata.

Scrivendo ora RUN seguito dal nome del file corrispondente alla lezione desiderata, ad esempio:

RUN "LEZ21

e premendo di seguito RETURN, verrà caricato il programma della lezione 21 ed eseguito.

Se sul disco che utilizzate esiste il file «PROLOGO», potrete digitare semplicemente RUN «PROLOGO» e RETURN dopodiché sul video vi apparirà un «menu» nel quale sono specificate tutte le istruzioni da eseguire per ottenere la stampa dell'introduzione, per l'indice o per la scelta delle lezioni.

NOTA: sull'ultima lezione, cioè la numero 50, è presente il file PROLOGO che contiene l'indice generale completo di tutte le lezioni. Lanciando questo programma otterrete un indice generale aggiornato di tutto il corso.

Per 2 drive

Le operazioni da eseguire in questo caso sono le stesse descritte precedentemente con la sola differenza che, disponendo di due drive, non è più necessario estrarre dal drive 0 il dischetto del BASIC + DOS in quanto il dischetto contenente la lezione di BASIC andrà inserito nel **drive 2**.

Per concludere, riportiamo di seguito l'elenco completo di tutti i dischi del corso con il numero di lezioni in essi contenute ed infine l'indice generale degli argomenti trattati in ogni singola lezione.

PER 1 SOLO DRIVE FLOPPY (costo di ogni dischetto L. 12.000)

DISCO 1/D (lezioni 1-2)
DISCO 2/D (lezioni 3-4)
DISCO 3/D (lezioni 5-6-7)
DISCO 4/D (lezioni 8-9)
DISCO 5/D (lezioni 10-11-12)
DISCO 6/D (lezioni 13-14)
DISCO 7/D (lezioni 15-16)
DISCO 8/D (lezioni 17-18)
DISCO 9/D (lezioni 19-20)
DISCO 10/D (lezioni 21-22)
DISCO 11/D (lezioni 23-24)
DISCO 12/D (lezioni 25-26)
DISCO 13/D (lezioni 27-28)
DISCO 14/D (lezioni 29-30)
DISCO 15/D (lezioni 31)
DISCO 16/D (lezioni 32-33)
DISCO 17/D (lezioni 34-35)
DISCO 18/D (lezioni 36-37-38)
DISCO 19/D (lezioni 39-40)
DISCO 20/D (lezioni 41-42-43)
DISCO 21/D (lezioni 44-45)
DISCO 22/D (lezioni 46-47-48)
DISCO 23/D (lezioni 49-50)

PER DUE DRIVE FLOPPY (costo di ogni dischetto L. 15.000)

DISCO 1 (lezioni 1-2-3)
DISCO 2 (lezioni 4-5-6-7)
DISCO 3 (lezioni 8-9-10)
DISCO 4 (lezioni 11-12-13-14)
DISCO 5 (lezioni 15-16-17)
DISCO 6 (lezioni 18-19-20)
DISCO 7 (lezioni 21-22-23)
DISCO 8 (lezioni 24-25-26-27)
DISCO 9 (lezioni 28-29-30)
DISCO 10 (lezioni 31-32-33)
DISCO 11 (lezioni 34-35-36-37)
DISCO 12 (lezioni 38-39-40-41)
DISCO 13 (lezioni 42-43-44)
DISCO 14 (lezioni 45-46-47-48)
DISCO 15 (lezioni 49-50)

In questo secondo elenco riportiamo gli argomenti che vengono trattati nelle diverse lezioni, per darvi la possibilità di scegliere, fra tutte, le sole lezioni che risultano di vostro maggior interesse.

- LEZIONE 1** = Concetto di PROGRAMMA e definizione delle VARIABILI NUMERICHE. Istruzione REM.
- LEZIONE 2** = Istruzioni di PRINT, END, CLS. Concetto di ISTRUZIONI MULTIPLE. Istruzioni di INPUT e GOTO.
- LEZIONE 3** = Elaborazione in DOPPIA PRECISIONE. Comandi di NEW, LIST, LLIST, BREAK, RUN, SAVE, DIR, LOAD.
- LEZIONE 4** = Distinzione fra i livelli DOS e BASIC. Precedenze nei CALCOLI. Definizione dei COMANDI DIRETTI.
- LEZIONE 5** = FORMATTAZIONE. Comando FORMAT. Errori di formattazione.
- LEZIONE 6** = Copiatura dei dischi. Comando COPY. Errori di copiatura. Copia di FILES.
- LEZIONE 7** = Comandi di RENAME, KILL, FREE, LIB.
- LEZIONE 8** = Comandi di AUTO e RENUM.
- LEZIONE 9** = Concetto di «ricorsività». Ciclo di FOR..TO..STEP..NEXT.
- LEZIONE 10** = Istruzione INPUT (seconda parte). Scrittura di un PROGRAMMA.
- LEZIONE 11** = Comandi di MEM e CLEAR. Istruzioni IF..THEN..ELSE. Comando FRE.
- LEZIONE 12** = Definizione di VARIABILI STRINGA: CHR \$, ASC e LEN.
- LEZIONE 13** = Istruzioni LEFT\$, RIGHT\$, MID\$. Comandi REVON e REVOFF.
- LEZIONE 14** = Istruzioni PRINT TAB e INKEY\$.
- LEZIONE 15** = Scrittura di un programma.
- LEZIONE 16** = Istruzioni di GOSUB e RETURN. Come COMPATTARE un PROGRAMMA. Concetto di CONCATENAMENTO.
- LEZIONE 17** = Istruzioni LPRINT, STR\$, VAL, STRING\$, INSTR.
- LEZIONE 18** = Concetto di FILE, RECORD e BUFFER. Gestione dei FILE SEQUENZIALI (prima parte).
- LEZIONE 19** = Gestione dei file sequenziali (seconda parte).
- LEZIONE 20** = Istruzione di ON ERROR GOTO e RESUME. Scrittura di un programma.
- LEZIONE 21** = Spiegazione del programma.
- LEZIONE 22** = Comando SAVE (seconda parte), MERGE, LIST (da DOS) e PRINT (da DOS).
- LEZIONE 23** = Funzioni di EDITING (1) : LIST, <VIRGOLA>, <PUNTO>, DELETE, EDIT.
- LEZIONE 24** = Funzioni di EDITING (2) : <L>, <SPAZIO>, <BACKSPACE>, <S>, <C>.
- LEZIONE 25** = Funzioni di EDITING (3) : <D>, <I>, <ESC>, <A>, <X>.
- LEZIONE 26** = Funzioni di EDITING (4) : <H>, <K>, <E>, <Q>, <N>.
- LEZIONE 27** = Comandi speciali da tastiera (prima parte).
- LEZIONE 28** = Comandi speciali da tastiera (seconda parte). Comandi LCOPY, STOP e CONT.
- LEZIONE 29** = FILE INTERNI. Istruzioni di DAT, READ e RESTORE.
- LEZIONE 30** = Istruzioni di INT e LINE INPUT.
- LEZIONE 31** = Scrittura di un programma e utilizzo di programma.
- LEZIONE 32** = Spiegazione del programma (prima parte).
- LEZIONE 33** = Spiegazione del programma (seconda parte). Istruzioni di REV, REVERS e KEYIN\$.
- LEZIONE 34** = Comandi di SIZE, BLINK, UNDER, HLIGHT, CURSOR, GRAPH.
- LEZIONE 35** = Comandi di SCROLL, MODE, BEEP e WAIT.
- LEZIONE 36** = Funzioni grafiche: SET, RESET, POINT.
- LEZIONE 37** = Funzioni grafiche: SET-RESET-POINT LINE, SET-RESET-POINT SQR, SET-RESET-POINT CIRCLE.
- LEZIONE 38** = Istruzioni grafiche: ARCHI, ELLISSI e SPIRALI.
- LEZIONE 39** = Istruzioni di PRINT , PRINT USING, LPRINT USING. Introduzione delle VARIABILI MATRICIALI. Istruzione DIM.
- LEZIONE 40** = Il programma GEFAM. SUBROUTINE del programma GEFAM.
- LEZIONE 41** = Gestione dei file RANDOM. Istruzioni MKD\$, MKIS, MKSS, CVD, CVI, CVS, FIELD, LSET, RSET, GET e PUT.
- LEZIONE 42** = Programma GEFAM: ARCHIVI. Istruzioni di ON..GOTO e ON..GOSUB. Programma GEFAM: MENU'.
- LEZIONE 43** = Programma GEFAM: REGISTRAZIONE MOVIMENTI; BILANCI; STAMPA.
- LEZIONE 44** = Programma GEFAM: CODICI; AZZERAMENTO; CHIUSURA; UTILIZZO.
- LEZIONE 45** = Istruzioni di ACCEPT\$, KEY, TAKE, CURSORE, POS, SWAP e PAGE.
- LEZIONE 46** = Istruzioni di CHANGE, HTRANS, LTRANS, LDROP, RDROP, LTRIM, RTRIM, LPAD, RPAD, RND, RANDOM.
- LEZIONE 47** = Istruzioni di DEFDBL, DEFINT, DEFSGN, DEFSTR, CDBL, CINT, CSGN, DEFFN, PEEK, POKE, INP, OUT.
- LEZIONE 48** = Funzioni coseno (COS), seno (SIN), tangente (TAN), arcotangente (ATN), logaritmo (LOG), esponente (EXP), radice quadrata (SQR), valore assoluto (ABS), segno (SGN), FIX e PI.
- LEZIONE 49** = Funzioni e comandi sui file da disco: EOF (End Of File), LINEINPUT #, PRINT #. Istruzioni USING, LOC e RUN R.
- LEZIONE 50** = Definizione di ERR, ERL, ERROR, TRON, TROFF, FIND e REF.

S.O.N.E. SISTEMA OPERATIVO compatibile CP/M

Già da tempo avevamo pensato di predisporre un nuovo sistema operativo che risultasse CP/M compatibile, da utilizzare nel nostro computer Z.80, in sostituzione del NE/DOS.

Abbiamo studiato questo nuovo sistema operativo, da noi denominato "SONE" (Sistema Operativo Nuova Elettronica), perché, giustamente, i lettori ci hanno fatto presente che di software, vale a dire "programmi", in DOS se ne trovano pochissimi in commercio, mentre in CP/M ne esistono un'infinità, tale da soddisfare qualsiasi esigenza.

L'inconveniente che ora si presenta a chi vuole utilizzare in tutta la sua potenza il "SONE" consiste nel fatto che sotto CP/M sono oggi disponibili dei programmi che occupano molto più di 100 kilobyte di memoria (vedi alcuni programmi gestionali di contabilità generale), e quindi, per utilizzare tutto il software disponibile occorre una adeguata capacità di memoria di massa, ben maggiore ai 102 Kilobyte disponibili in ciascun dischetto sotto DOS.

Per risolvere questo problema, potevamo scegliere due diverse soluzioni:

1 - Modificare radicalmente la scheda del floppy disk-controller, per aumentare la capacità dei dischetti, utilizzando dei drive floppy a doppia faccia, doppia densità, ma così facendo avremmo ottenuto un massimo di 800 kilobyte per dischetto.

2 - Lasciare invariata la configurazione iniziale del computer e lo schema di base del floppy disk controller, utilizzando ancora gli stessi drive floppy, e prevedere la possibilità di aggiungere una unità a disco rigido (HARD-DISK WINCESTER), in grado di immagazzinare fino a **12 megabyte**.

La prima soluzione, se da un lato risolve in modo molto semplice il problema della memoria di massa, dall'altra costringe tutti coloro che intendono adottare il nuovo sistema operativo SONE compatibile ai programmi CPM, all'acquisto forzato di almeno un nuovo drive floppy.

Adottando invece la seconda soluzione, si ha il vantaggio di lasciare il computer nella sua configurazione originale, perciò i floppy-drive già in vostro possesso possono essere tranquillamente riutilizzati e solo chi fosse interessato ad espandere la memoria di massa, potrà semplicemente applicare un'unità a disco rigido, ottenendo così una capacità di memoria di massa pari a **12 megabyte**.

Quest'ultima soluzione, da noi prescelta, consente perciò di ampliare le prestazioni del computer senza apportarvi alcuna modifica, in quanto tutte le routine per la gestione dell'HARD-DISK sono già state inserite all'interno del sistema operativo, con la possibilità di aggiungere questa espansione di memoria solo se lo si riterrà indispensabile per il lavoro futuro.

Con questo nuovo sistema operativo SONE, siamo certi di offrire ai nostri lettori un progetto valido e pienamente affidabile.

COME SI INSTALLA IL NUOVO SISTEMA OPERATIVO.

La configurazione minima del computer, necessaria per installare il nuovo sistema operativo SONE compatibile, è la seguente:

SCHEDA CPU (LX.382)
INTERFACCIA VIDEO (LX.529)
INTERFACCIA FLOPPY DISK (LX.390)
ESPANSIONE DI MEMORIA (LX.392 o LX.386)

Per quanto riguarda l'espansione di memoria, questa deve risultare da almeno 40 kilobyte e perciò potrete utilizzare una scheda dinamica da 32 K + una scheda statica da 8 K, oppure due schede di memoria dinamica, una delle quali con soli 16 K, per un totale di 48 K.

È anche possibile aggiungere ai 48 K così ottenuti un'ulteriore espansione di memoria statica da 8 K ed ottenere la massima configurazione, pari a 56 K.

Per installare il nuovo sistema operativo SONE, è necessario inoltre apportare alcune modifiche alla scheda LX.390 del floppy disk controller, in quanto la eprom che ora dovremmo inserire nello zoccolo presente su tale scheda, è del tipo 2716, cioè una eprom da 2 Kilobyte, che presenta una differente zoccolatura rispetto alle 2708 fin'ora utilizzate. Troverete la descrizione completa di tali modifiche sulla rivista n. 101/102 pag. 46-47. Chi volesse poi completare il computer con un Hard-Disk, troverà tutte le note ad esso relative sulla stessa rivista 101/102 da pag. 66 a pag. 75.

COME PROCEDERE

Una volta eseguite tutte le modifiche sulla scheda floppy, reinseritela sul bus del vostro computer e, posto nel drive 0 il dischetto contenente il nuovo **sistema operativo SONE**, premete il tasto del reset. Così facendo, sul video apparirà la scritta:

NEW FIRMWARE Vers. FS rev. 4.2

Dopo qualche istante, si metterà in moto il drive del primo floppy e sul monitor potrete leggere:

40K N.E. New Disk System - Vers. FS rev 1.6

A >

A questo punto il nuovo sistema operativo risulterà presente all'interno della memoria ed il computer rimarrà in attesa di un vostro comando.

Il segnale di attesa è contraddistinto, come nel CP/M, dalla lettera maiuscola "A" seguita dal simbolo >, accanto al quale risulterà posizionato il cursore.

Tale lettera indica che il computer lavora utilizzando come memoria di massa il floppy drive A, cioè il primo drive chiamato, sotto DOS, drive 0.

Il sistema SONE, come il CP/M, contraddistingue i drive floppy con una lettera anziché con un numero e, pertanto, il **primo drive** sarà chiamato **A**, mentre il **secondo B**.

Quindi, se il simbolo di attesa è A> si dice che il "**disco corrente**" è il **primo drive** e pertanto, se non viene specificato in altro modo, tutte le operazioni di scrittura e di lettura da disco fanno sempre riferimento al disco A.

Per modificare l'assegnazione del disco corrente, è sufficiente scrivere il nome del drive che si vuole selezionare, seguito dal simbolo dei due punti (:) e digitare RETURN; ad esempio, chi dispone di due drive e vuole selezionare il secondo, dovrà digitare : semplicemente:

B: (RETURN)

e sul monitor apparirà il segnale di attesa:

B>

specificando così che, ora, il disco corrente è il secondo drive, cioè il drive B. D'ora in poi quindi, in tutte le operazioni relative ai floppy disk, utilizzeremo questa notazione per distinguere fra loro i drive, o per specificare su che drive sta lavorando il computer.

COME FORMATTARE UN DISCO E DUPLICARE IL SISTEMA OPERATIVO

Quando si dispone di un programma "importante" (ed il sistema operativo è il programma "più importante" fra tutti quelli di cui potete disporre), è sempre buona abitudine farne una o due copie di sicurezza, in modo che, nella malaugurata ipotesi che si rovini involontariamente un disco (se avvicinate il disco ad un campo negativo di un altoparlante si cancellerà), si possa subito disporre di un "duplicato".

Nel dischetto contenente il nuovo sistema operativo SONE, è presente un programma per la formattazione, ed uno per la duplicazione del sistema operativo su di un altro disco, denominati rispettivamente FFORMAT.COM e PPUTSYS.COM. Vediamo ora come dovremo utilizzarli in pratica.

NOTA: È possibile eseguire queste operazioni possedendo sia uno, sia due drive e le differenze nella procedura sono praticamente irrilevanti; per questo motivo non riportiamo due paragrafi distinti per le due diverse configurazioni del computer, in quanto risulterebbero praticamente identici e, come tali, ripetitivi.

La prima operazione da eseguire per duplicare il sistema operativo su di un altro dischetto, è quella di formattare il dischetto di copia e pertanto vediamo subito come utilizzare il programma FFORMAT, che esegue appunto tale funzione.

Inserite il dischetto contenente il sistema operativo nel drive A e digitate:

FFORMAT

seguito da RETURN. Così facendo verrà letto dal disco A il programma di formattazione e caricato nella memoria del computer.

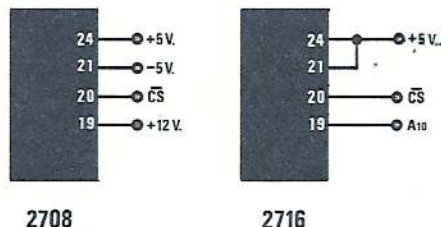


Fig. 104 Sostituendo nella scheda LX.390 la vecchia eeprom 2708 con la nuova 2716, siglata EP.683 perché già programmata, occorrerà modificare le connessioni dei piedini 21 e 19.

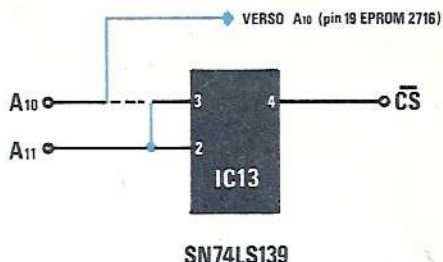


Fig. 105 Anche sull'integrato SN.74 LS139 (IC13) presente sempre nella scheda LX.390 dovrete collegare tra loro i piedini 2-3 ed interrompere il collegamento indicato A10.

Chi possiede un solo drive, dovrà ora estrarre dal drive A il dischetto originale ed inserire al suo posto il disco vergine da formattare, mentre, per chi ne possiede due, questa operazione non è indispensabile. Per maggior sicurezza, consigliamo ugualmente di sfilare dal drive A il dischetto originale per evitare, digliando comandi errati, di formattare quest'ultimo con evidenti, "deprecabili" conseguenze.

Il computer, prima di eseguire "fisicamente" la formattazione del disco, vi porrà alcune domande sulle caratteristiche del dischetto e sul tipo di formattazione da eseguire. Sul monitor infatti vi apparirà la scritta:

Floppy Disk formatter Vers. 4.0

Single or Double Side (S/D or ^ C to exit)?

Se il vostro drive è a singola faccia, risponderete digitando **S**, al contrario, se è a doppia, digiterete **D**. Pigiando **CTRL C**, come specificato fra parentesi nella scritta riportata sul video, uscirete dal programma di formattazione, ritornando al sistema operativo questo comando perciò sarà utile in un secondo tempo, terminata la formattazione.

Se pigerete il tasto **S** (per una singola faccia), oppure **D** (per una doppia faccia), sul video vi apparirà la scritta:

Do you want disk verify (Y/N)?

Letteralmente, la traduzione della frase riportata sul monitor è la seguente: "Vuoi la verifica del disco?". Digitando **Y** (cioè I, iniziale YES), il computer, terminata la formattazione del disco, eseguirà una verifica sul dischetto, per controllare se tutte le tracce del disco risultano formattate correttamente, mentre digitando **N** (che significa ovviamente NO), tale operazione non verrà eseguita.

Premendo un qualunque altro tasto, diverso da Y o da N, verrà riproposta la stessa domanda ed il computer rimarrà in attesa di una risposta corretta.

Scelta quindi una delle due possibilità di lavoro, con Y o con N, sul video vi apparirà la scritta:

Type disk to be format (A/B)?

che, tradotto letteralmente, significa: "Digita il nome del disco da formattare (A o B)".

Chi dispone di due drive ed ha inserito nel drive B il dischetto vergine da formattare, dovrà digitare **B**, mentre chi dispone di un solo drive, dovrà digitare **A**.

Per evitare errori di battitura, il computer richiederà ancora la conferma del nome del disco, riportando sul video la scritta seguente:

Are you sure for disk A (Y for YES. any key for no)?

cioè, tradotto letteralmente: "Sei sicuro del disco A? (Y per il SI, qualunque altro tasto per NO)". Ovviamente-

te, se avete digitato B, nella scritta sopariportata leggerete tale lettera al posto della lettera A da noi scelta come esempio.

A questo punto, digitando Y, inizierà la formattazione vera e propria del dischetto e sul monitor, in basso a destra, verrà riportato il numero della traccia che il computer sta formattando in quell'istante.

Giunto alla 39ª traccia, la formattazione sarà completata e, se avete selezionato la verifica della formattazione, verrà eseguita anche quest'ultima operazione, dopodiché il computer tornerà automaticamente all'inizio di tutta la procedura, riportando sul monitor la stessa domanda iniziale, cioè:

"Single or Double Side (S/D or ↑C for exit)".

Se volete formattare un altro dischetto sarà sufficiente ripetere le stesse operazioni fin qui esposte, mentre

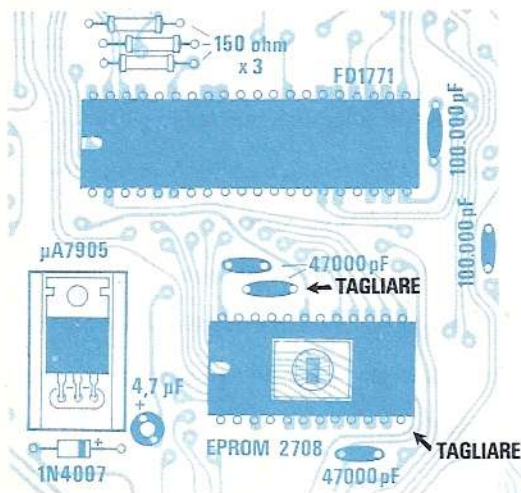
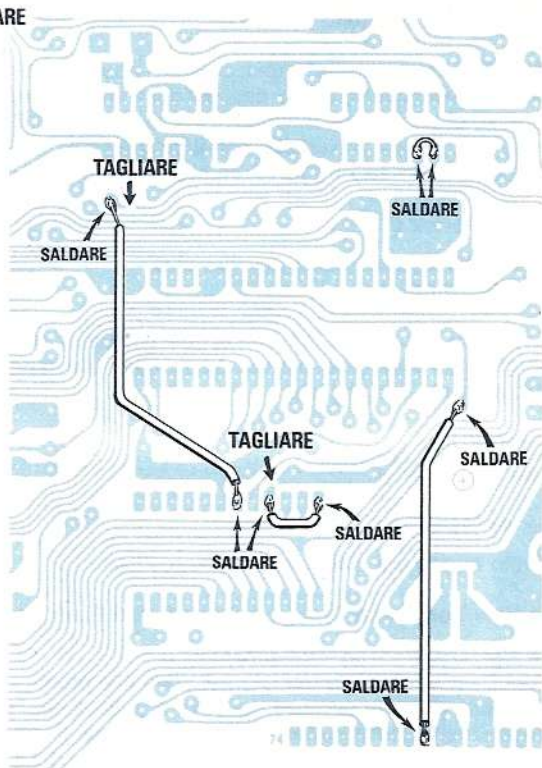


Fig. 106 Per effettuare le modifiche riportate in fig. 1 e 2, sulla parte superiore del circuito stampato LX.390 dovreste tagliare le due piste indicate con la freccia. Ovviamente la Eprom 2708, che in precedenza poteva risultare la EP.1390 o la EP.2390, andrà sostituita con la 2716 da noi siglata EP.683.

Fig. 107 Per completare la modifica sulla scheda LX.390 dovreste tagliare, sul lato opposto, le due piste sempre indicate dalle frecce, poi saldare tra loro i piedini 2-3 dell'integrato SN.74LS139 (vedi in alto a destra), i piedini 21-24 della eprom EP.683 (vedi al centro), infine collegare con due spezzoni di filo di rame isolato in plastica le piste chiaramente visibili in tale disegno.



se desiderate uscire da questo primo programma, dovrete **reinserire** il dischetto originale del "sistema operativo SONE nel drive A e quindi digitare **CTRL C**.

Dopo pochi istanti, riapparirà sul monitor il simbolo di attesa **A>** e potrete passare al programma per la duplicazione del sistema operativo semplicemente digitando:

PPUTSYS

seguito, al solito, da **RETURN**. Così facendo richiamerete il secondo programma contenuto nel dischetto originale, programma che verrà caricato nella memoria del computer al posto dell'**FFORMAT** appena abbandonato.

A questo punto dobbiamo aprire una breve parentesi per chiarire alcune particolarità da noi inserite in questo programma di duplicazione.

Molto spesso infatti, abbiamo riscontrato la presenza di innumerevoli copie "clandestine" di programmi o di sistemi operativi eseguiti spesso senza troppa cura su dischetti economici e senza nemmeno una semplice verifica di controllo, al solo scopo di lucro.

A questo proposito ci siamo accorti **che i dischetti** di cui i nostri lettori lamentavano difetti ed errori, attribuendoci la responsabilità della loro pessima qualità (sporcavano le testine), non erano gli stessi che noi avevamo messo in distribuzione.

Per risolvere questo problema, abbiamo inserito all'interno del SONE una protezione, che permetterà di duplicare 10 copie del sistema per tutte le vostre future esigenze. In questo modo sarete sempre protetti da copie inaffidabili e da duplicazioni non originali evitando, sin dall'inizio, spiacevoli spese di riparazione ed assistenza.

Il modo per ottenere uno o più duplicati del sistema operativo SONE è il seguente:

- Dal disco originale potrete ottenere 4 copie al massimo;
- Dalle prime due copie così ottenute, potrete ottenere altre 3 copie per ciascun dischetto, non più duplicabili.

In totale quindi, 4 copie dall'originale + 3 copie dal primo duplicato + 3 copie dal secondo duplicato, per un totale di 10 copie al massimo.

Detto questo, consideriamo ora come utilizzare nella pratica questo programma di duplicazione del sistema operativo, chiamato **PPUTSYS**.

NOTA: Il programma di copia, durante l'esecuzione delle operazioni, deve inizialmente accedere al **disco A** e scrivere su questo alcuni dati, per cui è **indispensabile togliere la fascetta di protezione alla scrittura posta su di un lato del dischetto**, in caso contrario, il computer darà una segnalazione di errore ed ovviamente non potrete proseguire con il programma.

Dopo aver scritto **PPUTSYS** ed aver digitato **RETURN**, sul monitor apparirà:

PutSys Vers.2.0

**Chose your system configuration size
from following ones! 40,48,56**

In pratica il computer richiede la configurazione della memoria, per adattare a questa il **nuovo sistema operativo da duplicare**, per cui, a seconda delle capacità di K delle schede di memoria inserite sul vostro BUS, se avete 40K digitate:

40 e RETURN

se ne avete 56K digitate:

56 e RETURN

Fatto questo sul video apparirà la scritta:

Set New Formatted Disk on Drive A and Push any Key

che significa, tradotto letteralmente:

"Poni il nuovo disco formattato nel drive A e premi un qualunque tasto".

Pertanto **dovrete estrarre** dal drive A il disco del "sistema operativo SONE" ed inserire al suo posto il **nuovo dischetto formattato** da duplicare. Fatto questo, premete un tasto qualunque. Terminata la duplicazione, il computer ricaricherà il sistema operativo dal disco appena duplicato, in modo da avere l'immediata conferma della correttezza di tutte le operazioni.