16 K di BASIC

Avere un BASIC da 16 K, residente su EPROM, sempre a disposizione del programmatore senza doverlo caricare da nastro o da disco è, oltre che una grande comodità d'uso, una nuova ed importante espansione del vostro computer Z80 N.E.

Chi possiede il computer Z80 N.E. nella versione con tastiera e monitor video ma senza floppy-disck, è costretto a servirsi del mini - Basic da 5,6 K disponibile su cassetta che, oltre ad avere un limitato numero di istruzioni, è anche abbastanza scomodo da usare in quanto bisogna ogni volta ricaricarlo, usando un registratore a cassetta.

Utilizzando la scheda che ora presentiamo, avrete immediatamente a disposizione, appena acceso il microcomputer, un Basic da 16 K completo di ogni istruzione e perciò l'uso risulterà decisamente più comodo e veloce.

Così volendo utilizzare il microcomputer in tutta la sua "potenza" di programmazione senza dover acquistare un floppy-disk, basta realizzare la scheda del Basic residente che offre la possibilità di realizzare un microcomputer con le sole schede LX380 (alimentatore), LX.381 (BUS), LX.382 (scheda CPU), LX.386 (espansione da 8K RAM o LX.392 (espansione da 32K RAM dinamiche), LX.387 (tastiera alfanumerica), LX.388 (interfaccia video).

In questo Basic abbiamo implementato delle funzioni specializzate per la gestione "ottimizzata" di qualsiasi registratore a cassetta che diviene in questa nuova configurazione del computer, una versatile unità di memoria esterna, di facile e comodo uso.

La scheda, come spiegheremo più dettagliatamente nel corso dell'articolo, è predisposta per essere innestata direttamente nel BUS del computer senza alcuna difficoltà.

Con questo nuovo progetto, il microcomputer Z80 N.E. diventerà un completo mezzo sia di studio che di sperimentazione pratica per tutti coloro che vogliono approfondire o iniziare le proprie conoscenze nel mondo dei computer e soprattutto nell'ambiente dei programmi, cioè nel SOFT-WARE, dove il BASIC, oggi, è praticamente il linguaggio più diffuso e conosciuto.

Assieme alle funzioni di programma tipiche del Basic stesso, all'interno di questa espansione sono presenti alcune funzioni MONITOR, che permettono di utilizzare il microcomputer anche a "livello di linguaggio macchina" per crearvi delle subroune o dei sottoprogrammi più veloci rispetto ad Istruzioni Basic che potranno poi essere in seguito richiamate da Basic.

Al termine dell'articolo troverete una lista completa e dettagliata di tutte le istruzioni del Basic residente in memoria e tale lista potrà esservi utile come "memorandum" riassuntivo per una rapida consultazione delle istruzioni del linguaggio o per apprendere dall'inizio la sintassi del linguaggio stesso.

SCHEMA ELETTRICO

La parte principale di questo circuito è costituita da un "banco" di 8 memorie EPROM (da IC1 a IC8) tipo 2516, nelle quali risiede il programma Basic da **16 Kilobyte.**

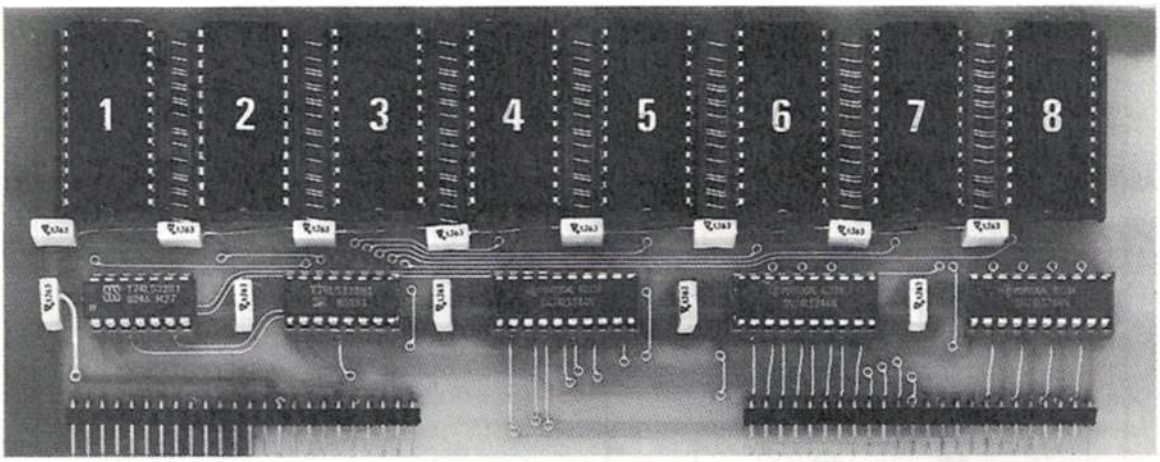
Essendo ciascuna memoria da 2 Kilobyte, la capacità totale della scheda è appunto di:

2 x 8 = 16 K

Per chiarezza di disegno, seguendo una consuetudine diffusa in tutti gli schemi elettrici nei quali sono presenti catene di integrati identici collegati tra di loro in parallelo, abbiamo riportato il numero dei piedini **solamente nella prima e nell'ultima memoria,** lasciando sottointeso che tale numerazione è analoga per tutte le altre memorie da IC1 ad IC8. Infatti, i piedini degli indirizzi e quelli dei dati nelle memorie di un computer, risultano collegati in parallelo su ciascun integrato cioè il piedino corrispondente all'indirizzo 0 (piedino 8) è collegato al piedino corrispondente su tutte le altre memorie ed anche, ovviamente, al BUS degli indirizzi del computer.

Per sapere quali sono i piedini d'ingresso degli indirizzi della memoria 2516, ne riportiamo di seguito la piedinatura:

INDIRIZZI	Piedino
0	8
1	7
2	6
3	5
4	4
5	3
6	2
7	1
8	23
9	22
10	19



residenti su EPROM

Analogamente, i piedini di uscita dei dati nella stessa memoria risultano i seguenti:

DATI	Piedino
0	9
1	10
2	11
3	13
4	14
5	15
6	16
7	17

Detto questo possiamo ora ritornare allo schema elettrico di fig. 1.

L'integrato IC9, presente in alto a sinistra è un TTL tipo 74LS244, nel cui interno sono presenti otto amplificatori di linea (Buffers). Questo integrato serve per prelevare dal BUS gli indirizzi del computer, dalla linea A0 alla linea A7, ad amplificarli e squadrarli ed infine inserirli nelle memorie.

Analogamente IC10, un TTL identico al precedente, serve ad amplificare i rimanenti indirizzi relativi alle linee da A8 ad A15. Gli indirizzi A8 - A9 -A10 vengono inviati sugli ingressi di indirizzamento delle EPROM (piedini 23-22-19) mentre A11-A12-A13-A14-A15, raggiungono la "logica di decodifica" degli indirizzi, costituita dall'OR IC11/A e dall'integrato IC12 SN74LS138.

Più dettagliatamente, IC12 serve ad indirizzare una delle otto EPROM presenti nella scheda, attivando il piedino di selezione della EPROM corrispondente (vedi pin 20 di IC1-IC8).

Le uscite 15 - 14 - 13 - 12 - 11 - 10 - 9 - 7 di IC12 fanno capo al piedino 20 di ognuna delle otto EPROM presenti nella scheda.

Il secondo integrato utilizzato nella "logica di decodifica" è IC11/A, un OR contenuto in un TTL tipo 74LS32. L'uscita di questa porta logica, sul piedino 8, è collegata sia all'ingresso di abilitazione del decodificatore degli indirizzi appena descritto (IC12) per comandarne l'attivazione o lo spegnimento in corrispondenza della zona utile di memoria che al piedino 5 dell'OR IC11/B la cui uscita (piedino 6), risulta collegata ai piedini di abilitazione 1 e 19 di IC13, un TTL tipo 74LS244 del tutto identico ad IC9 e IC10, qui utilizzato per amplificare i dati provenienti delle memorie EPROM della scheda ed applicarli al BUS dei dati del computer.

I terminali riportati ai due lati del circuito e siglati a sinistra 1B-2B-3B ecc., e a destra 24B - 23B - 22B ecc., sono in numeri di riferimento dei piedini del due connettori fissati in basso sulla scheda e corrispondono a quelli del BUS.

L'alimentazione per tutti gli integrati, viene prelevata tramite i due connettori di collegamento sul BUS del computer, quindi, inserendo la scheda nei connettori, viene automaticamente prelevata la tensione stabilizzata a 5 volt necessaria al circuito.

Terminata la descrizione dello schema elettrico, passiamo direttamente alla realizzazione pratica.

REALIZZAZIONE PRATICA

Per la realizzazione di questo progetto è necessario il circuito stampato a doppia faccia, con fori metallizzati, siglato LX.548.

Disponendo di tale stampato, il montaggio risulterà molto semplice in quanto la metallizzazione presente all'interno dei fori garantisce un perfetto

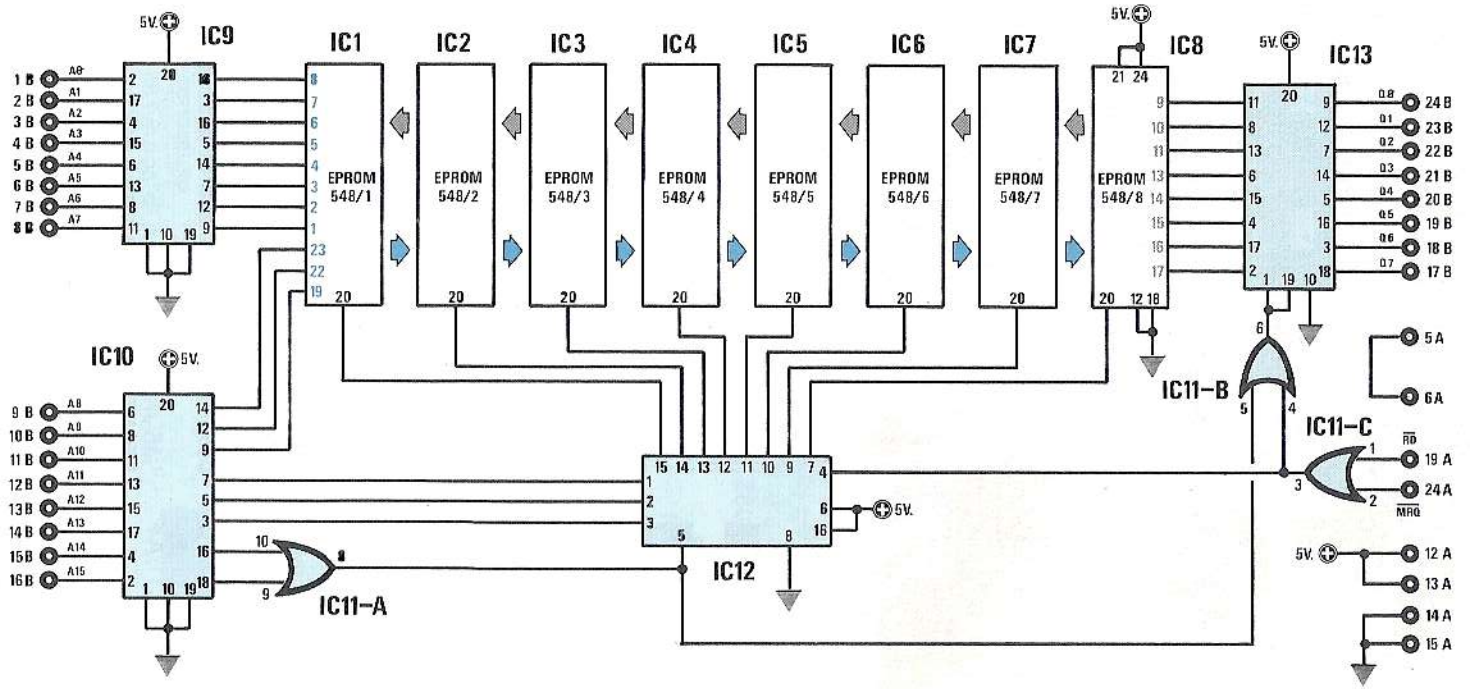


Fig. 1 Schema elettrico della scheda contenente, su 8 Eprom, il Basic da 16K. In tale scheda non sono stati inseriti i tredici condensatori da 100.000 pF applicati in parallelo sull'alimentazione di ogni integrato presente nel circuito come invece appaiono in fig. 2.

ELENCO COMPONENTI LX.548

IC1 = MM 2516 programmata (548-1)
IC2 = MM 2516 programmata (548-2)
IC3 = MM 2516 programmata (548-3)
IC4 = MM 2516 programmata (548-4)
IC5 = MM 2516 programmata (548-5)
IC6 = MM 2516 programmata (548-6)
IC7 = MM 2516 programmata (548-7)
IC8 = MM 2516 programmata (548-8)
IC9 = SN74LS244
IC10 = SN74LS244
IC11 = SN74LS32
IC12 = SN74LS138
IC13 = SN74LS244

collegamento elettrico fra le piste superiori e quelle inferiori.

Prima di iniziare il montaggio, controllate con una lente o in controluce, che sul circuito stampato non esistano delle piste in corto. Difetti del genere non dovrebbero mai presentarsi dato che, prima della consegna, ogni circuito viene controllato con un ingranditore per 10 ma non è da escludere che, alla persona addetta a tale controllo, su migliaia di stampati gliene possa sfuggire uno con una pista in corto e, se questo dovesse capitare proprio a voi, adottando l'accorgimento poc'anzi accennato, ve ne accorgereste immediatamente evitando così di perdere giorni interi nella ricerca del mancato funzionamento del circuito. Naturalmente, se dovesse accadervi una cosa del genere, riportate lo stampato al vostro fornitore, o se lo avete ricevuto per Posta, rispeditelo a noi, e vi verrà sostituito immediatamente.

Ora, date inizio alla realizzazione pratica saldando dapprima gli zoccoli per gli integrati e quindi i due connettori a 24 poli, necessari per innestare la scheda nel BUS del computer. Come vedesi nella foto e nei disegni, questi connettori vanno inseriti sullo stesso lato degli zoccoli.

Fatto questo inserite e saldate tutti i condensatori al poliestere e poichè questi risultano tutti di identica capacità non avrete il problema di identificarne il valore riportato sull'involucro.

A questo punto inserite negli zoccoli gli integrati ed è questa l'unica operazione che merita un attimo di attenzione da parte vostra.

Infatti, inserendo a rovescio uno di questi integrati, il circuito non potrà funzionare.

Rispettate perciò il verso di inserzione, posizionando la tacca di riferimento o il puntino presente in sua sostituzione in corrispondenza del piedino 1, come è riportato nello schema pratico di fig. 2.

Nell'inserire le memorie negli zoccoli, ricordate oltre che il verso di inserzione, di rispettare anche la sequenza della numerazione riportata su di esse inserendo quella con l'etichetta siglata 548-1 nel primo zoccolo a sinistra, come vedesi in fig. 2 (tenendo la scheda con i due connettori da 24 poli (A e B) rivolti verso il basso) e proseguendo in sequenza da sinistra verso destra, con la seconda, siglata 548-2, poi la terza, siglata 548-3 e così via fino all'ottava memoria, siglata 548-8.

Terminato il montaggio prima di inserirla nel BUS del vostro microcomputer dovrete eseguire le semplici operazioni di installazione che spiegheremo nel prossimo paragrafo.

PER COLLEGARE IL CIRCUITO AL COMPUTER

Il programma del linguaggio Basic da 16K inserito in questa scheda, è stato programmato a partire dalla **locazione 0000** fino alla **locazione 16.383** in decimale (che corrispondono alle locazioni da 0000 a 3FFF in esadecimale) e poichè originariamente, l'indirizzo di partenza nalla scheda CPU LX.382 è stato fissato alla **locazione 32.678** in decimale (che corrisponde alla locazione 8000 in esadecimale) bisogna quindi modificare tale scheda affinchè parta dalla locazione 0000. Tali modifiche sono comunque molto semplici e, per maggiore chiarezza le abbiamo riportate dettagliatamente in fig. 3 e fig. 4, come particolari dello schema pratico di montaggio apparso sul n. 68 a pag. 116.

In pratica, le modifiche da effettuare sono le seguenti:

1 - Sollevare dallo zoccolo il piedino 6 dell'integrato SN74LS00 (IC.8)
2 - Togliere l'integrato SN74LS109 (IC.13)
3 - Ponticellare a massa il piedino 10 dello zoccolo di tale integrato.

In questo modo, quando accenderete il computer, la CPU andrà automaticamente a leggere, come indirizzo di partenza, il contenuto della cella di memoria **0000**, indirizzo che corrisponde all'inizio del programma del linguaggio BASIC.

Ricordiamo che con il Basic presentato in questo articolo, **non è possibile utilizzare nè la scheda video-grafica (LX.529) nè l'interfaccia per Floppy-Disk (LX.390).**

Questa caratteristica è stata espressamente voluta in quanto, con la video-grafica ed il Floppy-Disck, è già disponibile da tempo il BASIC + DOS.

Questa scheda serve quindi per coloro che vogliono realizzare un microcomputer in versione economica ma ugualmente potente evitando di dover acquistare dei floppy-disck e sfruttando solo dei normali registratori a cassetta. Il computer in questa versione è perciò composto da:

1) BUS + alimentatore
2) Scheda CPU (LX.382) con le modifiche sopraindicate
3) Interfaccia video (LX.382)
4) Basic residente (LX.548)
5) Espansione di memoria dinamica (LX.392)
6) Tastiera alfanumerica (LX.387)

Per quanto riguarda l'espansione di memoria, bisogna considerare che, sul BUS del computer, è già presente la scheda del BASIC residente che occupa le locazioni dalla 0 alla 16.383 e perciò, in tale zona di memoria, non dovrà essere posta ovviamente nessun'altra scheda di espansione.

In pratica, utilizzando la scheda di espansione di memoria dinamica LX.392 dovrete eseguire i ponticelli P2 e P3 lasciando aperto il ponticello P4. Così facendo avrete a disposizione 32K di memoria dalla **locazione 16384** (che corrisponde alla locazione 4000 in esadecimale) fino alla **locazione 49151** (che corrisponde alla locazione BFFF in esadecimale). A nostro avviso tale capacità di memoria è più che sufficiente per qualsiasi programma scritto in questo linguaggio.

Chi volesse comunque espandere ulteriormente la memoria dovrà **necessariamente** utilizzare espansioni di memoria statiche (LX.386) ed indirizzarle dalla cella C000 (in esadecimale) in poi (vedi pag. 122 riv. 70 scheda n. 7).

Inoltre chi utilizza solamente le schede di memoria statica deve eseguire su tale scheda i ponticelli di indirizzamento per destinare la memoria dalla locazione da 4000 (in esadecimale) in poi (vedi pag. 121-122 riv. 70).

Una ultima precisazione: le schede di memoria statica hanno un consumo di corrente maggiore rispetto alle dinamiche e perciò è sconsigliabile utilizzare più di tre schede statiche contemporaneamente.

LE ISTRUZIONI DEL LINGUAGGIO BASIC

Perchè risulti comprensibile ciò che si vuole spiegare, è indispensabile, a nostro avviso, riportare le istruzioni del linguaggio Basic, non solo per leggerle ma anche per controllarle direttamente sul computer.

Solo in questo modo infatti, verificando subito ciò che state leggendo, imparerete più facilmente e con magggior profitto, le varie struzioni del linguaggio.

L'elenco di istruzioni che troverete di seguito, non possono nè vogliono essere una "lezione teorica di BASIC" ma solo un manuale di facile e rapida consultazione per imparare o per richiamare alla memoria la "sintassi" di questo linguaggio.

Inserite quindi la scheda nel BUS del vostro microcomputer e una volta fornita l'alimentazione, premete il tasto di reset e sul monitor apparirà la scritta:

MEMORY SIZE ?

Il computer chiede quanta memoria volete mettere a disposizione del BASIC. Con questa semplice procedura di inizio, si ha la possibilità di lasciare dello spazio di memoria libero per poter scrivere programmi in linguaggio macchina, richiamabili poi, come vedremo più avanti, anche durante l'esecuzione di un programma in BASIC.

Digitando semplicemente RETURN, il computer assegna automaticamente al BASIC tutta la memoria che trova disponibile (cioè tutte le schede di espansione RAM da voi inserite) altrimenti, scrivendo un numero decimale **maggiore di 18.000**, ad esempio 30.000, viene assegnato uno spazio di memoria pari al numero scritto meno circa 17.000 celle che sono impegnate rispettivamente:

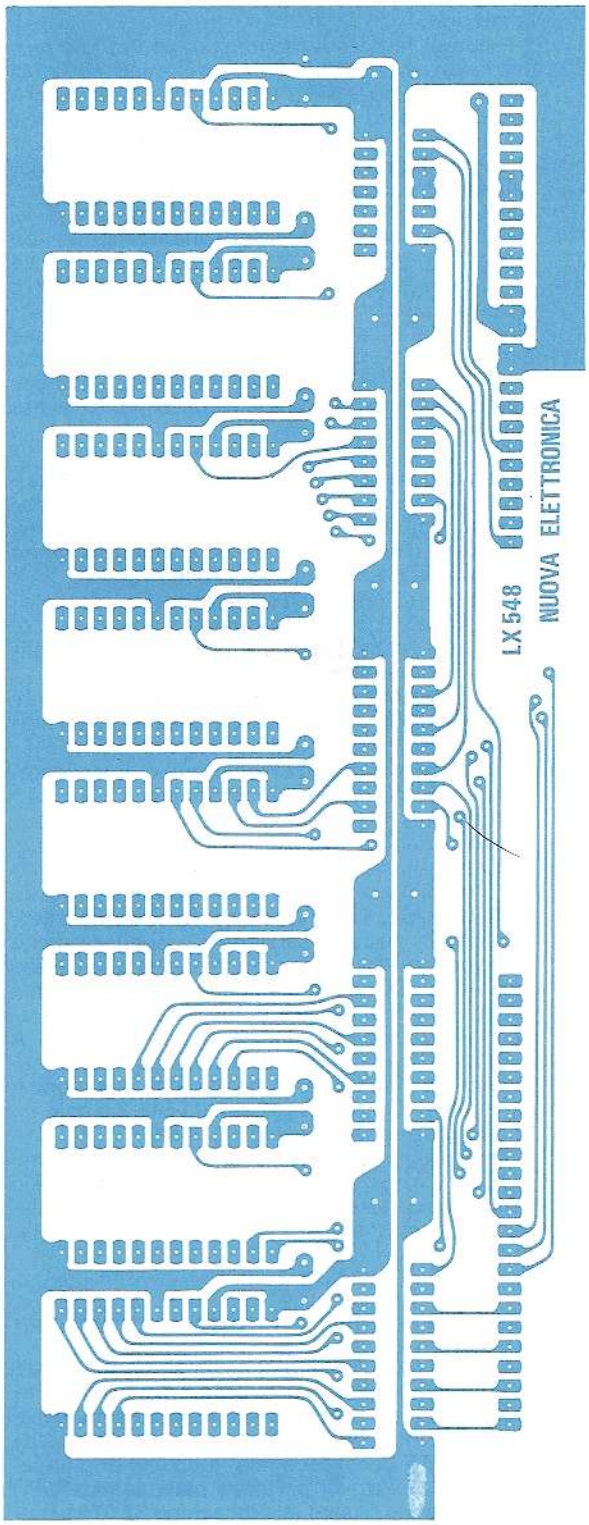
— 16.384 dall'espansione di memoria del BASIC
— circa 1 Kbyte di memoria per alcuni vettori di sistema del BASIC stesso.

Scrivendo quindi 30.000 viene assegnato ai programmi BASIC uno spazio di memoria di circa 13.000 celle di memoria. (Per l'esattezza 12.803 byte).

Dopo aver digitato RETURN o il numero corrispondente alla memoria disponibile e RETURN, sul monitor apparirà la scritta:

NUOVA ELETTRONICA BASIC 16
USED LINEFEED (Y/N) ?

La seconda domanda che il computer rivolge serve per inizializzare il sistema a seconda della stampante usata. Esistono infatti due modi di



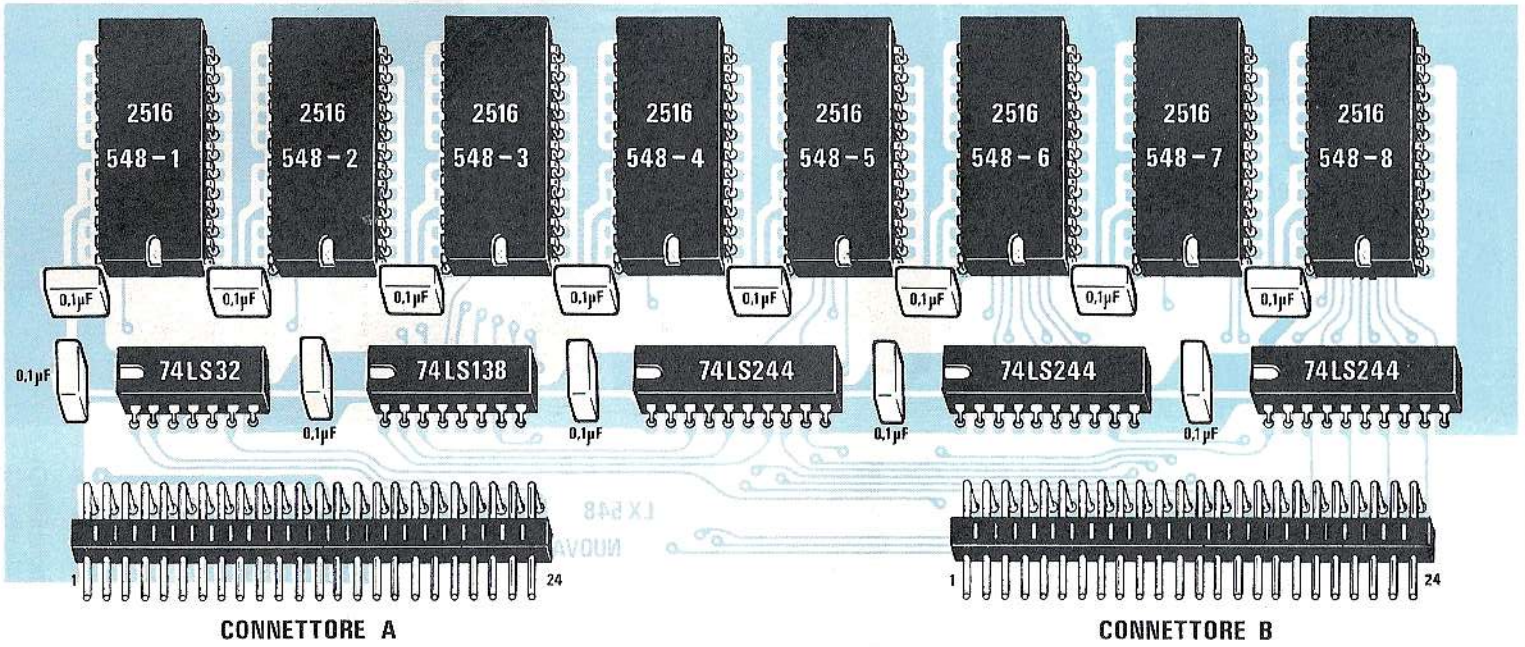


Fig.2 Vi ricordiamo che il circuito stampato (in alto è visibile il lato su cui effettuare le saldature) utilizzato per questa realizzazione è un doppia faccia con fori metallizzati, quindi le piste sui due lati sono già elettricamente collegate tramite la metallizzazione presente su ogni foro. Nello schema pratico qui riportato potrete vedere in che posizione vengono applicati i tredici condensatori da 100.000 pF. e in che ordine dovrete inserire le otto Eprom, rispettando la tacca di riferimento, sullo stampato.

stampare un testo e la differenza fondamentale consiste nell'avanzamento o meno della carta ogni volta che il computer invia un comando di ritorno della testina (RETURN). Nella prima ipotesi, la stampa avviene con LINEFEED automatico mentre nel secondo caso non si ha un LINEFEED automatico e l'avanzamento della carta deve essere comandato dal computer.

A questo punto dovrete digitare N se avete una stampante con il LINEFEED automatico, oppure Y in caso contrario. Digitando un qualunque altro tasto, riapparirà sul monitor la stessa scritta sopra-riportata. Digitate quindi o Y o N e, fatto questo, entrerete nel programma BASIC vero e proprio e perciò potrete iniziare ad utilizzare tutte le istruzioni riportate dettagliatamente qui di seguito. Per facilitare la ricerca in questa tabella riassuntiva delle istruzioni, le abbiamo elencate in ordine alfabetico ed inoltre, per renderne più chiaro il significato, abbiamo riportato accanto a queste, degli esempi "operativi" che, ricopiati fedelmente sul vostro computer, vi daranno la possibilità di constatare personalmente il modo di operare delle istruzioni stesse.

ABS (n) - Calcola il "valore assoluto" del numero **n** e lo pone nella variabile assegnata.

Per verificare tale istruzione ricopiate le seguenti linee di programma:

A = ABS (-23,5) (return)
PRINT A (return)
23,5 (È il valore contenuto nella variabile A)

AND - È un'istruzione che simula l'operazione effettuata da una porta logica tipo AND. Il risultato di questa operazione binaria è 1 se entrambi gli operandi valgono 1 ed è 0 in tutti gli altri casi.

Quando tale istruzione è utilizzata all'interno di un'istruzione IF, allora impone la verifica di entrambe le condizioni presenti nell'istruzione stessa (Vedi nel seguito l'istruzione IF)

ASC ("carattere") - Fornisce il valore decimale corrispondente al carattere ASCII scritto entro le parentesi. Ponendo nelle parentesi un carattere fra virgolette, viene fornito direttamente il corrispondente decimale del carattere mentre, scrivendo una variabile stringa, viene fornito il valore del PRIMO CARATTERE della stringa stessa, cioè se la variabile stringa è A$=DRT, il valore decimale fornito in uscita è quello della lettera D.

Esempio 1:

PRINT ASC ("A") (return)
65

Esempio 2:

A$ = "DRT" (return)
PRINT ASC (A$) (return)
68
(Che è il valore ASCII in decimale della lettera D)

ATN (numero) - Calcola il valore dell'arcotangen-

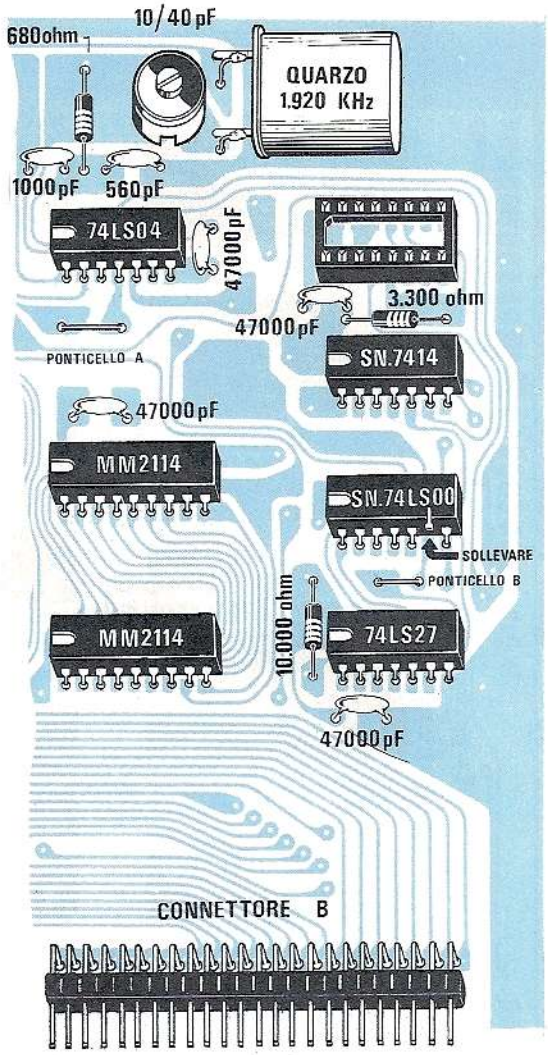


Fig.3 Inserendo nel Bus questa scheda contenente il Basic da 16K, per far sì che il computer funzioni regolarmente, dovrete effettuare, sulla scheda CPU LX.382, alcune piccole modifiche. In questo disegno abbiamo riprodotto il solo settore interessato di tale scheda, dove dovrete togliere dallo zoccolo l'integrato 74LS109 posto sotto al "quarzo" (vedi in alto lo zoccolo sprovvisto dell'integrato) poi sollevare il piedino 6 dall'interno dello zoccolo dell'integrato 74LS00.

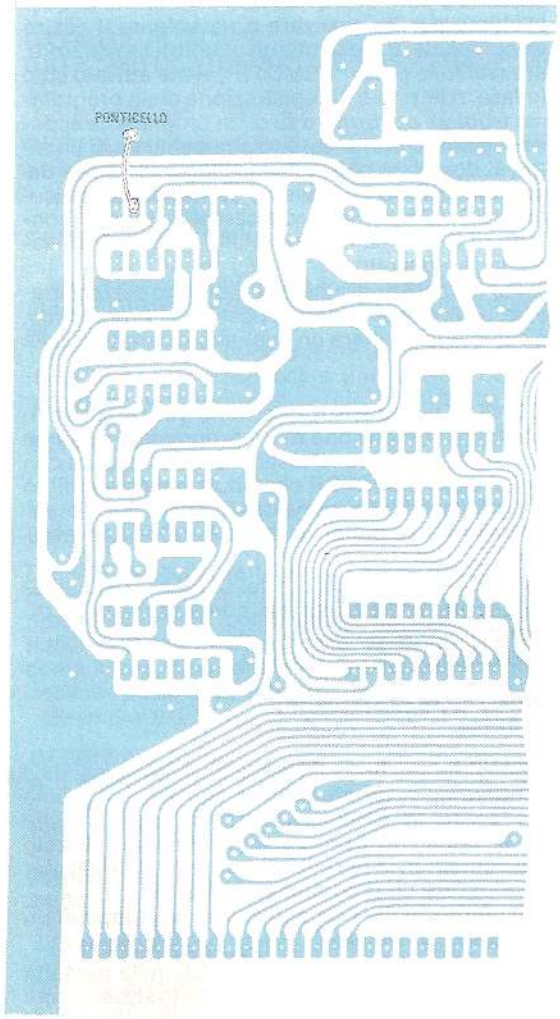


Fig.4 Dopo aver tolto l'integrato 74LS109 e sollevato il piedino sull'integrato 74LS00 (non consigliamo di tagliarlo per non avere un integrato utilizzabile solo per questa scheda) voltate la scheda dal lato opposto poi, come vedesi in alto su questo disegno, collegate con un corto spezzone di filo di rame a massa il piedino 10 dello zoccolo dove prima risultava inserito l'integrato stesso.
Consigliamo di tenere questo filo leggermente sollevato dallo stampato per non creare dei cortocircuiti.

te, espresso in radianti, del numero o della variabile numerica, scritta entro la parentesi.

Esempio 1:

PRINT ATN (3) (return)
1.24905

Esempio 2:

A = 5 (return)
PRINT ATN (A) (return)
1.3734

Esempio 3:

PRINT ATN (1)*4
3.14159

AUTO

È un'istruzione utile durante la scrittura dei programmi in quanto consente di ottenere una numerazione automatica delle linee di programma.

Digitando AUTO e RETURN sul video apparirà il numero 10 ed il cursore si posizionerà subito a destra di tale numero. Scrivendo una qualunque istruzione e RETURN, nella riga successiva apparirà il numero 20 ed il cursore si posizionerà subito a destra di tale numero.

Per uscire da questa procedura di autonumerazione è sufficiente premere contemporaneamente il tasto CONTROL (CTRL) e il tasto C.

L'autonumerazione può essere ottenuta anche in modo diverso da quello descritto, scrivendo l'istruzione AUTO, nella seguente forma:

AUTO n1, n2

dove n1 è il numero di linea da cui deve iniziare la autonumerazione ed n2 è l'incremento da dare linea per linea. Ad esempio, scrivendo:

AUTO 300,20

Si ottiene come prima linea di programma la linea 300 e, ad ogni RETURN, avrete di seguito 320-340-360 ecc.

CDBL (espressione) - Definisce in doppia precisione il termine seguente entro parentesi.

Esempio:

PRINT CDBL(10)/3 (return)
3,333333333333333

Nota: Avendo definito 10 come numero in doppia precisione, tutto il calcolo viene fatto in doppia precisione e perciò il risultato ottenuto è esatto.

Scrivendo invece PRINT CDBL (10/3) verrebbe prima calcolata l'operazione entro parentesi, normalmente eseguita in singola precisione, e poi stampato un numero in doppia precisione, con quindici cifre decimali, nel quale le ultime nove cifre, non essendo state calcolate nella prima operazione eseguita, non risulterebbero esatte.

CHR$ (codice) - Definisce il carattere ASCII corrispondente al codice decimale scritto entro la parentesi.

Esempio:

PRINT CHR$(65) (return)
A

Infatti il carattere "A" corrisponde, in ASCII, al numero decimale 65.

CINT (numero) - Approssima per difetto il valore del numero entro parentesi e accetta valori numerici compresi fra — 32768 e + 32767.

Esempio 1:

PRINT CINT (123.4) (return)
123

Esempio 2:

PRINT CINT (-238.5) (return)
-239

CLEAR numero - È un comando particolare ed è usato per riservare uno spazio di memoria per le "variabili stringa" pari al "numero" scritto a fianco dell'istruzione stessa. Normalmente tale spazio è 50, cioè sono riservati 50 byte per la memorizzazione delle stringhe.

CLOAD - Questa istruzione è utilizzata per leggere i programmi registrati su cassetta, tramite l'interfaccia per registratore LX.385.
L'struzione può essere scritta in varie forme:

- **CLOAD** (return) = Il computer legge il primo programma che incontra proveniente dal registratore e lo carica nella memoria. Ciò che era eventualmente preesistente in memoria, viene automaticamente cancellato.
- **CLOAD "A"** (return) = Il computer legge tutti i programmi provenienti dal registratore fino a quando non incontra il programma nominato "A". Letto tale programma, lo carica in memoria e termina la lettura. Il contenuto precedente della memoria viene anche in questo caso cancellato.
- **CLOAD?** = Il computer legge il primo programma proveniente dal registratore e lo confronta con quello presente in memoria, senza cancellare nulla di ciò che è presente in memoria. Al termine del confronto, viene stampato READY se non sono state riscontrate differenze oppure BAD in caso contrario. Questa istruzione è molto utile per verificare, dopo un salvataggio di un programma, se la copia ottenuta sul nastro è esente da errori.
- **CLOAD? "A"** = Analogamente a prima, viene letto e confrontato il programma proveniente dal registratore ma ora questa operazione si svolge solo quando, dal registratore, giungono i dati relativi al programma nominato "A". Anche in questo caso il computer non cancella nulla di ciò che era precedentemente presente in memoria e sul video apparirà READY se il programma letto è identico a quello presente in memoria oppure BAD se è stato riscontrato qualche errore.

CLS - Scrivendo questa istruzione si ottiene la cancellazione di tutta la pagina video ed il cursore viene posizionato in alto a sinistra.

CMON - È un'istruzione relativa al registratore a cassetta e permette di attivare il motorino di quest'ultimo per far avanzare o riavvolgere il nastro velocemente. È molto utile in quanto il registratore, attraverso la presa "REMOTE", viene attivato solo in fase di lettura o di registrazione di un programma e quindi per tornare su di un programma appena registrato o avanzare velocemente su di un altro, si dovrebbe ogni volta scollegare la presa del contatto REMOTE. Invece, scrivendo semplicemente CMON, l'interfaccia cassetta chiude il contatto di REMOTE ed è possibile usare tranquillamente il registratore.

Per fermare nuovamente il motorino del registratore, una volta terminata la ricerca del programma, è sufficiente digitare un qualunque carattere sulla tastiera.

CONT - Questa istruzione viene utilizzata per riprendere l'esecuzione di un programma dopo un arresto ottenuto con il tasto di BREAK oppure con un'istruzione di STOP contenuta nel listato di un programma. L'escuzione delle operazioni, riprende dallo stesso punto nel quale era avvenuta la fermata.

COS (n)- Calcola il valore del COSENO del numero n scritto entro parentesi, con n espresso in radianti.

Esempio:

PRINT COS (2.5) (return)
-.801144

CSAVE (nome file) - È l'istruzione che comanda il salvataggio di un programma su nastro. Il "nome file" scritto fra parentesi corrisponde al nome che viene assegnato al programma e che sarà utilizzato dal computer, in fase di lettura (vedi CLOAD), per riconoscere un programma dagli altri.

Il nome del file non può essere più lungo di un carattere. Scrivendo:

CSAVE (A) oppure **CSAVE (ABC),** sulla cassetta verrà registrato un programma con lo stesso nome del file, cioè A.

CSNG (n #) - Trasforma in singola precisione (5 cifre significative e arrotondamento dell'ultimo decimale), una variabile numerica definita in doppia precisione.

Esempio 1:

A # = 23.456789 (return)
PRINT CSNG(A #) (return)
23.45678

Esempio 2:

PRINT CSNG (.66666666667) (return)
1.666667

DATA n, m, x - Questa istruzione consente di memorizzare, all'interno di un programma, delle liste di dati di diverso contenuto, cioè dati numerici o caratteri alfanumerici. Non sono ammesse le espressioni numeriche, tipo 3 + (3 * B). I dati vengono memorizzati sequenzialmente, uno di seguito all'altro, separati fra loro da una virgola.

Esempio:

DATA 150, 75, 23.1, 10
DATA "AUTO", "TRENO", "AEREO", "TRASPORTO"
Per leggere questi dati vedi l'istruzione READ.

DEFDBL variabile,variabile. Definisce le "variabili" assegnate, in doppia precisione.
Esempio:

DEFBDL A, B, F (return)

Le variabili numeriche A,B ed F sono, d'ora in poi, trattate sempre come variabili in doppia precisione.

Per definire più velocemente un insieme di variabili in doppia precisione si può scrivere questa istruzione nella forma:

DEFDBL A—G

In questo modo tutte le variabili comprese, secondo l'ordine alfabetico, fra A e G, cioè A, B, C, D, E, F e G, vengono trattate in doppia precisione.

DEFINT variabile, variabile,... - È un'istruzione di definizione del tutto analoga a quella descritta precedentemente, con la differenza che ora le variabili presenti nell'istruzione vengono definite come intere, cioè variabili numeriche senza cifre decimali poste dopo il punto.
Esempio 1:

DEFINT A,D (return)
A = 45.67 (return)
C = 34.23 (return)
PRINT A (return)
45
PRINT C (return)
34.23

Esempio 2:

DEFINT A-D (return)
A = 45.67 (return)
C = 34.23 (return)
PRINT A (return)
45
PRINT C (return)
34

Nota: anche in questo caso, scrivendo nell'istruzione, due variabili separate dal segno -, tutte le variabili comprese alfabeticamente fra queste due lettere, sono trattate come intere.

DEFSNG variabile, variabile,... - È l'istruzione opposta a DEFDBL. Infatti definisce in singola precisione tutte le variabili scritte nell'istruzione o tutte le variabili comprese nell'intervallo fra le due lettere scritte nell'istruzione separate dal segno -.
Esempio:

DEFSNG A, F (A ed F sono variabili in singola precisione)
DEFSNG A-F (A, B, C, D, ed F sono variabili in singola precisione)

DEFSTR variabile, varibile,.. - Definisce come variabili stringa tutte le variabili scritte nell'istruzione. Se due variabili sono separate fra loro dal segno -, allora le definizione di "variabile stringa" è estesa a tutte le lettere comprese, alfabeticamente, fra le due lettere scritte.

DEFUSRn = ind. - È un'istruzione complessa che serve ad utilizzare, da BASIC, delle "routine" scritte in linguaggio macchina. Il numero "n" può variare da 0 a 9 per poter utilizzare fino a 10 routine diverse in linguaggio macchina. Il termine "ind" scritto dopo il segno = indica l'indirizzo di partenza del programma che si vuole utilizzare.
(Vedi anche dettagliatamente l'istruzione USR).

DELETE n1 - n2 - È un comando con il quale possono essere cancellate tutte le linee di programma specificate nell'istruzione con i due termini n1 ed n2. Questo comando può essere scritto in varie forme ottenendo, ovviamente, funzioni diverse:

- **DELETE 50** (Viene cancellata la sola linea 50)

- **DELTE 50-300** (Vengono cancellate tutte le linee di programma fra 50 e 300, comprese la 50 e la 300)

- **DELETE - 300** (Vengono cancellate tutte le linee del programma a partire dalla prima linea fino alla linea 300 compresa).

DIM variab. (n1, n2), stringa$ (n1, n2) - Permette di "definire" uno spazio in memoria da assegnare a liste di dati "vettorizzati". Tali dati possono essere immagazzinati in un "vettore" ad una dimensione oppure in matrici a due, tre o più dimensioni. Evidentemente, più sono grandi le "dimensioni" della matrice o del vettore, più sarà la memoria assegnata a questi dati e meno sarà la memoria poi a disposizione del programma stesso.

I numeri fra parentesi nell'istruzione sono le grandezze da assegnare ad ogni "dimensione" della matrice o del vettore definito. Si possono dimensionare vettori o matrici sia di numeri che di stringhe.
Esempio:

DIM A(20), B$(8,12) (return)
A(1) = 23.5 (return)
A(2) = 5 (return)
B$(3,5) = "BASIC" (return)
PRINT A (1) (return)
23.5
PRINT A(2) (return)
5
PRINT A(18) (return)
0
PRINT B$(3,5) (return)
BASIC
PRINT B$ (8,0) (return)
(spazio vuoto)

Nota: Chiedendo la stampa di un dato numerico non inserito - PRINT A(18) -, il computer risponde con uno 0, mentre per un dato di una variabile stringa - PRINT B$ (8,0) -, viene stampato una linea nulla.

EDIT - È un comando che porta il computer entro un sottoprogramma, chiamato appunto EDIT, che

permette di correggere e modificare il contenuto delle linee di programma. (vedi in seguito sotto il paragrafo EDIT).

ELSE - È un'istruzione utilizzata all'interno di una linea di programma ed è sempre preceduta da una istruzione di IF. Letteralmente ELSE significa "altrimenti" ed infatti, di seguito a tale istruzione, si trova sempre un comando che il computer dovrà eseguire nel caso che la condizione espressa dall'istruzione IF non risulti verificata. (Vedi l'istruzione IF).

END - Definisce la linea di chiusura del programma. Quando il computer, nell'esecuzione di un programma, incontra tale istruzione, si ferma e sul video appare la scritta:

READY

ERL - È un comando che fornisce il numero di linea nel quale si è verificato un errore nel programma. Se si utilizza tale istruzione senza che si siano verificati errori, il computer risponde con il numero 65535, che è il massimo numero di indirizzo decimale ammesso.

ERR - Questa istruzione è molto importante ed è utilizzata principalmente in programmi complessi In cui si vuole ottenere una "gestione degli errori".

Ogni errore, infatti, ha un proprio codice numerico e perciò può essere "riconosciuto" dal programma e può essere "gestito" senza che il computer si blocchi per causa sua. L'istruzione è utilizzata all'interno di "routine" per la gestione degli errori, assieme alle istruzioni:

- ON ERROR GOTO ind.
- RESUME
(Vedi istruzione seguente ERROR)

ERROR - Questa istruzione è utilizzata in due modi differenti all'interno dei programmi.

Utilizzata nella forma:

ERROR n

simula, all'interno di un programma, il tipo di errore specificato dal numero n.

Utilizzato come comando diretto, fornisce il significato per esteso del codice di errore scritto in n, cioè:

ERROR 2 (return)
SINTAX ERROR

Infatti, osservando la tabella del codice degli errori, il numero 2 corisponde ad un SINTAX ERROR.

La seconda funzione di questa istruzione è più complicata ed è utilizzata con l'istruzione di ON (vedi tale istruzione). La sintassi è la seguente:

ON ERROR GOTO ind.

dove "ind." è la linea di programma nella quale risiede la subroutine di gestione dell'errore. Al termine di tale subroutine, si deve tornare al programma principale utilizzando l'istruzione di RESUME n dove n è la linea di programma a cui si deve tornare. Se "n" non viene specificato, il ritorno avviene sulla linea nella quale si era verificato l'errore.

Per rendere più semplice capire come opera questa particolare funzione, provate a copiare Il programma riportato ed a provarlo:

```
10 CLEAR 15
20 ON ERROR GOTO 100
30 INPUT A $
40 GOTO 30
100 E = ERR/2 + 1
110 PRINT "CODICE ERRORE = " E
120 RESUME 30
```

Digitando RUN e RETURN, se vengono inseriti più di 15 caratteri, sul monitor apparirà la scritta:

```
CODICE ERRORE = 15
?
```

Come potete vedere il programma rientra in esecuzione senza fermarsi, come invece avverrebbe normalmente senza l'istruzione di ON ERROR GOTO.

FIX (espr) - Fornisce il **numero intero** equivalente all'espressione scritta fra parentesi.

Esempio:

```
PRINT FIX(10/3) (return)
3
```

FRE (A$) - È un comando che fornisce il numero di byte disponibili per le variabili si stringa.

FRE (0) - Fornisce la memoria utente ancora disponibile.

FOR variabile = n1 TO n2 NEXT variabile - L'istruzione FOR definisce un "ciclo di operazioni" da eseguire. La sequenza delle istruzioni che il computer dovrà eseguire è contenuta fra la prima istruzione di FOR e la successiva istruzione di NEXT.

Ad esempio, ponendo come "variabile" la lettera A (si poteva comunque scegliere una qualunque altra lettera) e definendo i due "n1" ed "n2" come 1 e 10 rispettivamente, si ha:

```
10 FOR A = 1 TO 10
20 PRINT A;
30 NEXT A
```

Questo programma viene eseguito dal computer seguendo questa sequenza di operazioni:
- Inizialmente la "variabile" A viene posta uguale al primo numero, cioè 1.
- Viene poi eseguita l'istruzione successiva, cioè la linea 20, dove viene comandata la stampa della variabile A e perciò sul video apparirà il numero 1.
- Passando alla linea 30, viene incrementato il valore della variabile A e si controlla che A non sia maggiore del numero n2., cioè 10. Se non lo è, il computer torna ad eseguire il ciclo di istruzioni descritto mentre se A risulta maggiore di 10, significa che il ciclo di operazioni è già stato interamente eseguito e quindi il computer esce dal ciclo ed eseguirà le istruzioni scritte alle linee seguenti.

Il risultato finale di questo esempio sarà perciò la stampa sul video della sequenza di numeri da 1 a 10.

Questa istruzione può essere scritta in forma più potente e complessa e cioè si possono programmare incrementi diversi della variabile A, cioè far incrementare il valore di A ad ogni ciclo di una quantità diversa da 1, cioè 2-3-10-56-0,2-1,4- ecc. ed inoltre può essere impostato un incremento negativo cioè, ad ogni ciclo, può venire sottratta una quantità programmata alla variabile A.

Per far questo è sufficiente aggiungere all'istruzione di FOR, l'istruzione STEP, con la seguente forma:

FOR variabile = n1 TO n2 STEP n3

Per maggior chiarezza riportiamo alcuni esempi nei quali abbiamo inserito i vari modi di scrittura di questa istruzione. A seguito di ogni esempio è riportato anche il risultato pratico del programma descritto, come voi stessi potrete constatare ricopiando fedelmente le istruzioni e provandole sul vostro computer.

Esempio 1:

```
10 FOR I = 10 TO 1 STEP-1
20 PRINT I;
30 NEXT I
RUN
10 9 8 7 6 5 4 3 2 1
```

Esempio 2:

```
10 FOR K = 0 TO 1 STEP .3
20 PRINT K;
30 NEXT K
RUN
O .3 .6 .9
```

Esempio 3:

```
10 J = 3:K = 8:L = 2
20 FOR I = J TO K + 1 STEP L
30 PRINT I;
40 NEXT I
RUN
3 5 7 9
```

Per completezza abbiamo riportato, nell'esempio 3, una forma "indicizzata" dell'istruzione FOR-NEXT.

Infatti, al posto dei tre numeri n1, n2 ed n3, in questo caso abbiamo inserito tre variabili numeriche J e K ed L, precedentemente definite alla linea 10.

GOSUB ind. - Richiamo di una subroutine. Il termine "ind." specifica la linea di programma nella quale risiede la subroutine chiamata. Alla fine di ogni subroutine è obbligatorio scrivere l'istruzione **RETURN**.

GOTO ind. - È un'istruzione di **"salto incondizionato"** che obbliga il computer a proseguire l'esecuzione del programma saltando alla linea specificata con il termine "ind."

IF espr. THEN istr. 1 ELSE istr. 2 - È un'istruzione di "TEST". Viene esaminata la condizione descritta in "espr." e quindi, se è verificata, viene eseguita l'istruzione "istr. 1", altrimenti viene eseguita l'istruzione "istr. 2".

Questo comando può essere scritto anche senza ELSE. In questo caso, se non è verificata la condizione descritta in "espr.", il programma prosegue eseguendo la linea successiva ad essa.
Esempio:

```
10 IF A = 4 THEN GOTO 100 ELSE A = 3
20 ..
```

In questo esempio, la condizione da verificare è che il valore della variabile A sia uguale a 4. Se tale condizione è verificata allora il programma salta alla linea 100, altrimenti viene posto in A il valore 3 ed il programma continua eseguendo l'istruzione presente alla linea 20.

INPUT "messagio"; variabile - Utilizzando questo comando è possibile inserire nel programma dei dati provenienti dall'esterno. Quando il computer incontra tale istruzione, l'esecuzione del programma viene arrestata e sul video appare la scritta inserita in "messaggio" seguita da un ?. A questo punto, ciò che viene scritto, è automaticamente inserito nella variabile definita nell'istruzione stessa. Se la variabile è una sola lettera (variabile numerica) allora si possono scrivere solamente numeri mentre se la variabile è una lettera seguita dal simbolo $ (stringa), si possono inserire sia numeri che lettere.

Esempio:

```
10 INPUT "Nome = "; B$
20 ...
RUN
NOME = ?
```

Scrivendo sulla tastiera le lettere corrispondenti al "Nome" richiesto, seguite da RETURN, queste verranno inserite nella variabile stringa B$.

INKEY$ - È un'istruzione analoga all'INPUT solo che il programma, passando da tale istruzione, non si ferma per eseguire l'INPUT ed il dato letto dalla tastiera viene messo all'interno del programma senza che l'esecuzione del programma stesso venga interrotta.

Esempio:

```
50 ..
60 B$ = INKEY$
70 ..
```

INSTR (stringa$, carat.) - Esegue una operazione sulle stringhe e fornisce come risultato un numero che corrisponde alla posizione del carattere (carat.) all'interno di una stringa (stringa$).

Esempio:

```
10 B$ = "S"
20 A$ = "BASIC"
30 X = INSTR (A$, B$)
40 PRINT X
RUN
3
```

INT (n) - Fornisce l'equivalente intero del numero scritto fra parentesi. L'istruzione esegue le stesse operazioni della CINT (vedi istruzione), solo che ora il campo dei numeri ammessi non è limitato.

LEFT$ (stringa $, n) - È una operazione sulle stringhe. Estrae dalla stringa definita nell'istruzione, un numero di caratteri pari ad n.

Esempio:

```
10 A$ = "NUOVA ELETTRONICA"
20 B$ = LEFT$ (A$, 3)
30 PRINT B$
RUN
NUO
```

LEN (stringa) - Fornisce un numero pari al numero di caratteri della stringa definita all'interno della parentesi.

Esempio:

```
10 A$ = "BASIC"
20 X = LEN (A$)
30 PRINT X
RUN
5
```

LET variabile = espres. - È un'istruzione di "assegnazione" e serve ad impostare un valore in una variabile.

Esempio:

```
LET B = 34.5
```

Nota: questa funzione è del tutto identica ad una assegnazione normale tipo B = 34.5 ed infatti è quest'ultima la forma di assegnazione più comunemente usata ed utilizzabile anche con questo tipo il BASIC).

LLIST - Esegue su stampante il listato del programma presente in memoria.

LINE INPUT stringa - Simile ad INPUT con la differenza che non visualizza il ? ed accetta tutti i simboli della tastiera.

LIST n1-n2 - È un comando diretto e genera la stampa su video del programma presente in memoria, partendo dalla linea "n1" e terminando alla linea "n2".

Se i termini n1 ed n2 non sono specificati, viene generata la lista su video di tutto il programma.

Esempio:

LIST 20-100 (Stampa le linee dalla 20 alla 100)
LIST 30 (Stampa la sola linea 30)
LIST -100 (Stampa tutte le linee dall'inizio del programma fino alla 100)

LOG (n) - Fornisce il logaritmo in base "e" del numero "n" scritto entro la parentesi.

LPRINT - È un'istruzione dedicata alla stampante. Quando viene dato questo comando, ciò che viene scritto di seguito viene riportato sulla stampante.

MEM - È una variabile che fornisce il numero di byte liberi disponibili in memoria.

MID$ (stringa, n1, n2) - È un operatore di stringa. Estrae dalla stringa definita nell'istruzione un numero di caratteri n2, a partire dal carattere n1.

Esempio:

```
10 A$ = "NUOVA ELETTRONICA"
20 B$ = MID$(A$,8,4)
30 PRINT B$
RUN
LETT
```

NEXT - Vedi l'istruzione FOR

NEW - È un comando diretto e cancella tutto ciò che è presente in memoria, azzera tutti i registri e tutte le variabili, ripristinando le condizioni di partenza iniziali.

NOT - È un "operatore logico" e simula il funzionamento di una porta logica NOT (invertitore), fornendo in uscita un 1 se il bit di ingresso è 0 e 0 se il bit di ingresso è 1.

Può essere utilizzato assieme ad altri operatori logici, per invertirne le funzioni.

Esempio 1:

```
PRINT NOT 23 (return)
-24
```

Esempio 2:

```
10 ..
20 IF A = D AND NOT A     3 THEN GOTO 100
30 ..
```

L'istruzione alla linea 20 è equivalente ad un NAND logico.

ON n GOTO n1.riga, n2.riga,.. - È un'istruzione complessa di "salto condizionato". A seconda del valore della variabile numerica "n", il computer passa ad eseguire il contenuto della linea di programma scritto dopo il GOTO.

Più precisamente se "n" è 1, salta all'indirizzo "n1.riga", se "n" è 2, salta alla "n2.riga", ecc.

Esempio:

```
50 ..
60 X = A
70 ON X GOTO 500, 600, 700
80 ..
```

Se X = 1, salta alla linea 500, se X = 2, salta alla 600, se X = 3, salta alla 700.

OR - È un "operatore logico" e simula il funzionamento di una porta logica OR, nella quale si ottiene un livello logico 1 quando almeno un ingresso è a livello logico 1.

Quando viene utilizzato all'interno di un'istruzione di IF, verifica la condizione dell'istruzione se almeno una delle due condizioni è verificata.

OUT porta, valore - Pone in uscita, sulla porta il cui indirizzo è specificato nell'istruzione, il valore scritto.

Esempio:

```
OUT 220,10
```

Pone sulla porta periferica di indirizzo decimale 220, il valore decimale 10.

val. = PEEK (ind.) - Legge un byte (in decimale) all'indirizzo di memoria specificato nella parentesi.
Esempio:

```
20 A = 174
30 A = PEEK(32450)
```

POINT (n1, n2) - È un'istruzione di semi-grafica e serve per testare se, sul video, il punto di coordinate "n1" ed "n2" è acceso o spento. Se è acceso, l'istruzione fornisce in uscita -1 mentre se è spento fornisce 0.
Esempio:

```
X = POINT(10,15) (return)
PRINT X (return)
```

Sul video apparirà il numero -1 se il punto corrisponde alle coordinate 10,15 è acceso oppure 0 se risulta spento (vedi istruzione SET per le coordinate del punto).

POKE ind., valore - Pone nella locazione di memoria specificata in decimale da "ind.", il valore decimale scritto nell'istruzione.
Esempio:

```
20 POKE 35623,126
```

Con questa istruzione viene posto, nella cella di memoria 35623, il valore 126.

POS (0) - Fornisce un numero corrispondente alla posizione orizzontale del cursore sul video.
Scrivendo tale istruzione dopo un'istruzione di PRINT, è importante ricordare di scrivere il simbolo ";" altrimenti il cursore andrà a capo della linea sul video e tale istruzione fornirà sempre risultato 0.

PRINT - È l'istruzione di stampa su video. Per stampare il contenuto di una variabile si scrive il nome della variabile di seguito a tale istruzione mentre per stampare una frase o un messaggio particolare, si deve porre fra virgolette (") il testo da stampare.
Esempio:

```
10 A = 123
20 B$ = "BASIC"
30 PRINT A
40 PRINT B$
50 PRINT "A,B$"
RUN
123
BASIC
A,B$
```

RANDOM - È un'istruzione relativa al "generatore di numeri casuali" presente all'interno del BASIC.
Inserendo tale istruzione all'interno di un programma nel quale viene utilizzata la funzione RND (vedi in seguito), viene reinizializzato il contatore del generatore e si ottengono così sequenze di numeri casuali più ampi e non ripetitivi.

READ variabile - Esegue una lettura da una lista di dati presente nel programma stesso (vedi DATA).
Le lettura avviene sequenzialmente dato per dato ed il valore estratto dalla lista viene inserito nella "variabile" definita nell'istruzione.
Esempio:

```
10 READ X
20 PRINT X;
30 IF X = 0 PRINT "FINE": END
40 DATA 1,2,3,45,5,0
RUN
1 2 3 45 5 FINE
```

RESTORE - Resetta l'istruzione di READ. Quando il programma incontra tale istruzione, la lettura del prossimo DATA inizia nuovamente dal primo dato.

REM - È un'istruzione che si usa all'interno di un programma per aggiungere dei commenti al listato. In questo modo risulterà più facile ricostruire le funzioni svolte dal programma scritto senza dover ristudiare tutto il programma stesso.

RESET - Vedi SET

RESUME - Vedi ERROR

RETURN - Vedi GOSUB

RIGHT$ (stringa,n) - È una funzione che opera sulle stringhe ed estrae, dalla stringa definita nell'istruzione, gli **ultimi "n" caratteri**.

RND (n) - È la funzione utilizzata per generare dei numeri casuali. Se il numero "n" all'interno della parentesi è 0, vengono generati numeri casuali decimali, compresi fra 0 ed 1 (ad esempio .3457). Se il numero "n" è maggiore di 1, allora vengono generati numeri **interi** (cioè senza decimali dopo la virgola), compresi fra 1 ed il numero scritto entro la parentesi.

RUN - È l'istruzione che comanda il lancio di un programma. Scrivendo semplicemente RUN e RETURN, il computer inizia ad eseguire il programma contenuto in memoria, a partire dalla prima istruzione.
Scrivendo invece RUN 500 e RETURN, l'esecuzione del programma avverrà dalla linea 500 in poi.

SET (n1,n2) - È un'istruzione di semi-grafica e serve per accendere un punto nell'area video, le cui coordinate sono specificate con i due numeri n1 ed n2 scritti entro parentesi. Come riferimento delle coordinate, il punto di origine (0,0) è il primo quadratino in alto a sinistra. Il numero n1 specifica la posizione **su di una riga** e varia da 0 (primo elemento a sinistra della riga) a 63 (ultimo elemento a destra della riga) mentre n2 specifica il **numero di riga** su cui si vuole accendere il quadratino. Nel video sono presenti 48 righe, numerate, a partire dall'alto verso il basso, da 0 a 47.
L'ultimo quadratino in basso a destra dell'area video avrà perciò coordinate (63,47).
Per spegnere il quadratino così acceso, si deve scrivere, con le stesse coordinate, l'istruzione:

```
RESET (n1,n2)
```

SGN (variabile num.) - È una funzione che osserva il segno della variabile numerica scritta entro pa-

rentesi. Se il segno di tale variabile è positivo, la funzione fornisce in uscita 1, se è negativo -1.
Esempio:

```
10 A = 12.4
20 B = SGN(A)
30 PRINT B
RUN
1
```

SIN (n) - Fornisce il valore della funzione SENO del numero scritto entro parentesi, dove n è inteso in radianti.

SYSTEM - È un comando diretto e porta il computer in **linguaggio macchina.** Digitando tale istruzione, entrerete perciò nel MONITOR del sistema. (Vedi paragrafo successivo).

SQR (n) - Fornisce il valore della radice quadrata del numero scritto entro parentesi.
Nota: Il numero "n" deve essere un **numero positivo** altrimenti si ha una notazione di errore.

STEP - Vedi l'istruzione FOR

STOP - È un'istruzione che comanda l'arresto dell'esecuzione del programma. Quando viene incontrata questa istruzione, il computer si blocca e stampa su video il numero di linea del programma nel quale ha incontrato l'istruzione STOP.

STR$ (espressione numerica) - Con questa istruzione è possibile trasformare una variabile numerica, descritta da una "espressione numerica" all'interno della parentesi, in una variabile stringa ed è l'unica istruzione in cui possono essere correlate due variabili di questo tipo.
Nella stringa viene ricopiato anche il segno e, se questo è positivo, il primo carattere della stringa è uno spazio. Nel caso che il segno sia invece negativo, il primo carattere della stringa è il simbolo "-".

STRING$ (n,carattere) - È una funzione di stampa o assegnazione e genera una stringa ripetuta "n" volte, del "carattere" definito nell'istruzione.
Esempio:

```
PRINT STRING $ (10,"X") (return)
XXXXXXXXX
```

Analogamente, è possibile anche la seguente sintassi:

```
A$ = STRING$ (20,"A")
```

TAB - È un'istruzione di stampa e viene usata assieme all'istruzione PRINT. La sintassi di tale istruzione è:

```
PRINT TAB (n)
```

Quando in una funzione di PRINT viene incontrata l'istruzione TAB, il cursore si porta sulla posizione di riga specificata da "n" entro parentesi.
È perciò l'equivalente del "tabulatore" delle macchine da scrivere tradizionali.

TAN (n) - Calcola il valore della TANGENTE del numero n scritto entro le parentesi, inteso in radianti.

THEN Vedi IF

TROFF - È un comando diretto e serve ad uscire dal programma TRON (vedi di seguito).

TRON - È un'istruzione particolare che serve a verificare il corretto funzionamento di un programma. Scrivendo tale istruzione prima del RUN, quando lancerete il programma appariranno sul video tutte le linee di programma che il computer via via esegue ed avrete così la possibilità di verificare il corretto svolgersi delle operazioni del programma stesso.
Esempio:

```
10 FOR A = 1 TO 3
20 PRINT A;
30 NEXT
40 END
TRON (return)
RUN
(10)(20)1(30)(20)2(30)(20)3(30)(40)
```

(Nota: I numeri 1, 2 e 3 fra le linee di programma riportate, sono la stampa del valore della variabile A, comandata dall'istruzione di PRINT alla linea 20

TO Vedi l'istruzione FOR

USING stringa;var.numerica - È un'istruzione di stampa e serve a riportare su video il valore di una variabile numerica (var.numerica) con il formato di scrittura definito dalla stringa. Per ottenere questa funzione si deve precedentemente definire la variabile stringa usata come maschera riferimento per il formato utilizzando la seguente sintassi:

```
Q$ = " ###, # " (Così viene definito un formato di 3 numeri interi ed 1 decimale)
W$ = " #### " (Così viene definito un formato di 4 numeri interi)
```

Esempio:

```
B$ = " ##.## " (return)
A = 1.3 (return)
PRINT USING B$; A (return)
1.30
```

(Nota: Vengono sempre aggiunti degli 0 alla parte decimale non definita)

USR (n) - Comanda un salto ad una routine scritta in linguaggio macchina, precedentemente scritta nello spazio di memoria libero e non destinato al BASIC. La sintassi completa dell'istruzione è la seguente:

A = USRn(arg.) dove n è il numero di subroutine in linguaggio macchina, definita con DEFUSRn (vedi istruzione relativa) mentre arg. è il parametro di trasporto fra il BASIC e la routine stessa. Per trasportare questo dato viene utilizzato il registro HL della CPU e scrivendo, come prima istruzione della USR l'istruzione:

```
CALL 0A7F (in assembler)
```

il computer preleva il contenuto della variabile "arg" e la pone nel registro HL.

Una volta terminato lo svolgersi di tutte le istruzioni scritte in questo programma in linguaggio macchina, si hanno due possibilità di ritorno al BASIC, una senza passare alcun dato, cioè non utilizzando l'assegnazione alla variabile "A" specificata nella chiamata ed una invece con il trasporto dal programma al BASIC di un dato elaborato.

Per tornare al BASIC senza passare alcun dato, è sufficiente scrivere:

RET (in assembler)

nella linea di programma nel quale si vuole ottenere il ritorno.

Se si vuole passare un dato al Basic, è sufficiente porre nel registro HL della CPU il dato elaborato e quindi scrivere il comando di ritorno al Basic come segue:

JMP 0A9A (in assembler).

Così facendo, il programma automaticamente pone il valore di HL nella variabile di trasporto "arg." e perciò la rende disponibile al programma BASIC principale.

VAL (stringa) - Fornisce il valore numerico corrispondente alla stringa definita fra parentesi. Il valore è calcolato estraendo dalla striga i soli caratteri numerici, posti di seguito all'inizio della stringa stessa.

Perciò il valore della stringa il cui contenuto è A12 è 0 mentre il valore della stringa 12A è 12. (Questa istruzione è l'inverso di STR$).

VARPTR (var) - È un'istruzione che fornisce l'indirizzo decimale della cella di memoria in cui è memorizzata la variabile specificata entro parentesi.

FUNZIONI SPECIALI

Vi sono due funzioni particolari, richiamabili con comandi diretti dal BASIC, e sono:

EDIT e SYSTEM

Esaminiamo separatamente queste due funzioni, totalmente differenti fra loro ma entrambe complesse.

LA FUNZIONE EDIT

Il comando EDIT serve a correggere delle linee di programma in cui sono stati commessi degli errori o nelle queli si vuole modificare il contenuto dell'istruzione.

La sintassi di tale comando è la seguente:

EDIT num. linea (return)

Così facendo, sul video apparirà la "linea" richiesta. Facciamo un semplice esempio pratico e supponiamo di scrivere il seguente programma:

```
10 FOR A = 1 TO 20
20 PRINT A
30 NEXT B
40 END
```

In questo listato è presente un errore alla linea 30, in quanto l'istruzione di NEXT deve essere riferita alla variabile A definita nell'istruzione FOR alla linea 10 e non alla variabile B, per altro inesistente in questo programma.

Si dovrà perciò correggere tale istruzione, ponendo A al posto di B.

Digitiamo perciò:

EDIT 30 (return)

e sul video apparirà:

30

La linea chiamata è ora disponibile ed è possibile eseguire le operazioni di modifica e correzioni volute.

Digitando la barra (spazio) sulla tastiera, il cursore avanza sulla linea chiamata di un carrattere alla volta mentre digitando il tasto ← (Back-Space) il cursore indietreggia allo stesso modo. Perciò, premendo 6 volte la barra, sul video apparirà l'intera linea 30.

Posizioniamo ora il cursore sulla lettera B, indietreggiando con il tasto ←, e quindi premiamo il **tasto C.** Così facendo, il programma EDIT esegue **un cambiamento di carattere** nella linea, ponendo, al posto di quello su cui è posizionato il cursore, il carattere che ora scriveremo. Digitiamo perciò

A e RETURN

e, per verificare le operazioni eseguite, scriviamo LIST. Come potrete constatare, nella linea 30, sarà ora presente l'istruzione NEXT A.

Abbiamo utilizzato questo primo esempio per illustrare nella pratica come si procede, in generale, lavorando con il programma EDIT e per definirne, fin dall'inizio, la caratteristica essenziale dell'EDIT, cioè quella di "correzione o modifica delle linee di programma".

Una regola fondamentale dell'EDIT è che, una volta chiamata la linea da correggere digitando:

EDIT num.riga e RETURN

il tasto che viene premuto successivamente è sempre interpretato dal programma EDIT come un **"comando di operazione"**, cioè, digitando uno dei caratteri che ora illustreremo, potrete comandare l'esecuzione di procedure diverse e specializzate a modificare, ricercare un carattere, cancellare parte della linea, inserire altri caratteri ecc.

Per uscire dal programma EDIT è sufficiente, una volta richiamata la linea, digitare RETURN. Come vedrete più nel dettaglio nella spiegazione delle varie istruzioni, esistono anche alcuni comandi che, automaticamente, escono dall'EDIT.

Iniziamo perciò a descrivere le varie funzioni assegnate ai caratteri della tastiera.

Carattere L - Visualizza tutta la linea editata ricopiandola sul video. Sotto, nella riga successiva, viene riscritto il numero corrispondente alla linea chiamata ed il cursore si posiziona automatica-

mente all'inizio di quest'ultima. Questo comando è molto utile per visualizzare tutta la linea da correggere senza dover battere ripetutamente la barra sulla tastiera.

Con questo comando si rimane in EDIT.

Esempio:

EDIT 30 (return)
30 L
30 NEXT A
30 ■

(Nota: il cursore è posizionato subito dopo il numero 30 ed il programma è pronto a ricevere un altro carattere di comando)

Carattere C - Sostituisce il carattere su cui è posizionato il cursore con il carattere digitato e si rimane ancora in EDIT.

Esempio:

EDIT 30 (return)
30 C M (return)
30 MEXT A

Nota: È stato sostituito al carattere N il carattere M. Se non fosse stato premuto RETURN, saremmo ancora all'interno dell'EDIT ed il programma sarebbe pronto a ricevere un altro carattere di comando.

È possibile sostituire inoltre anche più di un carattere per volta in una stessa linea, utilizzando questo stesso comando scritto nella forma:

nC dove "n" specifica il numero di caratteri da sostituire.

Carattere D - Cancella il carattere su cui è posizionato il cursore. Il carattere cancellato viene visualizzato fra due punti esclamativi e si rimane ancora In EDIT.

Esempio:

EDIT 30 (return)
30 D !N! (return)
30 EXT B

Nota: È stato cancellato il carattere N. Se non fosse stato premuto RETURN, saremmo ancora all'interno dell'EDIT ed il programma sarebbe pronto a ricevere un altro carattere di comando.

È possibile utilizzare il comando del carattere D in un'altra forma e cioè:
nD dove n è un numero intero. In questa forma vengono cancellati dalla linea un numero di caratteri pari ad "n", a partire dal carattere su cui è posizionato il cursore.

Esempio:

EDIT 30 (return)
30 3D !NEX! (return)
30 T A

Carattere I - Inserisce tutti i caratteri scritti, a partire dalla posizione in cui si trova il cursore. Per terminare l'inserzione dei caratteri, si deve premere il tasto **ESC**. Anche con questo comando, seguito dal tasto ESC, non si esce dall'EDIT a meno di non premere il tasto RETURN.

Esempio:

EDIT 10 (return)
10 (battere 6 volte lo spazio) **FOR A = I20 ESC** (return)
10 FOR A = 201 TO 20

Nota: Battendo sei volte la barra (spazio), si avanza di sei caratteri e perciò sul video appaiono, di seguito, i caratteri FOR A = . A questo punto si esegue l'inserimento digitando I20 e premendo il tasto di ESC.

È possibile comunque eseguire la stessa operazione digitando:

6 (Spazio)

Se non si preme il tasto di RETURN, si rimane in EDIT ed il programma è pronto ad accettare altri tasti di comando).

Carattere K - Cancella tutta la linea fino al carattere digitato dopo K. I caratteri cancellati vengono riportati sul video fra due punti esclamativi. Se non viene premuto il tasto di RETURN, si rimane all'interno dell'EDIT ed il programma è nuovamente disponibile ad accettare altri caratteri di comando, compreso ovviamente il carattere K.

Esempio:

EDIT 10 (return)
10 KR !FO! (return)
10 R A = 1 TO 20

Carattere X - Inserisce dei caratteri alla fine della linea. Digitando X il cursore si porta automaticamente alla fine della linea chiamata che viene perciò interamente visualizzata. Premendo il tasto II, Il cursore indietreggia e cancella i caratteri su cui passa. Per terminare l'inserzione si deve premere ESC ed in questo modo si rimane all'interno dell'EDIT. Se invece si preme RETURN si termina l'inserzione e si esce dall'EDIT.

Esempio:

EDIT 10 (return)
10 X FOR A = 1 TO 20 34 (return)
10 FOR A = 1 TO 2034

Carattere S (carattere) - Serve per cercare un carattere all'interno della linea e posiziona il cursore su tale carattere. Per modificare o inserire dei caratteri nella linea, si deve digitare un altro tasto dl comando.

È possibile inoltre scrivere questa istruzione nella forma:

nS (carattere) ed in questo modo vengono ignoratl i primi "n" caratteri uguali a quello cercato.

Carattere H - Cancella tutta la linea dalla posizione del cursore fino alla fine. Si rimane ancora in EDIT ed il programma è pronto ad accettare un nuovo carattere di comando.

Carattere Q - Annulla tutte le modifiche fatte sulla linea ed **esce dall'EDIT**. Questo comando serve ad eliminare le correzioni fatte su di una linea nel caso siano stati scritti dei dati ancora sbagliati durante la correzione della linea stessa.

Carattere A - Esegue le stesse operazioni del carattere Q con la sola differenza di **non uscire dall'EDIT**. Perciò digitando A, è poi possibile digitare un altro carattere di comando per proseguire nella correzione della linea.

Abbiamo così terminato la descrizione delle funzioni dell'EDIT. Passiamo quindi a descrivere l'ultimo comando di funzione implementato sul BASIC e cioè la funzione:

SYSTEM

Scrivendo tale istruzione seguita dal tasto di RETURN, sul video apparirà la seguente scritta:

N.E. MONITOR V. 1.0

Questo significa che siete entrati nel MONITOR del computer, ed è a questo "livello" che possono essere scritte le routine in linguaggio macchina, richiamabili direttamente da BASIC, come già abbiamo detto, parlando dell'istruzione USR.

Dal MONITOR infatti si ha la possibilità di interagire direttamente con le singole celle di memoria, con le periferiche I/O e con tutti i registri interni della CPU.

Il MONITOR, per svolgere tutte queste funzioni, mette a disposizione una serie di comandi diretti che sono:

B - Ritorna dal MONITOR al BASIC

P - Attiva la stampante sulla quale viene copiato tutto ciò che appare sul video (HARD-COPY)

Q - Disattiva la stampante.

R (return) - Visualizza tutti i registri nell'ordine:

Flag/A, B/C, D/E, H/G, Stak Pointer, Program Counter.

Per modificare il valore di questi registri è sufficiente scrivere, accanto ad R, il nome del registro voluto, cioè scrivere:

RA per il registro dei flag ed il registro A
RB per il registro B ed il registro C
RD per il registro D ed il registro E
RH per il registro H ed il registro G
RS per lo Stak Pointer
RP Per il Program Counter

D ind1,ind2 - Visualizza un'area di memoria, dall'Indirizo **ind1** (in esadecimale) all'indirizzo **ind2** (in esadecimale).

F ind1,ind2,dato - Pone in memoria, dall'indirizzo **ind1** all'indirizzo **ind2,** (in esadecimale), il dato (sempre in esadecimale) scritto nell'istruzione.

I ind. - Legge il dato presente in ingresso sulla periferica di indirizzo ind.

O ind,dato = Pone in uscita, sulla periferica di indirizzo ind, il dato scritto nell'istruzione.

H n1,n2 - Fornisce direttamente la **somma (n1 + n2)** e la **differenza (n1—n2)** in esadecimale dei due numeri definiti nell'istruzione.

M ind1,ind2,ind3 - Muove il blocco di memoria compreso fra **ind1** ed **ind2**, nella zona di memoria a partire dalla cella **ind3**.

G (return) - In questa forma, tale istruzione esegue un GO all'indirizzo presente nel program counter.

G ind (return) - Esegue le istruzioni del programma scritto in memoria a partire dalla cella di indirizzo ind.

G ind1,ind2 (return) - Esegue il programma scritto nelle celle di memoria a partire dalla locazione **ind1** e si ferma alla locazione **ind2**. È un'istruzione molto comoda per "testare" piccole parti di programma.

S ind - Sostituisce il dato contenuto nella cella di indirizzo specificato nell'istruzione, con quello digitato successivamente. Se si commette un errore di battitura, basta scrivere di seguito il codice operativo corretto dell'istruzione da sostituire in memoria **senza digitare nè lo SPAZIO nè RETURN**. Questi due comandi, oltre a convalidare il dato inserito nella cella di memoria chiamata, eseguono rispettivamente:
SPAZIO = avanzamento alla cella di memoria successiva, per un'ulteriore sostituzione.
RETURN = Esce dall'operazione di sostituzione.

CS ind1,ind2 (return) - Registra su cassetta tutto il programma compreso entro le celle di memoria dalla locazione ind1 alla locazione ind2.

CL (return) - Carica in memoria i dati del programma provenienti dal registratore. I dati vengono posti in memoria nelle stesse celle occupate dal programma originale.

CL n (return) - Carica dal registratore i dati relativi ad un programma. L'istruzione è del tutto analoga a CL appena descritta solo che i dati vengono posti in memoria partendo dalla locazione iniziale **a cui viene sommato il numero n.**

Abbiamo così terminato la descrizione di tutte le funzioni e di tutti i vari modi di operare, disponibili utilizzando la scheda di espansione presentata.

Lavorando con questo BASIC vi renderete rapidamente conto delle ampie possibilità di programma offerte da questo lingugagio e potrete così apprendere e verificare subito nella pratica come realizzare programmi in BASIC, iniziando da semplici listati fino ad arrivare a programmi elaborati e complessi.

COSTO DI REALIZZAZIONE

Tutto il necessario per la realizzazione di questa scheda con Basic da 16K residente, cioè un circuito stampato a doppia faccia con fori metallizzati siglata LX.548, tutti gli integrati completi di zoccoli, due connettori maschi, più le 8 Eprom già programmate ..... L. 158.000
Il solo circuito stampato LX.548 ..... L. 12.000