CBM

PROFESSIONAL COMPUTER

MANUALE D'USO

MODELLI 3032

CAPITOLO I

Congratulazioni e benvenuti nell'eccitante mondo dei personal computers. Scegliendo il PET 2001 avrete la certezza di avere un sistema di personal computer funzionante. In effetti se seguirete alcune semplici procedure descritte in questo manuale potrete iniziare ad operare con il vostro PET 2001 poco tempo dopo aver sballato il vostro calcolatore.

Le possibilita' potenziali del vostro PET sono illimitate. Questo manuale viceversa per la sua natura e' limitato. Esso non potra' quindi prevedere o spiegare tutte le domande che potranno sorgere.

Scrivete allora le vostre domande alla COMMODORE.

COMMODRE SYSTEM DIVISION

3330 Scott Blvd.
Santa Clara, Ca. 95050
U S A

360 Euston ROAD
London NWI 3B1
England

3370 Pharmacy Avenue
Agincourt
Ontario, M1W 2k4
Canada

PET e' un PERSONAL ELECTRONIC TRANSACTOR. E' costituito da un contenitore in metallo, in cui vi e' una scheda di controllo per il tubo a raggi catodici, la tastiera, la scheda del computer e un registratore Commodore. E' equipaggiato con un monitor televisivo in bianco e nero che visualizza i caratteri in un formato di 25 righe per 40 caratteri.

Al centro del vostro PET 2001 c'e' un microprocessore MCS 6502 che controlla totalmente le operazioni dello schermo, della tastiera, della cassetta e le periferiche addizionali che possono essere aggiunte al PET. Il tutto realizzato in modo che non possiate danneggiare il PET dalla tastiera. Il sistema operativo non puo' essere distrutto poiche' il software del computer o le istruzioni operative sono contenute nella memoria fissa (chiamata Read-Only-Memory). Questo modo di procedere permette sia all'utente inesperto che a quello sofisticato di usare il PET senza alcun pericolo.

Per soddisfare la necessita' di un utilizzatore esperto come quella di un inesperto, in questo manuale (originale) sono stati usati tre formati di stampa.

In corsivo sono riportati dei riassunti destinati al programmatore professionale. Quando si usa per la prima volta il manuale del PET bisogna spendere del tempo sulle spiegazioni piu' dettagliate che sono stampate nel carattere dei precedenti paragrafi. Dopo aver usato per un po' di tempo il PET le sezioni riassuntive possono essere usate per rivedere una particolare istruzione.

Questo tipo di formato in grossetto da una descrizione dettagliata di come il PET esegue una sezione. Queste parti sono per chi vuole usare il PET al livello di linguaggio macchina. Chi legge per la prima volta queste sezioni puo' trovare difficolta' nel capirle e si raccomanda di usarle solamente rileggendo il materiale dopo aver acquistato una maggior famigliarita' con il sistema operativo del PET.

Il linguaggio che sara' usato per comunicare con il PET e' chiamato BASIC, e' da tener presente che esistono altre pubblicazioni sul BASIC disponibili al pubblico. E' stata inclusa nell'appendice una lista di letture suggerite. Alcuni di questi manuali possono essere piu' adatti a soddisfare una specifica richiesta che non e' coperta da questo particolare manuale.

Lo strumento piu' adatto a comprendere cio' che il PET puo' fare e' il PET stesso. In molti casi, e' preferibile usare il PET con il testo piuttosto che continuare a leggere il testo stesso. In ogni caso , poiche' questo manuale non puo' evidentemente fornire tutte le informazioni tecniche relative agli aspetti dell'hardware o della programmazione del microprocessore MCS 6502 vi suggeriamo due altre pubblicazioni della COMMODORE: un primo manuale chiamato HARDWARE MANUAL che include tutte le descrizioni dei dispositivi ausiliari che generano i segnali di controllo necessari per permettere al PET di funzionare. Un secondo manuale chiamato PROGRAMMING MANUAL che contiene delle specificazioni dettagliate sul computer e sul linguaggio in cui esso lavora. Questi sono disponibili per l'acquisto presso il vostro fornitore o direttamente alla COMMODORE.

CAPITOLO 2

ESTRAZIONE DALL'INVOLUCRO E ACCENSIONE

E' necessario esaminare l'imballo per vedere se ci sono speciali istruzioni di sballamento e successivamente esaminare con cura il vostro PET per trovare qualsiasi danno nascosto. Se c'e' qualche danno e' necessario avvisare immediatamente il venditore ed il trasportatore.

Dopo aver tolto il PET dall'imballo e averlo posto su un opportuno supporto si inserisca la spina in una presa standard munita di collegamento a terra.

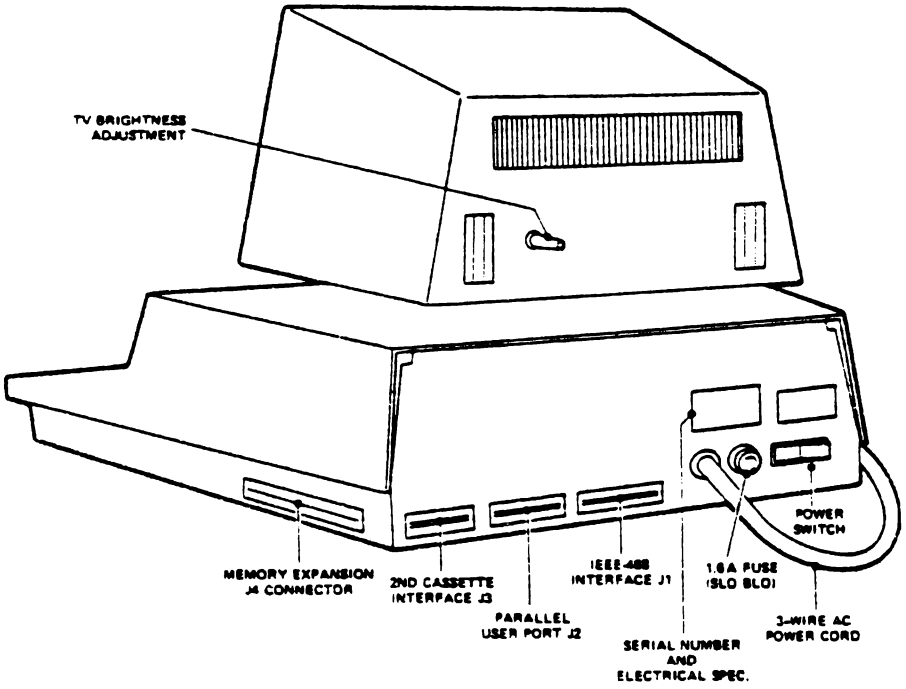


Figure 2.1. Rear view of PET 2001-8 showing switch, fuse, line cord and interfacing connectors

L'interruttore di accensione e' collocato sul retro del PET. Spostando l'interruttore verso sinistra si accende mentre verso destra lo si spegne.(Vi e' un punto bianco sull'interruttore d'accensione per indicare la posizione d'accensione o una etichetta ON/OFF).

Quando l'interruttore e' messo nella posizione ON, l'energia viene fornita immediatamente ai circuiti interni. Vi e' un circuito temporizzato che porta il calcolatore in uno stato noto e produce le condizioni di reset iniziali. Se lo schermo ha raggiunto le sue condizioni di regime prima di questo istante, si possono vedere sullo schermo una varieta' di strani caratteri che riflettono il contenuto corrente della memoria del computer che controlla lo schermo.

Il trasferimento della memoria dello schero allo schermo e' fatto con un circuito che opera al di fuori dal controllo del microprocessore e quindi anche quando il computer non e' operativo, lo schermo mostra sempre il contenuto corrente della memoria dello schermo.

Alla fine del ciclo di inserzione della tensione di alimentazione il computer inizializza la sua memoria interna, cancella temporaneamente lo schermo, quindi visualizza su di esso un messaggio come segue:

```
***COMMODORE BASIC***
   15359/31743 BYTES FREE
   READY
```

I numeri 15359/31743 si riferiscono alla quantita'di memoria disponibile all'utetente. La memoria e' strutturata in byte dove byte e' l'elemento fondamentale di dati del computer PET e corrisponde approssimativamente a una lettera o a un digit di informazione. Un PET da 16k mostrera' la cifra 15359 e un PET da 32k mostrera' che sono disponibili 31743 bytes.

Se il primo tentativo non dovesse aver successo e' necessario riprovare l'accensione aprendo l'interruttore e quindi richiudendolo immediatamente dopo.

Per ottenere il display, vengono usati quattro differenti tipi di memoria Read/Write di Utente,I/O (Input/Output) e memoria di schermo.

Le relazioni fra queste memorie sono illustrate in figura 2.2.

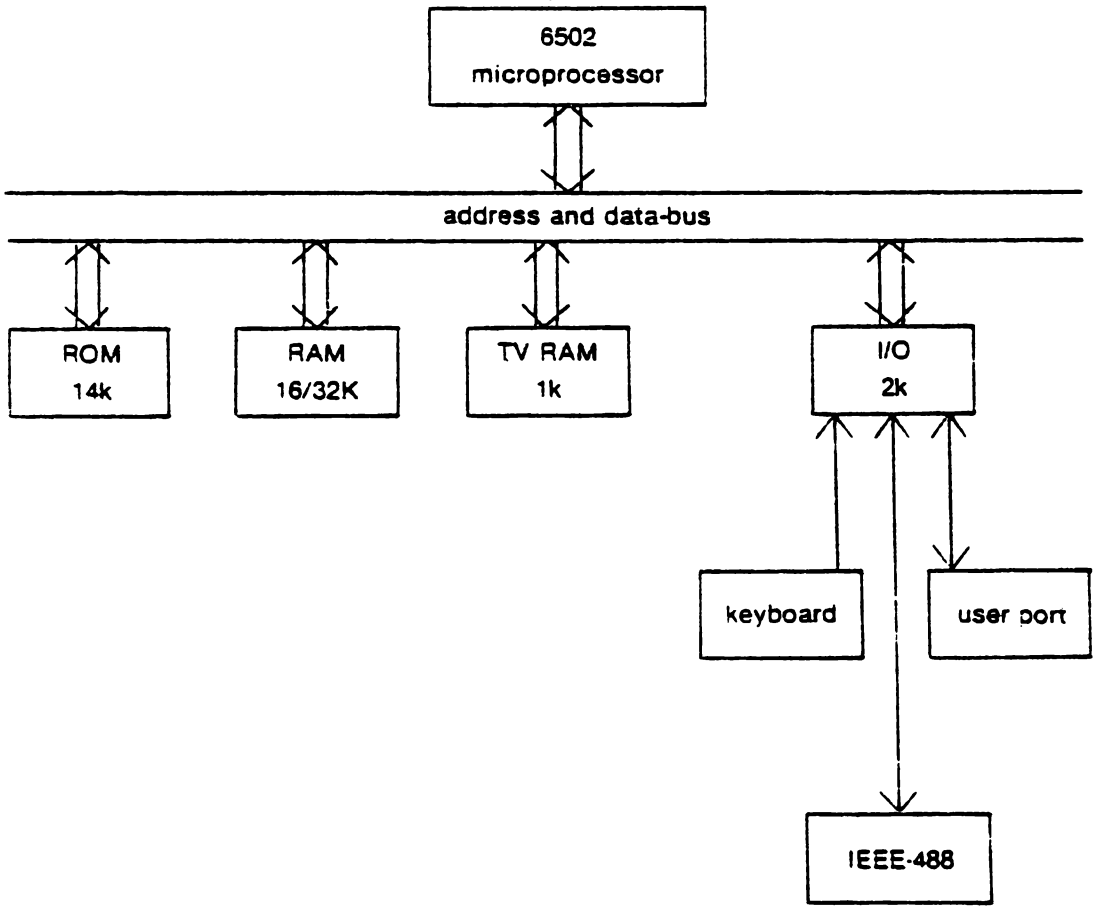


Figure 2.2. PET memory bus

ROM (READ ONLY - MEMORIA DI SOLA LETTURA)

La ROM permette al PET di eseguire le sue operazioni. In ciascun PET una memoria ROM di 14k contiene una serie di programmi scritti dalla Commodore che fanno l'escansione della tastiera e realizzano la scrittura sullo schermo, controllano l'Input; realizzano orologi in tempo reale ed eseguono comandi che l'utente batte sulla tastiera. La memoria di sola lettura non sono solamente le memorie di piu' basso costo per memorizzare questi dati ma anche permettono di dare la massima protezione contro gli errori dell'operatore e la piu' alta velocita' operativa della macchina. In tal modo infatti il sistema operativo e' indistruttibile dalla tastiera o per mezzo di un programma dell'utente.Questa macchina e' quindi non solamente disponibile per lavorare in BASIC dal momento della sua accensione ma anche il programma dell'utente non puo' danneggiare il sistema operativo BASIC.

MEMORIA DI INPUT/OUTPUT

Il secondo tipo di memoria e' quello che e' destinato a compiere l'operazione di Input/Output. Questa memoria contiene dei dispositivi di I/O chiamati PIA* e VIA** che permettono al PET di controllare individualmente i bit che manipolano il computer. Eccetto nei casi in cui si vogliono fare delle operazioni di I/O speciali, il programmatore non deve permettere che i suoi programmi interferiscono in qualche modo con queste aree.Il sistema operativo automaticamente manipola queste locazioni di memoria per eseguire le operazioni leggittime di I/O.

MEMORIA D'UTENTE A LETTURA/SCRITTURA - QUESTA MEMORIA E'ANCHE DETTA R.A.M.(O MEMORIA AD ACCESSO CASUALE)

Il terzo tipo di memoria e' lo spazio di memoria per i programmi di utente.(Nel resto di questo manuale essa viene chiamata area RAM).
In un PET standard da 8k essa e' collocata dalla locazione S0000 alla locazione decimale SIFFF.Una mappa dettagliata di tutta la memoria e' riportata in figura 2.3,e mostra la posizione della ROM. RAM.I/O della memoria di schermo e la loro collocazione standard.Potete vedere da questa mappa che i primi 1024 byte della memoria sono riservati per il sistema operativo e per i suoi svariati scopi che includono la bufferizzazione dei dati dalle cassette o dagli altri dispositivi di I/O. Il messaggio "7167BYTES FREE" e' il risultato di un analisi del BASIC che parte dalla locazione 1024 e cicla attraveso la memoria per determinare quali locazioni siano disponibili eseguendo quindi un test sul correttteo funzionamento della memoria.

Se il numero e' inferiore a 15359 o 31743,si sara' difronte a un problema di hardware.
BASIC puo' estrarre automaticamente fino a 32k di RAM purche' la memoria aggiunta sia contigua alla memoria che viene fornita con il PET.Questa memoria e' in realta' la memoria di lavoro della macchina; e' qui che i programmi vengono caricati e il BASIC fissa tutte le variabili del programma.

Nel seguito verranno discusse alcune tecniche per espandere questa memoria usando dei file e overlay di programma.

MEMORIA DELLO SCHERMO

La memoria dello schermo e' composta fisicamente dallo stesso tipo di integrati che vengono usati per la memoria standard del PET. Questa memoria viene costantemente scandita dal circuito di controllo del tubo a raggi catodici che estrae i bytes individuali e gli usa come indirizzo per uno speciale generatore di carattere a ROM, che fa apparire i caratteri sullo schermo. Come detto precedentemente, durante la discussione sull'accensione, questo modo di procedere e' totalmente automatico, ed il programmatore non ha nessun diretto controllo su di esso.

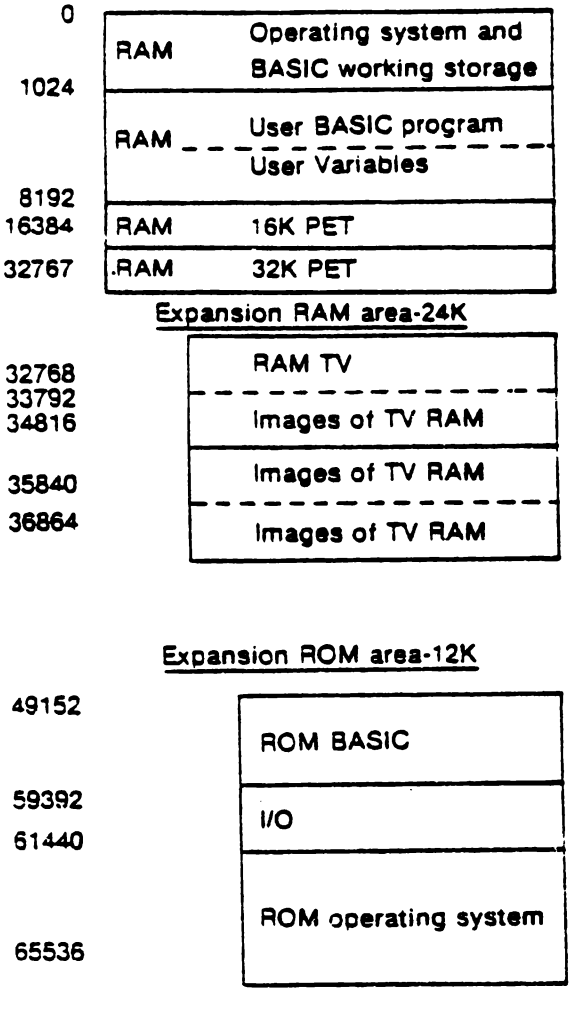


Figure 2.3. PET memory map

Per ciascun ciclo dello schermo TV (1/60 di secondo) l'hardware parte con l'indirizzo meno significativo ($8000) nella memoria di schermo ed inizia ad esplorare lo schermo partendo dall'angolo in alto a sinistra. Ciascun carattere nella memoria e' indirizzato nel generatore di caratteri 8 volte dando un carattere alto 8 righe sullo schermo. Il carattere ROM usato genera 8 punti ogni volta che viene indirizzato. Questi punti sono forniti in modo seriale allo schermo da sinistra a destra e dall'alto in basso. Questo da luogo ad un carattere di 8X8 punti senza spazi tra i caratteri.

Il controller del CRT cambia automaticamente l'indirizzamento della ROM che genera il carattere, a seconda che si stia scandendo la prima linea del carattere , la seconda ecc.

Si puo' cambiare la serie di caratteri sullo schermo caricando nell'indirizzo di memoria 59468 un 14 (un 12 fara' ritornare al primo set di caratteri). Dopo aver utlizzato per un certo tempo il primo insieme di caratteri e' opportuno testare se il vostro PET e' capace di utilizzare il secondo insieme. Quest'ultimo sostituisce con lettere minuscole l'insieme di caratteri grafici che e' disponibile nel primo insieme. Per comprendere meglio quanto detto e' opportuno vedere come i caratteri siano rappresentati nel PET e nella memoria.

RAPPRESENTAZIONE DEI CARATTERI NELLA MEMORIA DEL PET

Per rappresentare i caratteri nella memoria principale (RAM) viene usato il codice ASCII standard. Nel PET l'ottavo bit e' usato per indicare parole di comando BASIC o caratteri grafici per lo schermo PET.

B 6	0	0	0	0	1	1	1	1
5	0	0	1	1	0	0	1	1
4	0	1	0	1	0	1	0	1
3210								
0000	NUL	DLE	SP	0	@	P	SP	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	
1110	SO	RS	.	>	N	↑	n	
1111	SI	VS	/	?	■	←	o	

Figure 2.4. ASCII character set (7 bit code)

Esempio:
A e' rappresentato da 01000001
Il simbolo della picca e' rappresentato da 11000001
La memoria dello schermo e' organizzata con la rappresentazione differente che e' quella utilizzata nella memoria principale del PET.
Solamente 64 caratteri dall'insieme ASCII standard che sono normalmente stampabili.

B 6	0	0	0	0
1 5	0	0	1	1
1 4	0	1	0	1
3210				
0000	@	P		0
0001	A	Q	!	1
0010	B	R	"	2
0011	C	S	#	3
0100	D	T	$	4
0101	E	U	%	5
0110	F	V	&	6
0111	G	W	'	7
1000	H	X	(	8
1001	I	Y	)	9
1010	J	Z	*	:
1011	K	[	+	;
1100	L	\	,	<
1101	M	]	-	=
1110	N	↑	.	>
1111	O	←	/	?

Figure 2.5. ASCII 64 character set (6 bit code)

Essi sono gli stessi caratteri che sono direttamente disponibili sula tastiera del PET. La rappresentazione nella memoria dello schermo e' derivata dall'ASCII standard eliminando il bit 6; si ha in tal modo un codice a 6 bit dei caratteri presenti sulla tastiera. I caratteri grafici,o le maiuscole,sono rappresentati da un 1 nel sesto bit della memoria di schermo, permettendo quindila rappresentazione di ulteriori 64 caratteri. I caratteri rappresentabili sullo schermo del PET sono mostrati nella tabella difigura 2.6 E' bene notare che tutti i carattri grafici sono organizzati in modo da differire dalla corrispondente lettera unicamente per l'azionamento del tasto delle maiuscole sulla tastiera. Nella memoria di schermo l'VIII bit e' usato per ribaltare il campo di rappresentazione;cio' significa che nell'insieme dei punti forniti dal generatore di caratteri i punti neri vengono rimpiazzati con punti bianchi e i punti bianchi con punti neri.

B 6	0	0	0	0	1	1	1	1
1 5	0	0	1	1	0	0	1	1
1 4	0	1	0	1	0	1	0	1
3210								
0000	@	P		0	─	▌		┌
0001	A	Q	!	1	♠	●	▘	┴
0010	B	R	"	2	▐	▂	▝	┬
0011	C	S	#	3	─	♥		┤
0100	D	T	$	4	▗	▌	▁	▌
0101	E	U	%	5	▖	▗	▐	▙
0110	F	V	&	6	─	╳	▟	▛
0111	G	W	'	7	▐	○	▐	
1000	H	X	(	8	▐	✚	▂	▔
1001	I	Y	)	9	▚	▐	▶	▂
1010	J	Z	*	:	▚	◆	▐	▟
1011	K	[	+	;	▗	┼	▙	▪
1100	L	\	,	<	▟	▚	▪	▝
1101	M	]	-	=	▚	▐	▜	▟
1110	N	↑	.	>	▗	▀	▜	▝
1111	O	←	/	?	▐	▚	▁	▛

Figure 2.6. PET graphic character set (7 bit code)

Se viene usato il sistema operativo,esso automaticamente traduce i valori dal codice ASCII al codice di rappresentazione nella memoria di schermo.Sia il PRINT che l'imput diretto della tastiera da luogo a una traduzione automatica della memoria di schermo all memoria principale.

USO DELLA MEMORIA DI SCHERMO

Sono tre modi per inserire dei dati nella memoria di schermo.Il primo modo e' portare nell'appropriato indirizzo di memoria il desiderato carattere.Questo modo di procedere e' programmato solo quando la normale modifica dello schermo e' troppo lenta.Finche' il PET controlla direttamente lo schermo,non vi e' nesun effetto visibile per il fatto che lo schermo e il PET sono in contrapposizione per accedere alla memoria.Le routine presenti nel PET modificano la memoria di schermo solamente durante gli istanti in cui la memoria di schermo non viene usata per far apparire caratteri.Questo rallenta l'uso della memoria di schermo fino al 40% della memoria ottenibile con una operazione di forzamento memoria.

Il secondo modo per fa apparire dati sullo schermo e' la tastiera.Durante il periodo nel quale la tastiera e' abilitata,il carattere premuto sulla tastiera e' automaticamente visualizzato sullo schermo.

Il terzo modo di inserire dati sullo schermo e' l'uso di un comando PRINT in BASIC.Quando

PRINT"ABC"

e' inviato al BASIC, ne risulta che la successiva linea che viene stampata e':

ABC

Questo modo di procedere e' chiamato stampa di un campo letterale, e tutti i caratteri compresi tra le virgolette vengono visualizzati sullo schermo.
La successiva posizione in cui un carattere verra' visualizzato, se battuto sulla tastiera, viene indicato da un segnale pulsante chiamato cursore. Il cursore e' una indicazione visuale all'utente della successiva posizione di stampa sulla memoria di schermo. Lo schermo viene riciclato approssimativamente ogni sesantesimo di secondo, viene generato un interrupt. Questo da luogo ad un orologio in tempo reale nel computer e posiziona un contatore di lampeggiamento. Quando questo contatore raggiunge 37, il carattere individuato nella memoria di schermo dal pointer e' ribaltato sull'ottavo bit. Cio' fa si che il carattere di riferimento venga mostrato alternativamente in campo normale e inverso, dando un effetto visuale di lampeggiamento. Muovendo il puntatore si puo' stampare in qualsiasi posizione dello schermo. Questo e' fatto usando una combinazione della tastiera e del software chiamato editore di schermo, che manipola la memoria di schermo sotto controllo della tastiera.

CAPITOLO 3

La TASTIERA

Non appena il cursore lampeggiante appare sullo schermo il computer trasferisce i dati dalla tastiera alla memoria di schermo. I dati sulla tastiera vengono trasferiti per mezzo di una routine a interruzione alla memoria di schermo ogni volta che un nuovo tasto viene premuto. Solamente quando il ritorno carrello viene premuto i dati della tastiera vengono trasferiti al programma, viene cioe' trasferita una intera linea di caratteri alla volta.

Due eccezioni a quanto detto e in nessuno dei due casi si avra' il cursore lampeggiante. Uno di questi e' l'uso dell'istruzione GET, che sara' discussa piu' avanti, e l'altra e' quando i dati della tastiera vengono indirizzati direttamente usando dei programmi in linguaggio macchina. La tastiera del PET e' stata ottimizzata per l'uso come una tastiera da computer, tuttavia l'organizzazione e' simile a quella di una macchina da scrivere in modo che un normale dattilografo non si trovi spaesato.

Alcune importanti modifiche, tuttavia, sono state fatte:

1- per il grande uso che si fa con il computer di numeri e calcoli, e' stata aggiunta sulla parte destra della tastiera principale una tastiera da calcolatore.

2- I simboli sui tasti corrispodenti agli operatori matematici sono quelli normali per il BASIC.

3- I tasti per il movimento sullo schermo e per la correzione sono collocati sulla tastiera numerica.

4- I caratteri che sono normalmente utilizzati con il tasto delle maiuscole al di sopra dei numeri su una tastiera standard non richiedono viceversa l'uso del tasto delle maiuscole. Questi caratteri sono spesso usati nel BASIC ed e' conveniente non dover utilizzae il tasto delle maiuscole per adoperarli.

5- Tutti i caratteri standard sono minuscoli,in modo che l'uso del tasto delle maiuscole permette di utilizzare un insieme di 64 caratteri grafici.

LA TASTIERA DEL PET

La tastiera consiste in 73 tasti, inclusi due tasti di maiuscole,ciascuno dei quali puo' essere premuto per azionare il carattere che si trova sulla parte alta di ciascun tasto.I caratteri della parte bassa sono usati sempre a meno che uno dei due tasti delle maiuscole sia premuto. Ciascun tasto ha una sottile copertura in plastica trasparente che sta al di sopra della faccia superiore del tasto e che deve essere rimossa.Questa protezione e' messa per evitare danni ai tasti durante il trasporto. Per rimuovere il film, sollevare tale protezione plastica,in uno degli angoli,in modo da evitare di graffiare il di sopra dei tasti stessi.

Vi sono 64 caratteri sulla tastiera con 64 maiuscole sullo stesso tasto.Il resto della tastiera consiste di tasti di controllo.Molti di questi tasti di controllo sono ovvii:ad esempio il ritorno carrello o il comando di spostamento a destra o a sinistra del cursore.Il tasto di REVERS e permette di visualizzare tutti i successivi caratteri in modo inverso, scambiando cioe' il bianco con il nero.

Il tasto di REVERS opera in memoria.Ogni volta che il tasto viene premuto,la funzione Reverse diventa operazionale finche' non e' terminata da un RETURN;oppure premendo il REVERS-OFF (cioe' il tasto di Reverse assieme alle maiuscole).

Questo concetto di ribaltamento della funzione,ad esempio alto e basso,destra o sinistra,e' usato dappertutto in modo che la funzione complemnetare viene richiamata con il tasto della maiuscola.
La tastiera e' esplorata usando una 6420_PIA,un decoder da 4 a 10 linee e una routine ad interrup utilizzata dal controllo del tubo a raggi catodici.Ogni volta che si ha un segnale di interruzione la tastiera viene esplorata usando uno scansiometro da 2x5 righe essendo tale matrice ripetuta 8 volte slla tastiera stessa. Per incrementare una protezione contro i disturbi, la routine di scansione della tastiera memorizza il valore finale dell'ultima scansione in un buffer.
Con il tasto rilasciato, finche' un altro tasto viene premuto. Il tasto delle maiuscole e' un tasto speciale a chiusura multipla e viene trattato separatamente.Se uno o ambedue i tasti delle maiuscole sono premuti,il software condiziona un indicatore che e' usato per cambiare la decodifica del tasto.Tutte le chiusure dei tasti vengono tradotte usando una tabella memorizzata in ROM. Il tasto delle maiuscole viene codificato sull'VIII bit del carattere ASCII che viene poi tradotto nella rappresentzione di schermo in maniera standard.

Non appena la traduzione hardware e' stata fatta, il valore codificato viene trasferito in una coda della tastiera di 10 caratteri.La tastiera viene caricata ogni volta che un nuovo tasto viene premuto e viene scaricata alla velocita' permessa dal trasferimento dei caratteri allo schermo. Questa coda d'ingresso viene esplorata direttamente dalla routine GET per permette l'imput dei dati senza analizzare lo schermo. Lo stack d'ingresso viene esplorato dal programma di utente. Il programma di utente puo' esaminare il puntatore alla locazione 158 per determinare se esso e' maggiore o minore di zero;se esso e' maggiore di zero cio' significa che vi sono dati nella coda di tastiera. La coda di tastiera e' collocata alle locazioni da 623 a 632. Il primo carattere puo' essere estratto;tutti i caratteri seguenti vengono mossi verso il basso, e' un puntatore indice di caricamento decrementato di una unita'.L'operazione descritta e' pericolosa, a meno che non sia scritta in linguaggio macchina sulla basi di interrup mascherabili, in quanto una nuova chiusura di un tasto puo' memorizzare un nuovo valore durante il tempo nel quale si sta esaminanso la coda di tastiera e cambiare questa coda. Sia la routine GET che la routine d'ingresso da tastiera fanno tutto cio' automaticamente operando solamente durante l'interruzione o con interrup mascherabile.Tutto il tempo durante il quale il programma di correzione dello schermo e' operativo,uno speciale sistema operativo a due livelli,viene usato. Il primo livello abilita il cursore a lampeggiare e scrive i dati dalla tastiera alla memoria dello schermo alla posizione corrente del cursore. La routine allora muove il cursore di un carattere in basso nella memoria. Il procedimento e' ripetuto,tentando di mantenere la coda di tastiera vuota. Il secondo livello fa lampeggiare il cursore e traduce e memorizza i caratteri dalla tastiera nella coda di tastiera.In sostanza il primo livello del sistema operativo sorveglia costantemente le informazioni di imput alla ricerca di un ritorno carrello.Dopo che il ritorno carrello e' stato stampato,questa routine,automaticamente trasferisce l'intera linea al sistema operativo. Il resto del sistema operativo non conosce i caratteri finche' il ritorno carrello non e' stato stampato.

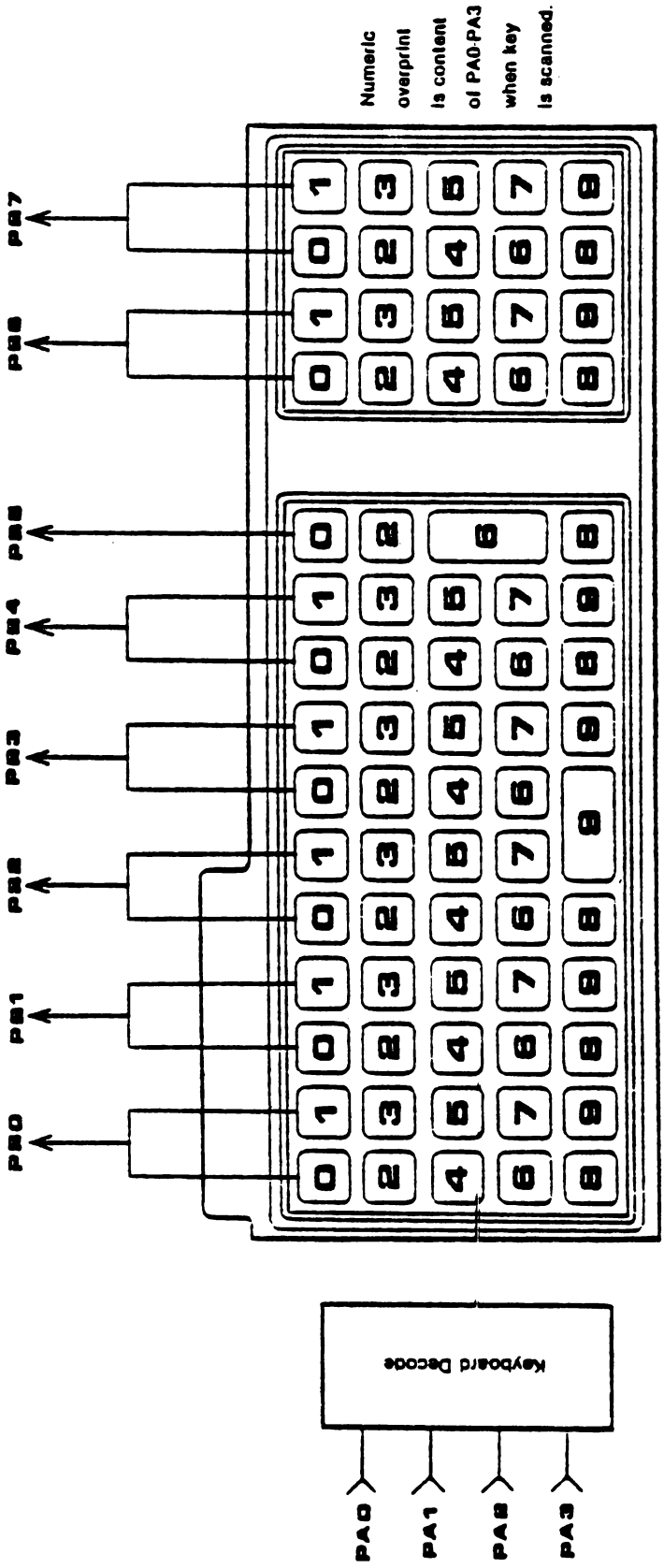


Figure 3.1. PET keyboard scan

PIA Data register addresses PA = 59408 PB = 59410

Questo permette la totale correzione della linea prima di consegnarla al sistema operativo. Un trucco interessante per i programmatori piu' esperti e' l'uso del PET per scrivere i propri programmi.Stampando una linea sullo schermo,forzando un ritorno carrello nella coda di tastiera e quindi-restituendo il controllo al BASIC, un nuovo numero di linea puo' essere fatto entrare nella memoria. Un altro esempio dell'uso della coda di tastiera e' la sequenza LOAD/RUN che e' incrementata nel programma di scansione della tastiera quando viene riconosciuto uno SHIFT-RUN, la routine allora automaticamente forza un "LOAD,RITORNO CARRELLO,RUN RITORNO CARRELLO" nella coda di tastiera.Quando il controllo viene restituito alla routine d'ingresso,il LOAD seguito da un RUN viene automaticamente trasferito nell'ordine giusto.
E' bene notare che la coda di tastiera e' lunga solamente 10 caratteri e se si supera tale lunghezza ,il vostro sistema puo' subirne conseguenze drammatiche. Il solo metodo per ovviare al superamento della lunghezza della coda e' spegnere il sistema e ripartire nuovamente.

PROGRAMMA DI CORREZIONE DELLO SCHERMO

Battendo sulla tastiera mentre il cursore e' attivo, fa si che quanto viene battuto sulla tastiera sia trasferito direttamente allo schermo.Questa funzione e' simile a quella di un semplice terminal di compiuter che richiede che voi battiate una linea errata finche' essa non e' giusta, ma il PET vi permette di correggere i vostri errori prima che la linea sia memorizzata. Il programma di correzione verra' meglio capito se si usera' il PET per seguire le discussioni che seguono, in quanto e' molto piu' semplice vedere che non descrivere gli esempi.
Per seguire questi esempi, due concetti sono necessari.Uno e' quello che quando si stampa un "?" il sistema operativo BASIC interpreta questo ? allo stesso modo dell'istruzione PRINT.
Il secondo concetto e' che quando si fa seguire a un ? le " tutti i caratteri dopo le " fino alle prossime " vengono tradotte dal BASIC come caratteri che voi volete stampare sullo schermo.
Il modo di operare del calcolatore che verra' usato in questa sezione e' conosciuto sotto il nome di modo diretto.(In contrapposizione a quello che viene chiamatomodo programmato).Il BASIC,cioe',eseguira' ciascuna istruzione non appena sara' stata battuta sulla tastiera e sara' stata fatta seguire da un ritorno carrello. Si vedranno in seguito che questo non e' normale modo di operare dei programmi. In questo modo infatti la macchina viene usata come super calcolatore.
La prima cosa da illustrare e' come la macchina stampi un semplice messaggio. Battendo sulla tastiera la linea ?"HI THERE seguito da un ritorno carrello il BASIC rispondera' stampando HI THERE. Si noti che ogni volta viene premuto un tasto sulla tastiera,il cursore si muove automaticamente di un posto a destra, permettendo di battere il successivo carattere e cosi' via , finche' non viene stampato il ritorno carrello. Quando si invia il ritorno carrello la scritta HI THERE appare immediatamente sullo schermo.
Si supponga ora di voler stampare la frase: questo e' un PET,ma nello scrivere la parola PET aver confuso il tasto P con il tasto B. Per poter correggere immediatamente questo errore vi e' un tasto che permette di cancellare un carattere gia' stampato.Questo tasto e' chiamato il tasto "DELETE", ed e' collocato nell'angolo superiore destro della tastiera.

Se si preme questo una volta,la lettera B sparira'.Stampando ora la P si nota che questo viene sovrastampato nella posizione in cui appariva precedentemente il B. Facendo seguire a questa P due caratteri ET e quindi il ritorno carrello, il calcolatore stampera' questo e' un PET, una linea di spazi bianchi e infine un READY. Il tasto di delete e' lo strumento fondamentale per le correzioni e permette di eliminare quanti caratteri fra quelli gia' inviati allo schermo si voglia e ristampare sulle loro posizioni.E' questa la forma piu' semplice di correzione. Essa e' realizzata incrementando il puntatore sullo schermo di una posizione e lascinado uno spazio bianco nella posizione in cui esso precedentemente si trovava. Si puo' rtornare indietro cancellando la scritta READY che e' immediatamente prima alla posizione del cursore semplicemente continuando a battere il tasto di delete.Si noti che premendo il tasto lentamente il cursore si muovera' di una posizione a ciascuna battuta,mentre se il tasto viene premuto con forza il cursore si movera' do parecchi caratteri prima di vederlo lampeggiare. Durante l'operazione,inoltre,il cursore puo' muoversi da una linea all'altra, infatti la memoria dello schermpo e' organizzata in modo che cancellando il precedente carattere nella memoria,il puntatore si muove indietro sul carattere precedente.A causa del fatto che i caratteri sono cancellati muovendosi da sinistra a destra lungo righe di 40 colonne, se ad esempio si cancella il carattere all'inizio di una linea e si aziona successivamente il tasto di cancellazione,il carattere che verra' eliminato sara' il quarantesimo carattere della linea precedente. Se si deve correggere qualcosa che si trovi sulla linea o sulla linea precedente a quella attualmente raggiunta, tale modo di procedere e' in effetti troppo lento.
Sul PET vi sono allora tre tasti che permettono il movimento del cursore. Un tasto fa muovere il corsore destra a sinistra; o viceversa; il secondo tasto lo muove in alto e in basso; il terzo tasto lo muove all'amgolo in alto a sinistra dello schermo e cancella lo schermo.

MOVIMENTO A DESTRA O A SINISTRA DEL CURSORE

Il tasto per il movimento verso destra del cursore fa muovere il puntatore di un carattere verso destra. Se si preme questo tasto ad esempio per cinque volte si puo' vedere che il cursore si muove di cinque posizioni verso destra.Evidentemente tutto questo viene fatto modificando il puntatore del cursore nella memoria. Il tasto per lo spostamento verso dinistra e' lo stesso tasto che viene usato per lo spostamento verso destra ed e' utilizzato premendo il tasto delle maiuscole prima di premere il tasto di spostamento.Se lo si batte per quattro volte, si vede che il puntatore si sposta verso sinistra di quattro posizioni. Quando si e' giunti all'inizio di una riga un'ulteriore pressione sul tasto di movimento verso sinistra fa spostare il puntatore all'estremita' della riga precedente.Questo tasto permette, evidentemente, di correggere con maggiore facilita' i testi che appaiono sullo schermo.Una possibilita' di correzione piu' rapida di questa viene offerta dall'uso del tasto di movimento in alto e in basso che e' disponibile sul vostro calcolatore.

MOVIMENTO IN ALTO E BASSO DEL CURSORE

L'azionamento del tasto per il movimento verso il basso fa muovere il puntatore di 40 colonne a destra rispetto alla sua attuale posizione.

Cio' fa si che l'effetto visivo sia praticamente uguale al muovere il puntatore di una linea sullo schermo.Infatti se si aziona il tasto precedentemente descritto per gli spostamenti verso destra 40 volte, si vedra' che il puntatore si trovera' nella medesima posizione sullo schermo una riga piu' in basso. Per muovere il cursore verso l'alto e' sufficiente azionare il tasto della maiuscola insieme al tasto di spostamento alto-basso.Questo fa si che il cursore si muova di una posizione verso l'alto ogni volta che il tasto sara' premuto.

CORREZIONE DELLO SCHERMO

Si possono ora usare i movimenti del cursore per portarlo nella posizione corrispondente alla seconda H della scritta HI THERE PET.Una volta che cio' sia stato fatto, si puo' cancellare la T premendo il tasto di cancellazione,il tasto delete. Si noti che tutti i caratteri alla destra del carattere cancellato vengono spostati verso sinistra di una posizione.Si vede,cioe',che la cancellazione e' uno spostamento di tutti i caratteri presenti in ,memoria di una posizione a sinistra, anziche' la sostituzione del carattere con uno spazio in bianco.

INSERZIONE E CANCELLAZIONE

Prima di analizzare le operazioni di inserzione e cancellazione, e' bene ricordare che la memoria dello schermo e' organizzata in modo che ogni singola linea consista di 40 caratteri.L'inserzione e la cancellazione si riferiscono ai caratteri di una linea. Non appena il tasto di cancellazione viene premuto, tutti i caratteri, a partire dalla posizione del cursore, fino alla fine della linea, vengono automaticamente spostati di un carattere a sinistra,, rimpiazzando il carattere che precede il cursore.Il cursore e' esso stesso spostato alla posizione del carattere rimpiazzato.L'ultimo carattere della linea viene automaticamente sostituito da uno spazio bianco.L'inserzione e' il contrario di quanto descritto.Si ritorni ora alla linea dalla quale e' stato eliminato il carattere T e si voglia inserire di nuiovo la T tra lo spazio bianco e HERE.Per fare cio' e' necessario inserire uno spazio in cui stampare la T. Cio' puo' essere fatto premendo una sola volta il tasto di inserzione insieme al tasto della minuscola. Premendo successivamente la T scritta viene ad essere ricostituita ed appare sullo schermo come HI THERE PET, mentre il cursore lampeggera' immediatamente dopo il carattere inserito. Per inserire piu' di un carattere, e' necessario premere il tasto di inserzione piu' di una volta; questo fa si che tutti i caratteri della linea si muovono verso destra e il cursore punti al primo carattere da inserire.Questo modo di operare permette di inserire diversi caratteri in ciascuna linea.Si noti che durante queste operazioni, il computer non fa niente altro che modificare quanto appare sullo schermo, cio' avviene in quanto non e' stato premuto il ritorno carrello che avverte il PET che la linea e' completa.Tutte le operazioni sono infatti eseguite attraverso il programma di correzione dello schermo.Tutte le correzioni sono compiute lavorando tra tastiere e memoria di schermo, senza interferire in alcun modo con il resto del sistema operativo.Questo permette all'utilizzatore di comporre dei testi perfetti e inviarli al computer prima di rendere il programma operativo, evitando tutti i passi intermedi di correzione.

LE LINEE SULLO SCHERMO DEL PET

Fisicamente, una linea sullo schermo consiste in 40 colonne d'informazione.Tuttavia, per tradizione nei calcolatori commerciali, molti dati sono organizzati per schede dati a 80 colonne, poiche' su 80 colonne si possono ovviamente inserire molti piu' dati che non su 40.Tuttavia, malgrado che lo schermo del PET possa visualizzare solamente 40 caratteri per ciascuna linea, l'utente ha tutte le possibilita'di utilizzo che nel caso di linee di 80 colonne. Tutto cio' viene realizzato permettendo che sullo schermo vengano definite delle linee con piu' di 40 caratteri. Infatti continuando a battere dei tasti quando si giunge alla fine di una riga, il cursore viene automaticamente spostato alla linea successiva. Lo schermo allora considera queste due linee come una unica linea ad 80 colonne, anziche' come due linee a 40 colonne.
Per permettere al PET di eseguire questa funzione vi e' all'inizio di ogni linea una tavola dei puntatori.Ciascuna linea ha un marcaggio che indica se essa e' l'inizio di una linea o una linea di continuazione. Il puntatore e' conservato mel bit di segno dek puntatore indice. Quando si ha un movimento del cursore verso l'alto o verso il basso, il programma di correzione esamina lo stato di questo puntatore di linea in modo da inizializzare il PET al numero di linea appropriato. Ogni volta che il puntatore e' attivato sullo schermo, vi e' un valore memorizzato che informa il calcolatore su quale e' la prima linea che appare sullo schermo stesso e su cui il cursore potra' opeare.La posizione sullo schermo viene adoperata come un puntatore separato che informa il PET se la linea e' maggiore o minore di 40 caratteri.Quando si ha un'uscita di qualche linea dello schermo, il puntatore di linea viene modificato in modo da mantenere l'informazione sulla prima linea.La tavola dei puntatori di linea e' collocata in memoria alle locazioni da 224 a 248.

SCORRIMENTO DELLO SCHERMO

Si abbia sullo schermo una mescolanza di linee da 40 a 80 colonne; si veda cosa accade se si tenta di muovere il cursore al di la della parte bassa allo schermo.Si faccia scorrere il cursore verso il basso finche' esso si trovi alla base dello schermo.Tentendo di passare alla linea successiva avviene che l'intero schermo si muove di una linea verso l'alto.Ogni volta che si tenta di stampare piu' di mille carattri sullo schermo, il programma di correzione dello schermo automaticamente fa muovere l'intero schermo di una linea verso l'alto. In effetti le linee si muovono verso l'alto sullo schermo di una o due linee a seconda dello stato della linea iniziale che compare sullo schermo stesso.Questo viene fatto nell'hardware testando il puntatore della seconda linea. Se viene riconosciuto che essa e' la continuazione in una linea a 80 colonne, tutto lo schermo viene mosso verso l'alto di due righe, cio' corrisponde a dire che l'ottantunesimo carattere dei mille caratteri presenti sullo schermo viene spostato all'inizio della memoria di schermo e che gli ultimi 80 caratteri della memoria vengono riempiti con degli spazi bianchi.Se la prima linea e' una linea di 40 colonne si ha lo spostamento di una sola riga, il quarantunesimo carattere viene mosso nella prima posizione della memoria e i 40 caratteri finali vengono riempiti con spazi bianchi.

Il cursore viene automaticamente posizionato all'inizio di questa zona di spazi bianchi e cio' si puo' vedere provando a muovere il cursore verso il basso. Il procedimento e' completamente automatico ed e' causato dal tentativo di inviare un ritorno di carrello oltre con uno spazio al di la del fondo dello schermo.Oltre al movimento non si ha nessuna altra azione.Si puo' vedere che se un programma che usi scrittura da luogo a scorrimento;esso e' troppo veloce per essere avvertito.Se il tasto di reverse viene mantenuto premuto durante la stampa,la velocita' di scorrimento viene diminuita di un fattore 20.

CAPO PAGINA E CANCELLAZIONE

Il tasto di HOME fa si che il cursore si porti all'angolo in alto a sinistra dello schermo (che corrisponde alla prima locazione della memoria di schermo).Mantenendo premuto il tasto di spaziatura e premendo il tasto di clear si ottieneuno schermo completamente cancellato con il cursore nell'angolo in alto a sinistra.I comamdi di cancellazione di capo pagina possono essere fatti su qualsiasi posizione dello schermo.

CAPITOLO 4

GLI ELEMENTI FONDAMENTALI DEL BASIC

Il processo creativo di combinare le istruzioni per risolvere un particolare problema, non puo' essere illustrato in questo testo. Questo manuale,infatti, non puo' risolvere il vostro particolare problema; esso puo' tuttavia fornirvi gli strumewenti in modo da utilizare il PET per risolvere il vostro problema.

L'ISTRUZIONE PRINT

Un calcolatore puo' manipolare un numero impressionante di dati,ma tutto cio' non ha senso se il risultato del calcolo non puo' essere visualizzato. Per questo motivo la descrizione del BASIC iniziera' dall'istruzione PRINT. Quando si stampa un testo,l'istruzione PRINT puo' essere abbrevviata con un "?".Un'istruzione quale:

```
PRINT "HELLO"
```

e' una istruzione al calcolatore che fa si che esso stampi sullo schermo quanto e'contenuto fra le "",nel caso in questione,una parola di saluto.Oppure l'istruzione:

```
PRINT1024*8
```

e' un'istruzione che fa stampare il prodotto di 1024 per 8.
E' bene notare che il BASIC permette di stampare piu' di un valore per volta.Anziche' far stampare una quantita' su una prima linea e una seconda quantita'su una seconda linea e' ad esempio possibile scrivere l'istruzione:

```
PRINT 1024)2,1024)3
```

che fa stampare il quadrato di 1024, alcuni spazi ed infine il cubo di 1024. Nella successiva sezione saranno contenuti dei dettagli sull'esatto formato di stampa.Qui si vuol solo far presente che un certo numero di dati possono essere stampati sia su una serie di linee successive che su una medesima linea. A meno che il calcolatore non sia stato istruito altrimenti per mezzo del comando CMD, tutte le uscite di stampa finiscono sullo schermo del computer stesso. I caratteri sono stampati nella prima posizione disponibile alla stampa sullo schermo, sotto controllo del BASIC e di un programma editor che memorizza la posizione sullo schermo.

Sebbene la rappresentazione fisica sullo schermo sia di 25 linee e di 40 caratteri, la stampa di linee fino a 80 caratteri;viene fatta eseguendo automaticamente un a capo dopo il quarantunesimo carattere su ciascuna linea. Il computer automaticamente fa scorrere la visualizzazione sullo schermo di una o di due linee quando esso raggiunge la visualizzaione di un migliaio di caratteri. Il comando di PRINT ha due forme principali. La prima permette la stampa di una singola variabile nello spazio ad essa destinato.Se il dato viene presentato in questa forma,il campo viene stampato dalla posizione corrente sullo schermo e facendo seguire la stampa con un ritorno carrello.Se i dati sono presentati nella forma PRINT A,B allora il BASIC ,automaticamente, stampa "A" partendo dalla posizione corrente sullo schermo, poi dopo aver imposto una spaziatura di 10 caratteri, stampa il valore "B" seguito da un ritorno carrello.Per impedire che il BASIC effettui il capo riga dopo la stampa, viene usato il ";".L'istruzione PRINT A;B; fa si che venga stampato A seguito immediatamente,senza alcuna spaziatura dal valore B.Il cursore viene posizionato alla fine del campo di B.Se la variabile A ha piu' di 7 caratteri, B viene stampato dopo una spaziatura di 20 caratteri, quando viene usata l'istruzione PRINT A,B.
Il BASIC in stampa usa le seguenti regole: quando la quantita' da stampare e' una stringa, non vi e' nessun carattere che preceda o segua tale stringa.Se la quantita' da stampare e' un numero, per prima cosa il BASIC esamina la sua grandezza.Se il numero e' minore di 0,01 o maggiore uguale a 999999999,2,il BASIC lo stampa usando l'annotazione scientifica.
Ad esempio:0,0034 e' stampato come 3,4 E-03,mentre il numero -1234567890,5 viene stampato come-1,2345678E+09.Se il numero ha un valore compreso tra 0,01 e 999999999,2 vengono stampate le 9 cifre piu' suguificative piu' il punto decimale se necessario. Gli zeri non significativi dopo il punto decimale non vengono stampati. Il BASIC stampa inoltre un carattere bianco dopo un numero (a meno che questo non venga stampato sotto forma di stringa).
Per utilizzare al meglio le possibilita' del PET di comporre un testo sullo schermo, e' bene notare che a differenza di parecchi BASIC, lo spazio che appare fra i campi di stampa e' sempre un caratterere di salto; ottenuto mediante il comando di spostamento a destra del cursore. Cio' causa che il puntatore di schermo sia avvanzato di un carattere. Poiche' il PET permette l'inclusione di tutti i comandi di posizionamento del cursore, come caratteri alfanumerico in una stringa, il programmatore ha un pieno controllo sulla posizione di stampa sullo schermo.I caratteri di controllo del cursore che possono venir usati come caratteri alfanumerici sono:il comando di cancellazione dello schermo, il comando di a capo pagina,il comando di spostamento a destra e a sinistra, in alto e in basso del cursore. Con l'uso di questi caratteri si possono comporre campi di qualsiasi lunghezza e di qualsiasi dimensione che partono da qualsiasi delle mille posizioni disponibili sullo schermo del PET. Come precedentemente discusso, la memoria di schermo del PET consiste in mille caratteri di memoria posizionati dallaposizione dalla locazione di memoria 8000 esadecimale.I caratteri sono presentati nella memoria di schermo in un codice ASCII a 6 bit concatenato con 2 bit addizionali.Uno di questi bit e' il campo inversoe il secondo e' un bit maiuscolo-minuscolo.

Quando si stampa sullo schermo la subroutine di stampa nel sistema opertivo,automaticamente traduce i caratteri ASCII nella forma della memoria di schermo.I caratteri di controllo dello schermo fanno semplicemente spostare il cursore sullo schermo stesso. Il carattere di capo pagina fa muovere il cursore all'inizio dello schermo. Il carattere di cancellazione fa spostare il cursore all'inizio dello schermo e inserisce in tutti i mille caratteri dello schermo la rappresentazione dello spazio bianco. Nel BASIC i numeri vengono rappresentati con qualita' binarie a 5 byte, eccetto il caso speciale degli interi, che vengono rappresentati con 2 byte. Tuttavia per quanto riguarda la stampa,il BASIC stampa gli interi allo stesso modo dei numeri in floating point.
Infatti, il BASIC, automaticamente converte gli interi in virgola mobile e la routine della stampa per i numeri in virgola mobile converte successivamente tali numeri in caratteri stampabili.

LE VARIABILI

Si e' gia' detto in precedenza che il PET puo' essere usato come un calcolatore di grosse dimensioni che esegue funzioni matematiche e quindi ne stampa i risultati.Tuttavia,in molti casi,il programma necessita di valori intermedii o di operazioni interattive. Per effettuare programmi a tutti i livelli, e' stato permesso l'uso di funzioni che possono assumereuna varieta' di valore volta per volta.Una funzione che puo' avere qualsiasi valore viene definita sia in algebra che in programmazione come variabile.Se non vi e' famigliare il concetto di varibile attraverso la matematica; un libro di algebra elementare o un testo elementare del BASIC potra'aiutarvi. Quanto verra' trattato in seguito riguardera' l'uso delle variabili.
Nel BASIC le variabili sono definite mediante due caratteri alfanumerici.Se la variabile e' una variabile numerica allora non vi e' nessun carattere particolare nel suo nome.Il carattere A e' considerato essere la variabile A.Il carattere AA e' una differente variabile. Il carattere A1 e' una terza variabile,ma tutte queste tre definizioni si riferiscono a valori numerici.Se la variabile contiene dati alfa-numerici, essa viene definita una stringa. Una variabile stringa deve terminare con il carattere $. Quindi,A e A$ sono rispettivamente un valore numerico ed una stringa e sono variabili differenti. AA$, evidentemente, e' differente da AA, ecc. Il BASIC distingue una variabile per il fatto che il primo carattere e' sempre un carattere alfabetico. Il secondo carattere puo' essere sia un carattere alfabetico che un carattere numerico. Una variabile intera termina con un percento(%). Esempio:A%.

LE MATRICI

Le matrici sono il quarto tipo di variabile che puo' essere definito nel BASIC. Una matrice e' differenziata dalla parentesi che la segue. La parentesi definisce il valore particolare nella matrice che deve essere usata nell'espressione.A(0,1) indica il primo valore della seconda riga di una matrice a due colonne ed e' differente da A,A$ e A%. Tutte queste variabili possono essere specificate nello stesso programma.Le definizioni di specifica e le tecniche di allocazione in memoria per ciascun tipo di variabile verranno date in seguito, in questo momento saranno presi in esame alcuni esempi di uso di ciascuna variabile.

L'uguale e' usato con due significati:se esso e' utilizzato in un'istruzione del tipo IF-THEN, l'uguale ha lo steso significato della funzione matematica;il valore alla sinistra dell'espressione viene comparato e deve esere uguale al valore alla destra;nell'altro caso,quando l'uguale segue una variabile come nell'espressione A=2+2, l'uguale fa si che il valore di A venga sostituito con il risultato dell'espressione che si trova sulla destra del segno di uguale.
Il BASIC originario richiede la parola LET prima di qualsiasi assegnazione di variabile,ma nel PET la parola LET e' opzionale e puo' esere omessa. A=2 e' l'equivalente a LET A=2. Il comando CLR mette tutte le varaibili nel PET a zero. Per capire come la variabile operi nel BASIC si daranno ora alcuni esempi da eseguire sul vostro PET. Si ricordi di premere il RETURN dopo che ciascun comando e' stato battuto sulla tastiera.

```
CLR
?A
Il PET stampera' 0
Ora battete
A=2+2
?A
Ora il PET stampera' 4
Battete
?B
Il PET stampera' 0
Ora rimpiazzate il valore di B con due volte il valore di A battendo:
B=2*A
?B
Il PET stampera' 8
Ora cambiate il valore di A battendo:
A=2+3
?A
Il PET stampera' 5
Se battete
?B
```

Il PET rispondera' lo stesso valore di prima.Finche' non darete una nuova espressione per B o rieseguirete come prima B=2*A,il alore di B rimarra' 8.

VARIABILI IN VIRGOLA MOBILE

Il BASIC opera sempre in aritmetica in virgola mobile;quindi per ciascuna variabile viene assegnato in memoria uno spazio che e' quello necessario a un numero in virgola mobile.4 bytes contengono una rappresentazione binaria di questo numero.Cio' permette di specificare fino a 9 cifre significative per il numero stesso. Per parecchi calcoli l'accuratezza viene limitata da questa rappresentazione.L'ulteriore byte serve a memorizzare un esponente che puo' raggiungere un valore massimo di +33. Gli esponenti piu' piccoli di -34 danno luogo a numeri troppo piccoli per essere distinti dallo zero.

VARIABILI STRINGA

Una variabile stringa puo' contenere una funzione,un numero,un carattere grafico o un carattere standard ASCII. Vi e' uno specifico insieme di varibili che permettono l'estrazione e l'impaccamento dei dati in stringhe;queste funzioni saranno rappresentate piu' avanti.La stringa e' limitata a 80 caratteri dalla dimensione del buffer di imput. Vi e' un insieme specifico di funzioni che permettono la costruzione di stringhe fino a 255 caratteri.

GLI INTERI

Le variabili in virgola mobile sono memorizzate nel BASIC mediante 5 bytes:uno per l'esponente e 4 per la mantissa in modo da ottenere l'accuratezza di 9 cifre significative. In parecchi casi,quando la quantita' e' una quantita' intera, la variabile puo' essere espressa in maniera piu' semplice. Per poter avere una maggiore efficenza della memoria,in modo particolare nel caso di matrici che possono occupare una quantita' notevole di memoria, il PET ha la possibilita' di memorizzare certi numeri come valori interi in 2 bytes. La rappresentazione e' binaria pura e permette di memorizzare numeri il cui valore sia compreso tra -32767 e +32767, la forma di memorizzazione,come detto, e' a 2 byte con il bit di ordine piu' alto del numero che contiene il segno.

USO DEL PROGRAMMA E ISTRUZIONI DIRETTE

Fino a questo momento e' stata usata una tecnica di programmazione che fa si che il PET risponde direttamente all'istruzione stampata. In questo caso il BASIC esegue immediatamente il comando che gli e' stato dato attraverso la tastiera non appena viene dato un ritorno carrello. Questo modo di operare viene chiamato,modo diretto. In questa maniera il PET viene usato come un super-calcolatore. Ad esempio se si vuole che il PET addizioni due numeri e divida il risultato per1/5 si puo',ad esempio,scrivere l'istruzione: ?(2+8)/5; se questa e' l'istruzione che e' stata battuta sulla tastiera,si avra' come risposta 2 seguito dalla frase READY.Il PET, cioe',obbedisce istantaneamente a qualsiasi istruzione assegnatagli attraverso la tastiera;eccetto quando e' in esecuzione un programma BASIC.Il modo diretto e' usuale per la correzione dei programmi per il calcolatore.Alle variabili,in questo caso,possono venir assegnati dei determinati valori e quindi puo' venir eseguito un piccolo pezzo del programma con una istruzione di GOTO per accertare se questo particolare pezzo del programma sia correttamente funzionante o meno.Inoltre,sempre con il modo diretto, nei programmi possono essere inseriti dei punti di rottura e si puo' testare lo stato delle variabili con dei comandi di stampa senza modificare il programma principale. Tuttavia eccetto che nei casi di correzione o nel caso in cui il PET venga usato come super calcolatore, per utilizzare la macchina come un vero calcolatore e' necessario scrivere o caricare un programma BASIC.La differenza tra l'esecuzione in modo diretto e quella a programma e' che parecchie istruzioni possono essere raggruppate assieme in ordine logico e il BASIC le eseguira' una in successione all'altra prima di restituire il controllo all'utente. Si supponga di volere che il BASIC stampi la frase HI THERE verticalmente anziche' orizzontalmente. Si puo' fare cio' con un programma,mentre non e' altrettanto facile ottenere lo stesso risultato in modo diretto.Le operazioni da eseguire per caricare un programma sono molto semplici. Qualsiasi istruzione si voglia sia trattata dal BASIC come un'istruzione di programma, deve essere preceduta da un numero di linea.Un numero di linea puo' essere un qualsiasi numero compreso fra 0 e 63999.
E' buona abitudine,quando si sviluppi un programma,assegnare i numeri di linea incrementandoli di 10 o di 100, cioe' anziche' numerare le istruzioni con i numeri 1,2,3, ecc., usare 10,20,30 ecc.Questo vi permette di avere uno spazio tra una istruzione e l'altra in modo da poter aggiungere successivamente delle nuove linee ed effettuare correzioni del vostro programma.

E' necessario a questo punto rimarcare che il BASIC eseguira' ciascuna istruzione in ordine crescente di numero di linea.Per stampare HI THERE verticalmente, ciascuna linea del programma stampera' una lettera del messaggio, bastera' che la prima linea sia numerata con 10 e tutte le successive abbiano un numero di linea con un incremento di 10.

```
10?"H"
20?"I"
30?"T"
40?"H"
50?"E"
60?"R"
70?"E"
```

Si ricordi che anche nella preparazione di un programma ogni linea deve essere chiusa con un ritorno carrello,come nel caso dei comandi diretti, in modo da informare il PET che la linea e' terminata. Le linee da 10 a 70 costituiscono un programma che permette al PET di stampare la frase voluta.

Il programma e' ora residente in memoria. Per eseguire il programma, e' necessario battere sulla tastiera RUN. Questo permette di stampare in verticale la frase HITHERE.

Si noti che non c'e' nessuno spazio fra la I e la T. Una delle ragioni per cui si sono usati dei numeri di linea spaziati di 10 e' che si puo' ora inserire una linea di correzione fra la linea 20 e la 30. Per prima cosa e' bene far apparire sullo schermo il programma, stampando il LIST. Questo fa si che il programma appaia sullo schermo come segue:

```
10 PRINT "H"
20 PRINT "I"
30 PRINT "T"
40 PRINT "H"
50 PRINT "E"
60 PRINT "R"
70 PRINT "E"
```

Si stampi ora:

```
25?""
```

Dopo aver premuto il tasto del ritorno carrello e rifatto il LIST del programma e dopo aver visto che la linea 25 e' stata inserita tra la linea 20 e 30; si puo' far girare il programma che da ora in verticale la stessa scritta inserendo uno spazio bianco fra la I e la T.

Questo esempio dimostra l'uso che si puo' fare dei numeri di linea e la possibilita' di inserire dei numeri di linea per eseguire delle correzioni in un programma. Vi e' un secondo modo per ottenere lo stesso risultato. Per prima cosa si cancelli lo spazio battendo sulla tastiera 25 seguito da un ritorno carrello. Si listi poi il programma e si veda che la linea 25 e' stata cancellata. Ora si posizioni il cursore sullo spazio che segue la I sulla linea 20, e si inserisca un carattere di spostamento verso il basso del cursore. Per prima cosa e' necessario premere il tasto di inserzione e solo successivamente il tasto che comanda lo spostamento del cursore verso il basso. Se questa operazione non viene eseguita nel modo corretto, il cursore si muovera' verso il basso immediatamente. Poiche' pero' il tasto di spostamento verso il basso del cursore e' stato premuto dopo il tasto di inserzione il cursore non si muovera' finche' l'istruzione 20 non passera' in esecuzione. Non si dimentichi nemmeno di premere il tasto della maiuscola prima di premere il tasto dell'inserimento.

Quando si fa nuovamente girare il programma, si puo' vedere che si ha lo stesso effetto del caso precedente inserendo uno spazio dopo la I. Allo stesso modo possono essere utilizzati i caratteri di controllo del cursore, ad es.: il carattere di "capo pagina" o il carattere "cancellazione".

Il programma di correzione dello schermo permette di modificare qualsiasi linea di programma presente sullo schermo stesso.

Il comando LIST ha diverse opzioni che aiutano a trovare la linea da correggere. Il comando LIST rintraccia i programmi e stampa il contenuto del programma memorizzato sullo schermo. Il comando LIST parte dal primo numero di linea presente in memoria e lista sullo schermo tutte le istruzioni fino all'ultima. E' possibile listare la singola linea, ad esempio LIST 20 fara' apparire sullo schermo solamente la linea 20, LIST 10-50 listera' tutte le linee dalla 10 alla 50 incluse, LIST-50 listera' tutte le linee i cui numeri sono compresi fra il primo presente in memoria e la linea 50 inclusa, l'istruzione LIST 50- listera' il programma a partire dalla linea 50 fino alla fine. Tutte le combinazioni delle istruzioni appena citate possono essere usate per trovare e correggere qualsiasi segmento del programma che si trovi in memoria.

Il BASIC e' un linguaggio interpretativo ed esegue un comando trovando l'ultima linea stampata sulla tastiera e analizzandola da sinistra a destra alla ricerca di parole chiave e di espressioni riconoscibili. Ogni volta che esso incontra una parola chiave come ad esempio PRINT, esso interpreta tale parola in un comando che significa qualche cosa per il BASIC. Le parole di comando sono caricate in memoria con il bit VIII alto in modo da informare il BASIC che tale dato e' una parola di comando o una parola chiave. Quando una linea di programma viene caricata nella memoria RAM con l'uso del ritorno carrello, il BASIC prende i numeri di linea e ricerca nella memoria finche' non trova lo stesso numero o il numero immediatamente superiore. Se il BASIC trova lo stesso numero di linea allora l'intera linea in memoria viene cancellata e al suo posto viene inserita una nuova linea. Nello stato di pre-interpretazione tutte le parole chiave sono rimpiazzate con un singolo carattere che contraddistingue la parola chiave stessa. Questo permette all'interpretatore di memorizzare i comandi nella forma piu' efficiente per quanto risuarda la memoria. Solamente i dati stampati dal programmatore, come caratteri alfanumerici, puntatori alle variabili e parole chiave vengono memorizzati. Ad esempio: PRINT, anziche' essere memorizzato come una sequenza di 5 caratteri, occupa in memoria solamente 1 carattere.

Il BASIC viene chiamato interpretatore poiche' l'esecuzione dell'istruzione viene fatta analizzando la parola chiave che permette l'esecuzione di quella linea di programma, poiche' eseguendo tale parola chiave sotto il controllo di una serie di subroutines. Questo modo di operare da luoso ad un sistema di memorizzazione dei programmi molto efficiente, ma ha dei tempi di esecuzione piu' lunghi di quelli che si avrebbero con un programma in linguaggio macchina. Poiche' il BASIC del PET usa in memoria dei contrassegni e memorizza questi contrassegni sui dispositivi di I/O ogni volta che il programma viene caricato o registrato, la codificazione dei dati sul nastro o in memoria non e' trasferibile ad altra macchina. Non e' generalmente possibile usare le istruzioni BASIC registrate su altre macchine.

Quando viene creato un programma BASIC si opera sotto due livelli di editor: il programma di correzione dello schermo e l'editor delle linee BASIC.Il programma di correzione dello schermo permette di cambiare caratteri su una linea finche'un ritorno carrello non la trasferisce alla memoria principale. IL programma di correzione delle linee BASIC permette di aggiungere nuove linee e modificare o cancellare vecchie linee. Per cancellare una linea, e' necessario battere sulla tastiera il numero di linea immediatamente seguito da un ritorno carrello.per modificare una linea,e' necessario da prima listarla sullo schermo, quindi modificarla ed infine battere un ritorno carrelloper rimemorizzarla.Per impiazzare una linea, e' necessario battere sulla tastiera lo stesso numero di linea con il nuovo testo e fare seguire tutto cio' con un ritorno carrello.
Vi sono due modi per eseguire un programma BASIC. IL pimo di questo e' battere sulla tastiera RUN. Il comando RUN dapprima azzera tutte le variabili del programma e inizializza i puntatori di programma. POi esegue istruzione del programma in ordine.Partendo con il numero di linea piu' basso.L'esecuzione continua finche' non vi sono altre istruzioni e viene incontrata l'istruzione END, oppure fino a quando non viene premuto il tasto di stop.L'istruzione RUN puo' avere anche un argomento, cioe' il numero della prima istruzione da eseguire.Ad esempio:se si batte sulla tastiera RUN 30, il programma iniziera' l'esecuzione dell'istruzione 30.L'istruzione RUN viene eseguita in modo diretto.Una istruzione di GOTO, anche eseguita in modo diretto, opera nello stesso modo del RUN eccetto che per il fatto che nessuna delle variabili viene rinizializzata.Il GOTO, naturalmente, deve spscificare il numero di linea della prima istruzione da eseguire, ad esempio: GOTO 30.

I LETTERALI

In parecchi esempi che sono stati fatti fino a questo momento, l'istruzione PRINT e' stata usata con i caratteri da stampare chiusi fra virgolette. Nel PET queste vengono chiamate strinshe letterali.Dati che possono essere memorizzati dal PET sono anche i numeri binari in virgola mobile.Tuttavia parecchi dati vengono utilizzati nei programmi non sono dati numerici che permettono un colloquio fra macchina ed operatore ma alfa-numerici.
Questi caratteri vengono specificati al PET con strinshe letterali piu' specificamente un letterale e' qualsiasi valore contenuto all'interno di una copia di virgolette. Per permettere la massima flessibilita' nell'uso dello schermo, il PET ha un insieme speciale di caratteri grafici e la possibilita' di memorizzare ed eseguire dei caratteri di controllo del cursore che vengono forniti ad esso per mezzo di letterali o con tecnica piu' soffisticata.

Si e' gia' visto nella sezione che trattava la tastiera del PET come esso memorizzi i dati in ASCII. I caratteri grafici sono memorizzati come un'estensione di questo insieme. I caratteri grafici vengono prodotti premendo il tasto della maiuscola e riutilizzando l'orisinale insieme di 64 caratteri, essi vengono accumulati in memoria con indicatore speciale per differenziarli dai caratteri minuscoli che compaiono sullo stesso tasto.

Qualsiasi combinazione dei caratteri presenti sulla tastiera del PET puo' essere inclusa in una letterale e compresi tutti i movimenti del cursore e il campo inverso.

Lo stesso programma di correzione dello schermo considera quando viene battuto sulla tastiera un letterale non appena una virgoletta viene stampata.Tutti i caratteri racchiusi tra virgolette vengono trasferiti direttamente allo schermo in forma tale che il software che trasferisce la linea d'ingresso al BASIC trasferira' il carattere di controllo quando necessario.I caratteri di controllo del movimento del cursore appariranno in modo speciale se usati in un letterale in campo inverso.Lo stesso carattere di campo inverso somigliera' ad una "R".Il capo pagina come una "S" la cancellazione come una "S" maiuscola o il cuore.Lo spostamento verso il basso del cursore e' una "Q",lo spostamento verso l'alto e' una "Q" maiuscola o un carattere vuoto.Lo spostamento del cursore a destra e' la parentesi destra e lo spostamento a sinistra e' la maiuscola di tale carattere e somiglia ad una linea verticale attraverso la quinta colonna dei punti che formano il carattere stesso. Il carattere d'inserzione e' un "T" maiuscolo che somiglia ad una seconda linea verticale. Puo' quindi in un letterale usare dei caratteri in campo inverso,ma si puo' passare in campo inverso con il suo carattere di controllo prima che qualsiasi altro carattere venga stampato. I soli caratteri che si possono usare in campo inverso tra virgolette sono quelli che saranno interpretati come caratteri di controllo. Il carattere di cancellazione e' il solo carattere di correzione che puo' logorare l'interno di un letterale.Quando su una linea e' stato stampato un numero dispari di virgolette non e' piu' possibile muovere il cursore sullo schermo finche' non si incontrano le virgolette di chiusura o viene stampato un ritorno carrello. Non e' quindi possibile coreggere qualche errore commesso o tentare di muovere il cursore dopo che una coppia di virgolette sono state stampate.Per poter eseguire la correzione e' necessario chiudere le virgolette o dare un ritorno carrello,quindi muovere il cursore ed eseguire la cancellazione.Un altro metodo per inserire il carattere di controllo del cursore in un testo esistente e' l'uso della funzione insert. Essa ha lo stesso effetto di una copia di virgolette aperte. Ad esempio,premendo tre volte il carattere di inserzione e tentando di eseguire poi un movimento del cursore il carattere di controllo apparira' in campo inverso come nel caso menzionato precedentemente.Questo modo di operare e' molto comodo in quanto permette di inserire tanti nuovi caratteri quanto e' il numero delle volte che il tasto di inserzione e' stato premuto. Cio' suggerisce di impratichirsi su tal modo di operare correggendo gli errori che potreste aver fatto via via battendo un vostro programma.La conoscenza di possibilita' di manipolazione dei caratteri grafici e di quelli del movimento cursore vi permettono qualsiasi possibilita' di eseguire grafici, purche' voi abbiate la pazienza di strutturare il programma. Vi si raccomanda di sviluppare la vostra abilita' stendendo dei programmi e sperimentando a fondo l'uso dei caratteri grafici e di movimento del cursore. Si ricordi che non e' possibile danneggiare la macchina. La cosa peggiore che vi puo' capitare e' di cancellare accidentalmente lo schermo dopo aver stampato su di esso una certa quantita' di cose utili.

IL CAMPO INVERSO

E' stato gia' visto negli esempi di funzionamento tra virgolette e del carattere di inserzione che non appena un modo di funzionamento e' stato stabilito per una determinata linea

esso sara' mantenuto dal PET finche' il modo stesso non sara'cancellato da un nuovo carattere di controllo o da un ritorno carrello.Il campo inverso funziona nello stesso modo. Esso rimane in effetto finche' una cancellazione del campo inverso non viene stampata oppure viene inviato un carattere di ritorno carrello.
Come e' stato precedentemente descritto nel paragrafo relativo alla memoria di schermo,il carattere di campo inverso viene memorizzato con un bit speciale per indicare che nella matrice dei punti che realizzano il carattere di schermo,i punti neri vengono sostituiti con punti bianchi e viceversa. Si puo' immediatamente vedere stampando un carattere in campo inverso che l'effetto ottenuto e' quello di un altro rilievo del carattere stesso sullo schermo; in effetti questa possibilita' permette di radoppiare il numero dei potenziali caratteri che il PET puo' far apparire sullo schermo. La possibilita' di campo inverso e' talmente comoda che non e' solamente implementata sul display del PET, ma in parecchi delle stampanti che possono venir adoperate con il PET.
Si veda ora un esempio di come il campo inverso lavori.Si cancelli lo schermo e si stampi su di esso HI seguito da uno spazio.Il successivo carattere sia il carattere di un campo inverso ed immediatamente dopo venga stampato THERE, infine si dia la canellazione del campo inverso, premendo il tasto del campo inverso assieme al tasto delle maiuscole, si stampi uno spazioe la scritta PET; si ottiene in tal modo una linea in cui la parola THERE assume una particolare evidenza.
Il campo inverso si attiva non appena il relativo carattere di controllo viene stampato e tutti i successivi caratteri che verranno inviati allo schermo saranno visualizzati in campo inverso finche' tale modo di funzionamento non verra' fatto cessare come precedentemente detto.Quanto esposto si applica sia a ingressi diretti da tastiera come a caratteri che vengono stampati da una stringa letterale.Si prenda ora in esame un esempio in cui il carattere di controllo per il campo inverso sia inserito in uma stringa letterale. Si stampi cioe',attraverso la tastiera, la seguente frase: ?"HI a questo punto si prema il tasto di campo inverso e si faccia seguire a tale carattere la scritta THERE;si cancelli ora il comando di campo inverso premendo contemporaneamente il tasto di campo inverso e delle maiuscole e si stampi infine PET". Si noti che il carattere di attivazione e disattivazione del campo inverso occupano uno spazio sullo schermo quando si programma e che essi appaiono nella rappresentazione inversa, ma che la scritta THERE non appare invece in campo inverso. L'effetto delle virgolette e' di posporre l'azione di un carattere di controllo al momento in cui la stringa letterale viene interpretata. Poiche' il campo inverso e' attivato e posiziona un bit di ciascun carattere sulla memoria di schermo,non e' richiesta nessuna posizione nella memoria di schermo per il carattere di campo inverso o per la sua disattivazione. Quando il flusso di caratteri viene ricevuto dal programma li stampera' sullo schermo stesso.Il campo inverso rimane attivato finche' un comando di disattivazione o un ritorno carrello non verra' premuto sulla tastiera.

TERMIMI ED OPERATORI

Le comunicazioni con il BASIC avvengono sia attraverso numeri sia attraverso stringhe letterali alfanumeriche. I numeri sono sempre presentati sullo schermo in forma decimale sebbene il microprocessore che opera nel PET lavori in modo binario.

E' possibile pero' trasferire da e per il calcolatore anche numeri binari. Quando un numero sara' assegnato senza nessuna specifica indicazione esso sara' considerato un numero rappresentato in forma decimale,quando viceversa, esso sara' preceduto dal simbolo di $ ad esempio: $0010 esso sara' considerato un numero esadecimale e in particolare per l'esempio fatto esso sara' uguale al numero decimale 16. Quando il BASIC riceve una intera linea interpreta i caratteri ricevuti e li divide in diverse classi;in particolare li dividera' in comandi,funzioni ed operazioni. La fase PRINT e' un comando al BASIC con una speciale funzione che il PET puo' eseguire.
Una funzione puo' essere ad esempio una radice quadrata o una variabile o una funzione speciale.Non appena si premera' sulla tastiera il tasto di "P GRECA" sara' disponibile ed usabile in una qualsiasi espressione la costante 3,14159265.
Un operatore e' un carattere che viene interpretato dal BASIC come una funzione aritmetica che deve essere eseguita nel calcolatore una espressione. Il BASIC permette di usare i seguenti insieme di operatori: il segno + fa si che due numeri rappresentati in virgola mobile siano addizionati e il risultato viene calcolato in un accumulatore in virgola mobile. La precisione e' limitata a 9 cifre significative. Il segno - sottrae il valore che si trova alla destra del - dal valore che si trova alla sinistra. L'asterisco * e' il simbolo usato in BASIC per la moltiplicazione.Il valore alla destra del segno di moltiplicazione e' moltiplicato per il valore che si trova a sinistra.La barra / e' il segno che viene usato nel BASIC per la divisione. Tutti i numeri alla sinistra della barra vengono divisi per il valore dell'espressione che si trova alla destra della barra stressa. La freccia rivolta versol'alto) e' il simbolo usato per l'elevamento potenza. Alla sinistra di tale segno si trova la base della potenza alla destra l'esponente. Le parentesi () chiuse ed aperte fanno si che i valori al loro interno siano una singola espressione. Ci devono essere tante parentesi aperte quante parentesi chiuse e il numero di copie viene valutato a partire dalla copia interna. Si ricordi che nel BASIC vi e' un ordine di precedenza nell'esecuzione delle operazioni. L'ordine gerarchico degli operatori e' il seguente: la moltiplicazione viene fatta per prima, seguita dalla divisione e dall'addizione e dalla sottrazione. Le espressioni all'interno delle parentesi calcolate per prime partendo dalla prima coppia di parentesi, cioe' da quella piu' interna.Il seguente insieme di esempi che possono essere usati sul vostro PET chiarira' l'uso degli operatori e il loro ordine di precedenza:

```
Addizione
?2+2
sottrazione
?4-2
Moltiplicazione
?6*2
Divisione
?12/2
Uso delle parentesi
?4+8/2
?(4+8)/2
Ordine degli operatori
?(2+4*(8-4)/2)*3
```

LE FUNZIONI

Vi sono tre funzioni disponibili in BASIC che sono caratteristici del PET. La prima di esse e' il P GRECO;ogni volta che questo carattere viene usato in espressione, il BASIC lo traduce nel valore di 3.14159265, esso puo' essere usato in qualsiasi momento in ogni espressione e sara' sempre considerato pari a tale numero.
La funzione TI$ e il valore TI sono due modi di comunicare con l'orologio in tempo reale che si trova all'interno del BASIC.Come e' stato detto precedentemente ogni volta che si ha un rinfresco dello schermo, cioe' ogni 1/60 di secondo, viene aggiornato un valore all'interno del PET. Questo valore e' utilizzato come un orologio in tempo reale a 24 ore. Tale valore e' disponibile al programmatore nella sua forma binaria attraverso l'espressione TI,che permette di caricare in memoria il valore corrente del tempo.Questo numero viene rappresentato su tre bytes binari e viene considerato come il numero di sessantesimo di secondo passati dall'avviamento dell'orologio.Con il suo uso si puo' valutare il tempo necessario a compiere una particolare operazione, e' sufficente in tal caso memorizzare il valore di TI nell'istante in cui la sequenza dei calcoli ha inizio e calcolare poi la differenza con il valore che TI assume alla fine di tale sequenza. La funzione TI$ presenta ed accetta dati in forma di ore,minuti e secondi. Quando l'espressione TI$ viene usata, essa presenta i dati in una stringa con due caratteri per le ore, due caratteri per i minuti e due caratteri per i secondi. Il valore del tempo viene cioe' presentato come in un orologio a 24 ore. Se ad esempio l'ora da rappresentare e' 10 minuti dopo l'una pomeridiana, la stringa si presentera' come 131000. Per posizionare il valore del tempo reale sara' sufficente battere sulla tastiera l'espressione TI$= seguita dal numero che deve essere impostato racchiuso fra virgolette. Ad esempio: preposizionare l'orologio alle ore 2:45 e 30 secondi pomeridiano si battera' sulla tastiera TI$="144530". Come per tutte le altre variabili, l'accensione dell'apparecchio, azzera l'orologio in tempo reale. E' necessario porre una certa attenzione nell'usare il valore TI. E' bene far notare che il valore di TI automaticamente viene azzerato, quando si passa alla mezzanotte.Le funzioni sono delle routine programmate dal BASIC che possono essere trattate con un singolo valore. Esse possono andare dal segno di P GRECO che e' una funzione predefinita al seno che e' la capacita' del BASIC di calcolare il seno di un numero. Quando il BASIC incontra il codice per una funzione calcola l'espressione per la funzione, calcola cioe' il valore risultante e usa tale valore nel comando.L'uso della funzione e' in verita' molto semplice. Se A uguale al seno di P GRECO radiante, l'espressione che dovra' essere scritta e':

```
A=SIN(P GRECO)
```

in questa istruzione vengono usate due funzioni: P GRECO e SENO; il BASIC valutera' tale espressione associando il valore 3.14159265 a P GRECO e calcolera' poi il valore della funzione seno e infine memorizzera' il risultato nella variabile A. Nell'espressione:

```
A=2*SIN(P GRECO)
```

dopo aver calcolato il seno, il BASIC moltiplichera'il risultato per 2 e lo memorizzera' in A.
Le funzioni trigonometriche seno, coseno,tangente e arco tangente sono tutte disponibili nel BASIC del PET.

L'argomento delle funzioni SENO,COSENO,TANGENTE dovra' essere espresso in radianti. Per convertire da gradi in radianti si dovra' moltiplicare il numero che esprime i gradi per P GRECO/180. Ad esempio: ?SIN(90*P GRECO/180) calcolera' il seno di 90 gradi. Ciascuna di queste funzioni viene calcolato per mezzo di tabelle.Poiche' P GRECO e' limitata a 9 cifre significative, in generale, i valori da assegnare agli argomenti delle funzioni trigonometriche dovranno essere minori di 1000 gradi o di 6 P GRECO. La precisione delle funzioni BASIC e' di 5 parti su 10 almeno finche' gli argomenti sono al di sotto di 20 radianti. L'espressioni che usano il valore in radianti sono funzioni del valore P GRECO che e' preciso solamente di una parte su 10 alla IX. L'arco tangente e' la sola funzione trigonometrica inversa che e' specificata come funzione nel BASIC. La funzione arco-tangente calcola il valore in radianti dell'espressione data come argomento. La risposta viene sempre data tra piu' o meno 17. La precisione e' di 5 parti su 10 a 10.Nell'uso normale il risultato e' assegnato in radianti. Per convertire un numero da radianti a gradi sara' necessario moltiplicare i radianti per 180 diviso P GRECO

```
?180/P GRECO*ATN(.5)
```

Per calcolare il valore dell'arco-seno e dell'arco coseno come funzione dell'arco tangente si puo' usare la seguente espressione generale:

```
ARC SIN (X)=ATN(X/SQR(-X*X+1)
ARC COS (X)=-ATN(X/SQR(-X*X+1)+1.5708
```

Ambedue l'espressioni forniscono risultati in radianti che puo' essere convertito in gradi moltiplicando l'espressione per 180/P GRECO. Si fa notare che in ambedue l'espressioni vi e' la possibilita' di eseguire una divisione per zero, cio' dara' luogo ad un errore nel BASIC.Prima di usare l'espressione e' necessario testare che tale divisione per zero non possa aver luogo.In particolare sara' necessario accertare che X non sia uguale ad 1.

FUNZIONI MATEMATICHE

Il numero piu' grande che il BASIC puo' manipolare e': -1.70141183E+38 o +1.70141183E+38. Qualsiasi tentativo di usare un numero piu' grande dara' luogo ad un errore di ?overflow. Il numero piu' piccolo che si puo' usare e che puo' essere distinto dallo zero e' 2.93873588E-39.Qualsiasi tentativo di usare un numero piu' piccolo dara' luogo ad un errore di underflow.Le funzioni matematiche del BASIC sono le seguenti:

ABS

La funzione ABS permette di calcolare il valore assoluto di un numero attraverso l'espressione ABS(X).La funzione calcolera' il valore dell'espressione come un numero positivo. Non vi e' nessuna perdita di precisione. Ad esempio:

```
ABS(-145)
```

dara' luogo al risultato +145.

INT

La funzione INT arrotonda per difetto un qualsiasi numero all'intero piu' vicino. Ad esempio:

```
INT(.23)=0
```

dara' un risultato pari a zero

```
INT(-2.5)=-3
INT(1.79)=1
```

La funzione INT non introduce nessuna perdita di precisione al di la del troncamento della parte decimale. Tuttavia piccole imprecisioni nell'argomento possono dar luogo a dei seri problemi.

Ad esempio il numero 4 puo' essere memorizzato dal BASIC come 3.99999999.Quando il numero 4 cosi' viene usato quale argomento per la funzione INT il risultato anziche' essere 4 sara' 3.

SGN

La funzione SGN restituisce come risultato un 1 se il segno del suo argomento maggiore di zero, uno zero se l'argomento zero e un meno 1 se il segno dell'argomento e' negativo ad esempio

```
SGN(-45)
```

dara' come risultato -1

```
?SGN(+10)
```

dara' come risultato 1

la funzione SQR calcola la radice quadrata di qualsiasi numero maggiore di zero.Se come argomento viene usato un numero minore di zero,il risultato e' la stampa del messaggio d'errore ILLEGAL QUANTITY ERROR.La precisione e' di 5 parti su 10 alla 10 sull' intero campo.

le due seguenti funzioni lavorano sulla base naturale di nepero e che vale 2.71828183.

EXP

il parametro di tale funzione definisce la potenza alla quale la base viene innalzata.

I limiti di tale parametro sono 88.02969189.quando per argomento si usa un numero che in valore assoluto sia maggiore dei limiti citati si produrra' un overflow.La forma in cui viene usata la funzionee'EXP(x).

LOG

la funzione LOG calcola il logaritmo natuale in base e del suo argomento.ad esempio:

```
LOG(32)
```

calcolera' il logaritmo naturale di 32.Per calcolare il logaritmo in base 10, e' sufficiente dividere il logaritmo in base e per il logaritmo in base e del numero 10; cioe' l logaritmo deimale di x e' usuale al logaritmo naturale di x fratto il logaritmo naturale di 10.

RANDOM

Le funzioni RANDOM vengono usate normalmente in programmi statistici e in programmi di gioco. Sono previste tre funzioni BASIC RANDOM. Il generatore di numeri RANDOM, cioe' di numeri a caso,usa un un algoritmo che produce un valore che si trova fra 0 ed 1.l'argomento puo' essere sia positivo, che zero o negativo. Un argomento positivo da luogo alvalore successivo della sequenza di numeri a caso generata da un algoitmo numerico nel BASIC.Tale sequenza partira' sempre dallo stesso valore che viene posizionato al lato della accensione dell'apparecchiatura.Tuttavia il valore iniziale puo' essere posizionato usando come argomento lo zero si ottiene un numero veramente casuale.IL programma BASIC legge 4 intervalli di tempo non correlati in relazione al presentarsi di eventi in tempo reale.Si ottiene un numero casuale zero se RND(0) viene connesso con alcuni eventi esterni, come ad esempio: l'inizzializzazione di un programma o la pressione di un tasto in risposta ad una domanda che si presenti sullo schermo.Un eccessivo ripetitivo in un programma alla funzione RANDOM non e' effettivamente casuale,poiche' le relazioni di tempo sono predicibili all'atto dell'avviamento del programma stesso. In una sequenza fissa di programma l'unico vero numero casuale e' il primo. Una soluzone a questo problema e' usare la RND di zero per generare un inizio sequenza casuale.La funzione RANDOM di un numero negativo non e' sempre un numero casuale.Il parametro viene considerato come inizio sequenza nella seneazione dei numeri a caso.

Questa tecnica puo' essere usata in programmi di debugging nel senso che una sequenza predicibile ripetitiva puo' essere ottenuta con la funzione RDN con il segno meno, durante lo sviluppo del programma. Un' altra tecnica per ottenere un numero casuale e' quella di usare la funzione RANDOM che abbia come argomento il tempo corrente.Sebbene che per gli scopi di gioco tale tecnica no sia tanto efficace come l'uso della funzione RANDOM con argomento zero. Si supponga di voler simulare in un programma di gioco, un dado. Inizialmente e' necessario posizionare il generatore di numeri a caso al suo valore iniziale con l'istruzione:

```
D=RND(-TI)
```

Successivamente, si puo' calcolare il valore del dado con l'istruzione:

```
D=INT(6*RND(1)+1)
```

PEEK, POKE

PEEK e' la funzione che permette all'utente di ispezionare qualsiasi locazione di memoria del PET. Il parametro della funzione PEEK e' l'indirizzo di memoria espresso in decimale e il risultato e' un numero decimale compreso fra 0 e 255. Il BASIC del PET e' realizzato in modo tale che qualsiasi indirizzo maggiore dell'indirizzo esadecimale C000 e' automaticamente azzerato. Questo modo di procedere, e' un vincolo legale imposto dalla compagnia che ha scritto il BASIC per proteggere il proprio software.

A titolo di esempio si voglia vedere il contenuto della locazione in memoria 25, sara' necessario battere sulla tastiera:

```
?PEEK(25)
```

POKE non e' una funzione bensi un comando. Esso permette all'utente di caricare un numero in una memoria di I/O o di lettura-scrittura. I parametri sono specificati in una lista che segue il comando. Il primo parametro e' l'indirizzo di memoria nel quale si vuol caricare l'informazione. Esso puo' essere compreso fra 0 e 65536. Il secondo parametro e' il valore che si vuol memorizzare. Esso deve essere compreso fra 0 e 255. Ad esempio se si volesse caricare il carattere A nella prima locazione della memoria di schermo sarebbe necessario scrivere:

```
POKE 32768,1
```

Alcune locazioni in memoria non possono essere cambiate (ROM) in particolare non possono essere cambiate le locazioni della memoria di sola lettura. Altre locazioni non devono essere cambiate, in particolare sara' necessario lasciare inalterate le locazioni di memoria delle variabili di sistema di I/O e del BASIC. Se si andranno a cambiare tali locazioni di memoria sara' poi necessario ressettare la macchina.

USR

La funzione USR serve a consegnare un parametro a un programma usando l'indirizzo di salto collocato nella memoria nelle posizioni 1 e 2. Si rimanda alla sezione relativa al linguaggio macchina per una dettagliata descrizione dell'uso di questa funzione.

FRE

La funzione FRE informa su come alcuni byte sono posizionati in memoria. Malgrado essa sia una vera funzione in quanto puo' essere usata in una espressione, essa viene normalmente usata nel modo diretto nella forma:

```
?FRE(0)
```

La funzione FRE da luogo ad una azione BASIC che viene chiamata letteralmente raccolta delle immondizie.Tale azione consiste nel riunire tutti i byite non usati in blocco di grosse dimensioni che puo' essere efficentemente allocato e usato, e infine alcune funzioni per aiutare nella formazione dei dati da stampare sullo schermo o su una stampante.

TAB

Questa funzione di formatazione posiziona il cursore alla colonna specificata nell'argomento della funzione stessa.L'argomento viene dapprima trattato dalla funzione INT. Il campo legale per l'argomento sta 0 e 255.Se il cursore si trova posizionato gia' al di la della locazione specificata, la funzione di tabulazione viene ignorata.Si noti che la funzione TAB usa dei caratteri di salto,e non degli spazi.

POS

Questa funzione restituisce come valore la posizione del cursore.La posizione e' rimessa a zero ogni volta che viene usato un ritornocarrello. Si noti che le posizioni di "a capo pagina","cancellazione" non alterano il valore POS sebbene il cursore venga posizionato nella prima colonna.

SPC

Questa funzione di formatazione stampa il numero di salti specificati nell'argomento, argomento che viene trattato dalla funzione INT.Il campo di valori legali per la funzione SPC si trova situato fra 0 e 255.Si noti che la funzione SPC con argomento zero da luogo a 256 salti.

CAPITOLO 5.

PROGRAMMAZINE ELEMENTARE

Uso di decisione nella stesura di un programma.Una delle massiori comodita' della programmazione BASIC e' la possibilita' di ritornare indietro e riesesuire alcune linee di programma.Cio' puo' essere fatto in due modi, senza condizioni con una istruzione di GOTO condizionamento con IF THEN.GOTO e' scritto per specificare un numero di linea obbiettivo alla quale l'esecuzione deve ritornare.GOTO puo'essere scritto anche con uno spazio tra GO e TO.Il BASIC del PET riconoscera'ambedue le forme di scrittura.
Ad esempio si potranno avere le seguenti forme di scrittura:

```
GO TO 50
GOTO 100
```

Istruzione IF THEN a tre forme:

IF(seguito dalla condizione)THEN(seguito da una istruzione)

IF(seguito dalla condizione)
GOTO(a un numero di linea)

IF(seguito dalla condizione)THEN(numero di linea)

Le condizioni vengono scritte come due espressioni aritmetiche separate da un operatore relazionale.Il BASIC del PET ha a disposizione sei operatori relazionali<,=,>=,<>,<=,>.

Fino a questo momento sono stati descritti dei programmi che esesuono delle funzioni singole in ordine seriale. Potrebbe a questo punto essere familiare il concetto che ad esempio viene esesuita per prima la linea 10 successivamente la linea 20 e cosi' via per le altre linee numerate in ordine crescente.

Se si volesse ad esempio calcolare e stampare i primi numeri fra 1 e 20, il loro quadrato e la loro radice quadrata, si potrebbe scrivere un programma nella sua forma lineare come segue:

```
10 PRINT 1,1,1
20 PRINT 2,2*2,SQR(2)
30 PRINT 3,3*3,SQR(3)
```

Il massior svantassio di questo modo di operare sta nel fatto che e' necessario scrivere 20 linee di programma sino ad arrivare alla linea

```
200 PRINT 20,20*20,SQR(20)
```

ANELLI INCONDIZIONATI

Facendo ricorso al concetto di variabile e con l'assiunta del concetto di anello, si puo' scrivere un programma che calcoli e stampi i valori fra 1 e 20, i loro quadrati e le loro radici quadrate senza dover scrivere tuttavia un lunso programma.

Il programma apparira' come segue:

```
10 PRINT"VALORE","QUADRATO","RADICE QUADRATA"
20 I=I+1
30 PRINT I,I*I,SQR(I)
40 GOTO 20
```

La linea 10 stampa una intesatzione per le colonne di numeri. Essa viene esesuita una volta sola.

La linea 20 il successivo numero da usare. La prima volta che questa linea viene esesuita I non e' mai stato usato e quindi ha il suo valore iniziale pari a zero.

La linea 30 stampa il valore assunto da I dal suo quadrato e dalla sua radice quadrata e corrisponde alle linee da 10 a 200 del programma precedente, con l'unica differenza che le costanti sono state rimpiazzate da variabili.
La linea 40 contiene un comando di GOTO che fa ritornare l'esecuzione indietro e la fa ripartire dalla linea 20. Il BASIC memorizza ciascuna linea di testa del programma facendo precedere ad essa un puntatore alla linea successiva.Usando questa tecnica l'interprete puo' esaminare rapidamente solamente il numero della linea,determinare se questa linea esista e trasferire l'esecuzione alla linea stessa. L'istruzione di GOTO non puo' solamente trasferire il controllo a una linea con un numero inferiore a quella attuale, ma puo' anche saltare a linee con numeri piu' elevati.Si vedra' in esempi futuri che il GOTO puo' permettere il salto di una porzione di programma.Mettendo in esecuzione il programma dell'ultimo esempio si vedra' che esso continuera' a stampare i valori di I finche' non si premera' il tasto di stop. Il rapido cambio di riga sulla memoria di schermo rendera' particolarmente impossibile leggere tali valori,ma l'uso del tasto di reverse riuscira' a rallentare tale operazione. Mantenendo premuto il tasto di reverse il cambio riga avverra' piu' lentamente per un fattore di 20. Per fermare l'esecuzione e' necessario premere il tasto di STOP. Se si vuole ripartire con il programma si puo' battere sulla tastiera CONT in modo che il programma incominci dal punto in cui e' stato interrotto oppure RUN con che il programma iniziera' dal principio. Per quanto riguarda l'esempio fatto si puo' osservare che,malgrado tale programma usi GOTO, esso non ha realmente risolto il problema che ci si era posti; cioe' di stampare esattamente i primi 20 numeri sullo schermo. Tuttavia, prima di considerare questo aspetto,si introduca un piccolo errore nel programma in modo da esaminare uno dei piu' comuni inconvenienti con uso poco corretto dell'istruzione GOTO. Si stampi l'istruzione 40 nel modo seguente:

```
        40 GOTO 10
```

e immediatamente si passi poi all'esecuzione, si vedra' che il programma stampera' alternativamente l'intestazione e i valori calcolati anziche' stampare un'unica intestazione nella parte alta dei valori calcolati. Il salto a una linea di programma errata e' il piu' comune errore che si puo' fare durante la programmazione. Nel caso in esame tale errore e' facilmente visibile e correggibile. Fermando il programma si puo' usare il programma di correzione dello schermo per correggerre la linea 40 scrivendo: 40 GO TO 20.

ANELLI CONDIZIONATI

L'istruzione IF THEN permette di eseguire l'istruzione che segue immediatamente il THEN solamente se una specifica condizione si e' verificata.La condizione puo' essere fissata mettendo uno dei sei operatri relazionali tra due espressioni. Gli operatori relazionali sono:

```
=  usuale
<> non usuale
>  maggiore di
<  minore di
>= maggiore usuale di
<= minore usuale di
IF A<B  THEN PRINT   "A LESS THAN B"
```

Se l'espressione e' vera, l'istruzione che si trova sulla stessa linea dell'istruzione IF viene eseguita.

Se l'espressione e' falsa, il programma salta alla prossima linea numerata. Per ricordare quali siano i segni di minore e maggiore si tenga presente che la punta della freccia sara' rivolta verso il valore che si vuole sia minore dell'altro. Nell'esempio precedente la linea 40 puo' essere sostituita dalle seguenti:

```
40 IF I<=20 THEN GOTO 20
```

L'istruzione IF THEN permette di prendere delle decisioni durante l'esecuzione del programma. Cio' permette di limitare il programma e di interpretare delle azioni non appena un certo avvenimento si verifichi.Nel caso dell'esempio il programma viene eseguito per I che va da 1 a 20 e finalmente passa oltre l'istruzione 40 non appena I diventa maggiore di 20.L'istruzione IF puo' essere anche utilizzata per saltare oltre una istruzione di GOTO incondizionato. Nell'esempio che stiamo esaminando si possono aggiungere due nuove linee e modificare la linea 40 nel modo seguente:

```
35 IF I=20 GOTO 50
        40 GOTO 20
        50 END
```

Il programma passerra' in esecuzione facendo variare I da 1 a 20 e non appena I sara' diventato =20 saltera' all'istruzione END. Parecchi interpretatori BASIC richiedono che sia inclusa un'istruzione di END come ultima istruzione del programma per terminarlo.L'END puo' essere usato opzionalmente nel BASIC del PET per forzare la fine nell'esecuzione del programma a un punto specifico.Le istruzioni IF THEN hanno tre forme. La prima e': IF espressione,GOTO numero di linea. La seconda e': IF espressione,THEN numero di linea, sottointendendo GOTO. La terza forma infine e': IF espressione, THEN istruzione. In questa ultima forma l'istruzione specificata dopo il THEN verra' eseguita se l'espressione che segue l'IF risultera' vera. In caso contrario si passera' immediatamente all'istruzione successiva. Un esempio di utilizzo di questa ultima forma dell'IF si puo' ottenere inserendo nel programma d'esempio che abbiamo fin qui considerato la seguente linea:

```
32 IF I=10 THEN PRINT"------"
```

Tale istruzione, in fase di esecuzione, fara' si che venga stampata una linea tra la decima e l'undicesima riga dei valori calcolati. E' bene ribadire che possibilita' operative dell'istruzione IF o IF THEN sono solemante due: o viene eseguita la linea successiva nel caso in cui la condizione non sia vera oppure nel caso in cui la condizione e' vera viene eseguita l'istruzione che segue il THEN.E' bene porre molta attenzione quando si usino piu' istruzioni dopo il THEN sulla stessa linea. Ad esempio nell'istruzione:

```
IF X=5 THEN 50:Z=A
```

la seconda istruzione Z=A non sara' mai eseguita poiche' la linea che verra' eseguita dopo IF, nel caso in cui la condizione sia vera, sara' la linea 50. Tuttavia in una istruzione del tipo:

```
IF X=5 THEN PRINT X:Z=A
```

Quando X sara' uguale a 5 sara' stampato X e Z sara' posto=A. In conclusione l'istruzione IF THEN permette di prendere una grande quantita' di decisioni durante l'esecuzione del programma.Cio' permette di limitare il programma e di intraprendere delle azioni non appena si verificano determinati casi nei vari punti del programma.L'istruzione IF THEN e' l'unione del concetto di salto incondizionato con il concetto di analisi di situazioni e permettono di usare il computer sia come elemento di controllo che come elemento di calcolo.

La combinazione intelligente di decisioni logiche con operazioni ripetitive e' quella che permette ai programmi di essere effettivamente efficienti.

INGRESSO DEI DATI

Un programma per poter funzionare deve poter accedere ai dati da elaborare. Ad esempio alcuni programmi potrebbero richiedere semplicemente dei dati molto semplici quali dei SI o dei NO in risposta a delle domande poste al programma stesso. Altri programmi ,ad esempio un programma di calcolo della paga potrebbe viceversa richiedere la retribuzione oraria, le ore lavorate, informazioni sulla tassazione. Nel BASIC del PET vi sono due modi per caricare questi valori nelle variabili.

ISTRUZIONI READ E DATA

Fino a quando non sono stati introdotti sistemi timeshare il BASIC poteva accettare dati solamente attraverso schede perforate.
All'interno del programma si trovano, quindi, in posizioni diverse parecchie istruzioni DATA. L'istruzione READ permetteva di estrarre i dati dal DATA e inserirli nelle variabili per usare poi tali valori durante l'elaborazine. Qaundo il BASIC e' diventato un linguaggio di tipo interattivo utilizzando le tecniche timeshere sono state introdotte delle istruzioni come INPUT e GET che permettono la comunicazione diretta con il programma BASIC. L'istruzione di READ e' stata ridotta ad un'istruzione che permette l'ingresso di parametri che non devono essere cambiati troppo spesso,come ad esempio: tabelle. La sintassi dell'istruzione READ e' data dal codice READ seguio da una lista della variabile in cui i dati devono essere caricati. Ad esempio:

```
READ A,B,C,D
```

L'istruzione READ utilizza i valori contenuti nelle istruzioni DATA via via che esse sono incontrate nel programma. Ad esempio l'istruzione DATA alla linea 10 e 30 dell'esempio successivo saranno utilizzate dall'istruzione READ alla linea 20. I valori contenuti in DATA sono utilizzati sequenzialmente e le virgole e la fine di ciascuna linea sono considerati elementi separatori dai dati.

```
10 DATA 2,-53,1E10
20 READ A,B
30 DATA 3.14,1,06E23
```

Gli spazi bianchi e i caratteri grafici sono automaticamente ignorati a meno che non siano racchiusi tra virgolette. Le virgolette sono considerate dei delimitatori per caratteri letterali.
Le stringhe possono essere inserite in un DATA senza virgolette,se non contengono caratteri numerici da considerare quali letterali. Ad esempio:

```
50 DATA ABC,DEF
```

Le virgole all'interno di una copia di virgolette non saranno considerate dal BASIC come separatori. Ad esempio:

```
60 DATA ",","","","
```

E' anche possibile mescolare dei valori numerici e alfanumerici nelle istruzioni DATA. Ad esempio:

```
10 DATA 123,ABC,345
20 READ A,A$,B
```

E' opportuno che il programmatore conosca le istruzioni DATA siano state inserite nella macchina oppure che usi un elemento di delimitazione alla fine dei dati. Se cio' non viene fatto i dati continuano a venire letti e il programma puo' giunsere alla fine delle istruzioni DATA.

Si genererà allora un errore quando il successivo READ verrà incontrato e sullo schermo verrà stampato il messaggio:?OUT OF DATA ERROR. Si avrà inoltre un errore di sintassi quando si tenterà di leggere un valore numerico in un campo alfanumerico.Le istruzione di READ e DATA sono implementate nel modo seguente:il primo byte del testo contiene uno zero. In realtà questo non è parte della prima linea, ma è una linea dummy che consiste solamente di un terminatore. Quando viene battuto RUN sulla tastiera,il puntatore delle istruzioni DATA viene indirizzato a questo byte. Poichè in questo byte è contenuto un terminatore il primo comando READ fa iniziare la ricerca della successiva linea DATA.Vi è poi un altro comando che il programmatore può usare per riutilizzare i dati immagazzinati nel DATA. L'istruzione RESTORE fa ripartire la riceca dati dall'inizio della memoria. Il seguente programma opererà rileggendo continuamente i dati contenuti nell'istruzione 10

```
10 DATA 10,20,30,40,50,60,70
20 I=1
30 READ A:PRINT A
40 I=I+1
50 IF I<8 THEN 30
60 RESTORE
70 GO TO 20
```

ISTRUZIONE INPUT

Come già detto,quando il BASIC è stato portato a un funzionamento interattivo, è stato introdotto il concetto di ingrsso dalla tastiera. Poichè il dispositivo di ingreso classico per il BASIC era la telescrivente, il formato dell'istruzione d'ingresso è stato limitato da questo dispositivo. Le possibilità delle operazioni d'ingresso da tastiera sono considerevoli quando vengono considerate collegate con la possibilità di utilizzare il programma di correzione dello schermo. La sintassi dell'istruzione INPUT è data dalla parola INPUT seguita da una lista di variabili. Tali variabili verranno caricate dai valori assegnati attraverso la tastiera secondo la sequenza specificata nella lista. Ad esempio:

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

verrà èseguito nel modo seguente: quando il BASIC incontrerà questa istruzione esso stamperà un punto interrogativo sullo schermo e attiverà il programma di correzione dello schermo facendo lampeggiare il cursore in attesa dei dati d'ingresso. Poichè da questo momento in poi si lavora sotto il controllo del programma di correzione dello schermo, sono disponibili i caratteri di movimento del cursore finchè non viene battuto sulla tastiera un ritorno carrello come terminatore. Dopo che questo ritorno carrello viene ricevuto dalla macchina i dati sono consegnati al BASIC un carattere alla volta.Il BASIC utilizzando i suoi buffer d'ingresso e le sue routine di traduzione interpreta poi i dati. Gli spazi bianchi in testa vengono poi soppressi, a meno che il dato d'ingresso non sia una stringa che richiede degli spazi bianchi nel qual caso è necessario racchiudere i caratteri d'ingresso fra virgolette. Il programma di correzione raccoglie solamente i caratteri che si trovano tra il punto interrogativo e la posizione corrente del cursore. I dati d'ingresso devono essere separati tra loro da virgole come nelle istruzioni DATA. Qando vengono consegnati più dati di quelli che sono richiesti, il BASIC risponderà con:

```
?EXTRA IGNORED
```

e considera' validi solamente tanti caratteri quanti sono quelli necessari a soddisfare la lista d'ingresso.Al contrario, quando non saranno stati consegnati dati in numero sufficente, il BASIC rispondera' con:

```
??
```

e fara' lampeggiare il cursore in attesa dei dati che mancano. Se durante l'interpretazione di un campo numerico verranno incontrati dei dati alfabetici il BASIC rispondea' con:

```
?REDO FROM START
```

nel PET, se l'ingresso sara' seguito solamente da un ritorno carrello senza che sia stato stampato niente altro, il BASIC considerera' tale azione come una terminazione del programma, allo stesso modo del tasto di stop. Questo particolare modo di funzionamento e' una eredita' dei giorni delle telescriventi, quando tale modo operativo era la maniera piu' conveniente di terminare un programma.
Durante l'attesa di dati il PET inibisce il funzionamento del tasto di stop.
L'istruzione di INPUT permette anche di eseguire delle opeazioni speciali per indicare all'utente quali caratteri d'ingresso sono desiderati e in che forma essi devono essere consegnati. Un letterale che segue il comando di INPUT e' stampato prima che venga stampato il punto interrogativo. Ad esempio:

```
10 INPUT"BIRTHDAY";A
```

verra' stampato come:

```
BIRTHDAY?
```

e il calcolatore rimarra' in attessa del valore che specifichera' in forma numerica il giorno del compleanno e lo carichera' in un campo numerico. Qui di seguito si riporta un esempio di ingresso in un programma che calcola il terzo lato di un triangolo rettangolo:

```
10 INPUT"FIRST LEG";A
20 INPUT"SECOND LEG";B
30 IF A=0 OR B=0 THEN 10
40?"THIRD IS";SQR(A*A+B*B)
50 GOTO 10
```

Facendo eseguire questo programma e consegnando come valori 3 e 4 rispettivamente, si otterra' come risultato 5. Il programma puo' essere modificato per vedere come si possono cambiare i valori su una singola linea. Si cancelli la linea 20, si listi la linea 10, tale linea venga poi modificata nel modo seguente:

```
10 INPUT"FIRST LEG,SECOND LEG";A,B
```

Quando il programma viene eseguito i valori 3 e 4 devono essere consegnati in una unica operazione e separati tra loro da una virgola. E' evidente che ambedue i modi di operare sono accettabili, tuttavia e' buona pratica di programmazione evitare che l'utente possa far confusione, quando piu' campi vengono riuniti sulla stessa linea. Non e' infatti buona pratica mescolare sulla stessa linea valori numerici e alfanumerici come nell'esempio seguente:

```
10 INPUT"NAME,BIRTHDAY";A$,A
```

L'ISTRUZIONE GET

Il maggior problema presentato dall'istruzione di INPUT e' che essa non permette di lavorare realmente in real time. Durante il tempo adoperato dall'utente per introdurre i dati e premere il tasto del ritorno carrello, tutti i procedimenti di elaborazione vengono arrestati. Il BASIC del PET e' stato fornito di una speciale funzione che permette di accettare dalla tastiera un carattere alla volta o di accertarsi se un tasto e' stato premuto.

Il comando e' GET. Come sintassi GET e' identico a INPUT. E' possibile specificare una lista di variabili, ma in generale tale modo di operare non e' il migliore in quanto lo scopo del GET e' tenere sotto sorveglianza la tastiera e ritornare al programma non appena un tasto viene premuto. Quando viene specificato un valore numerico come ad esempio:

```
GET A
```

solamente tasti numerici vengono accettati come ingresso.
La pressione di qualsiasi altro tasto dara' luogo al messaggio:

```
?SYNTAX ERROR
```

L'uso di un valore numerico puo' dar luogo a confisione perche' il valore restituito e' zero, se nessun tasto viene premuto. In caso contrario si avra' un valore compreso fra 1 e 9 per i tasti da 1 a 9. L'uso migliore del GET si ha con le variabili stringa. Se nessun tasto viene premuto, la stringa avra' valore nullo e lunghezza pari a 0;in caso contrario la stringa conterra' il carattere corrispondente al tasto premuto. E' opportuno riferirsi alla sezione successiva per una spiegazione dettagliata dell'uso delle stringhe secondo questa tecnica. L'istruzione GET richiama una routine che esamina il buffer degli interrup di tastiera. Se il buffer e' vuoto la variabile contiene un valore nullo o 0. Se vi sono caratteri, il primo di esso viene estratto dalla coda e restituito al programma. Poiche' la lunghezza della coda e' di 10 caratteri richiamando l'istruzione GET 10 volte in un anello si e' sicuri che la coda sara' vuota quando si attendera' una risposta. Questo modo di operare e' particolarmente nei giochi interattivi.
L'esempio seguente permette di ottenere un'attesa finche' un tasto non verra' premuto e fara' proseguire il programma non appena si avra' la pressione di un tasto.

```
10 GET A$
20 IF A$=""THEN 10
```

nel caso esaminato, "" e' il letterale che non contiene caratteri e la stringa e' nulla.

CAPITOLO 6

TECNICHE AVANZATE DI PROGRAMMAZIONE

IN quanto precede, sono state descritte quasi esclusivamente funzioni numeriche, ma quasi tutti i programmi operano anche con grandezze alfanumeriche.Il BASIC contiene un insieme di funzioni che permettono di lavorare con questi dati. I dati alfanumerici possono essere definiti come una connessione continua di caratteri che vengono visti dal BASIC come valore di una singola variabile. Nel BASIC del PET, il simbolo viene usato per esprimere variabili che sono stringhe o dati alfanumerici.Tutte le regole che si applicano alle normali variabili, si applicano anche alle variabili stringhe.
Seguendo le convenzioni per la designazione dei nomi si puo' creare una variabile A$ che e' diversa dalla variabile A% o dalla variabile A. Si batta a titolo di prova A$="NOW IS THE TIME" e PRINT A$ per esaminare il valore della stringa.In questo modo puo' venir definita una stringa di lunghezza fino a 70 caratteri, a seconda del numero di cifre del numero di linea, cioe' si puo' creare una stringa con tanti caratteri quanti stanno in una linea. Tuttavia, la limitazione del numero di caratteri che possono essere memorrizzati in una stringa e' 255 si possono costruire stringhe piu' lunghe di quanto non siano le stringhe che possono essere fatte entrare con una unica linea.L'accumulazione di caratteri provenienti da un dispositivo di ingresso/uscita e la costruzione di dati viene fatta attraverso la concatenazione di stringhe. L'operatore che viene usato e' " + ".

Si puo' modificare l'espressione A$ dell'esempio precedente, battendo sulla tastiera A$=A$+"FOR ALL". Si stampi poi A$ si vedra' che il letterale che verra' stampato ha uno spazio in testa. A differenza dei numeri che vengono formattati dal BASIC, il valore di un letterale viene preso come sta'. Una stringa puo' contenere qualsiasi combinazione di bit incluse quelle combinazioni che formano il carattere di controllo come ad esempio, lo spostamento del cursore e il ritorno carrello. Tutto cio' verra' illustrato piu' avanti.

Come detto il BASIC permette di usare stringhe di lunghezza fino a 255 caratteri. Queste possono essere fatte uscire sullo schermo o su qualsiasi dispositivo d'uscita che accetti piu' di 79 caratteri, l'ingresso tuttavia e' limitato a 79 caratteri a causa della dimensione del buffer d'ingresso. Questo problema puo' essere superato spezzando le stringhe in sottostringhe prima di inviarle in INPUT o usando l'istruzione GET per far entrare ciascun carattere individualmente. Le sottostringhe cosi' ottenute o i caratteri individuali possono essere ricombinati nella stringa originale mediante la concatenazione.

CONFRONTO DI STRINGHE

In figura 2.6 e' riportata la tabella dei caratteri ASCII, essa contiene l'ordine in cui i caratteri vengono rappresentati nel PET quando due stringhe vengono paragonate. I caratteri contenuti in un set di stringhe vengono paragonati partendo dal carattere che sta piu' a sinistra e proseguendo fino alla fine del campo di definizione.

Con riferimento alla tabella dei caratteri ASCII, si supponga di paragonare una stringa che contenga una "A" da un'altra che contenga una "B" nella stessa posizione. Il risultato sara' che la seconda stringa e' considerata maggiore della prima.

Una stringa che contenga solamente lo spazio bianco e' considerata minore di una stringa che contenga il carattere 1, il quale a sua volta e' minore del carattere "A" che e' minore del carattere "B". La stringa che contiene unicamente il carattere "A" e' minore della stringa "ABC" o di qualsiasi altra stringa che abbia il carattere "A" come primo carattere. Tutti i caratteri vengono confrontati in sequenza e il primo carattere che differisce sulle due stringhe definisce la relazione fra le stringhe. Si possono quindi in tal modo usare le stesse funzioni relazionali sia per le srtinghe che per i numeri, in particolare i simboli:

<> per esaminare la relazione di diverso
= per uguale
< per minore
> per maggiore

Un uso immediato del confronto fra stringhe puo' essere quello di costruire delle liste ordinate alfabeticamente o dei files o dei direttori telefonici. Il confronto puo' anche essere usato per ricercare un dato in questa lista ordinata o nell'elenco telefonico.

Si invita il lettore di questo mauale a provare il seguente programma:

```
10 INPUT A$
20 INPUT B$
30 IFA$=B$THEN?"A$=B$":GOTO10
40 IFA$<B$THEN?"A$<B$":GOTO10
50 PRINT"A$>B$":GOTO10
```

NUMERI E CODICI ASCII

Le seguenti due coppie complementari di operazioni sulla stringa e sui numeri permetteranno di usare le stringhe in maniera non convenzionale.

STR$

La STR$ e' funzione di un argomento. L'argomento dev'essere numerico ed essa restituisce il carattere che rappresenta l'espressione numerica. Ad esempio:

```
10 X=3.1
20 ?STR$(X)
```

Eseguendo il programma sullo schermo verra' stampato 3.1
I numeri positivi verranno preceduti nella loro stringa equivalente da uno spazio bianco. I numeri negativi avranno il segno meno nella posizione corrispondente.

VAL

VAL e' una funzione complementare di STR$. Essa converte una stringa in un numero che puo' essere usato per i calcoli. Se il primo carattere diverso dallo spazio bianco che si incontra nella stringa e' un carattere non numerico allora la funzione VAL restituira' un valore nullo. In altre parole la funzione VAL puo' convertire tanti caratteri numerici quanti ne incontra prima di trovare un carattere non valido. Ad esempio:

```
?VAL("3.14 AB")
```

restituira' il valore 3.14

CHR$

Si e' gia' visto in precedenza che in una stringa si possono avere tutti i normali caratteri ASCII introducendoli o attraverso una stringa racchiusa tra virgolette o direttamente dall'organo di INPUT, tuttavia parecchi dispositivi richiedono i caratteri di

controllo che non possono essere prodotti con i normali mezzi. Ad esempio: una stampante PET usa il ritorno carrello assieme al tasto delle maiuscole come carattere di terminazione per indicare un ritorno carrello non seguito da un avanzamento linea per eseguire in tal modo delle sovrastampe. La funzione CHR$ permette allora di specificare questi caratteri di controllo semplicemente assegnando il numero di codice ASCII del carattere stesso. La funzione CHR$ quindi, converte un numero nella rappresentazione interna ASCII. Il valore dell'argomento deve essere compreso fra 0 e 255. Ad esempio:

```
10 A$=CHR$(65)+CHR$(66)
20 PRINT A$
```

sullo schermo appariranno i caratteri A B.
In questo esempio il numero 65 e' il codice ASCII per il carattere A e il numero 66 il codice ASCII per il carattere B. Il programma converte tali codici nei caratteri prima di concatenarli e stamparli.

ASC

La funzione ASC converte un carattere in un codice ASCII e tale codice puo' essere poi usato in calcoli numerici. Il parametro d'ingresso della funzione ASC e' una stringa. Ad esempio eseguendo l'istruzione:

```
?ASC("A")
```

sullo schermo comparira' il numero 65.Se la stringa e' formata da piu' di un carattere la funzione ASC restituira' il codice del primo carattere della stringa.Ad esempio:

```
?ASC("123")
```

dara' come risultato,sullo schermo, il numero 49.Il codice ASCII per "1" e' 49.

SEGMENTI DI STRINGA

In molti casi sarebbe desiderabile poter avere solamente una parte di una stringa ad esempio quando si voglia sviluppare una lista in ordine alfabetico. Si supponga ad esempio che in corrispondenza con una istruzione di INPUT venga battuto sulla tastiera il nome di una persona.Esso puo' essere battuto facendo entrare dapprima il nome poi una eventuale iniziale del secondo nome e infine il cognome. Per l'ordinamento alfabetico e' necessario tuttavia che il nome venga ignorato e l'ordinamento stesso venga eseguito sulla base del cognome.Per separare parti di stringa e usarne in una espressione,il BASIC del PET fornisce tre funzioni. Come si vedra', la maggior parte dei programmi che operano su stringhe si limiteranno ad usare queste tre funzioni per analizzare pezzi di una stringa complete. Nell'immediato seguito verranno presentate queste tre funzioni e definite tutte e tre;malgrado che esse siano essenzialmente la stessa funzione. Vengono date tre possibilita' d'uso semplicemente per comodita' dell'utlizzatore.Le tre funzioni sono:

La funzione LEFT$
La funzione RIGHT$
La funzione MID$

La funzione LEFT$ (stringa,I) restituisce gli "I" caratteri piu' a sinistra della stringa che appare in argomento.Se "I" e' negativo,zero o maggiore di 255, allora sullo schermo apparira' il messaggio ILLEGAL QUANTITY ERROR. LA funzione RIGHT$ (stringa,I) restituisce gli "I" caratteri piu' a destra della stringa argomento. Quando "I" e' minore di zero o maggiore di 255, sullo schermo viene stampato il messaggio ILLEGAL QUANTITY ERROR. Per la funzione MID$ esistono due forme.

La prima, la piu' generale, e' MID$ (stringa,I,J).In tal caso vengono restituiti i "J" caratteree della stringa argomento che partono dalla Iesima posizione. Se "I" e' maggiore della lunghezza della stringa, viene restituita una stringa nulla. Quando "I" o"J" sono negativi o maggiori di 255 viene stampato sullo schermo il messaggio ILLEGAL QUANTITY ERROR. Se "J" e' maggiore del numero di caratteri che possono essere prelevati dalla stringa, allora vengono restituiti tutti i caratteri dalla posizione "I" alla fine della stringa stessa. La seconda forma e' MID$ (stringa,I) ed ha lo stesso effetto della forma precedente quando "J" e' maggiore della lunghezza della stringa.Vengono in tal caso restituiti tutti i caratteri che partono dalla posizione "I" fino alla fine della stringa. Se "I" e' maggiore della lunghezza della stringa allora viene restituita una stringa nulla,se viceversa "I" e' negativo o maggiore di 255 sullo schermo viene stampato il messaggio:ILLEGAL QUANTITY ERROR.
Queste tre funzioni, quindi, permettono di estrarre dalla stringa argomento un certo numero di caratteri iniziali, finali o inseriti nella stringa a partire dalla posizione "I". Nell'esempio che si faceva precedentemente per individuare il cognome dei dati di un individuo e' sufficiente analizzare i caratteri a partire dal carattere che si trova all'estrema destra e retrocedere finche' non s'incontra lo spazio bianco. Per implementare questo programma e' tuttavia sufficente una sola delle funzioni descritte.

LUNGHEZZA DI UNA STRINGA

La funzione LEN fornisce l'esatto numero di caratteri contenuto in una stringa. Vengono conteggiati anche i caratteri che non appaiono in stampa sullo schermo e agli spazi bianchi. Le informazioni di stringa vengono memorizzate dal BASIC in un vettore da 3 bytes. Due bytes contengono il puntatore alla locazione di memoria in cui la stringa e' memorizzata,mentre il terzo bytes e' la lunghezza, la funzione LEN estrae questo byte. Si puo' ora scrivere una programma di uso generale per estrarre il cognome da un nome completo.

```
10 INPUT"NAME:FIRST,MI,LAST",A$
20 I=LEN(A$)
30 IF MID$(A$,I,1)=""THEN 60
40 I=I-1
50 IF I>0 GOTO 30
60 PRINT"LAST NAME=";MID$(A$,I+1)
```

In queto esempio sono state usate le due forme di MID$. Nella linea 30 la funzione e' stata usata specificando nel primo parametro la lunghezza. Tale linea e' usata per ricercare il carattere di spazio bianco che delimita il cognome. Nella linea 60, viceversa, non e' stata specificata alcuna lunghezza.

MEMORIZZAZIONE DELLE STRINGHE

Le stringhe vengono accumulate nello spazio che si trova fra la fine del vostro programma BASIC e la piu' alta locazione di RAM.Ogni volta che viene aggiunta una stringa,una catena si allunga verso il basso a partire dalla posizione piu' alta della memoria. La memorizzazione e' ottimizzata in modo da non creare mai una copia di una stringa assegnata ad un letterale. In altre parole, se viene eseguita un'espressione A$=B$, sia A$ che B$ saranno copia della stessa stringa. Si richiede una nuova stringa solamente se viene eseguita una concatenazione o una operazione di INPUT; un sempio consistente dell'uso delle funzioni stringa. Usando le funzioni stringa descritte si puo' scrivere una routine che mescola un mazzo di carte e le distribuisce una per volta.

La routine che segue ha applicazione in parecchi giochi, come il poker o il bridge. Si noti l'uso dei simboli delle carte che sono disponibili sul PET.

```
105 PRINT"□"    :REM SET UP DECK WITH ALL 52 CARDS
110 C$="A♠2♠3♠4♠5♠6♠7♠8♠9♠T♠J♠Q♠K♠"
120 C$=C$+"A♥2♥3♥4♥5♥6♥7♥8♥9♥T♥J♥Q♥K♥"
130 C$=C$+"A♦2♦3♦4♦5♦6♦7♦8♦9♦T♦J♦Q♦K♦"
140 C$=C$+"A♣2♣3♣4♣5♣6♣7♣8♣9♣T♣J♣Q♣K♣"
190 REM PULL A CARD
200 R=2*INT(LEN(C$)*RND (1)/2+1)-1
201 N$=MID$(C$,R,1):Y$=MID$(C$,R+1,1)
430 REM SHRINK THE DECK
432 IFR>1THENT$=LEFT$(C$,R-1):GOTO435
433 T$=""
435 C$=T$+MID$(C$,R+2)
439 REM PRINT A CARD
440 PRINTN$;Y$,
450 IFLEN(C$)>=1THEN200
455 REM ENDOF DECK
460 INPUT"ANOTHER DEAL    ■■■";Z$
470 GOTO105
READY.
```

All'inizio la stringa C$ viene riempita con i valori del mazzo di carte.Due caratteri rappresentano ciascuna carta, il seme e' il valore.In fase di distribuzione N$ contiene il valore e Y$ contiene il seme. La stringa C$ che rappresenta il mazzo di carte si restringe via via in modo che una carta possa essere distribuita una volta sola.L'istruzione 105 cancella lo schermo. La stringa C$ viene inizializzata nelle istruzioni che vanno da 110 a 140. C$ viene concatenato in quanto l'assegnazione del letterale e' troppo grande per poter essere eseguita su una sola linea.
L'istruzione 200 usa la funzione RND per generare un indice all'interno della stringa C$. L'indice RANDOM sara' compreso tra un valore pari a 1 e un valore pari alla lunghezza della stringa C$-1. Nell'istruzione 201 l'indice e' usato per estrarre N$ (il valore) eY$ (il seme) dalla stringa C$ per mezzo della funzione MID$. Le Istruzioni da 432 a 435 rimuovono la carta dalla stringa in modo che essa non possa essere distribuita di nuovo. Poiche' il secondo argomento della funzione LEFT$ non puo' essere nullo, viene eseguita nell'istruzione 432 una verifica che R>1 in modo di prevenire la stampa del messaggio ILLEGAL QUANTITY ERROR. L'istruzione 440 stampa ciascuna carta via via che essa e' estratta.L'istruzione 450 esegue il test di fine mazzo e l'istruzione 460 permette al giocatore di mescolare di nuovo il mazzo.

FUNZIONI DEFINIBILI DELL'UTENTE

Fino a questo punto sono state spiegate le funzioni intrinseche al BASIC. Nel campo della matematica si usano pero' molte piu' funzioni, specialmente trigonometriche. Si potrebbe, e' vero, pensare di scrivere un programma per approssimare in linea certe funzioni, ma cio' e' veramente noioso e dal punto di vista della documentazione un'espressione per quanto semplice potrebbe divvenire non facile a capirsi.Fortunatamente nel BASIC esiste la possibilita' di definire funzioni in termini di altre funzioni.Una funzione viene definita in un'istruzione DEF;

ad esempio:

```
100 INPUT B
110 INPUT C
120 DEF FN A(V)=V/B+C
```

Il nome della funzione e' "FN" seguito da qualsiasi nome legale di variabile.Si ricordi che il nome di una variabile puo' essere sia una singola lettera, sia una lettera seguita da una successiva lettera o da un numero. Ad esempio nomi legali per funzioni sono:

```
FNX
FNJ7
FNK0
FNR2
```

Le piu' severe limitazioni per le funzioni definite dall'utente stanno nel fatto che ciascuna d'esse deve poter essere definita interamente su una lina di 80 caratteri e che le funzioni stringa non possono essere definite. Le variabili che si trovano nella parentesi che segue il nome della funzione vengono chiamate variabili dummy.Una funzione puo' essere definita mediante qualsiasi espressione ma deve avere solamente un argomento.Le variabili che vengono usate nelle espressioni vengono considerate variabili globali (hanno cioe' lo stesso valore che nel resto del programma) e per il calcolo viene usato il loro valore corrente. Dopo aver definito una funzione puo' essere usata in un programma, ad esempio:

```
130 Z=FNA(3)
140 ?Z
```

Quando viene definita una funzione nella tavola delle variabili viene caricato semplicemente un nome di variabile.Il primo carattere di questo nome ha 18 bit, cioe' il bit piu' significativo messo a 1 per indicare che il nome stesso e' un nome di variabile. Con questo nome vengono associati due puntatori: un indirizzo della posizione del testo in cui la funzione e' immagazzinata e un indirizzo in cui e' memorizzata la variabile dummy. Il programma che esegue la funzione e' un programma rientrante in modo che la funzione possa essere definita in termini di altre funzioni definite dall'utente.Se durante l'esecuzione lo spazio disponibile per lo stack viene completamente utilizzato in quanto si ha a che fare con procedure ricorsive si avra'un errore di traboccamento di memoria. Sono riportate funzioni definite dall'utente pronte per essere usate in programmi BASIC in memoria 6.1

FUNCTIONS EXPRESSED IN TERMS OF BUILT-IN BASIC FUNCTIONS

```
SECANT,SEC(X)
   DEF FNA(X)=1/COS(X)
       FOR X<>P GRECO/2

COSECANT,CSC(X)
   DEF FNB(X)=1/SIN(X)
       FOR X<>0

CONTANGENT,COT(X)
   DEF FNC(X)=COS(X)/SIN(X)
       FOR X<>0

INVERSE  SINE,ARCSIN(X)
   DEF FND(X)=ATN(X/SQR(-X*X+1)
       FOR ABS(X)<1
```

```
 INVERSE COSINE,ARCCOS(X)
    DEF FNE(X)=-ATN(X/SQR(-X*X+1))+P GRECO/2
        FOR ABS(X)<1

INVERSE SECANT,ARCSEC(X)
    DEF FNF(X)=ATN(SQR(X*X-1))+(SGN(X)-1*P GRECO/2
        FOR ABS(X)>1

 INVERSE COSECANT,ARCCSC(X)
    DEF FNG(X)=ATN(1/SQR(X*X-1))+(SGN(X)-1)*P GRECO/2
        FOR ABS(X)>1

 INVERSE COTANGENT,ARCCOT(X)
    DEF FNH(X)=-ATN(X)+P GRECO/2
        FOR ANY X

 HYPERBOLIC SINE,SINH(X)
    DEF FNI(X)=(EXP(X)-EXP(-X))/2
        FOR ANY X

 HYPERBOLIC COSINE,COSH(X)
    DEF FNJ(X)=(EXO(X)+EXP(-X))/2
        FOR ANY X

 HYPERBOLIC TANGENT,TANH(X)
    DEF FNR(X)=-EXP(-X)///(EXP(X)+EXP(-X))*2+1
        FOR ANY X

 HYPERBOLIC SECANT,SECH(X)
    DEF FNL(X)=2/(EXP(X)+EXP(-X))
        FOR ANY X

 HYPERBOLIC COSECANT,COSH(X)
    DEF FNM(X)=2/EXP(X)-EXP(-X))
        FOR X <>0

 HYPERBOLIC COTANGENT,COTH(X)
    DEF FNN(X)=EXP(-X)/(EXP(X)+EXP(-X))*2+1
        FOR X<>0

INVERSE HYPERBOLIC SINE,ARCSINH(X)
    DEF FNO(X)=LOG(X+SQR(X*X+1))
        FOR ANY X

 INVERSE HYPERBOLIC COSINE,ARCCOSH(X)
    DEF FNP(X+SQR(X*X-1))
        FOR X>=1

 INVERSE HYPERBOLIC TANGENT ,ARCTANH(X)
    DEF FNQ(X)=LOG((1+X)/(1-X))/2
        FOR ABS(X)<1

 INVERSE HYPERBOLIC SECANT,ARCSECH(X)
    DEF FNR(X)=LOG((SQR(-X*X+1)+1)/X)
        FOR 0<X<=1

 INVERSE HYPERBOLIC COSECANT,ARCCOSH(X)
    DEF FNS(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
        FOR X<>0

 INVERSE HYPERBOLIC COTANGENT,ARCCOTH(X)
    DEF FNT(X)=LOG((X+1)/(X-1))/2
        FOR ABS(X)>1
```

ISTRUZIONI DI GOSUB E RETURN

Si e' visto ora che le funzioni definite dall'utente, funzioni con un singolo argomento, possono essere usate come qualsiasi altra funzione intrinseca. La maggiore limitazione delle funzioni definite dall'utente sta nel fatto che essa e' realizzata mediante una singola espressione algebrica e deve essere contenuta in una sola linea. Tuttavia in molti casi e' necessario ripetere un certo numero di istruzioni durante lo svolgimento di un programma. Queste istruzioni possono essere raccolte in un unico gruppo ed eseguiuite mediante un'istruzione di GOSUB, ad esempio:

```
GOSUB 5000
```

Le linee di programma che sono cosi' riunite vengono chiamate subroutine. L'istruzione di GOSUB permette di andare ad eseguire le istruzioni della subroutine. Essa differisce dal GOTO in quanto il GOSUB ricorda qual'e' la linea da cui si e' partiti per andare alla subroutine stessa e ritorna alla linea immediatamente successiva automaticamente dopo che la subroutine e' stata eseguita. Una subroutine viene memorizzata come una serie di linee in linguaggio BASIC che partono al numero di linea specificato nell'istruzione GOSUB. L'ultima linea della subroutine deve essere un'istruzione di RETURN. Questo permette al BASIC di far ripartire l'esecuzione alla linea immediatamente successiva all'istruzione di GOSUB una volta che la subroutine sia stata terminata di eseguire.Ad esempio:

```
10 REM THIS IS THE MAINE LINE CODE
20 GOSUB 50
30 STOP
50 REM THIS IS A SUBROUTINE
60 RETURN
```

In un programma scritto in tal modo, le linee verrebbero eseguite nella sequenza riportata:

```
10-20-50-60-30
```

Ogniqualvolta che viene eseguita un'istruzione di GOSUB lo stack viene caricato con 5 bytes:nel primo un contrassegno di GOSUB e successivamente due bytes, uno per il numero di linea e l'altro per l'indirizzo del testo dell'istruzione GOSUB.Il numero di linea che segue il codice GOSUB viene poi utilizzato come numero di linea da eseguire immediatamente; e una routine di GOTO abilita il salto.L'istruzione di RETURN rimette a posto il numero di linea e l'indirizzo del testo estraendo questi dati dallo stack per riprendere l'esecuzione del programma dal punto immediatamente successivo a quello in cui si trova l'istruzione di GOSUB. Tutti i punti d'ingresso dei FOR di fronte all'istruzione di GOSUB vengono eliminati. La limitazione fisica sul numero di GOSUB che possono essere contemporaneamente in funzione e' di 23 richiami a subroutine. Esempi di subroutine:
si considera la funzione fattoriale:
n!=1x2x3x...xn
Questa funzione non puo' essere definita attraverso una definizione di funzione d'utente. Tuttavia essa puo' essere realizzata mediante il semplice programma che segue:

```
10   INPUT N
100  I=1/NF=1
110  NF=NF*I
120  I=I+1
130  IF I<=N GOTO 110
140  PRINT NF
```

Queste routine comprese fra le linee 100 e 140 potrebbe essere usata diverse volte durante l'esecuzione di un programma assegnando diversi valori ad N.

Ad esempio si supponga di voler calcolare il coefficente binominale:

$$\binom{M}{R} \qquad \frac{M}{R!\ (M-R)!}$$

il programma che realizza tale calcolo e' il seguente:

```
 10 PRINT"M=";INPUT M
 15 PRINT"R=";INPUT R
 20 N=M:GOSUB 100:X=NF
 30 N=R:GOSUB 100:Y=NF
 40 N=M-R:GOSUB 100:Z=NF
 50 BC=X/(Y*Z)
 60 PRINT BC
 70 END
100 I=1:NF=1
110 NF=NF*I
120 I=I+1
130 IF I<=N GOTO 110
140 RETURN
```

Esesuendo questo programma ed assegnando ad M il valore 11 e a R il valore 6 il risultato sara' 462.
Le subroutine agiscono come una"black box"o una funzione complessa all'interno del programma. Certe variabli fisse vengono usate per introdurre dati, ed altre variabili fisse o talvolta la stessa variabile vengono usate per ottenere i risultati. Ad esempio: nelle subroutine comprese fra le linee 100 e 140, la variabile"N" e'l'ingresso e la variabile "NF" e' l'uscita.

```
     ----------
     I        I
N--->I SUB100 I ---> NF
     I        I
     ----------
       NF=N!
```

Ponendo la variabile "N" usuale rispettivamente a "M","R","M-R" si ottiene in "NF" il valore "M!","R!" e "(M-R)!. Ovviamente per una buona funzionalita' le subroutine non devono richiedere variabili;dall'ingresso esse possono viceversa realizzare delle funzioni specifiche per la stampa in forma speciale sullo schermo.

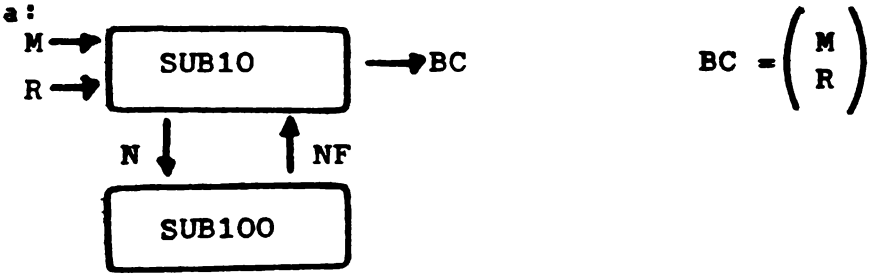
```
     ----------
     I        I
     I  SUB   I ----> print form
     I        I
     ----------
```

RICHIAMI INTERNI E SUBROUTINE

Il programma dell'esempio precedente puo' essere esso stesso reso un subroutine per essere utilizzato come una routine che calcola il coefficente binomiale e possa venir usato all'interno di un altro programma. Semplicemente cambiando la linea 70 in

```
70 RETURN
```

La subroutine che cosi' si ottiene e che verra' chiamata SUB 10, inizia la linea 10 e termina alla linea 70 ed ha la seguente struttura:



I richiami a subroutine dall'interno di un'altra subroutine vengono chiamati richiami interni. Spesso tale tecnica di operare viene chiamata annidamento di subroutine. Nell'esempio fatto la SUB 10 esegue un richiamo interno alla SUB 100,oppure, in altre parole,la SUB 100 e' annidata nella SUB 20. E' abbastanza frequente il caso di programmi che utilizzano subroutine annidate in altre subroutine che a sua volta sono annidate in altre subroutine e cosi' via. L'unico limite a tale tecnica operativa e' la quantita' di memoria disponible.Le subroutine possono essere annidate in piu' che una subroutine. Ad esempio una subroutine di input che accetta uno specifico carattere dalla tastiera stampa a cursore lampeggiante e stampa sullo schermo il carattere dato; puo' essere chiamata in qualsiasi istante sia dal programma principale che da altre subroutine.

PRECAUZIONI DA OSSERVARE

Un errore comune che si commette nell'usare delle subroutine e'permettere al programma principale di invadere una subroutine che lo segue. Questo fatto dara' luogo a un messaggio di errore e piu' esattamente a RETURN WITHUOT GOSUB ERROR. Per evitare tale inconveniente e' necessario inserire in posizione opportuna,nel programma,un'istruzione di STOP o un'istruzione di END.Talvolta si ha la tendenza ad usare comunque una subroutine. Quando una subroutine e' usata una volta sola allora e' bene incorporare le istruzioni che la formano nel programma principale in modo da risparmiare tempo d'esecuzione e spazio di memoria occupata. In conclusione si puo' dire che le subroutine sono degli strumenti incredibilmente potenti per la programmazione e permettono di strutturare i programmi in blocchi.

ANELLI FOR-NEXT

Le istruzioni FOR-NEXT semplificano la scrittura di programmi BASIC permettendo di realizzare delle strutture complesse ad anello con una singola istruzione. Ad esempio:

```
FOR I=A TO B STEP C
```

La fine dell'anello e' specificata dall'istruzione

```
NEX
```

Sono permessi anche degli anelli FOR-NEXT annidati uno dentro l'altro purche' ciascun anello usi un'unica variabile. L'uso di nomi di variabili identici puo' dar luogo a degli errori che visualizzano sullo schermo il messaggio NEXT WITHOUT FOR. Uscendo dal ciclo FOR-NEXT attraverso un salto fa si che dallo stack esca l'indicazione del FOR. La migliore via per eseguire un'uscita del tipo e' assegnare alla variabile un valore pari al suo valore massimo,indi uscire dall'anello attraverso il NEXT. Gia' in precedenza e' stato mostrato come operazioni ripetute possono essere fatte usando una variabile contatore come ad esempio "I" nel programma seguente:

```
10 I=1
20 I=I+1
30 IF I<=10 THEN GOTO 20
```

Nel caso illustrato qualsiasi procedura di calcolo che sia realizzata con le linee da 21 a 29 sara' ripetuto 10 volte.In piu' la variabile"I" variera' il suo valore tra 1 e 10 incrementando ogni volta il suo valore di 1. Il processo di anello puo' essere generalizzato nel caso seguente:

```
10 I=A
20 I=I+C
30 IF I<=B THEN GOTO 20
```

Il valore di "I" variera' ora fra A e B incrementando ad ogni giro il suo valore di C. Poiche'tale procedura e' molto comune nella stesura di programmi,il BASIC permette di fare cio' attraverso l'struzione di FOR-NEXT.La stessa cosa,infatti,verra' fatta secondo le seguenti istruzioni:

```
10 FOR I=A TO B STEP C
20 NEXT
```

"I" e' la variabile contatore, "A" e' il valore iniziale,"B" il valore finale e "C" e' l'incremento o passo."A","B" e"C" possono essere non solo delle costanti, ma anche delle espressioni aritmetiche valide. Ad esempio:

```
10 FOR I=A(2)+1 TO J*2 STEP-1
```

In altre parole, la variabile contatore, puo' essere qualsiasi valore floating,ma non puo' essere un intero, ad esempio 1% o una variabile con indice come I (1,4). Quando il valore dell'incremento e' 1(C=1) si puo' omettere di includere lo STEP nell'istruzione FOR. Si esamini ora il seguente esempio:

```
10 REM COMPUTATION OF FACTORIAL
20 NF=1
30 FOR I=1 TO N
40 NF=NF*I
50 NEXT
```

Si noti come questo programma sia molto piu' corto e piu' chiaro di quello scritto precedentemente per eseguire la stessa funzione.Non appena un FOR viene eseguito, un punto d'ingresso a 16 bytes viene caricato nello stack. Prima di fare tutto cio' viene eseguito un test per vedere se vi sono altri punti d'ingresso caricati nello stack con la stessa variabile d'anello. In tal caso questo punto d'ingresso FOR e tutti gli altri punti d'ingresso FOR che sono stati fatti in precedenza vengono eliminati dallo stack. L'operazione descritta viene fatta in modo che un programma salti fuori a meta'di un anello di FOR e non tenga occupati inutilmente 16 bytes dello spazio di stack. L'istruzione NEXT cerca il piu' prossimo punto d'ingresso di stack oppure la variabile specificata come parametro sull'istruzione NEXT e ressetta lo stack a questo punto. Se non e' possibile trovare il punto di reset si avra' un errore di NEXT WITHUOUT FOR. Anche l'esecuzione di un'istruzione di GOSUB carica nello stack un punto d'ingresso a 5 bytes. Quando viene eseguita l'istruzione RETURN, lo stack viene esplorato alla ricerca di un punto d'ingresso FOR che non puo' essere chiuso. Qando tutti i punti d'ingresso FOR sono stati esplorati viene posizionato un pointer al punto d'ingresso GOSUB.Questo assiciura che se si esegue un GOSUB a una sezione di programma in cui e' stato fatto aprire un anello di FOR, che pero' non viene chiuso, l'istruzione di RETURN possa comunque ritrovare il piu' recente punto d'ingresso GOSUB. Il RETURN elimina dallo stack i punti GOSUB e tutti i punti d'ingresso FOR fatti prima dell'inizializzazione del salto a subroutine.

ANELLI FOR-NEXT ANNIDATI UNO DENTRO L'ALTRO

Gli anelli FOR-NEXT, allo stesso modo delle subroutine, possono essere annidati uno dentro l'altro. Cio' significa che un anello FOR-NEXT puo' essere contenuto in un altro anello FOR-NEXT e cosi' via.E' tuttavia molto importante che per ciascun anello venga usata una variabile contatore diversa, come appare evidente nel seguente esempio:

```
10 FOR I=1 TO 10
15 PRINT "I"
20 FOR J=1 TO 10
25 PRINT "J"
30 FOR K=1 TO  10
35 Print "K"
40 NEXT
50 NEXT
60 NEXT
```

A prima vista le linee comprese fra 40 e 60 di questo esempio possono generare confusione in quanto non e' possibile individuare qual'e' il FOR che corrisponde a ciascun NEXT. Opzionalmente si puo' specificare una variabile dopo NEXT. Questa variabile deve avere lo stesso nome della variabile contatore usata nel corrispondente FOR; comunque essa non e' richiesta dal BASIC per eseguire l'anello FOR-NEXT.Si potrebbero quindi cambiare le istruzioni da 40 a 60 del precedente esempio con:

```
40 NEXT K
50 NEXT J
60 NEXT I
```

Nel BASIC del PET e' permesso anche scrivere un'unica linea NEXT per terminare tutti e tre gli anelli FOR, nella fattispecie:

```
40 NEXT K,J,I
```

Si puo' tuttavia avere un errore di "NEXT WITHOUT FOR" se non si pone estrema cura nello specificare l'ordine di K,J,I.Tuttavia e' interessante comprendere come questo modo di operare possa rendere piu' compatta la scrittura di un programma e aumentare la potenzialita' dell'istruzione FOR-NEXT quando tale istruzione viene utilizzata per realizzare anelli annidati uno dentro l'altro. E' necessario a questo punto fare alcune osservazioni. La variabile contatore puo' essere cambiata di valore durante un anello. Ad esempio:

```
10 FOR I=1 TO 8
20 X=X+1
30 IF I=7 THEN I=8
40 NEXT
50 PRINT X
```

questo programma eseguira' il calcolo dell'espressione:

```
X=1+2+3+4+5+6+7=28
```

Un'altra osservazione importante e' bene farla quando si vuol uscire da un anello FOR-NEXT usando un'istruzione di salto, e' conveniente in questo caso assegnare alla variabile contatore il suo valore finale e poi uscire dall'anello attraverso l'istruzione NEXT.Ad esempio:

```
10 FOR I=1 TO 10
20 IF FNA(X)=0 THEN I=10
30 NEXT:RETURN
```

al posto di:

```
10 FOR I=1 TO 10
20 IF FNA(I)=0 THEN RETURN
30 NEXT
```

VARIABILI CON INDICE

Non e' necessario dichiarare attraverso un'istruzione DIM le matrici se esse hanno solamente una dimensione e contengono meno di 10 elementi. Il numero totale degli elementi di una matrice puo' essere calcolato moltiplicando le dimensioni di ciascun indice aumentato di 1 per l'altro indice. Ad esempio la matrice "A"(9,8) contiene (9+1)*(8+1) elementi. Gli indici partono da 0 e salgono via via fino al massimo valore.

Gli indici partono da 0 e salgono via via fino al massimo valore. Gli elementi sono ordinati nel modo seguente:

```
          A(0,0)      A(0,8)
          A(1,0)      A(1,8)
fino a:   A(9,0)      A(9,8)
```

I limiti del numero di indici e il valore massimo di ciascun indice sono determinati dall'ammontare di memorie disponibili. Il BASIC del PET permette che il numero totale degli elementi di una matrice sia al massimo 256 . Ciascun elemento della matrice richiede come minimo 5 bytes di memoria.
Se una matrice a una sola dimensione richiede piu' di 10 elementi l'istruzione DIM deve precedere qualsiasi utilizzo della matrice stessa nel programma.In caso contrario si avra' un errore di REDIM'ED ARRAY. Ad esempio la lista di un conto di bilancio:

```
  ---------------
1 I  $100         I
2 I  $1 3 5       I
3 I  $57.86       I
4 I  <$9 8 7>     I
5 I  $2 2         I
6 I  <$6 3 >      I
7 I  $5 0         I
8 I  <$2 1>       I
9 I  $2 1         I
  ---------------
```

Si supponga di voler scrivere un semplice programma che permette di introdurre un numero di conto e l'operazione fatta e' ottenere un totale per ciascun conto, indicando con A1,A2,A3,A4,A5 il saldo di ciascun conto. Un programma possibile potrebbe essere il seguente:

```
10 INPUT"ACCOUNT,CHARGE";I,C
20 IF I=1 THEN A1=A1+C
30 IF I=2 THEN A2=A2+C
```

Un programma di tale tipo richiederebbe l'esecuzione di una quantita' notevole di operazioni in parallelo per eseguire la somma su ciascun conto. Il programma puo' essere notevolmente semplificato ricorrendo ad una matrice.Ogni elemento della matrice puo' esere individuato attraverso il suo indice. Se come indice si usa il numero di conto, il programma precedente puo' essere semplificato nel modo seguente:

```
10 INPUT"ACCOUNT,CHARGE";I,C
20 A(I)=A(I)+C
30 GOTO 10
```

La lista che siottiene in talmodo e' una lista che presenta 9 righe e una colonna e utilizza internamente una matrice a una dimensione. Si puo' ottenere una tabella a piu' colonne utilizzando matrici a piu' dimensioni. Ad esempio mediante una matrice a due dimensioni si puo'ottenere la seguente tabella:

```
ACCOUNT  I BALANCE   OF TRANSACTIONS
-----------------------------------
  1      I $100              1     I
  2      I $135              1     I
  3      I $57.86*           1     I
  4      I <$987>            1     I
  5      I $22               1     I
  6      I <$63>             1     I
  7      I $50               1     I
  8      I <$21>             1     I
  9      I $21               1     I
         -------------------------
```

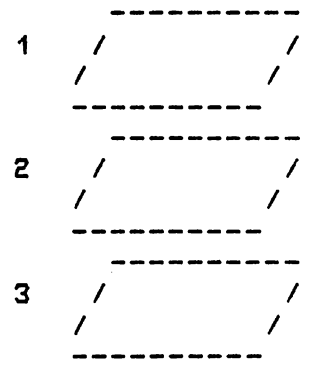
In questo caso la tabella ha 9 righe e 2 colonne. Per individuare il singolo elemento in questo caso, e' necessario specificare due indici, l'indice di riga e l'indice di colonna. Ad esempio, la quantita' indicata con un asterisco in riga 3 e colonna 1 viene individuata dai due indici 3 e 1.
Per permettere di utilizzare in un programma BASIC, tabelle multidimensionali, esiste un'istruzione che descrive il numero di righe e di colonne contenute in una matrice. E' questa l'istruzione "DIM" . Per la tavola appena illustrata di 9 righe e due colonne si dovra' scrivere:

```
DIM A(9,2)
```

Qualora quindi si volesse tener conto anche del numero di operazioni eseguito per ciascun conto il programma precedentemente illustrato va cosi' modificato:

```
10 INPUT"ACCOUNT,CHARGE";I,C
20 A(I,1)=A(I,1)+C
30 A(I,2)=A(I,2)+1
40 GOTO10
```

Si supponga ora di avere a che fare con una tabella per ciascuna di 5 compagnie, ciascuna delle quali abbia 9 conti e ciascun conto abbia un saldo al quale si vuole associare il numero di operazioni eseguite. Tutto cio' potrebbe essere fatto accatastando fogli di carta su ciascuno dei quali viene riportato l'insieme delle informazioni relative a ciascuna compagnia e numerando questi fogli.



Il semplice esempio che abbiamo qui riportato serve ad introdurre il concetto delle matrici multidimensionali. Queste matrici corrispondono a quelle usate in matematica.
Nella matematica un vettore e' un insieme ordinato di numeri.
V=(V1,V2,....Vn)
Tale vettore ha n componenti e viene chiamato vettore di n dimensione. Ad esempio:
V=(3,9,2)
e' un vettore a tre dimensioni.
L'ordine e' importante, infatti:
W=(3,2,9)
e' ancora un vettore a 3 dimensioni ma diverso da V.
I vettori possono essere memorizzati usando le variabili con indice. Queste variabili vengono usate allo stesso modo in cui venivano usate le variabili ad esempio X oppure I% oppure ancora A$.
In esse sara' memorizzato un qualsiasi valore si voglia oppure esse conteranno lo zero o la stringa nulla se nessun valore sara' stato specificato.
Allo stesso modo di come si fa per i vettori, anche per le variabili con indice si puo' indicare l'esecuzione di un gran numero di operazioni usando una semplice singola annotazione. Questo modo di procedere e' particolarmente comune quando con gli anelli "FOR-NEXT" come mostra il seguente esempio:

Esempio, prodotto scalare.
Il prodotto scalare di due vettori "V" e "W" e' un vettore indicato con V.W le cui componenti sono V(i)xW(i).
Ad esempio nel caso a 4 dimensioni, se:
V=(V1,V2,V3,V4)
e W=(W1,W2,W3,W4)
allora V.W=(V1xW1,V2xW2,V3xW3,V4xW4)
Si supponga ora di avere i due seguenti vettori:
V=(5,6,7,11,4,6,1) W=(9,5,2,1,0,3,2)
Un programma che calcola il prodotto scalare fra i vettori v e w sara' il seguente :

```
10 FORI=1TO7:READ V(I):NEXT
20 FORI=1TO7:READ W(I):NEXT
30 FORI=1TO7:VW(I)=V(I)*W(I):NEXT
40 FORI=1TO7:?VW(I):NEXT
50 DATA 5,6,7,11,4,6,1
60 DATA 9,5,2,1,0,3,2
```

VARIABILE STRINGA CON INDICE
E' stato gia' detto in precedenza che le variabili con indice possono essere:

```
decimali:A(I).
intere:  A%(I)
stringhe:A$(I)
```

L'uso delle variabili stringa con indice e' estremamente usuale come mostra il seguente programma che stampa un istogramma del prodotto lordo nazionale degli Stati Uniti dal 1966 al 1974.

Il list del programma e' il seguente:

```
10 SP$="":FORI=1TO40:SP$=SP$+" ":NEXT
20 A$(1)="■"+"▌ ":A$(2)="■"+"▌ ":A$(3)="■"+"▌":A$(4)="■"+"▌ ":A$(5)=" ▌"
21 A$(6)=" ▌":A$(7)=" ▌"
30 FORI=0TO8:READV(I):NEXT
40 PRINT"□"SPC(8)"GROSS NATIONAL PRODUCT"
50 PRINTSPC(12)"(IN $ BILLIONS)■"
100 FORI=0TO8
110 X=V(I)/50:Y=INT(X)
120 PRINT"■"STR$(66+I)" $"STR$(V(I));
130 PRINT"■"LEFT$(SP$,Y-9)A$(8*(X-Y))"■"
140 NEXT
200 DATA 753,796,869,936,982,1063,1171,1307,1413
READY.
```

Le variabili con indice V(0),V(1),.....V(8) sono il prodotto nazionale lordo per ciascuno dei 9 anni. Le stringhe con indice A$(0),A$(1),....A$(7) permettono di ottenere un grafico con la risoluzione illustrata nella seguente figura:

string	prints	ASC
A$(0)	null(by default)	
A$(1)	▌	165
A$(2)	▉	180
A$(3)	▉	181
A$(4)	▉	161
A$(5)	▉	182 (R)
A$(6)	▉	170 (R)
A$(7)	▉	167 (R)

L'INTESTAZIONE

GROSS NATIONAL PRODUCT
(IN $BILLIONS)

viene stampata dalle istruzioni che si trovano fra la linea 40 e 50 e quindi l'anello FOR-NEXT racchiuso fra le linee 100 e 140 stampa le 8 barre. La linea 120 stampa ciascuna barra mentre la linea 130 porta di una riga in alto il cursore e quindi stampa il valore dell'anno, la funzione STR$(66+I) e il prodotto nazionale lordo seguito dalla funzione STR$(V(I)).
Ciascuna barra e' realizzata con Y spazi bianchi in campo inverso e la stringa A$(8*(X-Y)). Il valore di Y e' determinato dalla formula:

```
Y=INT(V(I)/45)
Y=INT(GNP/45)
```

Il valore di 45, qui usato, e' semplicemente un fattore di scala. Le proporzioni delle barre rimangono le stesse quando vengono usati valori diversi da 45.
Il dimensionamento esatto nella lunghezza della barra viene realizzato usando la variabile stringa con indice:

```
A$(8(X-Y))
```

L'espressione 8(X-Y) puo' avere un valore compreso fra il valore decimale 0 e il valore 7.99...9 ma A$ automaticamente tronca la parte decimale.

ISTRUZIONI DI DIMENSIONAMENTO

Quando si usa una variabile che abbia piu' di 10 elementi si deve usare un'istruzione di dimensionamento. Essa ha la forma "DIM A$(K)", dove K e' il piu' grande valore che l'indice di A$ puo' assumere nel programma che si sta scrivendo. Quando una variabile viene ridimensionata senza che ci sia stata un'istruzione di 'CLR' oppure quando un'istruzione di dimensionamento e' posteriore all'uso della variabile cui e' riferita si ha un errore di ?REDIM'D ARRAY ERROR.
L'istruzione di dimensionamento riserva in memoria, lo spazio per il numero di elementi specificato comprendendo fra questi anche l'elemento il cui indice e' 0.E' quindi buona norma nello scrivere i programmi di iniziare ad usare l'indice da 0 e non da 1.

Poiche' in memoria le variabili sono divise fra variabili semplici e matrici; l'inserzione di una variabile semplice e' un po' piu' complicata quando e' gia' stata definita una matrice. Per prima cosa l'intera matrice viene spostata verso l'alto della memoria di 7 bytes e i puntatori vengono aumentati di 7. Infine la variabile semplice puo' essere inserita alla fine della zona di memoria delle variabili semplici.

SE E' STATA DEFINITA UNA MATRICE DI GROSSE DIMENSIONI E INIZIALIZZATA PRIMA CHE VENGA SEGNATA UNA VARIABILE SEMLICE, SI PUO' PERDERE UN NOTEVOLE TEMPO DI ESECUZIONE PER MUOVERE LA MATRICE OGNI VOLTA CHE LA VARIABILE SEMPLICE VIENE DEFINITA. LA MIGLIOR STRATEGIA DA SEGUIRE IN QUESTO CASO E' ASSEGNARE UN VALORE A TUTTE LE VARIABILI SEMPLICI SCONOSCIUTE, PRIMA DI ASSEGNARE LA MATRICE.QUESTO FATTO PUO' OTTIMIZZARE LA VELOCITA' D'ESECUZIONE.

Nota, effetto delle istruzioni NEW e CLR sui puntatori dei dati:

ISTRUZIONE CLR

Il puntatore delle stringhe viene posto usuale al valore piu' alto di memoria.Il puntatore dei dati viene posizionata all'inizio delle variabili. La fine delle variabili semplici viene posizionata all'inizio delle variabili.

ISTRUZIONE NEW

Il puntatore delle stringhe viene posto usuale alla posizione piu' alta della memoria. Il puntatore dei dati viene posto usuale alla linea di testo -1.La fine della tavola delle variabili viene posto usuale all'inizio della linea di testo +3.L'inizio delle variabili viene posto usuale alla linea di testo+3.

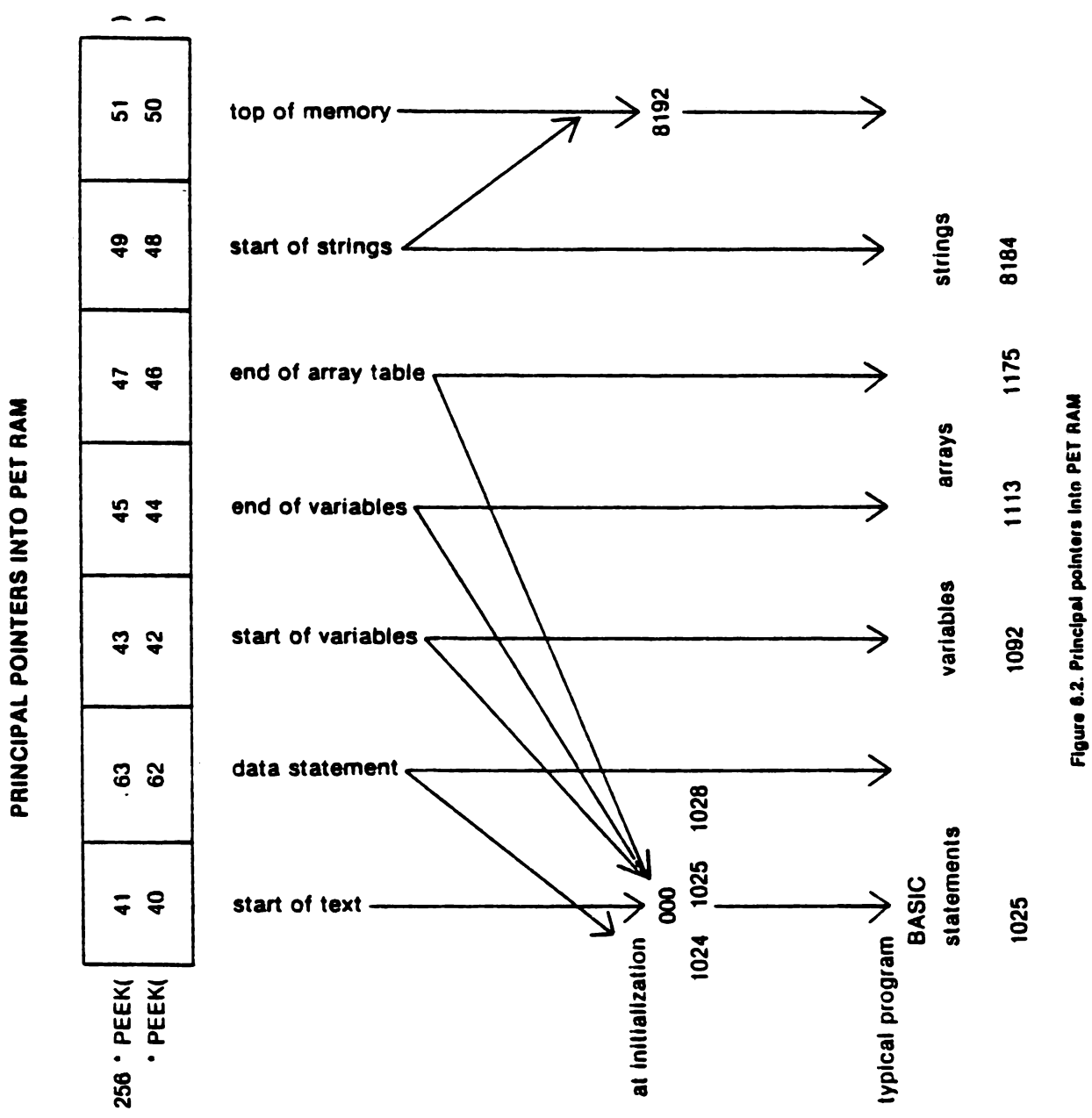


Figure 6.2. Principal pointers into PET RAM

APITOLO 7

NTERFACCIA E LINEE DEL PET

Come indicato in figura7.1,vi sono 4 connettori accessibili ttraverso finestre sul retro e sul lato del PET che permettono l'utente di interfacciare il calcolatore con dispositivi sterni.I connettori usati, come anche mostrati in figura 7.2, ono normali connettori per circuito stampato e vengono collegati ul bordo della sheda su cui e' assemblato il PET stesso. Vi sono ontatti utili su ambedue le facce del circuito mpato.L'dentificazione dei contatti per i connettori J1, J2 e' llustrata anch'essa nella figura 7.2

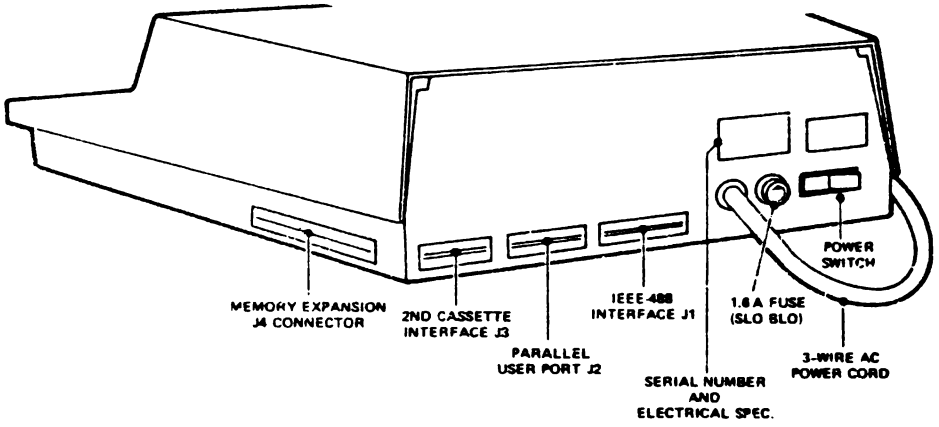


Figure 7.1. Simplified view of CBM showing switch, fuse, line cord and interfacing connectors.

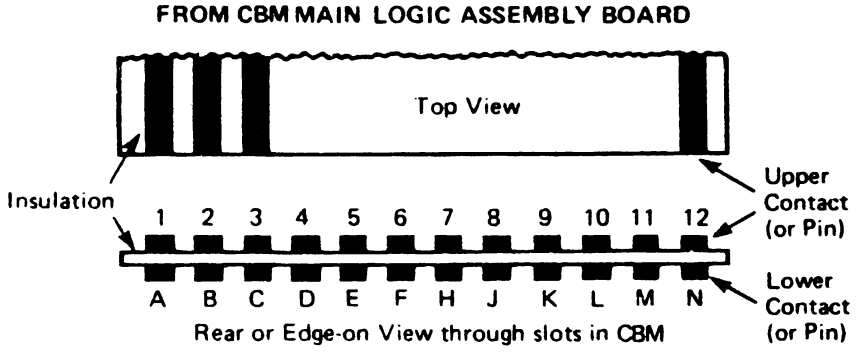


Figure 7.2. Simplified views of edge connectors J1 and J2 to illustrate contact identification convention.

INTERFACCE IEEE-488 (Connettore J1)

Non viene usato il connettore standard IEEE-488 sul PET. Al suo posto viene adoperato un normale connettore per circuiti stampati a 12 posizioni e 24 contatti con spaziatura frai contatti di 0,156 pollici, questo fatto per mettere compatibilita' con tutte le altre connessioni al PET

Degli intagli sono posizionati tra i piedini 2-3 e 9-10, per impedire l'inversione del connettore.
In tavola 7.3 e' riportata l'identificazione dei contatti del PET, le connessioni per un connettore standard IEEE, le nomenclature IEEE le definizioni dei segnali.
Le possibilita' di carico e le impedenze di linea per le connessioni sono quelle standard delle specifiche IEEE-488.

CBM Pin Characters	Standard IEEE Connector Pin Numbers	IEEE Signal Mnemonic	Signal Definition/Label
Upper Pins			
1	1	DI01	Data input/output line #1
2	2	DI02	Data input/output line #2
3	3	DI03	Data input/output line #3
4	4	DI04	Data input/output line #4
5	5	EOI	End or identify
6	6	DAV	Data valid
7	7	NRFD	Not ready for data
8	8	NDAC	Data not accepted
9	9	IFC	Interface clear
10	10	SRQ	Service request
11	11	ATN	Attention
12	12	GND	Chassis ground and IEEE cable shield drain wire
Lower Pins			
A	13	DI05	Data input/output line #5
B	14	DI06	Data input/output line #6
C	15	DI07	Data input/output line #7
D	16	DI08	Data input/output line #8
E	17	REN	Remote enable
F	18	GND	DAV ground
Lower Pins			
H	19	GND	NRFD ground
J	20	GND	NDAC ground
K	21	GND	IFC ground
L	22	GND	SRQ ground
M	23	GND	ATN ground
N	24	GND	Data ground (DI01-8)

Table 7.3. CBM contact identification characters. IEEE-488 identification characters, associated labels and descriptions.

CONNETTORI PER L'INTERFACCE IEEE

Nella tabella 7.4 e' riportata una lista dei produttori dei connettori piu' frequentemente usati per la connessione con il PET.

Manufacturer	Part Number
Cinch	251-12-90-160
Sylvania	6AG01-12-1A1-01
Amp	530657-3
Amp	530658-3
Amp	530654-3

Table 7.4. Receptacles recommended for CBM IEEE-488 connectors or parallel user port.

CONNETTORI IEEE-488

I connettori standard IEEE-488 non sono direttamente collegabili al PET; alcuni di tali connettori sono riportati in tabella 7.5 con il nome del relativo fornitore.

Connector Manufacturer	Identifier	Description
Cinch	5710240	Solder-plug
Cinch	5720240	Solder-receptacle
Amp	552301-1	Insulation displacement plug
Amp	552305-1	Insulation displacement receptacle

La COMMODORE ha in vendita cavi di 1 metro di lunghezza che recano ad una estremita' un connettore duale IEEE-488 e all'altra estremita' un connettore per circuito stampato. Informatevi presso il vostro venditore o alla COMMODORE per i prezzi e i termini di consegna.

PORTE DI UTENTE PARALLELE

Le linee di questa interfaccia sono portate fuori dal PET attraverso un pettine realizzato sul circuito stampato a 12 posizioni e 24 contatti, con una spaziatura di 0,156 pollici fra i contatti.
Nella tavola 7.4 sono riportati i connettori che possono venire utilizzati per effettuare il collegamento.
Gli intagli sono posizionati fra i piedini 1-2 e i piedini 10-11.
La tavola 3.1 illustra l'identificazione dei piedini, le corrispondenti etichette e la loro descrizione.
Si noti che le connessioni 1-12, cioe' la linea superiore dei contatti, (vedere figura 7.6) e' utlizzata per essere usata dal servizio di assistenza PET.
Quando si usano i programmi diagnostici incorporati in ROM , viene usato uno speciale connettore esso collega alcuni dei contatti superiori con i contatti inferiori. E' quindi bene sapere che la parte superiore del connettore nei piedini da 1 a 12 deve essere usata solamente con estrema precauzione.

Pin Identification Character	Signal Label	Signal Description
1	Ground	Digital ground.
2	T.V. Video	Video output used for external display, used in diagnostic routine for verifying the video circuit to the display board.
3	IEEE-SRQ	Direct connection to the SRQ signal on the IEEE-488 port. It is used in verifying operation of the SRQ in the diagnostic routine.
4	IEEE-EOI	Direct connection to the EOI signal on the IEEE-488 port. It is used in verifying operation of the EOI in the diagnostic routine.
5	Diagnostic Sense	When this pin is held low during power up the CBM software jumps to the diagnostic routine, rather than the BASIC routine.

Table 7.6. Parallel user port information.
CBM pin identification characters, the corresponding signal labels and their descriptions.

Table 7.6. Parallel user port information (continued).

Pin Identification Character	Signal Label	Signal Description
6	Tape #1 READ	Used with the diagnostic routine to verify cassette tape #1 read function.
7	Tape #2 READ	Used with the diagnostic routine to verify cassette tape #2 read function.
8	Tape Write	Used with the diagnostic routine to verify operation of the WRITE function of both cassette ports.
9	T.V. Vertical	T.V. vertical sync signal verified in diagnostic. May be used for external TV display.
10	T.V. Horizontal	T.V. horizontal signal verified in diagnostic may be used for TV display.
11, 12	GND	Digital ground.
A	GND	Digital ground.
B	CA1	Standard edge sensitive input of 6522VIA.
C	PA0	Input/output lines to peripherals, and can be programmed independently of each other for input or output.
D	PA1	
E	PA2	
F	PA3	
H	PA4	
J	PA5	
K	PA6	
L	PA7	
M	CB2	Special I/O pin of VIA.
N	GND	Digital ground.

VERSATILE INTERFACE ADAPTER

Le linee che si trovano sul lato inferiore del connettore delle porte d'utente provengono da un versatile Interface Adapter (via MOS Tecnology part. N 6522).

I segnali CA1, PA0-7, e CB2, sono direttamente connessi a una VIA 6522 standard posizionata all'indirizzo esadecimale E840(corrispondente all'indirizzo decimale 59456).

La porta parallela consiste di 8 linee bidirezionali di I/O da PA0 a PA7, di una linea di handshake per il controllo di queste 8 linee, di nome CA1, che puo' essere anche usata per altri ingressi sensibili al fronte e in fine da un collegamento di grande potenza, CB2. Esso ha tutte le possibilita' di CA1, ma puo' anche essere usato come ingresso o uscita dello shift resister della VIA.

Una descrizione dettagliata della via e' riportata nella tabella 7.7. Tutti i segnali della via che non sono connessi alle porte d'utente vengono utilizzati dal PET per controllo interno. Si noti che l'utente deve evitare di utilizzare questi segnali.

Table 7.7 shows the decimal and hexadecimal addresses in the CBM associated with the VIA.

Decimal	Hexa-Decimal	$E840+	Addressed Location
59456	E840	0000	Output register for I/O port B.
59457	E841	0001	Output register for I/O port A with handshaking.
59458	E842	0010	I/O Port B Data Direction register.
59459	E843	0011	I/O Port A Data Direction register.
59460	E844	0100	Read Timer 1 Counter low order byte Write to Timer 1 Latch low order byte.
59461	E845	0101	Read Timer 1 Counter high order byte. Write to Timer 1 Latch high order byte and initiate count.
59462	E846	0110	Access Timer 1 Latch low order byte.
59463	E847	0111	Access Timer 1 Latch high order byte.
59464	E848	1000	Read low order byte of Timer 2 and reset Counter interrupt. Write to low order byte of Timer 2 but do not reset interrupt.
59465	E849	1001	Access high order byte of Timer 2; reset Counter interrupt on write.
59466	E84A	1010	Serial I/O Shift register.
59467	E84B	1011	Auxiliary Control register.
59468	E84C	1100	Peripheral Control register.
59469	E84D	1101	Interrupt Flag register (IFR).
59470	E84E	1110	Interrupt Enable register.
59471	E84F	1111	Output register for I/O Port A, without handshaking.

Table 7.7. VIA 6522 Decimal and Hexadecimal addresses in CBM.

LA PROGRAMMAZIONE DELLE PORTE DI UTENTE

Le linee dei dati da PA0 a PA7 vengono programmate individualmente per funzionare come ingresso o uscita. Questa operazione puo' essere fatta usando un comando software POKE 59459 in modo da caricare un numero nel registro di direzione dei dati. La tabella 7.8 illustra un pratico esempio di selezione ingresso-uscita. Questa programmazione deve essere fatta all'inizio una volta per tutte. Dopo di cio' un'istruzione POKE 59471 viene usata per pilotare i piedini programmati come uscita, mentre l'istruzione PEEK 59471 viene usata per leggere gli ingressi.

Command Statement	Binary Representation	Lines	Mode
POKE 59459,255	11111111	PA0-7	Output
POKE 59459,0	00000000	PA0-7	Input
POKE 59459,240	11110000	PA0-3	Input
		PA4-7	Output

Table 7.8. Parallel user port example.
Programming of lines PA0-7 for input/output operation.

INTERFACCIA PER LA SECONDA CASSETTA
(Connettore J3)

Questa interfaccia viene portata all'esterno attraverso un connettore a pettine di 6 posizioni e 12 contatti realizzato sul circuito stampato del PET.La spaziatura fra i contatti e' di 0,156 pollici fra connettore e contatto.(Vedere la figura 7.9) Un intaglio e' posizionato fra i piedini 2-3 per evitare l'inversione del connettore. Qusta porta e' riservata all'uso di una seconda casseta a nastro magnetico. Qualsiasi altra connessione viene fatta a rischio dell'utente. Si noti che i 5 volt non possono essere usati all'esterno come alimentazione.In tabella 7.10 sono riportati i caratteri d'identificazione dei piedini, i nomi dei segnali corrispondenti e la descrizione del loro significato. La tabella 7.11 indica viceversa i costruttori e numero di catalogo di connettori adatti alla connessione della seconda cassetta.

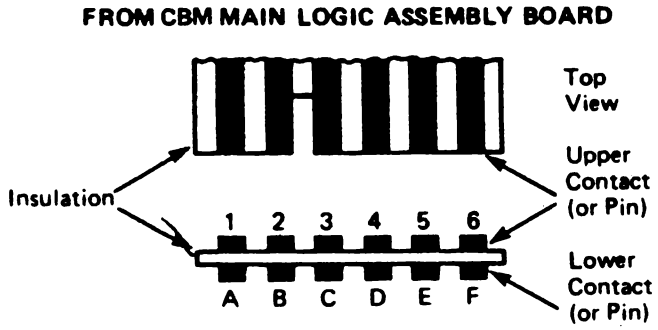


Figure 7.9. Simplified view of edge connector J3 with contact identification.

Note A-1, B-2, etc., imply a pin A to pin1, pin B to pin 2, connection.
In some special units, pins 1 through 6 were not connected.

Pin Identification Characters	Label	Description
A-1	GND	Digital ground.
B-2	+5	Positive 5 volts to operate cassette circuitry only.
C-3	Motor	Computer controlled positive 6 volts for cassette motor.
D-4	Read	Read line from cassette.
E-5	Write	Write line to cassette.
F-6	Sense	Monitors closure of mechanical switch on cassette when any button is pressed.

Table 7.10. Second cassette interface port.
CBM pin identification characters, labels and associated descriptions.

CONNETTORE PER L'ESPANSIONE DELLA MEMORIA (Connettore J4-J9)

Il connettore per l'espansione della memoria permette l'accesso alle linee di ingresso-uscita bafferizzate e decodificate che provengono dal microprocessore 6502.La figura 7.12 illustra uno schema semplificato del connettore a 80 posizioni usato. Lo spazio tra i contatti e' di 0,1 pollice.Si noti che 40 connessioni "B" sul lato superiore sono i ritorni di terra per le corrispondenti 40 connessioni "A" sul lato inferiore. La scheda stampata su cui e' montato il PET e' stata progettata per poter essere collegata a una scheda figlia per attraverso i conettori di espansione di memoria J4 eJ9 e i connettori di espansione dell'alimentazione J10 e J11.La tabella 7.12 illustra i connettori che l'utente puo' usare per effettuare la connessione.Tutti i connettori hanno i piedini spaziati di 0,1 pollice. J4 eJ5 hanno una configurazione 2x25 mentre J10 e J11 cono connettori 2x7. La tabella 7.13A illustra le connessioni per l'alimentazione,mentre la tabella 7.13B illustra le connessioni per l'espansione di memoria.La tabella 7.13C illustra i numeri di piedino usati nel PET,i nomi di segnali e la loro spiegazione.

Table 7.11. A selection of suitable receptacles for connecting with the CBM second cassette edge connector J3.

Manufacturer	Identifier
Sylvania	6AJØ7-6-1A1-Ø1
Viking	2KH6/1AB5
Viking	2KH6/9AB5
Viking	2KH6/21AB5
Amp	53Ø692-1
Sullins	ESM6-SREH
Cinch	25Ø-Ø6-9Ø-17Ø

Manufacturer	contact grid	identifier
Spectra-strip	2x7	8Ø2-1Ø4
Spectra-strip	2x7	8Ø2-114
Spectra-strip	2x25	8Ø2-Ø5Ø
Spectra-strip	2x25	8Ø2-15Ø
Circuit-Assembly	2x7	CA-14-IDSC
Circuit-Assembly	2x25	CA-50-IDSC

Table 7.12. A selection of suitable receptacles for connecting with CBM daughter board pin connectors J4, J9, J1Ø, and J11

Daughter board power connections table 7.13A

J1Ø (● ● ○ ● ● ● ● 1 2 3 4 5 6 7)

pin #	function
1	-5V Raw power
2	-5V Raw power
3	key
4	+12V Raw power
5	+12V Raw power
6	Ground
7	Ground

J11 (● ○ ○ ● ● ● ● 1 2 3 4 5 6 7)

pin #	function
1	+9 unregulated
2	key
3	key
4	+9 unregulated
5	ground
6	+9 unregulated
7	Ground

Memory expansion bus table 7.13B

Side A ●
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

Side B ●

J9

pin #	function	pin #	function
Side A1	ground	14	BA12
2	BAØ	15	BA13
3	BA1	16	BA14
4	BA2	17	BA15
5	BA3	18	SYNC
6	BA4	19	$\overline{IRQ}$
7	BA5	20	Memory Management
8	BA6	21	BØ2
9	BA7	22	BR/W
10	BA8	23	$\overline{BR/W}$
11	BA9	24	$\overline{DMA}$
12	BA1Ø	25	ground
13	BA11	Side B1-25	ground

J4

pin #	function	pin #	function
Side A1	ground	14	$\overline{SEL\ 6}$
2	BDØ	15	$\overline{SEL\ 7}$
3	BD1	16	$\overline{SEL\ 8}$
4	BD2	17	$\overline{SEL\ 9}$
5	BD3	18	$\overline{SEL\ A}$
6	BD4	19	$\overline{SEL\ B}$
7	BD5	20	$\overline{CAS}$
8	BD6	21	$\overline{RAS}$
9	BD7	22	$\overline{RES}$
10	$\overline{SEL\ 2}$	23	$\overline{RDY}$
11	$\overline{SEL\ 3}$	24	$\overline{NMI}$
12	$\overline{SEL\ 4}$	25	ground
13	$\overline{SEL\ 5}$	Side B1-25	ground

Table 7.13C shows the PET pin numbers, line labels and line descriptions.

Connector Pin Numbers	Line Labels	Line Description
J9-1	GND	Logic Ground
J9-2	BAØ	Address bit Ø, used for memory expansion. Buffered.
J9-3	BA1	Address bit 1, used for memory expansion. Buffered.
J9-4	BA2	Address bit 2, used for memory expansion. Buffered.
J9-5	BA3	Address bit 3, used for memory expansion. Buffered.
J9-6	BA4	Address bit 4, used for memory expansion. Buffered.
J9-7	BA5	Address bit 5, used for memory expansion. Buffered.
J9-8	BA6	Address bit 6, used for memory expansion. Buffered.
J9-9	BA7	Address bit 7, used for memory expansion. Buffered.
J9-25	GND	Logic Ground.
J9-10	BA8	Address bit 8, used for memory expansion. Buffered.
J9-11	BA9	Address bit 9, used for memory expansion. Buffered.
J9-12	BA10	Address bit 1Ø, used for memory expansion. Buffered.
J9-13	BA11	Address bit 11, used for memory expansion. Buffered.
J9-14	BA12	Address bit 12, used for memory expansion. Buffered.
J9-15	BA13	Address bit 13, used for memory expansion. Buffered.
J9-16	BA14	Address bit 14, used for memory expansion. Buffered.
J9-17	BA15	Address bit 15, used for memory expansion. Buffered.
J9-19	$\overline{\text{IRQ}}$	Interrupt request line to the microprocessor.
J9-21	$\overline{\text{BØ2}}$	Buffered phase 2 clock.
J9-22	BR/W	Buffered read/write from 6502 microprocessor.
J4-10	$\overline{\text{SEL 2}}$	4K byte page address select for memory locations 2ØØØ-2FFF.
J4-11	$\overline{\text{SEL 3}}$	4K byte page address select for memory locations 3ØØØ-3FFF.
J4-12	$\overline{\text{SEL 4}}$	4K byte page address select for memory locations 4ØØØ-4FFF.
J4-13	$\overline{\text{SEL 5}}$	4K byte page address select for memory locations 5ØØØ-5FFF.
J4-14	$\overline{\text{SEL 6}}$	4K byte page address select for memory locations 6ØØØ-6FFF.
J4-15	$\overline{\text{SEL 7}}$	4K byte page address select for memory locations 7ØØØ-7FFF.
J4-16	$\overline{\text{SEL 8}}$	4K byte page address select for memory locations 8ØØØ-8FFF.
J4-17	$\overline{\text{SEL 9}}$	4K byte page address select for memory locations 9ØØØ-9FFF.
J4-18	$\overline{\text{SEL A}}$	4K byte page address select for memory locations AØØØ-AFFF.
J4-19	$\overline{\text{SEL B}}$	4K byte page address select for memory locations BØØØ-Bfff.
J4-22	$\overline{\text{RES}}$	Reset for 6502 microprocessor. Note: connected to 74LS00 output.
J4-23	$\overline{\text{RDY}}$	Ready line to the microprocessor.
J4-24	$\overline{\text{NMI}}$	Non maskable interrupt to microprocessor.
J9-1	GND	Logic ground.
J4-2	BDØ	Data bit Ø. Buffered.
J4-3	BD1	Data bit 1. Buffered.
J4-4	BD2	Data bit 2. Buffered.
J4-5	BD3	Data bit 3. Buffered.
J4-6	BD4	Data bit 4. Buffered.
J4-7	BD5	Data bit 5. Buffered.
J4-8	BD6	Data bit 6. Buffered.
J4-9	BD7	Data bit 7. Buffered.
J4-20	$\overline{\text{RAS}}$	Dynamic RAM RAS.
J4-21	$\overline{\text{CAS}}$	Dynamic RAM CAS.
J4-25	GND	Logic Ground.

COMANDI BASIC ADDIZIONALI

In questo momento il lettore del manuale si sara' probabilmente famigliarizzato con l'uso dei comandi INPUT e PRINT.Il comando INPUT permette l'introduzione di dati nel calcolatore. Il comando PRINT permette di visualizzare sullo schermo i dati. Questi comandi sono comuni a qualsiasi BASIC di qualsiasi calcolatore. Per aumentare la flessibilita' del calcolatore PET sono stati aggiunti diversi altri comandi ai classici comandi BASIC. I prodotti futuri BASIC ne terranno conto e si avvantaggeranno di questo fatto incrementando le possibilita' operative. In generale,aumenta la flessibilita' di scambio tra il PET e i dispositivi periferici, grazie all'uso di questi comandi extra. Per comunicare con qualsiasi dispositivo, si puo' usare una combinazione dei seguenti comandi addizionali:

a)OPEN/CLOSE apre e chiude un file logico

b)PRINT# scrive un dato dal PET sul dispositivo di INPUT/OUPUT

c)CMD ha lo stesso significato del Print ma lascia il dispositivo IEEE attivo in ascolto sul bus dopo l'esecuzione del comando

d)INPUT# Legge un dato dal dispositivo I/O e lo trasferisce al PET

e)GET# Legge un carattere dal dispositivo I/O e lo trasmette al PET

PARAMETRO DEI COMANDI DI INGRESSO/USCITA

Per poter usare i comandi addizionali, appena decritti,e' necessario prendere in considerazione 4 parametri.:

a) Numero logico del file (LF)
b) Numero del dispositivo (D)
c) Indirizzo secondario (SA)
d) Nome del file (FN)

Questi parametri, ad esempio, possono apparire quando si usa il comando OPEN# all'interno di un'istruzione:

OPEN# LF,D,SA,FN

FILE LOGICO

I file vengono usati per memorizzare e per rileggere i dati,come ad esempio, nel caso di file su nastro magnetico su disco. Una conveniente estensione di questa idea e' guardare qualsiasi dispositivo che possa ricevere, e,o generare dei dati come un file logico. Si immagini,ad esempio, che un volmetro digitale esterno sia collegato in modo da trasmettere su richiesta, attraverso l'IEEE bus al PET le letture di tensione. Nello stendere il programma per il volmetro, il programmatore dovra' aprire un file e assegnare un numero di file logico per riferirsi al volmetro.Dopo che questa operazione sara' stata fatta, il PET potra' usare un comando "READ",nella fattispecie INPUT che usera' il numero di file logico appena definito per individuare il volmetro.Quando poi l'acquisizione dei dati del volmetro sara' finita, il file logico dovra' essere chiuso. Piu' generalmente i vantaggi offerti dall'uso dei file logici sono:

a) qualsiasi combinazione di numero di dispositivo e di indirizzo secondario puo' essere associata con un unico numero di file logico all'interno di un programma.
b) file diversi possono essere associati allo stesso dispositivo per mezzo di numeri di file logico distinti.

Questo approcio e' stato usato nello sviluppo del nuovo sistema di memorizzazione su disco per il PET.
c) Dopo che il numero di file logico e' stato definito in una istruzione OPEN ,in un programma solamente questo numero viene usato nelle seguneti istruzioni di INPUT/OUPUT.Cio' elimina la necessita' di ulteriori ridefinizioni del numero di dispoditivo dell'indirizzo secondario,quando usato, e del nome del file, quando usato.
Sebbene da un punto di vista logico sia possibile pensare di usare un numero di file logici, grande quanto si vuole, durante l'esecuzione di un programma; il massimo numero di file che puo' essere controllato dal PET, allo stesso tenpo, e' 10.Il numero di file logico deve essere un intero compreso fra 1 e 255.

NUMERO DEL DISPOSITIVO

Tutti i dispositivi con cui il PET comunica vengono identificati attraverso un numero. I primi 4 sono riservati alla seguente periferica:

0 la tastiera

1 nastro magnetico a cassetta montato su pannello,quando non specificato, viene usata questa cassetta magnetica.

2 cassetta magnetica addizionale

3 schermo video

Tutti gli altri dispositivi devono essere dispositivi IEEE, il controllo viene trasferito ad essi con un numero compreso fra 4 e 30.Eccetto in casi speciali un numero ben preciso deve essere associato a ciascun dispositivo periferico per permettere al PET di comunicare con un particolare dispositivo usando il bus parallelo IEEE-488. Su parecchi dispositivi IEEE la codifica di questo numero viene fatta per mezzo di interruttori, o in caso di prodotti piu'economici, per mezzo di connessioni saldate.

INDIRIZZO SECONDARIO

Il concetto d'indirizzo secondario puo' essere nuovo per chi non ha mai lavorato con un bus IEEE. L'uso di un indirizzo secondario permette di realizzare periferie intelligenti che possono funzionare in modi diversi. Ad esempio nella stampante PET si hanno i seguenti 6 indirizzi secondari:

0 stampante normale.Viene preso sempre questo indirizzo secondario se nell'OPEN questo non viene specificato.

1 stampa sotto controllo di un'istruzione di format

3 posizionamento del numero di linee da stampare per pagina

4 attiva l'uso di messaggi diagnostici estesi

5 byte di dati per caratteri programmabili.

In definitiva, cambiando l'indirizzo secondario usato per comunicare con un dato dispositivo fisico, le sue caratteristiche operative possono essere totalmente cambiate, quando lo si desideri.Parecchi dispositivi IEEE hanno le loro particolari convenzioni sugli indirizzi secondari che devono essere seguite. Le relative specifiche possono essere ottenute consultando il manuale di quel particolare dispositivo.
Le cassette magnetiche PET hanno uno speciale insieme di regole per l'indirizzo secondario:

0 in tal caso il nastro e' usato per le operazioni di lettura.Se non viene specificato l'indirizzo secondario,esso viene posto sempre pari a 0.

1 Il nastro viene usato per le opeazionidi scrittura.

2 il nastro viene usato per le operazioni di scrittura forzando sul nastro un contrassegno di fine nastro,quando il file viene chiuso.

Gli indirizzi secondari possono avere un valore fra 0 e 31.

NOME DEI FILE

Nei dispositivi ad accesso casuale in cui possono essere memorizzati piu' file, per identificare uno di questi file e' obbligatorio l'uso di un nome.Nel caso dei nastri, il nome del file e' desiderabile, anche se si ha un solo file sul nastro,poiche' cio' facilita l'identificazione dei nastri.Per le due cassette magnetiche del PET, un nome dei file, puo' essere qualsiasi combinazione fino a 128 caratteri.Quando il nome del file viene testato, esso viene analizzato in ordine di caratteri crescenti.Si supponga che in fase di scrittura di un file sia stato assegnato ad esso il nome di 8 caratteri,COUNTING.Se si vuole, il file puo' essere ricercato attraverso il nome abbreviato come ad esempio,COU. In tal caso verra' preso e aperto il primo file nella cui intestazione si trovino questi tre caratteri consecutivi.Tuttavia questo modo di operare potrebbe far prendere dei file indesiderati il cui nome sia ad esempio:COUNT oppure COUNTRY, piuttosto che COUNTING. E' quindi opportuno specificare il nome del file in modo completo per evitare errori.Per gli altri dispositivi che usano nomi per il file, e' bene consultare le istruzioni del dispositivo per accertare le specifiche per l'uso dei nomi di file.

OPERAZIONI SUI FILE DELLE CASSETTE SU NASTRO MAGNETICO

Il PET dedica una speciale attenzione alle due cassette nastro-magnetiche che puo' gestire. Le unita' nastro sono modificate in modo tale che il PET abbia il controllo sui movimenti del motore. Il calcolatore puo' anche avvertire se i tasti di PLAY,REWIND e FAST FORWARD sono premuti; cio' viene fatto per mezzo di un singolo interruttore montato sull'unita' nastro.Si noti che lo stesso interruttore e' usato per testare l'azionamento di tutti e tre i nastri; se qualsiasi di questi tre pulsanti e' premuto, il PET lo interpretera' come se fosse stato premuto il tasto PLAY e si comportera' di conseguenza.A causa del tipo di meccanismo che viene usato sull'unita' nastro, l'utente deve fare da solo il riavvolgimento, l'avvolgimento veloce, lo stop,il caricamento e l'espulsione dei nastri.Egli inoltre dovra' azionare la registrazione premendo il bottone di registrazione in coincidenza con il bottone PLAY.Dopo che l'appropriato tasto e' stato premuto, il PET ha un totale controllo sul movimento del nastro. Opportuni messaggi vengono visualizzati sullo schermo per avvertire l'utente di completare la funzione voluta, cioe' di premere il tasto di ascolto o il tasto di registrazione.Inoltre l'utente sara' anche avvertito quando riavvolgere e far avvanzare velocemente il nastro.Le due unita' nastro del PET sono comandate indipendentemente e le varie linee di controllo permettono di leggere dati dalla cassetta 1 oppure leggere dati dalla cassetta 2,controllare il motore della cassetta 1,controllare il motore della cassetta 2, vie e' viceversa un'unica linea di scrittura.

TECNICA PER LA REGISTRAZIONE DEI FILE

La struttura dei dati impiegati nei file su nastro assicura la massima utilizzazione della memoria e la massima affidabilita' della registrazione. Per ottenere cio' il PET registra i dati in due consecutivi blocchi a due frequenze. Tutti i dati sono ripetuti totalmente e per mezzo di un test d'errore incorporato nel PET, e' possibile correggere gli eventuali sbagli di registrazione utilizzando i dati ridondanti che sono memorizzati nel secondo blocco di dati.

Durante la lettura viene usata una speciale tecnica di correlazione della velocita', per correggere gli errori dovuti al fatto che le unita' a nastro registrano a velocita' variabile e per compensare la normale resistenza all'avvanzamento, caratteristica dei nastri. Per correggere le caratteristiche di ciascua partenza e arresto sul nastro e sincronizzare correttamente ciascun blocco, un tono singolo viene scritto fra i blocchi. Il segnale e' usato pr sincronizzare sia la posizione che la velocita' del nastro.Questo tono viene registrato per lunghezze variabili all'inizio e tra i blocchi del nastro stesso.Quando si inizia un file, il PET scrive questo tono per circa 10 secondi, in modo da poter riconoscere automaticamente, in lettura, l'intestazione.I blocchi sul nastro sono poi separati da durate di questo tono piu' corte.

INTESTAZIONI DEI FILE

Un'importante caratteristica da soddisfare e' quella di permettere all'utente di registrare un file su nastro.Per facilitare questa operazione e per mettere il test delle etichette di ciascun file,il primo dato registrato fisicamente sul nastro per ogni operazione,e' un'intestazione.L'intestazione e' della stessa identica forma di un normale blocco di dati, eccetto che per il primo carattere di ciascun blocco che contiene un carattere di identificazione, il quale abilita il sistema operativo a distinguere tra blocchi di programmi, blocchi di dati,intestazioni e contrassegni di fine nastro.Il PET e' in grado di registrare nell'intestazione di ciascun file un nome lungo fino a 128 caratteri. Questo nome verra' ricercato e testato nelle varie operazioni di OPEN e CLOSE.

BUFFER SUL NASTRO

Un'altra necessita' e' necessario soddisfare durante le operazini di scrittura sul nastro ed e'di permettere all'utente di dare il comando di scrittura indipendentemente da quanto stia accadendo in quel momento sul nastro stesso. Cio' viene fatto associando permanentemente al nastro un blocco di memoria con le funzioni di buffer di dati.Un buffer di 192 caratteri e' collocato all'indirizzo decimale 634 per la cassetta numero uno, seguito da un buffer di 192 caratteri che inizia all'indirizzo decimale 826 per la cassetta numero due.Le intestazioni dei file vengono scritte dapprima in questo buffer e poi scritte sul nastro.I dati del file vengono accumulati nel buffer del nastro finche' non si superano i 192 caratteri,poi il contenuto viene scritto sul nastro o nel caso di un programma di lettura,il successivo blocco di dati viene letto nel buffer.Le intestazioni dei file e tutti i blocchi di dati sono quindi lunghi 192 caratteri.I buffer del nastro non vengono usati nel caso di file programma, poiche' essi vengono scritti sul nastro direttamente dalle locazioni di memoria in cui il programma risiede.Per poter ricaricare gli origramma nelle opportune posizioni di memoria, nell'intestazione del nastro esistono informazioni sulle locazioni iniziale e finale del programma stsso.Il programma e' scritto sul nastro nell'usuale forma di due consecutivi blocchi ridondanti.

FILE MULTIPLI

Per poter caricare piu' di un file sul nastro, l'utente deve avere la possibilita' di aggiungere file ad un nastro e conoscere quando un nastro e' finito.E',quindi,importante che il sistema sia in grado di registrare un'indicazione di"fine file"(end of file) e di"fine nastro"(end of tipe).

Nel caso di file di dati un contrassegno di "fine file"viene aggiunto immediatamente dopo l'ultimo carattere dei dati. Questo carattere viene reso, durante la lettura, disponibile all'utente attraverso la parola di stato; il contrassegno di "fine file" e' inserito automaticamente quando un file che sta venendo scritto viene chiuso.Quando si ha invece a che fare con file programma, poiche' tutti i dati sono sempre contenuti in un singolo blocco, la fine del blocco significa autoamticamente fine del programma. Per indicare che e' stata raggiunta la fine del nastro viene scritta sul nastro una speciale intestazione.Quando questa intestazione viene incontata durante la ricerca di un file,il PET ferma automaticamente il nastro e indica "file not found" all'utente.Un tipico nastro con piu'file puo',ad esempio,contenere dapprima un file di dati,poi un file programma, seguito da un "end of tape" come illustrato nell'esempio di figura7.14

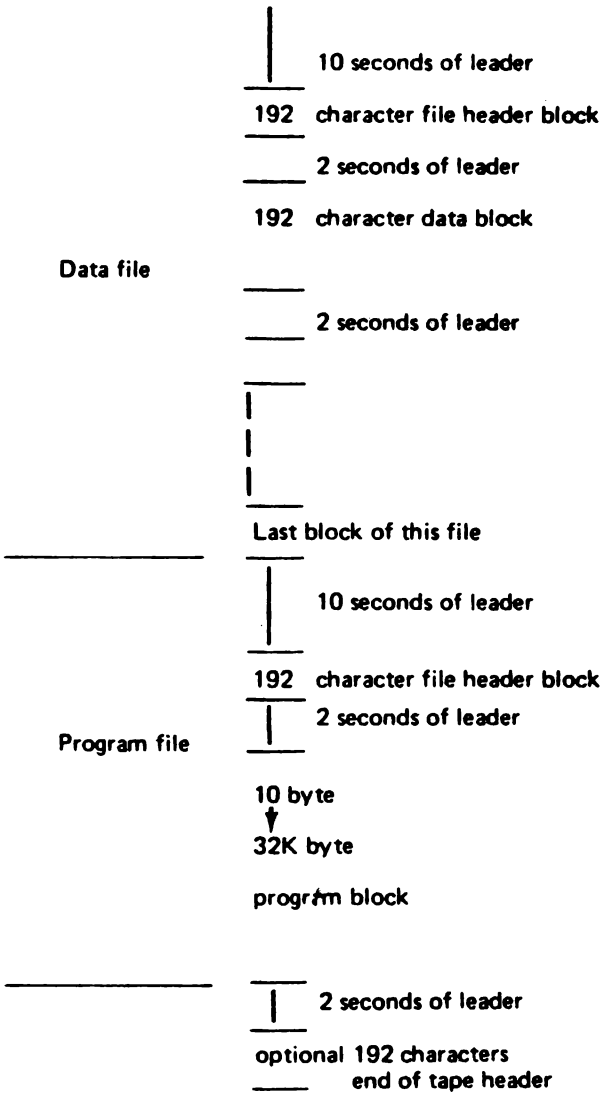


Figure 7.14. An example of multiple file structure.

OPERAZIONI LOGICHE SU FILE DI INGRESSO-USCITA

GENERALITA'

Queste operazioni possono essere suddivise in tre passi:
a)Apertura del file:con questa operazione si forniscono al PET tutte le informazioni che gli sono necessarie per lavorare con i file.
b)Leggere dati dal nastro o scrivere dati sul nastro nell'appropriato file logico.
c)Chiudere il file:permette al PET di rilasciare il periferico e il file logico.

Questi passi saranno discussi in dettaglio nelle pagine seguenti.

APERTURA DEI FILE

Per informare il BASIC su che file esso dovra' lavorare, e' necessario, come prima cosa, aprire il file.Questa operazione viene fatta con la seguente istruzione:

OPEN file logico, dispositivo, indirizzo secondario, nome del file.

Piu' specificatamente l'istruzione consiste del comando OPEN seguito dal numero di file logico, poi dal numero del dispositivo al quale il file e' assegnato; successivamente ancora l'indirizzo secondario se esistente, comunicato durante l'interazione del BASIC con il file e per ultimo il nome del file fisico, se esiste.Questa istruzione e' interpretata dal BASIC, puo' quindi usare dei numero di file logici calcolati, la stessa cosa,ovviamente, si puo' dire anche per il numero di periferico e per l'indirizzo secondario.Questa possibilita' e' estremamente interessante quando si lavora con dispositivi a file multipli come i dischi.
La parola chiave"OPEN" e il numero di file logico sono essenziali per aprire un file; con questa opeazione si assegna infatti al dispositivo di numero che verra' usato nelle operazioni di "read" attraverso l'istruzione "INPUT#"o di "write" attraverso l'istruzione PRINT#. Il numero della periferica e' opzionale; se non viene inserito nell'istruzione, il valore che viene considerato sara' sempre "1".Il nome del file e' opzionale, per le unita' a nastro magnetico,sebbene venga spesso usato: viceversa, le unita' di memorizzazione a disco, il nome e' essenziale.

ESEMPI DI ISTRUZIONI OPEN

Le istruzioni OPEN 1,2,1 viene interpretata dal sistema operativo come segue:

(LF) Il primo numero 1 assegna il numero di file logico informando il calcolatore che e' stato aperto il file logico 1.
(D) Il file logico 1 viene poi assegnato all'unita' a nastro numero 2, attraverso il 2 che apparira' nell'istruzione.
(SA) L'unita' a nastro numero 2 viene poi predisposta per eseguire operazioni di scrittura sul nastro attraverso il secondo 1 dell'istruzione.
(FN) In questo caso non viene assegnato alcun nome di file.
In modo del tutto analogo l'istruzione OPEN3 viene interpretata nel seguente modo:(F)
(LF)Viene aperto il file logico numero 3.
(D) Poiche' non vie e' nessun numero di dispositivo il file logico 3 viene assegnato all'unita' nastro numero 1.
(SA) Poiche' non vie e' nessun indirizzo secondario l'unita' a nastro numero 1 viene predisposta per operazioni di lettura.

(FN) Anche in questo caso non viene assegnato nessun nome di file.
Se il numero dispositivo di una stampante e' "4" l'istruzione OPEN 12,4,1 viene interpretata come segue:
(LF) Viene aperto il file logico numero 12.
(D) Questo file viene assegnato al dispositivo numero 4.
(SA) L'indirizzo secondario fa si che la stampante stampi sotto controllo di un'istruzione di format.
(FN) In questo caso non e' possibile assegnare alcun nome al file.

LOAD

Un caso speciale di comando di apertura e' il LOAD di un file che abbia nome:un'istruzione di LOAD viene fatta con la seguente forma:

LOAD name, device number

Il sistema operativo genera automaticamente un'istruzione di OPEN usando l'indirizzo secondario appropriato per effettuare l'operazione di "LOAD". Sul dispositivo cosi' attivato viene iniziata una ricerca per trovare un programma il cui nome e' quello specificato nell'istruzione LOAD. Dopo che il programma e' stato trovato, esso viene automaticamente letto dal dispositivo e caricato nella memoria partendo dall'indirizzo specificato nell'intestazione del file.Gli errori di lettura che possono verificarsi durante l'esame del primo blocco vengono automaticamente corretti dal secondo blocco.Alla fine del ciclo di caricamento viene eseguita una somma di prove checksum. Se si ha un errore di checksum o se si e' in presenza di un errore che non e' correggibile, il sistema operativo stampa automaticamente ?LOAD ERROR e ferma il caricamento del programma. Se il caricamento del programma viene fatto operando in modo diretto, alla fine del caricamento viene eseguita la funzione di cancellazione, inizializzando tutte le variabili.Se l'operazione di caricamento e' invecie chiamata da un programma, allora il PET tratta questa operazione come un overlay.Il nuovo programma viene caricato nello spazio usato dal precedente programma, ma i valori di tutte le variabili vengono mantenuti usuali a quelli raggiunti nel programma precedente. Questo fatto permette ad un programma di chiamare un altro passandogli tutti i parametri che gli saranno assegnati.L'unico vincolo a quanto appena esposto sta nel fatto che il programma chiamato deve avere diemsioni usuali o minori del programma chiamante.Poiche' il BASIC rimpiazza completamente il programma corrente, non e' possibile avere un singolo programma principale e diverse subroutine in overlay, tuttavia,includendo il programma principale in ciascun overlay, si puo' ottenere lo stesso effetto con una piccola perdita di velocita'.L'uso dei nomi dei file e degli overlay permettono di scrivere dei programmi strutturati di grande ampiezza con complessita' notevole.

OPERAZIONE DI VERIFY

L'istruzione di VERIFY e' un caso speciale di LOAD.Essa dovrebbe venir eseguita dopo aver registrato qualsiasi programma.Il comando di VERIFY fa si che il BASIC esegua le stesse operazioni dell'istruzione LOAD, con la differenza che i dati non vengono caricati in memoria, ma vengono comparati con il contenuto della memoria.Se vengono incontrati degli errori, sia nel primo che nel secondo passo,il PET stampera' sullo schermo ?VERIFY ERROR, e' necessario allora registrare una seconda volta il programma sul nastro in quanto la copia precedente non e' utilizzabile.

Durante l'operazione di VERIFY, la parola di stato ha i seguenti significati:
Codice 4 significa blocco corto
Codice 8 significa blocco lungo
Codice 16 significa errore di checksum su nastro
Codice 32 significa errore di checksum su nastro

ISTRUZIONE SAVE

Anche l'istruzione SAVE provoca un'operazione di apertura e chiusura automatica di un file. L'istruzione SAVE si realizza mediante il seguente formato:

SAVE nome,numero di dispositivo

Se il dispositivo fisico e' uno delle due unita' nastro, il sistema operativo inizia automaticamente a registrare un'intestazione e apre un file su nastro con il nome appropriato.L'intestazione viene scritta inserendo in essa l'inizio e la fine della zona in cui verra' caricato successivamente il programma. Se il dispositivo e' un'unita' IEEE-488 viene inviato uno speciale messaggio di apertura che sta ad indicare che il PET sta inviando un file programma.Immediatamente dopo il programma viene scritto direttamente dalle sue locazioni di memoria o sul nastro o sul bus IEEE-488. Se l'operazione di SAVE avviene su nastro, viene calcolato un checksum e registrato quando il programma e' stato tutto scritto come dato ridondante.Qando la registrazione e' finita, il nastro viene automaticamente fermato e posizionato per il dato successivo.

CARATTERISTICHE SPECIALI IEEE-488

Sul nastro gl'indirizzi di inizio e fine di un programma vengono memorizzati e riletti dall'intestazione del file.Per un uso efficiente dei dispositivi IEEE-488, l'indirizzo di partenza di un programma viene caricato nei primi due bytes di dati con il comando di SAVE e vengono riletti da queste posizioni con un LOAD.

CONSIDERAZIONI SULL'ISTRUZIONE OPEN IN RELAZIONE AI DISPOSITIVI IEEE-488

Quando un comando sceglie un dispositivo che ha un numero pari a 4 o maggiore di 4, il sistema operativo assume che tale dispositivo e' del tipo IEEE-488. Se inoltre l'istruzione OPEN non specifica alcun nome di file, non viene inviato nulla sul bus IEEE-488. Se invecie viene specificato un nome di file, il sistema operativo invia al dispositivo, il cui numero e' quello specificato nell'istruzione OPEN, una sequenza per richiedere il suo uso come ascoltatore con un indirizzo secondario che e'"OR" del numero esadecimale "FO" e dell'indirizzo secondario specificato nell'istruzione OPEN.I periferici forniti dal Commodor, come ad esempio,i sistemi di memoria di massa a disco floppy,useranno questo indirizzo secondario e il nome del file per trasmettere i dati al dispositivo ascoltatore su cui e' stato aperto il file.

MODI D'OPERAZIONE SUI FILE SU NASTRO MAGNETICO

I file su nastro magnetico possono essere aperti per due scopi distinti:
a)per scrivere dal PET sul nastro
b)per leggere dal nastro e inviare i dati al PET

APERTURA DI UN FILE PER SCRIVERE SUL NASTRO DAL PET

Il diagramma di flusso della figura 7.15 mette in evidenza le interazioni fra PET e utente e descrive le funzioni del PET quando viene aperto un file per scrivere su nastro. Il blocco iniziale illustra che vi sono due modi di aprire un file:
a) aprire un file attraverso l'operazione di SAVE per scrivere un programma.
Si noti che se un file su nastro viene aperto direttamente dalla tastiera,allora sullo schermo appare il messaggio WRITING NAME. Se invece il file viene aperto sotto controllo di un programma e i tasti PLAY e RECORD sono premuti, allora non apparira' nessun messaggio sullo schermo. In tale maniera qualsiasi altra scritta che sia stata fatta apparire sullo schermo dal programma corrente non viene disturbata.

OPEN PER LEGGERE DAL NASTRO AL PET

Il diagramma di flusso della figura 7.16 mette in evidenza l'interazione fra il PET e l'utente e descrive le funzioni che il PET svolge quando si apre un file per leggere da nastro.Il blocco iniziale mostra che vi sono due modi di aprire il file:
a)aprire il file per leggere da nastro dei dati.
b)aprire il file attraverso l'istruzione di LOAD per caricare un programma in memoria.
Si noti che se il file viene aperto direttamente dalla tastiera, allora sullo schermo apparira' il messaggio PRESS PLAY,SEARCHING FOR NAME,FOUND NAME.Se viene usata l'istruzione di LOAD, allora il BASIC,inizializza le variabili del programma caricato.Se il fila viene aperto sotto controllo del programma e il bottone di PLAY del registratore e' stato pemuto in un tempo precedente, nessun messaggio apparira' sullo schermo in modo da non disturbare cio' che viene visualizzato dal programma in atto.Non e' necessario inizializzare le variabili del BASIC.

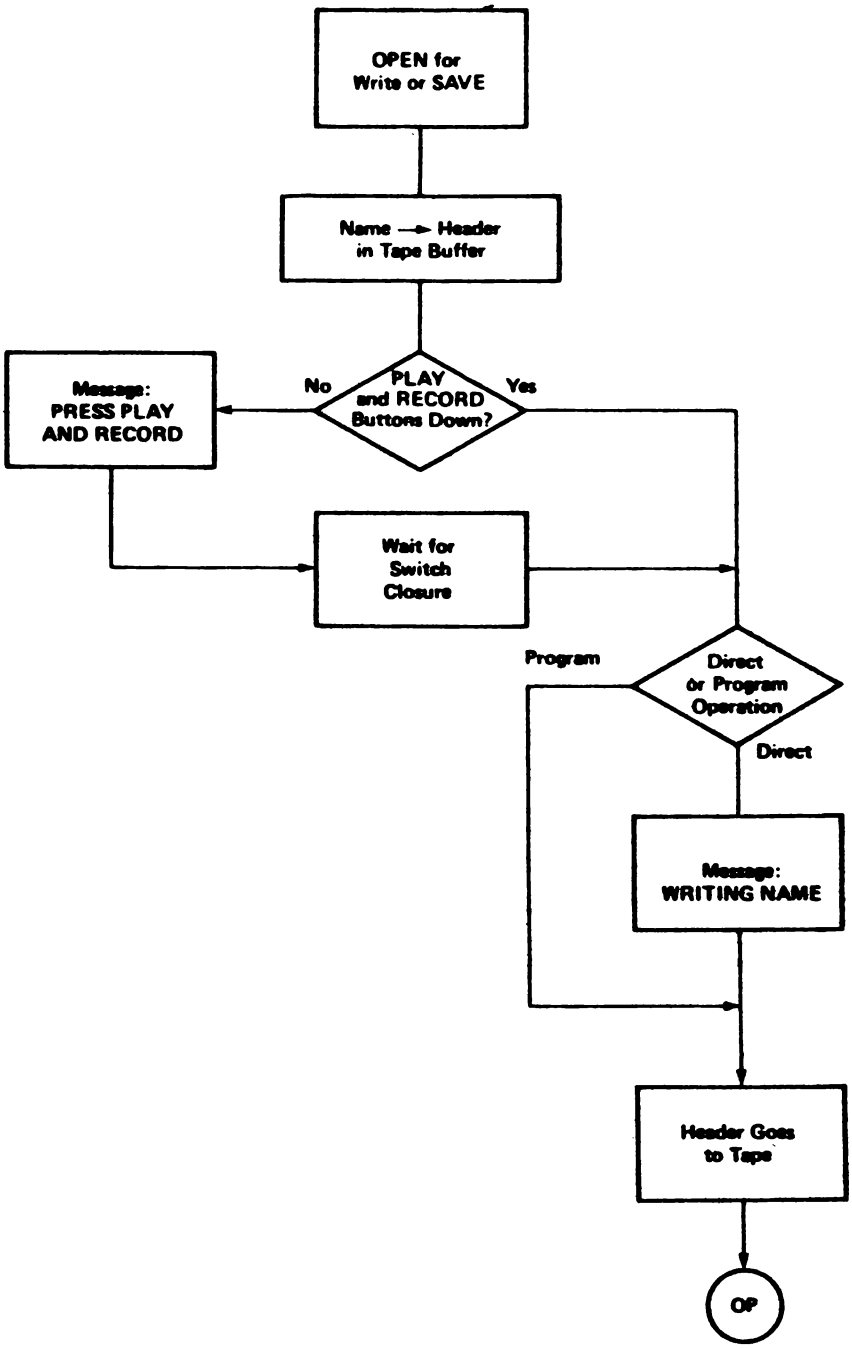


Figure 7.15. OPEN for write from CBM: PRINT#,CMD or SAVE.
OP = operating system takes over.

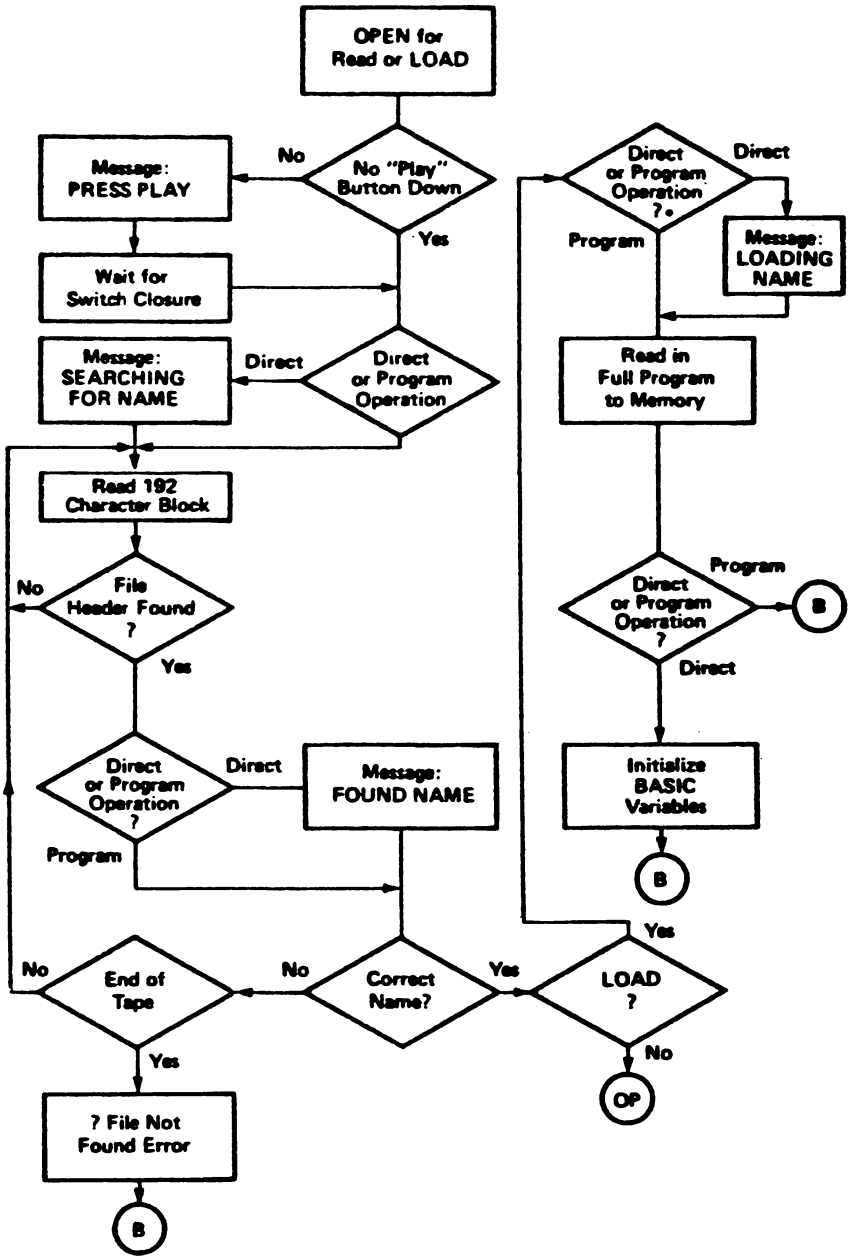


Figure 7.16. OPEN for read to CBM: INPUT# or LOAD
OP = Operating system takes over. B = BASIC takes over.

INGRESSO DEI DATI - NOZIONI GENERALI -

L'uso del codice "INPUT" in questo contesto implica l'ingresso dei dati nel PET da qualsiasi organo periferico.

INPUT# - INGRESSO DI STRINGHE E DI VARIABILI -

INPUT# e' il comando usato per iniziare il trasferimento di dati da un dispositivo INPUT/OUPUT al sistema operativo. Il formato dell'istruzione e':

INPUT# numero di file logico,A,A$,B,B$,ecc

Dove A,A$,B e B$ sono variabili numeriche o variabili stringa da far entrare dal dispositivo periferico scelto al sistema operativo un carattere alla volta.A causa delle regole dell'interprete BASIC che si applicano a questa istruzione d'ingresso, tutti i ritorno carrello, le virgole, i campi terminatori, i caratteri nulli e gli spazi bianchi che precedono il dato, eccetto che nelle stringhe e tutti gli altri caratteri di controllo sono automaticamente cancellati.Non e' sempre possibile mescolare dati numericie alfabetici su un dispositivo di INPUT/OUPUT.Se e' stato specificato un campo numerico, solamente i dati numerici,nella forma standard che il BASIC si aspetta, vengono accettati, in caso contrario viene stampato sullo schermo un messaggio ?BAD DATA ERROR. Se esiste qualche incertezza sul tipo di dati da far entrare nel calcolatore e' necessario utilizzare solamente le stringhe ed usare poi i vari comandi di manipolazione delle stringhe per processare i dati e caricarli nell'appropriata variabile.Esempio di un'istruzione INPUT#: se X rappresenti una serie di 50 numeri memorizzata su un file su nastro chiamato VECTOR e si assuma che il tasto PLAY sia stato appena premuto sull'unita' numero 1.Il programma che segue leggera'i 50 numeri uno alla volta e li visualizzera' successivamente sullo schermo.

10 OPEN 1,1,0"VECTOR" apre il file logico numero 1.Assegna il file alla cassetta 1. Apre l'unita' a nastro per la lettura.Inizia la ricerca del file fisico di nome VECTOR.
20 FOR K=1 to 50 legge 50 numeri uno alla volta dalla cassetta 1.
30 INPUT#1,X
40 PRINT X Visualizza i numeri sullo schermo
50 NEXT K
60 CLOSE 1 Quando i 50 numeri sono stati letti viene chiuso il file logico numero 1.

ISTRUZIONE GET - TRASFERIMENTO DI CARATTERI

Non tutti i periferici sono in grado di trasferire dati in una forma che sia direttamente accettabile dal BASIC. Vi e' tutta una serie di dati binari e combinazioni che il BASIC ignora e sebbene parecchi dispositivi IEEE-488 rispondano correttamente con formato di dati che e' direttamente accettabile dal BASIC, alcuni altri non sono in grado di eseguire tale funzione.In aggiunta, molto spesso,e' utile al programmatore avere un immediato accesso al carattere che e' in corso di trasferimento.L'istruzione GET permette di prendere il singolo carattere dal bus IEEE-488 o dalla cassetta magnetica e mettere tale carattere nel campo che viene specificato immediatamente dopo il codice GET#.La forma dell'istruzione e':

GET# file logico, campo

INGRESSO DA NASTRO

Quando si legge da un file su nastro i dati vengono all'utilizzatore I/O indipendentemente.

Ogni volta che si incontra un'istruzione INPUT# oppure GET# da un'unita' di I/O che sia il nastro 1 o 2 viene chiamata una subroutine speciale,che inizia l'ingresso da nastro.Non appena il BASIC richiede un carattere,tale carattere viene fornito da un appropriato buffer da nastro.Quando il buffer e' vuoto la routine d'ingresso da nastro sospende il programma di utente e legge un blocco di dati dal nastro nel buffer e quindi trasferisce il successivo carattere al BASIC. Se si verifica un errore di lettura, esso viene messo in evidenza nella parola di stato.Quando nel buffer viene incontrato un contrassegno di fine del file, il bit difine del file della parola di stato viene messo ad uno e viene stampato automaticamente un ritorno carrello.Alla fine di un comando, il BASIC richiama un'altra routine che rida' l'ingresso alla tastiera e informa che l'operazione sul file e' stata completata.

SEQUENZE DI INPUT SUI DISPOSITIVI IEEE-488

I comandi INPUT# e GET# vengono utilizzati nella stessa sequenza che e' stata appena descritta.Quando il comando viene incontrato per la prima volta, viene chiamata una routine di inizializzazione d'ingresso da IEEE-488, essa invia una richiesta d'attenzione al dispositivo e all'indirizzo secondario che erano state specificate per quel file logico nella sequenza OPEN. Alla fine della sequenza di attenzione il PET si mette in condizioni di ascolto e vi rimane in attesa di un segnale DAV che indica che il singolo carattere e' stato ricevuto. Se il segnale DAV viene ricevuto entro 65 millisecondi tale carattere viene consegnato al BASIC o comunque che ha richiamato la routine IEEE-488.Ogni volta che questa routine viene richiamata si ha la stessa sequenza di operazioni. Se il bus non risponde entro 65 millisecondi, la routine IEEE-488 termina automaticamente la sequenza e da luogo ad un errore di lettura nella parola di stato in modo da indicare che nessun carattere e' stato ricevuto. Se durante la lettura di un carattere la routine IEEE-488 si accorge che la linea EOI e' stata alzata, tale routine da luogo ad una informazione di fine dei dati nella parola di stato e restituisce un ritorno carrello finche' il comando non e' stato esaurito.Alla fine del comando, il BASIC,richiama una subroutine di terminazione che rinizializza le operazioni sulla tastiera e invia un comando di liberazione al bus IEEE-488, liberando per successivi comandi.

LIMITAZIONI PER I BUFFER D'INGRESSO

Sebbene i dati vengano trasferiti al sistema operativo un carattere alla volta, per poter rimanipolare,il BASIC li accumula in un buffer di 80 posizioni. Il buffer deve essere terminato da un ritorno carrello. Nel PET, se si leggono piu' di 80 caratteri, si puo' avere la situazione che il sistema operativo funzioni male,in quanto alcune variabili del sistema operativo vengono sporcate.Per far funzionare nuovamente il PET e' necessario spegnerlo e riaccenderlo immediatamente dopo. E' necessario porre estrema attenzione a questo vincolo quando si usino sistemi con nastri o con dischi.In altre parole e' necessario scrivere sui nastri o sui dischi un ritorno carrello in modo che non ci siano mai blocchi di piu' di 80 caratteri che non siano separati da un ritorno carrello.Per superare queste limitazioni quando si abbiano dei dispositivi che inviano piu' di 80 caratteri per blocco, puo' essere utilizzato il comando GET per costruire le stringhe appropriate senza incorrere in incovenienti.

USCITA DEI DATI - GENERALITA' -

L'uso dei codici "PRINT" e "WRITE" si riferiscono all'uscita dei dati dal PET verso qualsiasi periferico.

PRINT#

Il comando PRINT# deve essere seguito da un numero di file logico,da una virgola e quindi dati che devono essere stampati.Ad esempio:

PRINT# numero di file logico,dati

I dati vengono trasferiti un carattere alla volta al periferico fisico e messo in relazione con il file logico specificato nell'istruzione OPEN relativa. Parecchi delimitatori del file come ad esempio le virgole vengono automaticamente cancellati dal BASIC ;sebbene questo fatto non abbia una grande importanza nella fase di scrittura, si deve ricordare che quando si leggera' dal nastro o da qualsiasi altro dispositivo I/O questi delimitatori del file devono essere forzati. Questo puo' venir fatto inserendo un CHR$(44) oppure "," tra i campi in modo da evitare che essi vengano scritti come un singolo campo.Ad esempio l'istruzione:

PRINT# LF,A;B$;C$
verra' inviata come
ABC
senza delimitatori:
l'istruzione:
PRINT#LF,A;CHR$(44)B$;CHR#(44);C$
oppure
PRINT#LF,A",";B$;",";C$
verra' inviata in uscita come:
A,B$,C$,CR
dove CR sta ad indicare un ritorno carrello
oppure le istruzioni:
PRINT#LF,A
PRINT#LF,B$
PRINT#LF,C$
appariranno in uscita come:
A CR B$ CR C$ CR

Poiche' il BASIC formata sempre l'uscita verso qualsiasi dispositivo come se stesse scrivendo sullo schermo.L'istruzione PRINT LF,A,B contiene diversi caratteri bianchi tra il valore di A e di B, mentre l'istruzione PRINT LF;B non ha questi caratteri bianchi.

NOTA:sebbene ambedue le istruzioni INPUT# e PRINT# operano virtualmente nello stesso modo dei loro equivalenti INPUT e PRINT nel BASIC,il comando abbreviato ? che puo' essere usato al posto di PRINT, non puo' essere applicato al posto di PRINT#.Il codice ?# e PRINT# sono riconosciuti e riportati a due differenti caratteri di contrassegno che vengono utilizzati in maniera diversa dal BASIC. Il codice ?# verra' stampato eseguendo una lista come PRINT#, ma in esecuzione dara' luogo a ?SYNTAX ERROR.

ESEMPI DI ISTRUZIONE PRINT#

Il programma che segue, stampa una serie di numeri da 1 a 50 sulla stampante del PET, uno alla volta.

10 OPEN 5,4,0 apre il file logico #5.Assegna il file logico #5 al periferico #4 (stampante del PET) il modo di stampare e' quello normale corrispondente all'indirizzo secondario "0".
20 FOR K=1to50 stampa la serie dei 50 numeri sulla stampante.
30 PRINT#5,K
40 NEXT K
50 CLOSE 5 chiude il file logico #5.

Per scrivere la stessa serie di numeri su una cassetta sull'unita' a nastro #2 e' necessario solamente cambiare la linea che contiene l'istruzione OPEN, purche' si adoperi lo stesso numero logico di file:

100PEN 5,2,1 Apre il file logico #5.Assegna il file logico #5 alla #2 (unita' a cassetta) il modo di scrittura corrisponde a una scrittura senza "end of tape"e corrisponde all'indirizzo secondario 1.

```
20 FOR k=1to50 registra la serie dei 50 numeri sul nastro.
30 PRINT#5,K
40 NEXT K
50 CLOSE 5  Chiude il file logico #5.
```

Nell'esempio appena illustrato i dati vengono accumulati in un buffer di 192 caratteri, un carattere alla volta.Quando si supera la capacita' del buffer l'ingresso dei dati viene sospeso, il nastro parte e il comando del buffer viene scritto su nastro. Il buffer viene inizializzato per accettare altri 192 caratteri e il rogramma prosegue.

USCITE SUL BUS IEEE-488

L'istruzione PRINT# fa si che venga richiamata una subroutine d'uscita che inizializza per operazioni di OUPUT un periferico IEEE-488.In primo luogo l'uscita che normalmente avviene sullo schermo viene riassegnata a quel dispositivo fisico che e' stato scielto come file logico nella routine di apertura.Un comando di porsi in ascolto viene inviato sul bus IEEE-488 al dispositivo fisico all'indirizzo secondario specificati per il file logico nell'istruzione OPEN. A questo punto il BASIC invia un carattere alla volta ad una subroutine che lo trasferisce sul bus, il PET,cioe', agisce come il trasmettitore e tutti i dispositivi indirizzati diventano ricevitore.Quando il BASIC ha finito d'eseguire il comando PRINT# viene richiamata un'altra subroutine del sistema operativo in modo che il PET invii un comando di liberazione al bus e rimette l'uscita sullo schermo. In tal modo il bus rimane libero per una successiva operazione.

COMANDO CMD

Normalmente, ciascun comando di stampa, opera in unione con un unico dispositivo di uscita, ed alla fine del comando il bus viene liberato.In alcuni casi,tuttavia, e' opportuno avere la possibilita' di colloquiare con piu' di un dispositivo sul bus; per permettere cio',viene fornito il comando speciale CMD.L'istruzione CMD e' virtualmente identica a PRINT#, eccetto che alla fine del trasferimento dei dati , non viene chiamata la routine di liberazione, lasciando quindi che il periferico sia appeso al bus come ricevitore.Il sistrema operativo, dunque, continua a trattare l'ultimo dispositivo che e' stato individuato dal comando CMD come il dispositivo di uscita primario per il BASIC. I comandi PRINT o LIST vengono diretti a questo dispositivo anziche' allo schermo video. Piu' specificatamente l'istruzione CMD del dispositivo di stampa, seguito dall'istruzione LIST, fa si che la lista venga eseguita su carta. Tuttavia, poiche' ne' l'istruzione CMD ne' il comando LIST fanno terminare le operazioni sul bus in riferimento al dispositivo prescelto, e' necessario una istruzione PRINT per far terminare un'istruzione CMD.

ESEMPI DI COMANDO CMD

Per listare sulla stampante:

```
OPEN 3,4   4 e' il numero della stampante
CMD 3
LIST   si ottiene in tal modo una lista anziche' sullo
```

schermo, sulla stampante

Per scrivere e per usare un disco allo stesso tempo si usera' il seguente programma:

*CMD 3 - Il file logico 3 viene aperto alla stampante.

PRINT# 15,A,B,C - il 15 e' il numero di file logico del floppy disc che deve essere stato aperto precedentemente.

Tali istruzioni faranno si che A,B e C verranno registrate sul floppy,ma verranno anche stampate sulla stampante.Per controllare un dispositivo di INPUT si potra' usarela seguente sequenza di istruzioni:

**CMD 3 - Attiva la stampante.

INPUT# 15,A,B,C - Legge dal floppy.

Si otterra' in tal modo che i dati vengano letti dal floppy e trasferiti nelle variabili A,B e C, tuttavia si avra' anche contemporaneamente al trasferimento, una stampa.

CHIUSURA DEI FILE

E' opportuno che qualsiasi file logico che e' stato aperto durante un programma sia chiuso quando non e' piu' necessario; nel caso di file su nastro o su disco essi devono essere chiusi prima che il programma termini. E' bene tener a mente le seguenti regole:

a) Le operazioni del PET possono diventare incoerenti se il numero totale di file logico in azione, eccede la decina.

b) Se un file logico associato a un'unita' nastro non viene chiuso non verra' registrato nessun "end of file" alla fine del file fisico su nastro. Se tale nastro poi viene nuovamente letto in memoria, il PET non avra' modo di riconoscere la fine del file e se un dato non voluto o erroneo e' presente a causa di una precedente registrazione, esso verra' letto in memoria.

ESEMPIO DI UN'ISTRUZIONE DI CLOSE

Per chiudere qualsiasi file e' sufficiente dare il seguente semplice comando:

```
CLOSE file logico
```

Se si vuol chiudere il file logico numero 5, sara' sufficiente dare il comando:

```
CLOSE 5
```

CHIUSURA DEI FILE SU NASTRO

Quando viene aperto un file su nastro vengono eseguite due operazioni successive:nel primo carattere disponibile per i dati viene registrato un contrassegno di "end of file", poi il buffer del nastro viene inviato sulla cassetta. Se nell'istruzione OPEN viene scielta l'opzione "end of tape",sulla cassetta viene registrato anche un blocco d'intestazione "end of tape".

CHIUSURA DEI PERIFERICI IEEE-488 FORNITI DI NOME

Per i dispositivi IEEE-488, che sono stati aperti con un nome di file, tramite l'istruzione OPEN, viene inviata una speciale sequenza di comando d'ascolto con un indirizzo secondario speciale formato dalla cifra esadecimale E0 e l'indirizzo secondario specificato nell'istruzione OPEN.Questo permette di chiudere dispositivi come file su dischi tramite il controllo di periferica.

RIVELAZIONE D'ERRORE

Il concetto generale del sistema operativo del PET e' quello di permettere all'utente di operare in formato libero; cio' consente di leggere o scrivere su nastri, dischi e stampanti nella maniera piu' comoda.

Poiche' l'INPUT/OUPUT e' totalmente in formato libero, e' molto importante che il sistema operativo sia in grado di informare l'utente quando si ha un errore di trasmissione o si incontra una fine dei dati.

PAROLA DI STATO

Per facilitare la rilevazione degli errori nelle operazioni di I/O, il PET usa una "parola di stato" nel senso che un byte viene manipolato da ciascuna operazione di I/O e puo' essere richiamato dal programmatore in qualsiasi momento attraverso la funzione ST. Ciascun bit della parola di stato ha un significato generale per tutte le operazioni e un significato particolare per ciascun dispositivo I/O. La tavola 7.17 mostra gli errori in relazione alla "parola di stato" per unita' a cassette magnetiche.

ST Bit Position	ST Numeric Value	Cassette Read	IEEE R/W	Tape Verify + Load
0	1		Time out on write	
1	2		Time out on read	
2	4	Short block		Short block
3	8	Long block		Long block
4	16	Unrecoverable read error		Any mismatch
5	32	Checksum error		Checksum error
6	64	End of file	EOI line	
7	-128	End of tape	Device not present	End of tape

Table 7.17. Status Word (ST) values correlated with tape cassette, unit and IEEE bus read/write errors.

ERRORI DEI DISPOSITIVI IEEE-488

Basicamente vi sono tre errori che possono avvenire durante un trasferimento sul bus IEEE-488. Il primo caso si ha quando l'intero bus non risponde ad una sequenza di richiesta d'attenzione. Se cio' avviene la routine IEEE-488 alza il bit 7 della parola di stato che ha il significato di dispositivo NOT PRESENT. Il PET allora fa terminare il programma corrente stampando su schermo ?DEVICE NOT PRESENT ERROR.Se il bus risponde correttamente alla richiesta d'attenzione, ma quando il PET tenta di scrivere il primo carattere sul bus, il dispositivo fisico non e' presente, cosa indicata dallo stato basso del segnale NRFD o NDAC, allora il PET, da'ancora un'indicazione di dispositivo NOT PRESENT. Il secondo errore ha luogo durante il processo di trasferimento dei dati al dispositivo.Se il bus non risponde entro un determinato tempo o se cessa di rispondere per mezzo di segnali NRFD o NDAC portati ambedue alti, si avra' un'indicazione di errore mettendo ad 1 il bit 0 della parola di stato.Il terzo errorre avviene durante la lettura tramite il bus IEEE-488, quando il periferico non invia il segnale DAV entro 65 millisecondi;in tal caso viene messo ad 1 il bit 1 della parola di stato.

ERRORI SULL'UNITA' A NASTRO

Le cassette testano i dati solamente durante la lettura.Gli errori che possono essere rilevati sono:

1) BLOCCO CORTO (parola di stato pari a 4) Leggendo un blocco dal nastro,un tono di spaziatura viene incontrato prima che il numero di byte che ci si aspettava sia stato letto dal blocco.La causa possibile puo' essere il tentativo di leggere un file di caricamento corto come un record di dati.

2) BLOCCO LUNGO (parola di stato pari ad 8) Leggendo un blocco da nastro non s'incontra un tono di spaziatura dopo che il numero che ci si aspettava di byte e' stato letto dal blocco.La causa possibile e' il tentativo di leggere come dati un file di caricamento lungo.

3) ERRORE DI LETTURA IRRECUPERABILE (parola di stato pari a 16) Causa:piu' di 31 errori nel primo blocco dei blocchi di ridondanza o un errore che non puo' essere corretto in quanto e' avvenuto nella stessa posizione in ambedue i blocchi.

4)ERRORE DI CHECKSUM (parola di stato pari a 32) Dopo un LOAD o la lettura di un dato viene calcolata una somma di prova sui byte che si trovano nella RAM e tale somma viene confrontata con un byte ricevuto dal dispositivo di INPUT.Se essi non coincidono,si origina un errore di CHECKSUM.

5)FINE DEL FILE(parola di stato pari a 64) Questo bit viene messo a 1 quando viene incontrato un contrssegno di fine del file dati su un record su nastro.

6) FINE DEL NASTRO (parola di stato pari a 128) E' stato incontrato un contrassegno di EOT.

ESEMPI DI USO DELLA FUNZIONE ST

Da quanto detto appare evidente che il PET non e' in grado di riconoscere errori durante la scrittura su nastro o durante la lettura o la scrittura dallo schermo.La normale tecnica di programmazione e' quella che fa eseguire ad un'istruzione di INPUT# o GET# un test sul valore dello stesso. Poiche' la parola di stato con singolo byte di memoria e lo stato cambia per ciascun nuovo comando di I/O;lo stato e' una grandezza rapidamente transiente. Ad esempio:

```
100 INPUT#2,A
110 INPUT#5,B
120 IF TS=0 THEN 200
```

Questo programma testa il risultato del trasferimento dei dati solamente dal file logico 5.Il risultato della lettura del file logico 2 e' perso. Allo stesso modo il programma:

```
100 INPUT#2,A
110 PRINT A
120 IF ST=0 THEN 200
```

fa si che la parola di stato ragguaglia solamente sullo stato della stampa anziche' dare informazioni sullo stato della lettura dal file 2. Una via corretta per usare la parola ST e' la seguente:

```
100 INPUT#2,A,B,C
110 IF ST=0  THEN 200
120 IF ST=64 THEN 300
130 IF ST=2  THEN 400
```

Ciascun errore puo' in tal modo essere individuato e si possono intraprendere le successive operazioni che si rendano necessarie attraverso l'istruzione:

140 IF ST AND mask THEN - mask rappresenta il bit che si vuol testare.

TECNICHE D'INTERROGAZIONE

Una tecnica per interrogare i dispositivi IEEE-488 lenti, come ad esempio, dei dispositivi di campionamento che richiedono anche diversi minuti per rispondere e'quella di usare l'istruzione INPUT# dal dispositivo; quando lo stato indica un time out, si puo' allora passare ad altra routine o ciclarla sull'istruzione INPUT# finche' l'errore non viene eliminato, quando finalmente non ci sara' piu' errore il dato corretto sara' finalmente pronto e potra' essere letto e quindi successivamente utilizzato.Usando questa tecnica di campionamento si puo' servire tutta una serie di dispositivi lenti quando viene eseguito un programma utilizzando il real time clock e la sequenza di errore per time out dell'istruzione INPUT#.

VALORI ASSEGNATI AI PARAMETRI QUANDO ESSI NON VENGONO SPECIFICATI

Parameter	Default Value	Default Operation
Device #	D=1	Cassette #1 selected
Secondary address	SA=0	On tape files open for read On IEEE-488 devices, no secondary address is sent.

Table 7.18. Default values.

Statement	Equivalent (Default) Parameter Values	Operation
OPEN 1	OPEN 1,1,0	Open logical file #1 for cassette #1 read no file name
OPEN 1,2	OPEN 1,2,0	Open logical file #1 for cassette #2 read no file name
OPEN 1,2,1	OPEN 1,2,1	Open logical file #1 for cassette #2 write no file name
OPEN 1,2,1, "DAT"	OPEN 1,2,1, "DAT"	Open logical file #1 for cassette #2 write file named "DAT"

Table 7.19. Example of default parameters.

INTRODUZIONE AL BUS IEEE-488

Questo bus consiste di 16 linee di segnali che sono divise funzionalmente in tre gruppi:

1) il bus di trasmissione di dati
2) il bus di controllo
3) il bus di gestione

Inoltre, il bus IEEE-488 puo' sopportare tre classi di dispositivi:

a) trasmettitori:in ogni istante, solamente un dispositivo, puo' trasmettere dati dal bus di dati.
b) ricevitori: tanti dispositivi,quanti sono necessari,possono ricevere dati simultaneamente dal bus.

c)controllore:il PET e' il solo controllore che e' fornito per l'IEEE bus nell'applicazione che si sta descrivendo.

CONTROLLO DEL BUS E DELLA PERIFERICA

Dalla cartolina in circuito stampato su cui e' montato il circuito del PET e' ricavato sul bordo un connettore a 12 posizioni e 24 contatti, per ulteriori informazioni riferirsi alla tavola 7.19 ed alla figura 7.2
E' necessario porre in luce alcune limitazioni fisiche che si hanno quanso si connettono delle periferiche al bus IEEE:
a) la massima lunghezza del cavo connesso fra PET e periferica e' di 20 metri.
b) La massima distanza fra due periferiche connesse al bus e' 5 metri.
c) Il massimo numero di periferica e' 15.

CBM Contact Identification	Bus	IEEE Label	CBM Contact Identification	Label Description
1	DATA	DI01	1	Data INPUT/OUTPUT LINE #1
2		DI02	2	Data INPUT/OUTPUT LINE #2
3		DI03	3	Data INPUT/OUTPUT LINE #3
4		DI04	4	Data INPUT/OUTPUT LINE #4
5	MANAGER	EOI	5	End or identify
6	TRANSFER	DAV	6	Data valid
7		NRFD	7	Not ready for data
8		NDAC	8	Data not accepted
9	MANAGER	IFC	9	Interface clear Same as CBM reset
10		SRQ	10	Service request
11		ATN	11	Attention
12		SHIELD	12	Chassis ground and IEEE cable shield
A	DATA	DI05	13	Data INPUT/OUTPUT LINE #5
B		DI06	14	Data INPUT/OUTPUT LINE #6
C		DI07	15	Data INPUT/OUTPUT LINE #7
D		DI08	16	Data INPUT/OUTPUT LINE #8
E	MANAGER	REN	17	Remote enable (REN) always ground in the CBM
F	GROUNDS	GND6	18	DAV ground
H		GND7	19	NFRD ground
J		GND8	20	NDAC ground
K		GND9	21	IFC ground
L		GND10	22	SRQ ground
M		GND11	23	ATN ground
N		LOGIC GND	24	Data ground (DI01-8)

Table 7.20. IEEE bus group, label and contact identification number.

IL BUS DEI DATI

Questo bus e' composto di 8 linee bidirezionali che trasmettono i segnali relativi ai dati attivi nello stato basso. Il dispositivo piu' lento che si usa in quel momento sul bus controlla il ritmo di trasferimento dei dati;il modo di trasferimento e' a un byte alla volta con gli 8 bit in parrallelo.Sui bus dei dati sono trasmesse anche le informazioni di controllo e gli indirizzi dei periferici.Questi vengono differenziati dai dati da un segnale ATN vero durante il loro trasferimento.Il bit piu' significativo (MSB) si trova sulla linea DI-08.Per una spiegazione delle abbreviazioni dei segnali come ad esempio DI-08 ci si riferisca alla figura 7.23

MODI DI TRASMISSIONE DEI DATI

Sul bus dei dati e' valida qualsiasi configurazione di bit quando si trasmettono dati verso le periferiche.

IL BUS DI TRASFERIMENTO

Il bus di trasferimento e' composto da tre linee che controllano il trasferimento dei dati sul bus dati.I segnali trasmessi vengono usati nella procedura di collegamento come illustrato in figura 7.21.Questi segnali sono:

a) NRFD non pronto per i dati.
b) NDAC dato rifiutato.
c) DAV dato valido.

Si noti che il segnale DAV e' originato dal trasmettitore,mentre i ricevitori creano i segnali NFRD e NDAC.
Per la descrizione dettagliata di questi segnali ci si riferisca alla tabella 7.23.

LA PROCEDURA DI COLLEGAMENTO

Quando un trasmettitore invia un byte di dati a uno o piu' ricevitori,viene usata la seguente procedura di controllo, in modo da assicurare che l'operazione vada a termine con successo. La funzione essenziale della procedura di collegamento e' quella di assicurare che:

a) tutti i ricevitori siano pronti ad accettare dati.
b) che vi sia sul bus dei dati un dato valido.
c) che i dati siano ricevuti da tutti i periferici.

Il trasferimento dei dati ha luogo ad una velocita' che e' determinata dal dispositivo periferico che si trova in quel momento in ascolto sul bus ed ha la minima velocita' di ricezione. Questo fatto permette l'interconnessione di dispositivi che lavorano a differenti velocita' ed a differenti possibilita' di manipolazione dei dati.La sequenza di eventi che ha luogo durante il trasferimento di byte di dati dal trasmettitore ai ricevitori e' mostrata nel diagramma di flusso di figura 7.21.

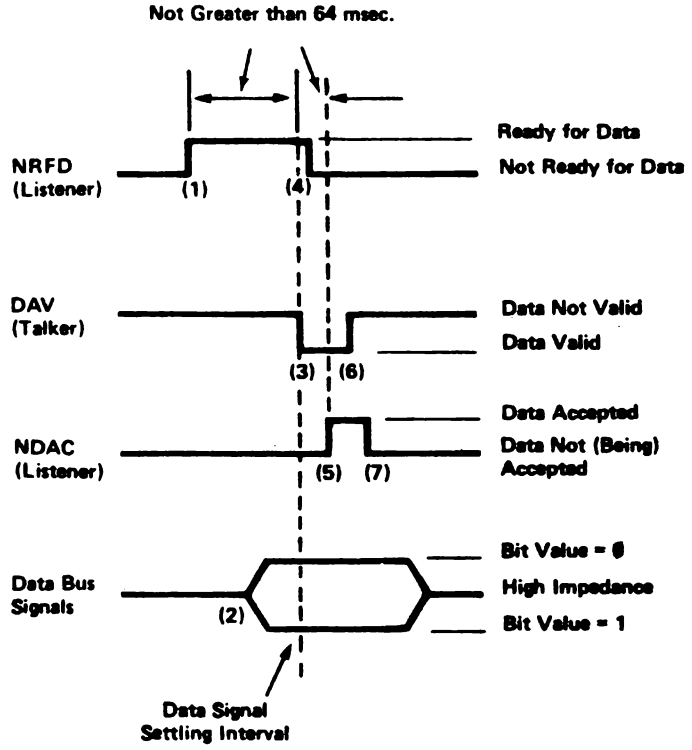


Figure 7.21. Transfer bus handshake sequence.

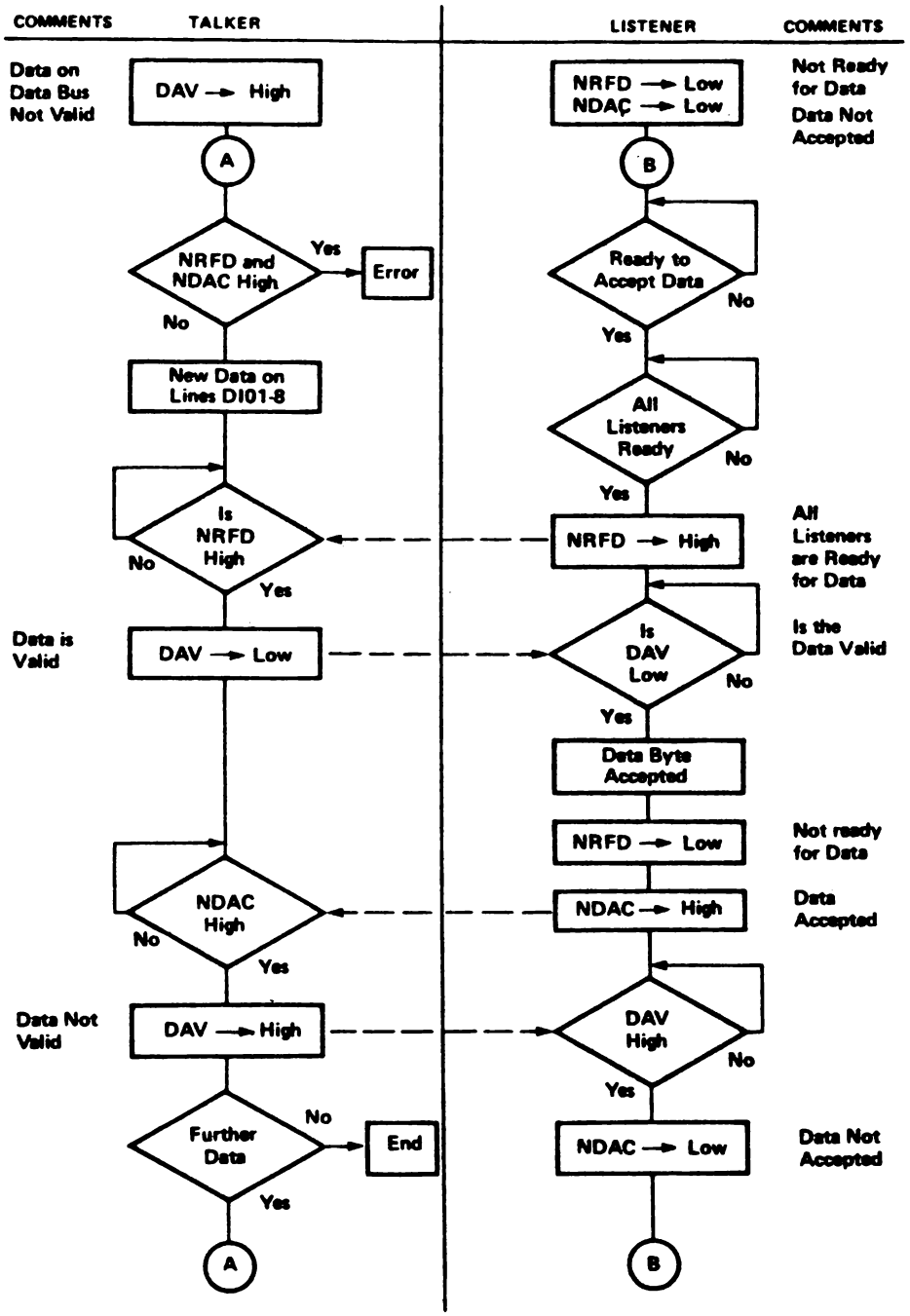


Figure 7.22. Sequence of events during a data byte transfer from the talker to the listners. Broken lines indicate the testing of transfer bus signal logic levels.

La figura 7.22 mostra la tamporizzazione relativa dei segnali sul bus di trasferimento durante una tipica procedura di collegamento; i numeri della descrizione che seguira' immediatamente, si riferiscono ai cambiamenti dei livelli dei segnali logici della figura 7.22 con riferimento ai numeri chiusi in parentesi.
1) Il segnale NRFD si porta allo stato alto corrispondente allo stato logico falso e indica che tutti i ricevitori sono pronti per il successivo byte dei dati.
2) Il trasmettitore invia il prossimo byte dei dati sul bus dei dati e posiziona i segnali relativi ai dati.Cio' puo' avvenire prima, dopo o durante la transizione(1).
3) Il trasmettitore controlla il segnale NRFD,quando questo segnale viene trovato allo stato alto il trasmettitore porta allo stato basso,che corrisponde ad un livello logico vero,il segnale DAV per informare i ricevitori che i dati sul bus-dati sono ora validi.
4) Non appena un ricevitore si accorge che il segnale DAV si e' portato allo stato basso, allora. porta il segnale NRFD basso anch'esso; i dati vengono ora accettati da tutti i ricevitori alla loro velocita', ciascun ricevitore alla fine della ricezione rilasci il segnale NDAC.
5) Il segnale NDAC si porta allo stato alto, che corrisponde al livello logico falso, quando il piu' lento dei ricevitori ha accettato il dato.
6) Il trasmettitore posiziona il segnale DAV allo stato alto, corrispondente al livello logico falso,per indicare che i segnali del bus non sono piu' validi.
7) I ricevitori si accorgono che il segnale DAV si e' posrtato allo stato alto di conseguenza posiziona il segnale NDAC allo stato basso completando la sequenza di collegamento.Quando ciascun ricevitore ha processato il dato, il segnale NRFD viene rilasciato.Questo fatto fa terminare la sequenza per il primo trasferimento dei dati.La sequenza ora puo' essere ripetuta di nuovo ripartendo dal punto 1 finche' tutti i dati che era necessaria trasmettere sono stati inviati ai periferici.

VINCOLI SULLE TEMPORIZZAZIONI NEL BUS PET-IEEE

E' necessario porre in luce alcune limitazioni sulle temporizzazioni in modo da evitare perdite di dati:
a) Quando il PET funge da ricevitore, esso aspetta che il segnale DAV si porti allo stato basso entro 64 millisecondi dall'istante in cui ha posizionato allo stato alto il segnale NRFD.
b) Quando il PET funge da trasmettitore,esso si aspetta che il segnale NDAC vada allo stato alto entro 64 millisecondi dopo che il segnale NRFD e' stato posto a livello alto.
Se questi limiti vengono sorpassati, il PET cessa di trasferire dati e posiziona l'appropriato bit della parola di stato ST.Ci si riferisca alla tabella 7.24.

IL BUS DI GESTIONE

Questo gruppo di 5 linee di segnali controlla lo stato del bus dei dati e definisce i suoi segnali;questi segnali possono essere dati,indirizzi o informazioni di controllo definibili in altre parole come comandi alle periferiche.
I 5 segnali di gestione sono:

a) ATN-ATTENZIONE Mediante questo segnale si puo' definire ciascun dispositivo quale ricevitore o trasmettitore.
iu' comoda.

b) EOI-FINE DELLA IDENTIFICAZIONE indica che l'ultimo byte di dati e' stato trasferito.

c) IFC-AZZERAMENTO DELL'INTERFACCIA Inizializza il bus dei dati. Trasmettitori e ricevitori vengono passati a uno stato di riposo.Questo segnale e' lo stesso che si ha durante il raset del PET.

d) SRQ-RICHIESTA DI SERVIZIO Un dispositivo periferico informa il controllore che ha bisogno di essere servito. Non e'implementato nel BASIC ma e' disponibile nel PET.

e) REN-ABILITAZIONE REMOTA E' collegato permanentemente a massa nel PET.

I SEGNALI IEEE E LE LORO DEFINIZIONI

Le 16 linee di trasmissione del bus IEEE-488 sono assegnate ciscuna ad uno specifico segnale. La tabella 7.23 indica il gruppo del bus, il nome, l'abbreviazione e la descrizione funzionale per ciascuno di questi segnali.

CONVENZIONE SUI LIVELLI LOGICI

Il valore logico vero o 1 corrisponde allo stato basso con uscite di tipo collettore comune.Cio' permette di ottenere che ciascun periferico possa mantenere il bus nello stato logico 1 o vero.

BUS GROUP	SIGNAL ABBREV.	NAME	FUNCTIONAL DESCRIPTION
MANAGER	ATN	ATTENTION	Il PET che in quel momento funge da controllore,posiziona questo segnale allo stato basso quando invia dei comandi sul bus dei dati.Quando il segnale ATN e' basso,solamente indirizzi di periferica e messaggi di controllo possono essere presenti sul bus dei dati.Quando il segnale ATN e' alto,solamente i periferici,precedentemente assegnati,possono trasferire dati.
TRANSFER	DAV	DATA VALID	Quando il segnale DAV e' basso,cio' sta ad indicare che il dato e' valido sul bus dei dati.
MANAGER	EOI	END OF IDENTIFY	Quando viene inviato sul bus l'ultimo byte di dati,il trasmettitore puo' mettere a 1 il segnale EOI.Il PET posiziona sempre il segnale EOI allo stato basso,quando invia l'ultimo byte di dati.

MANAGER	IFC	INTERFACE CLEAR	Il PET invia il suo reset interno come un segnale IFC allo stato basso per inizializzare tutti i periferici ad uno stato di riposo. Quando il PET viene acceso o viene ressettato il segnale IGC si porta allo stato basso per circa 100 millisecondi.
TRANSFER	NDAC	DATA NOT ACCEPTED	Questo segnale viene mantenuto allo stato basso quando un ricevitore sta leggendo un dato.Quando il byte e' stato totalmente letto,il ricevitore riporta il segnale NDAC allo stato alto.Questi segnala al trasmettitore che il dato e' stato accettato.
TRANSFER	NRFD	NOT READY FOR DATA	Quando il segnale NRFD si trova allo stato basso,uno o piu' ricevitori non sono pronti a ricevere il successivo byte di dati.Quando tutti i periferici sono pronti il segnale NRFD si porta allo stato alto.
MANAGER	SRQ	SERVICE REQUEST	Non e' implementato nel BASIC,ma e' disponibile all'utente del PET.
MANAGER	REN	REMOTE ENABLE	Il segnale REN viene mantenuto basso dal controllore del bus.Il PET. ha il corrispondente piedino collegato a massa,il che fa si che il segnale REN sia permanentemente allo stato basso.
DATA	DIO1-8	DATA I/O LINES 1 THROUGH8	Questi segnali rappresentano i bit di informazione sul bus dei dati.Quando un segnale DIO va a uno stato basso esso rappresenta un livello logico 1;quando va allo stato alto rappresenta un livello logico Ø.
GENERAL	GND	GROUND	Connessione di massa:sul bus IEEE si hanno 6 ritorni a massa dei segnali di controllo e gestione, un ritorno per i segnali dati e un collegamento alla terra del circuito e allo chassis.

LA PAROLA DI STATO

ST e' una variabile del BASIC che puo' essere usata per testare le operazioni di INPUT/OUTPUT. ST viene ad assumere determinati valori compresi tra -128 e 127.La tabella 7.24 mostra i codici di stato che possono venir realizzati da operazioni sul bus IEEE-488.

ST	ERROR	EXPLANATION
1	TIME OUT ON LISTENER	Il periferico IEEE non ha risposto entro l'intervallo di time out di 65 millesecondi.
2	TIME OUT ON TALKER	Il dispositivo IEEE non ha fornito un segnale attivo di"dato valido"non ha cioe' messo il segnale DAV allo stato basso entro l'intervallo di time out di 65 millesecondi.
64	END OR IDENTIFY (EOI)	Il segnale EOI si e' portato a livello basso quando l'ultimo byte dei dati e' stato trasmesso sul bus IEEE.Si noti che non tutti i periferici generano un segnale EOI. E' necessario consultare per ciascun periferico il relativo manuale.
-128	DEVICE NOT PRESENT	Il periferico richiesto non risponde quando viene interrogato;questo fatto da luogo ad un messaggio d'errore e il sistema operativo si riporta a livello di comando del PET.

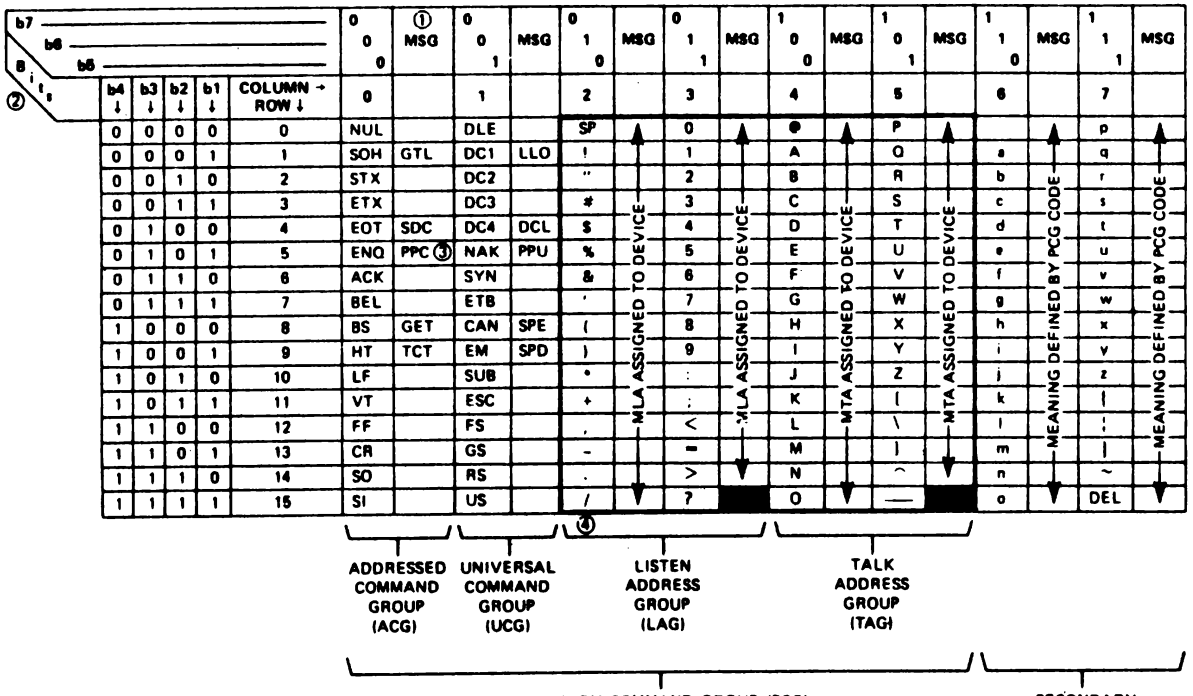
tabella 7.24

IEEE-488 REGISTRO DEGLI INDIRIZZI

La tabella 7.24 illustra gli indirizzi hardware del bus IEEE-488 utilizzati nel PET. Un tentativo di controllare il bus attraverso le istruzioni di PEEK o POKE,puo' dar luogo a risultati non soddisfacenti se vengono superati gli intervalli di time out previsti per i dispositivi 488.

Hex Address	Decimal Address	Bits	IEEE	Mode
E820	59424	0-7	DI01-8	Input
E822	59426	0-7	DI01-8	Output
E821	59425	3	NDAC	Output
E823	59427	3	DAV	Output
		7	SRQ	Input
E810	59408	6	EOI	Input
E840	59456	0	NDAC	Input
		1	NRFD	Output
		2	ATN	Output
		6	NRFD	Input
		7	DAV	Input

Table 7.25. IEEE-488 hardware addresses and signal information.



NOTES: ① MSG = INTERFACE MESSAGE

② b_i = DI01 . . . b_7 = DI07

③ REQUIRES SECONDARY COMMAND

④ DENSE SUBSET (COLUMN 2 THROUGH 5). ALL CHARACTERS USED IN BOTH COMMAND & DATA MODES.

Table 7.26. Code assignments for "Command Mode" of operation.
(SENT AND RECEIVED WITH ATN TRUE)

CAPITOLO 8

USO DEL PET PER LA PROGRAMMAZIONE IN LINGUAGGIO MACCHINA

I programmi in linguaggio macchina vengono eseguiti molto piu' rapidamente che i programmi BASIC che devono venir interpretati prima di essere eseguiti. Sul PET, il linguaggio macchina puo' essere usato per comunicare con le porte d'utente, eseguire della musica o scrivere sullo schermo ad alta velocita'. Se non avete mai provato a programmare il 6502, e' opportuno che vi procuriate e consultiate i due libri menzionati nel Capitolo 1, prima di procedere alla lettura di questo capitolo. Nel PET vi sono due modi per creare in memoria un programma in linguaggio macchina ed eseguirlo. Il primo si avvale del BASIC.Come messo in luce precedentemente, vi sono due istruzioni BASIC:l'istruzione PEEK e l'istruzione POKE che permettono di operare come con linguaggio macchina per controllare le istruzioni INPUT o OUTPUT o per caricare e leggere delle locazioni individuali di memoria.Il secondo metodo per programmare e' quello del monitor.
Un monitor ha essenzialmente tre funzioni:esamina e deposita dei byte in memoria e salta al codice d'esecuzione.Queste operazioni sono realizzabili attraverso le istruzioni PEEK,POKE e SYS nel BASIC.La principale limitazione del BASIC sta nel fatto che tutti i byte devono venir convertiti in codice decimale prima di usarli. Un monitor disponibile per il PET permette di lavorare interamente in notazione esadecimale,tuttavia al 6502 non interessa affatto in che base voi lavoriate in quanto la sua rapprsentazione interna e' binaria.

LA PROGRAMMAZIONE IN LINGUAGGIO MACCHINA ATTRAVERSO L'USO DEL BASIC

Con l'uso dell'istruzione POKE e' possibile costruire in una stringa di locazione di memoria un insieme di istruzioni che siano una subroutine in linguaggio macchina e che possa essere usata da un programma individuale.Per implementare questa subroutine sono necessarie 4 considerazioni fondamentali:

1) cosa la subroutine debba fare,
2) come implementarla,
3) dove caricare il programma,
4) come stabilire la comunicazione fra la subroutine ed il BASIC.

La decisione su cosa il programma debba fare e come implementarlo e' lasciata al programmatore attraverso l'uso del manuale di programmazione del 6502. Per allocare il programma e' necessario decidere se esso sia un piccolo programma da usare solo temporaneamente o se viceversa tale programma debba essere operativo per tutto il tempo in cui il programma BASIC e' operativo in macchina. Per capire dove sia meglio caricare il programma nella memoria e' bene rivedere la mappa della memoria del PET. Tutti gli indirizzi di programma di pagina zero vengono utilizzati dal sistema operativo e sono di solito modificati da programma a programma. La pagina 1 di memoria viene usata dall'uso normale dello stack e delle correzioni di INPUT/OUTPUT da nastro. La pagina 2 contiene una serie di variabili che vengono usate durante l'esecuzione dei programmi.Quindi le locazioni di memoria da 634 a 1023 vengono usate come buffer per la prima e la seconda cassetta.Se un programma non usa ingressi o uscite verso il nastro magnetico queste aree non verranno utilizzate dal BASIC.

Se viene usata solamente la prima cassetta, l'area relativa al buffer della seconda cassetta e' disponibile. Se ambedue le cassette vengono usate durante il programma, oppure se tale area non e' sufficente per scriverci il programma,allora lo spazio tra la fine del programma BASIC e quello in cui il BASIC immagazzina le sue variabili si rende disponibile al programmatore.In qualsiasi istante durante l'esecuzione di un programma, un'operazione di PEEK nelle locazioni 124 e 125 permette di conoscere la locazione di inizio della zona di memoria in cui il BASIC sistema le variabili. La zona di memoria che si trova al di sotto di queste locazioni non verra' utilizzata durante l'esecuzione del programma BASIC e potra' essere usata con l'accortezza di adottare un piccolo margine di sicurezza proporzionale alla dimensione dello spazio dati usato dal programma BASIC.Le locazioni di memoria descritte possono venir contate dall'istruzione FRE. Il problema finale consiste nel come caricare il programma nelle locazioni di memoria. Sebbene mediante l'uso di un programma di monitor si possano caricare dei programmi in linguaggio macchina, Questa procedura involve due passi di caricamento, dapprima deve essere caricato il programma in linguaggio macchina, poi deve essere caricato il programma BASIC.Ovviamente questa procedura non e' sempre valida, se il programma deve essere caricato nei buffer delle cassette magnetiche.Un'altra tecnica e' quella di assemblare il programma all'interno di un programma BASIC, caricando le istruzioni in linguaggio macchina nelle istruzioni DATA.Le istruzioni DATA possono poi essere lette all'inizio dell'esecuzione del programma BASIC e caricate mediante l'istruzione POKE nell'appropriate locazioni di memoria.

IL COMANDO SYS

Quando si debba trasferire il controllo al programma in linguaggio macchina,vi sono due modi diversi di procedere.La via che e' preferibile sta nell'istruzione SYS che trasferisce totalmente il controllo dal BASIC al programma in linguaggio macchina, alla fine il controllo puo' venir restituito al BASIC per mezzo di un mormale ritono da subroutine. L'istruzione SYS puo' essere usata per trasferire il controllo a qualsiasi altro programma, quale il monitor in linguaggio macchina o linguaggi successivi che possono rendersi in futuro disponibili. Se viene incontrata la seguente istruzione:

```
10 SYS (634)
```

alla linea 10 il BASIC cede il controllo del computer al programma allocato alla posizione di memoria 634.La forma generale dell'istruzione SYS e':

```
SYS (indirizzo di partenza)
```

L'indirizzo di partenza puo' essere un valore calcolato,comunque esso deve dar luogo a un numero positivo non piu' grande di 65535.

NOTA IMPORTANTE: l'esecuzione di programmi in linguaggio macchina non tiene conto di tutte quelle protezioni che il BASIC possiede per continuare a funzionare anche quando l'utente commette un errore.Da questo punto di vista quindi ogni volta che il controllo viene trasferito dal BASIC ad un vostro programma si puo' avere una situazione in cui degli errori di programma in linguaggio macchina causino la cessazione del funzionamento della macchina stessa.

Per aiutarvi a risolvere questo tipo di problema, e' bene usare un monitor in linguaggio macchina, quando si debbano sviluppare dei programmi che non siano dei banali programmi di piccolissime dimensioni.In ogni caso, quando il sistema perde qualsiasi possibilita' di controllo, l'unico intervento possibile e' di spegnere e riaccendere immediatamente dopo la macchina. Per restituire il controllo al BASIC l'ultima istruzione del programma in linguaggio macchina che viene eseguita deve essere una istruzione RTS. Il BASIC allora riiniziera' ad interpretare l'istruzione che si trova immediatamente dopo l'istruzione SYS.Per trasferire dati da e per il programma di utente i dati devono venir caricati tramite l'istruzione POKE in delle locazioni di memoria che siano temporaneamente indisturbate durante l'esecuzione della routine BASIC. I risultati del programma avviato con l'istruzione SYS possono essere letti all'interno del programma BASIC attraverso l'istruzione di PEEK.

LA FUNZIONE USR

Esistono parecchi programmi,in particolare routine matematiche, per i quali e' piu' conveniente passare parametri da e per il BASIC usando la funzione USR in modo da ottenere risultati direttamente processabili dal BASIC. L'istruzione USR viene specificata con un parametro. Il BASIC calcolera' l'espressione per i suoi parametri e colloca il risultato del calcolo in un accumulatore in virgola mobile che il BASIC usa per tutte le sue funzioni.Si noti che se nessun parametro viene trasferito, l'accumulatore in virgola mobile non viene inizializzato dall'utente o tramite qualsiasi altra tecnica, mentre prima dell'esecuzione della funzione USR e' stato usato dal BASIC in un gran numero di modi. USR richiama una routine che esegue un programma in linguaggio macchina. Poiche' USR e' una funzione, e' possibile includere la funzione come parte di un'istruzione BASIC, come ad esempio in: IF USR(A)=1,THEN ecc.In questo caso il parametro A verra' passato alla funzione USR in accumulatore in virgola mobile.Il risultato dell'accumulazione in virgola mobile, quando l'utilizzatore ritorna al BASIC,verra' comparato a 1 e verra' eseguita la funzione logica. Il comando SYS e' il modo piu' usuale per trasferire il controllo a una procedura in linguaggio macchina in cui le variabili del programma principale non vengono usate. USR e' il modo piu' usuale quando si vuol implementare un nuovo comando BASIC. Vi e' un'importante considerazione da fare quando si usi USR. USR adopera delle locazioni per le variabili pre-assegnate: le locazioni 1 e la 2.Queste locazioni devono essere inizializzate con il valore esadecimale dell'indirizzo di partenza in cui il programma in linguaggio macchina e' memorizzato. Cio' puo' essere fatto in qualsiasi momento nel programma principale attraverso l'istruzione di POKE caricando l'equivalente decimale e la parte meno significativa dell'indirizzo nella locazione 2 e la parte piu' significativa dell'indirizzo nella locazione 1, ad esempio:

```
10 POKE 1,122
20 POKE 2,2
30 IF USR(A)=1THEN ecc
```

SUBROUTINE BASIC UTILI

Vi e' una serie di subroutine nel BASIC che permettono ai programmi in linguaggio macchina di calcolare valori nell'accumulatore in virgola mobile.

Queste funzioni vengono chiamate istruzioni di salto a subroutine. Il parametro specificato nella funzione USR viene calcolato, convertito nell'equivalente binario in virgola mobile con segno esponente a mantissa e caricato in una serie di 6 byte che vengono chiamati accumulatori in virgola mobile. In questi 6 byte vengono caricati rispettivamente:

$5E segno e esponente
$5F la mantissa con il suo byte piu' significativo
$60 byte della mantissa
$61 byte della mantissa
$62 byte della mantissa
$63 il byte meno significativo della mantissa
$64 segno della mantissa

L'esponente viene calcolato in modo che il numero sia rappresentato in forma normalizzata. Questo esponente viene memorizzato come un numero binario assoluto con segno.Gli esponenti negativi,infatti,non vengono memorizzati nella rappresentazione complemento A2. Il massimo esponente e' 10 alla 28. Il minimo esponente e' 10 alla meno 39 e viene caricato come 00.Un esponenete 0 viene usato per segnalare che il numero e' pari a 0.

ESPONENTE	VALORE APPROSSIMATO
FF	10
A2	10
7F	10
02	10
00	10

Poiche' l'esponente e' in realta' una potenza di 2,esso potrebbe essere interpretato come il numero di spostamenti a sinistra o spostamenti a destra a seconda che tale esponente sia positivo o negativo che e' necessario compiere sulla mantissa normalizzata per ottenere la rappresentazione binaria del valore memorizzato.Poiche' la mantissa e' sempre in forma normalizzata,il bit piu' alto del byte significativo e' sempre 1.Questo garantisce sempre almeno una precisione di 40 byte che e' approssimativamente equivalente a 9 cifre significative decimali con alcuni bit per l'arrotondamento. Se un numero vale 0,esso non avra' dunque i byte tutti 0 nella mantissa.L'unico contrassegno per uno 0 e' l'esponente.Ci si riferisca alla figura 8.1 per avere degli esempi di esponenti e di mantissa. Se la mantissa e' positiva, allora il suo byte di segno e' 0.Una mantissa negativa fa si che questo byte sia -1.

EXAMPLE FLOATING POINT NUMBERS

	exponent	mantissa	mantissa	mantissa	mantissa	Sign of mantissa
1E38	FF	96	76	99	52	00
4E10	A4	95	02	F9	00	00
2E10	A3	95	02	F9	00	00
1E10	A2	95	02	F9	00	00
1	81	80	00	00	00	00
.5	80	80	00	00	00	00
.25	7F	80	00	00	00	00
1E-4	73	D1	B7	59	59	00
1E-37	06	88	IC	14	14	00
1E-38	02	D9	C7	EE	EE	00
1E-39	00	A0	00	00	00	00
0	00	00	00	00	00	
-1	81	80	00	00	00	FF
-10	84	A0	00	00	00	FF

Figure 8.1.Example floating point numbers.

Le variabili in virgola mobile del BASIC sono memorizzate in 5 byte anziche' in 6 byte come nell'accumulatore in virgola mobile.Da quanto e' gia'stato detto, infatti,si puo'notare che il byte piu' significativo della mantissa ha sempre il bit piu' alto diverso da 0. Se si e' sicuri, quindi, che il numero sara' sempre rappresentato in questa forma si puo' utilizzare questo bit per indicare il segno della mantissa, liberando quindi il byte usato per il segno. Il sesto byte viene usato nell'accumulatore in virgola mobile per semplificare quando la mantissa viene ruotata.Il contenuto dell'accumulatore in virgola mobile puo' essere convertito in un intero su due byte chiamando la subroutine FLPINT che e' memorizzata all'indirizzo esadecimale D0A7.Il byte piu' significativo dell'intero viene restituito alla locazione esadecimale B3 e il meno significativo nella locazione esadecimanle B4.

```
        10 A=USR(2)
 contents of FAC after USR call
    82 80 00 00 00 00
          JSR FLPINT
 contents of FAC after conversion
 82 00 00  I  00 02  I  00 00
           integer value
```

Un intero puo' essere convertito in un numero in rappresentazione in virgola mobile caricando il byte piu' significativo nel registro A e il byte meno significativo nel registro Y, quindi chiamando la routine INTFLP alla locazione esadecimale D278.

```
L D A  MSB
LDY    LSB
JSR    INTFLP
USABLE I/O ROUTINES
Read a line,pass a character
             $FFCF return cher in 0
             no other ress changes
Print a character on screen
             $FFD2 Char in A
             no ress changed
Test for stop key
             $FFE1 returns=,<>
             only A changed
Get a character from keyboard
             $FFE4
             char or if nome null(00)
```

SOMMARIO

Vi sono due modi per comunicare fra il BASIC e un programma in linguaggio macchina.Il piu' semplice di essi e' l'istruzione SYS mediante la quale il controllo del calcolatore viene passato al programma in linguaggio macchina che si trova memorizzato all'indirizzo specificato nel comando SYS.Per implementare le proprie funzioni vi e' una funzione chiamata USR che quando le locazioni di memoria 1 e 2 vengono propriamente inizializzate in modo da puntare su un programma in linguaggio macchina, calcola un parametro specificato nella funzione stessa e restituisce il risultato al programma principale usando l'accumulatore in virgola mobile. Una serie di subroutine di utilita', disponibili nel BASIC, permettono sia alla funzione USR o al comando SYS di fare delle operazioni sull'accumulatore in virgola mobile senza che il programmatore debba girare e scrivere dei programmi se non quelli chiamati.In tutti i casi, l'uso di programmi in linguaggio macchina,e' riservato ai programmatori molto esperti.In questo caso la protezione delle ROM per errori di codice viene perduta.I programmi in linguaggio macchina vengono di solito adoperati quando il BASIC o non e' sufficentemente veloce oppure la funzione che si desidere non e' implementata.

PROGRAMMA MONITOR IN LINGUAGGIO MACCHINA TIM e' il programma TERMINAL INTERFACE MONITOR per i microprocessori MOS TECHNOLOGY'S 65XX.Esso e' stato espanso e adattato per funzionare sul PET COMMODORE. Il PET usa la versione a cassetta magnetica di questo monitor. L'esecuzione viene trasferita dall'interno del BASIC al TIM nel comando SYS. Per caricare il vostro monitor e' necessario prendere la cassetta con il programma MONITOR e caricarla sull'unita' a nastro,con il lato monitor rivolto verso l'alto.Battere poi LOAD"MONITOR" e quando il programma e'stato caricato, battere RUN.I comandi inviati alla tastiera del PET fanno si che il programma TIM possa iniziare l'esecuzione di un programma, visualizzarlo modificare registri e locazioni di memoria o caricare in memoria o su nastro magnetico dati binari.Quando si modifica la memoria, il programma TIM esegue automaticamente la lettura dopo aver fatto una scrittura in modo da assicurarsi che all'indirizzo di memoria specificata ci sia memoria di tipo scrivibile e che essa risponda correttamente.Il programma TIM e' provvisto di diverse subroutine che possono essere richiamate dal programmatore.

Queste includono le possibilita' di leggere o scrivere caratteri sul display video, di scrivere un byte in formato esadecimale e di stampare una sequenza di ritorno carrello avvanzamento riga. I programmi del comando TIM sono i seguenti:

M visualizzazione del contenuto della memoria.
R visualizzazione del contenuto dei registri.
G inizio dell'esecuzione.
X ritorno al BASIC.
L caricamento di una posizione di memoria.
S scrittura su nastro magnetico.

Esempio:

```
    M DISPLAY MEMORY
   .M C000,C010
   .: C000 1D C7 48 C6 35 CC EF C7
   .: C008 C5 CA DF CA 70 CF 23 CB
   .: C010 9C C8 9C C7 74 C7 1F C8
```

Quando si vuole inviare un comando di visualizzazione di contenuto della memoria, l'indirizzo di partenza e di fine devono essere specificati mediante numeri esadecimali a 4 cifre.Quando il cursore viene mosso per eseguire un correzione e viene premuto il RETURN il doppio punto informa il monitor che e'stato fatto entrare un nuovo dato.

```
    R DISPLAY REGISTERS
   .R PC SR AC XR YR SP
   .: C6 ED 00 20 00 F5
```

I registri vengono messi da parte e rimessi poi al loro posto dopo ciascuna entrata e uscite dal programma TIM. Essi possono essere modificati o precaricati come nell'esempio di visualizzazione della memoria appena citato. Il punto e virgola informa il monitor che i registri sono stati modificati.

```
    G BEGIN EXETUTION
   .G C38B
```

Il comando GO puo' avere come opzione l'indirizzo del programma bersaglio. Se non si specifica nulla, il PC viene preso come bersaglio e il contenuto del PC puo' essere posizionato mediante l'istruzione R appena vista.

```
    X EXIT TO BASIC
   .X


READY
```

Il comando X fa si che il BASIC riassuma il comando. Durante questa operazione la memoria non viene alterata in nessun modo e il BASIC riprende le operazioni allo stesso modo in cui operava prima che il monitor fosse richiamato.

```
    L LOAD
   .L,"PROGRAM NAME",01
```

Con il comando LOAD non si puo' omettere nessun parametro. Il numero di periferico e il nome di file devono essere completamente specificati.I comandi al sistema operativo per gli interventi dell'operatore sono gli stessi che per il BASIC.

Gli indirizzi di memoria sono caricati come specificati nell'intestazione del file, che viene registrato in conseguenza di un comando di SAVE. Le SUBROUTINE in linguaggio macchina possono essere caricate dal BASIC, ma deve essere posta molta cura a non usare delle variabili BASIC come pointer delle variabili.

```
   S SAVE
  .S,"PROGRAM NAME",01,0400,076D
```

Ovviamente anche per il comando SAVE non puo' essere omesso alcun parametro.Sia l'indirizzo di partenza che quello di fine devono essere specificati. Per cancellare il comando si puo' usare sia il tasto del RETURN che il tasto di STOP, cio' fa cancellare il comando di DISPLAY MEMORY,LOAD oppure SAVE.

INTERRUZIONIE PUNTI D'ARRESTO

BRK e' un'istruzione d'interruzione software che fa si che la CPU interrompa l'esecuzione, metta da parte il PC e registri P nello stack e quindi salti attraverso un vettore alle locazioni esadecimali 21B e 21C. Il programma TIM inizializza questo vettore in modo che punti allo stesso ingresso della CALL. A meno che l'utente non modifichi questo vettore, il programma TIM assumera' il controllo quando viene eseguito un'istruzione BRK,stampera' B* per indicare che l'ingresso e' stato fatto attraverso un punto d'arresto anziche' stampare un C* che indica un ingresso attraverso CALL.Esso stampera' altresi' il contenuto dei registri come nel comando R e si mettera' in uno stato di attesa dei comandi se parte dall'utente.Si noti che dopo un'istruzione BRK il PC punta al byte che segue l'istruzione BRK:quindi l'utente puo' sciesliere di manipolare le istruzioni di BRK notando che tale istruzione agisce come un'istruzione a due byte, esso dovra' quindi depositare nel PC due byte al di la dell'istruzione BRK.
IRQ da luogo nel PET a una ISR che aggiorna il clock ed esegue una scansione della tastiera ogni sessantesimo di secondo.Se il vettore viene alterato la subroutine in linguaggio macchina non lo rinizializza e sara' necessario fare un reset attraverso lo spegnimento dell'apparecchiatura.
NMI non e' disponibile nel PET. La linea nel microprocessore che corrisponde a questo interrup e' permanentemente mantenuta alta.REST porta ad una inizializzazione del BASIC.La memoria viene cancellate,ed e' necessario ricaricare e riavviare il programma TIM attraverso un comando SYS.

CHIAMATE DEL PROGRAMMA MONITOR TIM E LOCAZIONI SPECIALI

JSR	WRT	$FFD2	type a character
JSR	RDT	$FFCF	input a character
JSR	GET	$FFE4	get a character
JSR	CRLF	$FDDO	type a CR
JSR	SPACE	$FDCD	type a space
JSR	WROB	$E775	type a byte
JSR	RDOB	$E7B6	read a byte
JSR	HEXIT	$E7EO	Ascii ti hex in A

USO DELLA MEMORIA

$0A-$22	zero page
$400-$76A	absolute RAM

Le locazioni esadecimali 23 e 5A sono locazioni di pagina zero relative ai buffer d'ingresso del BASIC che possono essere usati quando il BASIC non le usa. Il secondo buffer per le cassette alle locazioni esadecimali 33A e 3FF sono locazioni ben protette se il periferico non e' presente. Le altre locazioni di memoria possono essere usate,ma con considerevole rischio,dipendendo da qual'e' la sezione del PET che viene utilizzata in quel momento.

PROCEDURA DI CHECKOUT DEL MONITOR

1) Accendere il PET normalmente in modo di portarsi a livello di comando del BASIC.Battere sulla tastiera SYS 1024,verra' allora mostrato sullo schermo la seguente sequenza:

```
   B* PC SRAC XR YR SP INV
  .;  29 00 88 89 FE FF E68A
```

I valori riportati in questo esempio possono, o essere diversi da quelli che apparirranno sullo schermo del PET.Tuttavia, il primo e l'ultimo valore dovrebbero coincidere con quelli dell'esempio.

2) La visualizzazione dei registri e il messaggio standard di visualizzazione.Esso contiene un C* per identificare l'ingresso fatto attraverso chiamata, seguito dal contenuto dei registri della CPU:il contatore di programma,lo stato del processore,l'accumulatore,il registro indice X, il registro indice Y e lo stack pointer.E' da notare che tutti gli ingressi e le uscite del programma TIM sono in base 16, sono cioe' cifre esadecimali, le cifre utilizzate nell'annotazione esadecimale sono: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Dopo aver stampato il contenuto dei registri della CPU,il programma TIM, e' pronto a ricevere comandi. TIM indica questo stato di attesa stampando il carattere di avviso che consiste in un punto su una nuova linea.

3) Il contenuto dei registri della CPU possono anche essere visualizzati attraverso l'uso del comando R. Battendo una R e premendo il tasto di RETURN il programma di monitor rispondera' come appena descritto, ma senza l'asterisco.

4) I valori visualizzati posono essere modificati attraverso il programma di edit dello schermo e la linea puo' essere reinserita in memoria premendo poi il tasto di RETURN.Si ricordi di battere gli spazi che delimitano i campi e utilizzare numeri decimali di 4 cifre per gli indirizzi e di due cifre per il contenuto di ciascun byte.

5) La memoria puo' essere visualizzata e modificata usando il comando M. Battendo:

```
     .M 0100 0107
```

si avra' sullo schermo la visualizzazione della seguente stringa:

```
     0 1  3  4  5  6  7
     .:0100 20 00 30 30 30 30 30
```

Anche questi dati possono essere modificati attraverso l'uso del programma di edit e reinseriti in memoria premendo il tasto di RETURN.

6) Si usi ora il comando M e il ; per inserire il seguente programma di test di nome CHSET esso permette di stampare sul terminale il set di 64 caratteri ASCII.

Il comando M viene usato per visualizzare le locazioni di memoria sullo schermo del PET.

Poi sara' possibile usare il programma di correzione su ciascuna linea insieme al tasto di RETURN per modificare la memoria.

```
*=$33A
CRLF=$4F2 FDDO
WRT=$FFD2
33A 20 F2 04 ;     CHSET JSR CRLF
33D A2 20                LDX #$20
33F 8A             LOOP  TXA
340 20 D2 FF             JSR WRT
343 E8                   INX
344 E0 60                CPX #$60
346 D0 F7                BNE LOOP
348 00                   BRK
349 4C 3A 03             JMP CHSET
.M  033A,034B
.: 033A 20 F2 04 A2 20 8A 20 D2
.: 0342 FF E8 E0 60 D0 F7 00 4C
.: 034A 3A 03
```

7) Il programma CHSET e' assemblato in modo da risiedere nel buffer della seconda cassetta. Battendo sulla tastiera:

```
.GO033A
```

tale programma andra' in esecuzione.Esso dara' luogo sullo schermo alla seguente lista:

```
!"#$%'()*,-./0123456789:;=?&ABCDEFGHIJKLMNOPQRSTUVWXYZ[/]
B* PC SR AC XR YR SP INV
  0349 3B 5F 60 8D FE E68A
```

si noti l'indirizzo contenuto nel PC. E' possibile battendo il tasto G rieseguire nuovamente un programma senza specificare nessun indirizzo.

8) Si voglia ora collegare il programma CHSET con il BASIC. Per prima cosa e' necessario rimpiazzare l'istruzione di BRK alla locazione esadecimale 348 con un RTS (RITORNO DA SUBROUTINE) cio' consiste nel cambiare il contenuto della locazione 348 da 00 a 60.

9) Si cambi il vettore della funzione USR alle locazioni 1 e 2 in modo che esso punti alla subroutine che inizia alla locazione esadecimale 33A.

```
.:0000 4C 3A 03
```

10) Si esca dal programma di monitor e si rientri nel BASIC attraverso l'istruzione:

```
.X
READY
```

11) Si provi che il collegamento sia stabilito usando sia l'istruzione SYS che USR come comandi diretti nel modo seguente:

```
A=USR(0)
SYS(3*256+3*16+10)
```

```
CBM RESIDENT MONITOR.....PAGE 0001

LINE # LOC     CODE        LINE

2514  FD11                 ;COPYRIGHT 1978 BY
2515  FD11                 ;COMMODORE INTERNATIONAL LIMITED




2517  FD11                 NCMDS   =8
2518  FD11  A9 43          CALLE   LDA #'C          ;CALL ENTRY
2519  FD13  85 B5                  STA TMPC
2520  FD15  D0 16                  BNE B3
2521  FD17  A9 42          BRKE    LDA #'B          ;BREAK ENTRY
2522  FD19  85 B5                  STA TMPC
2523  FD1B  D8                     CLD
2524  FD1C  4A                     LSR A            ;C SET FOR PC CORRECTION
2525  FD1D  68                     PLA
2526  FD1E  8D 05 02               STA YR           ;SAVE Y
2527  FD21  68                     PLA
2528  FD22  8D 04 02               STA XR           ;SAVE X
2529  FD25  68                     PLA
2530  FD26  8D 03 02               STA ACC          ;SAVE ACCUMULATOR
2531  FD29  68                     PLA
2532  FD2A  8D 02 02               STA FLGS         ;SAVE FLAGS
2533  FD2D  68             B3      PLA
2534  FD2E  69 FF                  ADC #$FF         ;PC-1 FOR BREAK
2535  FD30  8D 01 02               STA PCL
2536  FD33  68                     PLA
2537  FD34  69 FF                  ADC #$FF
2538  FD36  8D 00 02               STA PCH
2539  FD39  A5 90                  LDA CINV         ;SAVE CUURENT IRQ VECTOR
2540  FD3B  8D 08 02               STA INVL
2541  FD3E  A5 91                  LDA CINV+1
2542  FD40  8D 07 02               STA INVH
2543  FD43  BA                     TSX              ;SAVE CURRENT STACK POINTER
2544  FD44  8E 06 02               STX SP
2545  FD47  58                     CLI              ;CLEAR INTERRUPT DISABLE
2546  FD48  20 D0 FD       B5      JSR CRLF         ;PRINT ENTRY DATA
2547  FD4B  A6 B5                  LDX TMPC         ;TYPE OF ENTRY (B OR C)
2548  FD4D  A9 2A                  LDA #'*
2549  FD4F  20 84 E7               JSR WRTWO        ;WRITE '*C' OR '*B'
2550  FD52  A9 52                  LDA #'R          ;DISPLAY REGISTERS COMMAND
2551  FD54  D0 1A                  BNE S0           ;SKIP TO INTERPRET COMMAND

2553  FD56  A9 02          STRT    LDA #2           ;USER COMMAND INPUT
2554  FD58  85 77                  STA TXTPTR       ;COMING FROM TEXT BUFFER
2555  FD5A  A9 00                  LDA #0
2556  FD5C  85 BE                  STA WRAP         ;ADDR WRAP AROUND FLAG
2557  FD5E  A2 0D                  LDX #CR          ;START PROMPT WITH CRLF
2558  FD60  A9 2E                  LDA #'.          ;A PROMPTING '.'
2559  FD62  20 84 E7               JSR WRTWO
2560  FD65  20 EB E7       ST1     JSR RDOC         ;INPUT COMMAND LINE
2561  FD68  C9 2E                  CMP #'.          ;IGNORE PROMPTING '
2562  FD6A  F0 F9                  BEQ ST1
2563  FD6C  C9 20                  CMP #$20         ;SPAN BLANKS
2564  FD6E  F0 F5                  BEQ ST1
```

```
CBM RESIDENT MONITOR.....PAGE 0002

LINE # LOC     CODE        LINE

2566  FD70  A2 07        S0      LDX #NCMDS-1     ;LOOKUP COMMAND
2567  FD72  DD E8 FD     S1      CMP CMDS,X
2568  FD75  D0 0B                BNE S2
2569  FD77  86 B4                STX SAVX         ;INDEX OF COMMAND IN TABLE
2570  FD79                ;INDIRECT JMP FROM TABLE BY
2571  FD79                ; PUSHING TARGET ADDRESS-1
2572  FD79                ; THEN RTS
2573  FD79  BD E8 FD              LDA ADRH,X
2574  FD7C  48                    PHA
2575  FD7D  BD F0 FD              LDA ADRL,X
2576  FD80  48                    PHA
2577  FD81  60                    RTS
2578  FD82  CA            S2      DEX
2579  FD83  10 ED                 BPL S1           ;LOOP FOR ALL COMMANDS

2581  FD85  6C FA 03              JMP (USRCMD)     ;ALLOW USER COMMANDS




2583  FD88  A5 FB         PUTP    LDA TMP0
2584  FD8A  8D 81 02              STA PCL
2585  FD8D  A5 FC                 LDA TMP0+1
2586  FD8F  8D 80 02              STA PCH
2587  FD92  60                    RTS

2589  FD93                ;DISPLAY MEM SUBR. SET AR=NUMBER
2590  FD93                ;OF MEMORY BYTES DISPLAYED.
2591  FD93                ;TMP0=ADR OF MEM DISPLAYED
2592  FD93                ;
2593  FD93  85 B5         DM      STA TMPC
2594  FD95  A0 00                 LDY #0
2595  FD97  20 CD FD      DM1     JSR SPACE        ;WR N BYTES
2596  FD9A  B1 FB                 LDA (TMP0),Y     ;(TMP0)=ADR
2597  FD9C  20 75 E7              JSR WRDB
2598  FD9F  20 B5 FD              JSR INCTMP
2599  FDA2  C6 B5                 DEC TMPC
2600  FDA4  D0 F1                 BNE DM1
2601  FDA6  60                    RTS
2602  FDA7                ;READ AND STORE BYTE.
2603  FDA7                ;NO STORE IF SPACE OR TMPC = 0.

2605  FDA7  20 B6 E7      BYTE    JSR RDDB
2606  FDAA  90 0D                 BCC BY3          ;SPACE
2607  FDAC  A2 00                 LDX #0           ;STORE BYTE
2608  FDAE  81 FB                 STA (TMP0,X)
2609  FDB0  C1 FB                 CMP (TMP0,X)     ;VERIFY WRITE
2610  FDB2  F0 05                 BEQ BY3
2611  FDB4  68                    PLA              ;ERROR: CLEAR STACK
2612  FDB5  68                    PLA
2613  FDB6  4C F7 E7              JMP ERROPR
2614  FDB9  20 B5 FD      BY3     JSR INCTMP       ;INC TMP0 ADR
2615  FDBC  C6 B5                 DEC TMPC
2616  FDBE  60                    RTS
```

```
CBM RESIDENT MONITOR.....PAGE 0003

LINE # LOC      CODE        LINE

2618  FDBF  A9 02        SETR   LDA #<FLGS       ;SET TO ACCESS REGS
2619  FDC1  85 FB               STA TMP0
2620  FDC3  A9 02               LDA #>FLGS
2621  FDC5  85 FC               STA TMP0+1
2622  FDC7  A9 05               LDA #5
2623  FDC9  60                  RTS

2625  FDCA  20 CD FD     SPAC2  JSR SPACE
2626  FDCD  A9 20        SPACE  LDA #$20
2627  FDCF  2C                  .BYT $2C
2628  FDD0  A9 0D        CRLF   LDA #$D
2629  FDD2  4C B2 FF             JMP $FFD2

2631  FDD5                      ;INCREMENT (TMP0,TMP0+1) BY 1
2632  FDD5  E6 FB        INCTMP INC TMP0          ;LOW BYTE
2633  FDD7  D0 06               BNE SETUR
2634  FDD9  E6 FC               INC TMP0+1       ;HIGH BYTE
2635  FDDB  D0 02               BNE SETUR
2636  FDDD  E6 BE               INC WRAP
2637  FDDF  60           SETUR  RTS



2639  FDE0                      ;COMMAND AND ADDRESS TABLE

2641  FDE0  3A           CMDS   .BYT ':'          ;MODIFY MEMORY
2642  FDE1  3B                  .BYT ';'          ;ALTER REGISTERS
2643  FDE2  52                  .BYT 'R'          ;DISPLAY REGS
2644  FDE3  4D                  .BYT 'M'          ;DISPLAY MEMORY
2645  FDE4  47                  .BYT 'G'          ;START EXECUTION
2646  FDE5  58                  .BYT 'X'          ;WARM START BASIC
2647  FDE6  4C                  .BYT 'L'          ;LOAD MEMORY
2648  FDE7  53                  .BYT 'S'          ;SAVE MEMORY
2649  FDE8  FE           ADRH   .BYT >ZZ1
2650  FDE9  FE                  .BYT >ZZ2
2651  FDEA  FE                  .BYT >ZZ3
2652  FDEB  FE                  .BYT >ZZ4
2653  FDEC  FE                  .BYT >ZZ5
2654  FDED  FF                  .BYT >ZZ6
2655  FDEE  FF                  .BYT >ZZ7
2656  FDEF  FF                  .BYT >ZZ8
2657  FDF0  B8           ADRL   .BYT <ZZ1
2658  FDF1  96                  .BYT <ZZ2
2659  FDF2  22                  .BYT <ZZ3
2660  FDF3  57                  .BYT <ZZ4
2661  FDF4  CE                  .BYT <ZZ5
2662  FDF5  B6                  .BYT <ZZ6
2663  FDF6  10                  .BYT <ZZ7
2664  FDF7  10                  .BYT <ZZ8
```

CBM RESIDENT MONITOR.....PAGE 0004

```
LINE # LOC      CODE        LINE

2666  FDF8  0D            REGK    .BYT CR.
2666  FDF9  20 20
2667  FDFD  20 50                 .BYT ' PC  IRQ  SR AC XR YR SP'

2669  FE15  98            ALTRIT TYA
2670  FE16  48                    PHA
2671  FE17  20 D8 FD              JSR CRLF
2672  FE1A  68                    PLA
2673  FE1B  A2 2E                 LDX #'.
2674  FE1D  20 84 E7              JSR WRTWO
2675  FE20  4C CA FD              JMP SPAC2

2677  FE23  A2 00         DSPLYR LDX #0
2678  FE25  BD F8 FD     D2      LDA REGK,X
2679  FE28  20 D2 FF              JSR $FFD2
2680  FE2B  E8                    INX
2681  FE2C  E0 1D                 CPX #29
2682  FE2E  D0 F5                 BNE D2
2683  FE30  A0 3B                 LDY #';
2684  FE32  20 15 FE              JSR ALTRIT
2685  FE35  AD 00 02              LDA PCH
2686  FE38  20 75 E7              JSR WROB
2687  FE3B  AD 01 02              LDA PCL
2688  FE3E  20 75 E7              JSR WROB
2689  FE41  20 CD FD              JSR SPACE
2690  FE44  AD 07 02              LDA INVH
2691  FE47  20 75 E7              JSR WROB
2692  FE4A  AD 08 02              LDA INVL
2693  FE4D  20 75 E7              JSR WROB
2694  FE50  20 BF FD              JSR SETR
2695  FE53  20 93 FB              JSR DM
2696  FE56  F0 39                 BEQ BEQS1

2698  FE58  20 EB E7     DSPLYM JSR RDOC
2699  FE5B  20 A7 E7              JSR RDOA        ;READ START ADR
2700  FE5E  90 34                 BCC ERRS1       ;ERR IF NO SA
2701  FE60  20 97 E7              JSR T2T2        ;SA TO TMP2
2702  FE63  20 EB E7              JSR RDOC        ;SKIP DELIMITER
2703  FE66  20 A7 E7              JSR RDOA        ;READ END ADR
2704  FE69  90 29                 BCC ERRS1       ;ERR IF NO EA
2705  FE6B  20 97 E7              JSR T2T2        ;SA TO TMP0, EA TO TMP2

2707  FE6E  20 81 F3     DSP1    JSR STOP1       ;TEST FOR STOP KEY
2708  FE71  F0 1E                 BEQ BEQS1
2709  FE73  A6 BE                 LDX WRAP
2710  FE75  D0 1A                 BNE BEQS1
2711  FE77  38                    SEC             ;DOUBLE BYTE COMPARE
2712  FE78  A5 FD                 LDA TMP2
2713  FE7A  E5 FB                 SBC TMP0
2714  FE7C  A5 FE                 LDA TMP2+1
2715  FE7E  E5 FC                 SBC TMP0+1
2716  FE80  90 0F                 BCC BEQS1       ;EA LESS THAN SA
2717  FE82  A0 3A                 LDY #':
2718  FE84  20 15 FE              JSR ALTRIT
2719  FE87  20 6A E7              JSR WRDA
```

```
CBM RESIDENT MONITOR.....PAGE 0005

LINE # LOC     CODE        LINE

2720  FE8A  A9 00           LDA #0
2721  FE8C  20 93 FD        JSR DM          ;DISPLAY 0, INCR TMP0
2722  FE8F  F0 DD           BEQ DSP1

2724  FE91  4C 56 FD    BEQS1  JMP STRT

2726  FE94  4C F7 E7    ERRS1  JMP ERROPR

2728  FE97               ;ALTER REGISTERS

2730  FE97  20 B6 E7     ALTR   JSR RDDB        ;SKIP 2 SPACES
2731  FE9A  20 A7 E7            JSR RDDA        ;CY=0 IF SP
2732  FE9D  90 03               BCC AL2         ;SPACE
2733  FE9F  20 88 FD            JSR PUTP        ;ALTER PC
2734  FEA2  20 CF FF     AL2    JSR $FFCF
2735  FEA5  20 A7 E7            JSR RDDA
2736  FEA8  90 0A               BCC AL3
2737  FEAA  A5 FB               LDA TMP0
2738  FEAC  8D 08 02            STA INVL
2739  FEAF  A5 FC               LDA TMP0+1
2740  FEB1  8D 07 02            STA INVH
2741  FEB4  20 BF FD     AL3    JSR SETR        ;SET TO ALTER R'S
2742  FEB7  D0 0A               BNE A4

2744  FEB9               ;ALTER MEMORY - READ ADR AND DATA

2746  FEB9  20 B6 E7     ALTM   JSR RDDB        ;SKIP 2 SPACES
2747  FEBC  20 A7 E7            JSR RDDA        ;READ MEM ALTER ADR
2748  FEBF  90 D3               BCC ERRS1       ;CY=0, IF SPACE,ERR
2749  FEC1  A9 08               LDA #8          ;SET CNT = 8
2750  FEC3  85 B5        A4     STA TMPC
2751  FEC5  20 EB E7     A5     JSR RDDC
2752  FEC8  20 A7 FD            JSR BYTE
2753  FECB  D0 F8               BNE A5
2754  FECD  F0 C2        A9     BEQ BEQS1
2755  FECF  20 CF FF     GO     JSR $FFCF
2756  FED2  C9 0D               CMP #$0D        ;IF CR, EXIT
2757  FED4  F0 0C               BEQ G1
2758  FED6  C9 20               CMP #$20        ;IF NOT SPACE, ERR
2759  FED8  D0 BA               BNE ERRS1
2760  FEDA  20 A7 E7            JSR RDDA
2761  FEDD  90 03               BCC G1
2762  FEDF  20 88 FD            JSR PUTP
2763  FEE2  AE 06 02     G1     LDX SP
2764  FEE5  9A                  TXS             ;ORIG OR NEW SP VALUE TO SP
2765  FEE6  78                  SEI
2766  FEE7  AD 07 02            LDA INVH
2767  FEEA  85 91               STA CINV+1
2768  FEEC  AD 08 02            LDA INVL
2769  FEEF  85 90               STA CINV
2770  FEF1  AD 00 02            LDA PCH
2771  FEF4  48                  PHA
2772  FEF5  AD 01 02            LDA PCL
2773  FEF8  48                  PHA
2774  FEF9  AD 02 02            LDA FLGS
```

```
CBM RESIDENT MONITOR.....PAGE 0006

LINE # LOC      CODE        LINE

2775  FEFC  48              PHA
2776  FEFD  AD 03 02        LDA ACC
2777  FF00  AE 04 02        LDX XR
2778  FF03  AC 05 02        LDY YR
2779  FF06  40              RTI

2781  FF07  AE 06 02  EXIT  LDX SP
2782  FF0A  9A              TXS
2783  FF0B  4C 89 C3        JMP READY       ;EXIT TO BASIC WARM STAR

2785  FF0E  4C F7 E7  ERRL  JMP ERROPR

2787  FF11            ZZZ1  =BUF+7

2789  FF11            ;MACHINE LANGUAGE LOAD ROUTINE

2791  FF11  A0 01     LD    LDY #1
2792  FF13  84 D4           STY FA          ;DEFAULT DEVICE #1
2793  FF15  88              DEY
2794  FF16  84 D1           STY FNLEN
2795  FF18  84 9D           STY VERCK
2796  FF1A  A9 02           LDA #>ZZZ1      ;PLACE TO STORE NAME
2797  FF1C  85 DB           STA FNADR+1
2798  FF1E  A9 07           LDA #<ZZZ1
2799  FF20  85 DA           STA FNADR
2800  FF22  20 CF FF  L1    JSR $FFCF
2801  FF25  C9 20           CMP #' '
2802  FF27  F0 F9           BEQ L1          ;SPAN BLANKS
2803  FF29  C9 0D           CMP #CR
2804  FF2B  F0 1A           BEQ L5          ;DEFAULT TO LOAD
2805  FF2D  C9 22           CMP #'"'
2806  FF2F  D0 DD     L2    BNE ERRL        ;FILE NAME MUST BE NEXT
2807  FF31  20 CF FF  L3    JSR $FFCF
2808  FF34  C9 22           CMP #'"'
2809  FF36  F0 24           BEQ L8          ;END OF NAME
2810  FF38  C9 0D           CMP #CR         ;DEFAULT A LOAD
2811  FF3A  F0 0B           BEQ L5
2812  FF3C  91 DA           STA (FNADR)Y
2813  FF3E  E6 D1           INC FNLEN
2814  FF40  C8              INY
2815  FF41  C0 10           CPY #16
2816  FF43  F0 C9     L4    BEQ ERRL        ;FILE NAME TOO LONG
2817  FF45  D0 EA           BNE L3
2818  FF47  A5 B4     L5    LDA SAVX
2819  FF49  C9 06           CMP #6
2820  FF4B  D0 E2     L6    BNE L2          ;NOT A LOAD
2821  FF4D  20 22 F3        JSR LD15
2822  FF50  20 E6 F8        JSR TWAIT
2823  FF53  A5 96           LDA SATUS
2824  FF55  29 10           AND #SPERR
2825  FF57  D0 F2     L7    BNE L6          ;LOAD ERROR
2826  FF59  4C 56 FD        JMP STRT
2827  FF5C  20 CF FF  L8    JSR $FFCF
2828  FF5F  C9 0D           CMP #CR
2829  FF61  F0 E4           BEQ L5          ;DEFUALT LOAD
```

```
CBM RESIDENT MONITOR.....PAGE 0007

LINE # LOC     CODE       LINE

2030  FF63  C9 2C          CMP #','
2031  FF65  D0 F8    L9    BNE L7          ;BAD SYNTAX
2032  FF67  20 B6 E7       JSR RDOB
2033  FF6A  29 0F          AND #$F
2034  FF6C  F0 D5    L10   BEQ L4          ;DEVICE 0
2035  FF6E  C9 03          CMP #3
2036  FF70  F0 FA    L11   BEQ L10         ;DEVICE 3
2037  FF72  85 D4          STA FA
2038  FF74  20 CF FF       JSR $FFCF
2039  FF77  C9 0D          CMP #CR
2040  FF79  F0 CC          BEQ L5          ;DEFAULT LOAD
2041  FF7B  C9 2C          CMP #','
2042  FF7D  D0 E6    L12   BNE L9          ;BAD SYNTAX
2043  FF7F  20 A7 E7       JSR RDOA
2044  FF82  20 97 E7       JSR T2T2
2045  FF85  20 CF FF       JSR $FFCF
2046  FF88  C9 2C          CMP #','
2047  FF8A  D0 F1    L13   BNE L12         ;MISSING END ADDR
2048  FF8C  20 A7 E7       JSR RDOA
2049  FF8F  A5 FB          LDA TMP0
2050  FF91  85 C9          STA EAL
2051  FF93  A5 FC          LDA TMP0+1
2052  FF95  85 CA          STA EAH
2053  FF97  20 97 E7       JSR T2T2
2054  FF9A  20 CF FF  L20   JSR $FFCF
2055  FF9D  C9 20          CMP #$20
2056  FF9F  F0 F9          BEQ L20
2057  FFA1  C9 0D          CMP #CR
2058  FFA3  D0 E5    L14   BNE L13         ;MISSING CR AT END
2059  FFA5  A5 B4          LDA SAVX
2060  FFA7  C9 07          CMP #7
2061  FFA9  D0 F8          BNE L14
2062  FFAB  20 A4 F6       JSR SV5
2063  FFAE  4C 56 FB       JMP STRT

2065  FFB1                 ZZ1=ALTM-1
2066  FFB1                 ZZ2=ALTR-1
2067  FFB1                 ZZ3=DSPLYR-1
2068  FFB1                 ZZ4=DSPLYM-1
2069  FFB1                 ZZ5=GO-1
2070  FFB1                 ZZ6=EXIT-1
2071  FFB1                 ZZ7=LD-1
2072  FFB1                 ZZ8=LD-1
```

COMANDI

Solitamente il comando viene dato dopo che il BASIC ha stampato READY. Questo è chiamato "Command Level" (livello di comando). I comandi possono venire usati come statement di programma. Alcuni comandi come LIST, NEW e LOAD determineranno l'esecuzione del programma solo quando avranno finito.

NOME	ESEMPIO	SCOPO / UTILIZZO
CLR	CLR	Cancella le variabili, resette i puntatori per "FOR" e "GOSUB" ,RESTOREs data.
LIST	LIST	Fa una lista del programma corrente
	LIST 100-	Opzionalmente inizia ad una riga specifica. Può essere fermato premendo il tasto STOP. (Il BASIC finirà la lista alla riga corrente).
	LIST X	Fa una lista soltanto della riga X.
	LIST -X	Fa una lista dall'inizio del programma alla riga X.
	LIST X - Y	Fa una lista di righe da X a Y incluse. N.B. Durante l'esecuzione della lista tenendo schiacciato il tasto RVS questa verrà rallentata ad approssimativamente 2 righe al secondo.
RUN	RUN	Inizia l'esecuzione del programma in memoria dal numero di statement più basso. Il RUN cancella tutte le variabili (fa un CLR) e riinizializza i DATA. Se avete fermato il programma e volete continuare l'esecuzione da un certo punto, usate il RUN seguito dal numero della riga.
NEW	NEW	Cancella il programma corrente e tutte le variabili.
CONT	CONT	Continua l'esecuzione del programma dopo aver schiacciato il tasto STOP oppure dopo aver eseguito uno STOP statement. Non potete continuare dopo aver commesso un errore, dopo aver modificato il programma oppure prima di iniziare l'esecuzione del vostro programma. Uno degli scopi principali del CONT è il debugging. Supponiamo ad un certo punto nell'eseguire il vostro programma non vi venga stampato alcunchè. Ciò può avvenire a volte perchè il programma sta eseguendo un numero molto elevato di calcoli ma può avvenire anche perchè siete caduti in un "infinite loop" (anello infinito). Un anello all'infinito è una serie di BASIC statement

dai quali non si può uscire. Il PET continuerà ad eseguire continuamente una serie di statements finchè non interverrete oppure toglierete la corrente. Se avete il sospetto che il programma sia in un anello all'infinito premete lo STOP e la riga che il BASIC avrà eseguito sarà stampata. Dopo che il BASIC avrà stampato READY potrete usare il PRINT per scrivere alcuni valori delle variabili. Dopo aver esaminati questi valori potrete raggiungere la convinzione che il programma abbia funzionato correttamente. Dovrete poi scrivere CONT per continuare l'esecuzione del vostro programma da dove è stato lasciato o scrivere un GOTO diretto per riprendere l'esecuzione del programma ad una riga diversa. Potete usare (LET) per assegnare ad alcune vostre variabili diversi valori. Ricordate se premete lo STOP in un programma e volete poi in un secondo momento riprenderlo non si devono essere verificati errori e non dovete inserire nessuna nuova linea. Se fate ciò non vi sarà possibile continuare ed avrete un errore continuato. E' impossibile continuare un comando diretto. Il CONT riprende sempre l'esecuzione dallo statement del programma da eseguire quando avete premuto lo STOP.

OPERATORI

SIMBOLI	ES. DI STATEMENT	SCOPO / UTILIZZO
=	B=-A	Negazione. N.B. 0+A è una sottrazione mentre -A è una negazione.
↑	130 PRINT X↑3	ESPONENTI. (eguale a X*X*X nell'esempio 0*0*0*0 ; 0 elevato a qualsiasi potenza da sempre 0)
*	140 X=R*(B*D)	MOLTIPLICA
/	150 PRINT X/1.3	DIVISIONE
+	160 Z = R+T	ADDIZIONE
-	170 J = 100 - I	SOTTRAZIONE

REGOLE PER ELABORARE LE ESPRESSIONI

1) Le operazioni con ordine di precedenza più alto devono essere eseguite prima delle più basse. Questo significa che le moltiplicazioni

e le divisioni devono essere eseguite prima delle addizioni e sottrazioni. Ad esempio 2+10/5 = 4 e non 2.4. Quando operazioni con uguali precedenze sono riscontrate in una formula quelle più a sinistra devono essere eseguite prima: 6-3+5=8, e non -2.

2) L'ordine in cui vengono eseguite le operazioni può essere specificatamente espresso tramite l'uso di parentesi. Ad esempio per sommare 5 a 3 e poi dividere per 4 useremo (5+3)/4=2. Invece se avessimo usato 5+3/4; avremmo ottenuto 5.75 (5+3/4).

L'ordine degli operatori gerarchicamente usato per elaborare le espressioni è il seguente iniziando da quella più alta: (N.B. le operazioni elencate nella stessa riga hanno la stessa precedenza.)

1) Le formule fra parentesi sono sempre elaborate per prime.
2) ↑ — Elevamento a potenza
3) Negation — -X dove X può essere una formula
4) * / — Moltiplicazione e divisione
5) + - — Addizione e sottrazione
6) Operatori Relazionali
   - = uguale
   - <> diverso
   - < minore di
   - > maggiore di
   - <= minore o uguale a
   - >= maggiore o uguale a

   (Le seguenti 3 operazioni sono logiche)
7) NOT — Negazione logica a livello di bit; essa prende soltanto la formula alla sua destra come argomento.
8) AND — AND logico a livello di bit
9) OR — OR logico a livello di bit.

Espressioni che usino operatori relazionali avranno sempre un valore vero (-1) oppure falso (0). Quindi (5=4)=0, (5=5)=-1,(4>5)=0,(4<5)=-1 ecc. Ogni valore che non sia Ø viene considerato vero.

La clausola THEN e IF è eseguita quando il risultato dell'IF non è uguale a 0. Cioè IF X THEN è equivalente a IF X<>0 THEN....

SIMBOLO	ESEMPIO	SCOPO / UTILIZZO
=	10 IFA=15THEN40	un'espressioneèè uguale a un'espressione
<>	70 IFA<>0THEN5	un'espressione è diversa da un'espressione
>	30 IFB>100THEN8	un'espressione è maggiore di una espressione
<	160 IFB<2THEN10	un'espressione è minore di un'espressione
<=,=<	180 IF100<=B+CTHEN100	un'espressione è minore o uguale a un'espressione
>=,=>	190 IFQ=>RTHEN50	un'espressione è maggiore o uguale a un'espressione
AND	2 IFA<5ANDB<2THEN7	Se l'espressione 1(A<5) e l'espressione 2(B<2) sono ambedue vere, l'esecuzione del programma passerà alla linea 7.

OR	IFA<1ORB<2THEN2	Se l'espressione 1(A<1) oppure l'espressione 2(B<2) oppure ambedue sono vere,l'esecuzione del programma passerà alla linea 2.
NOT	IF NOT Q3 THEN 4	Se Q3 è falso si salterà alla linea 4 (NOT VERO=FALSO).

AND OR e NOT possono essere usati per manipolazioni di bit e per eseguire operazioni booleane.

Queste tre operazioni convertono i loro argomenti a sedici bit, cioè in numeri interi compresi fra -32768 e +32767 nella rappresentazione complemento a 2. Dopodichè eseguono una specifica operazione logica sugli stessi e resituiscono un risultato nello stesso campo. Se l'argomento non fa parte di tale campo risulterà un errore.

Le operazioni vengpno eseguite a livello di bit, ciò significa che ciascun bit del risultato è ottenuto esaminando il bit nella stessa posizione per ogni argomento.
La seguente tabella di verità mostra la relazione logica fra bits:

OPERATORE	ARG. 1	ARG. 2	RISULTATO
AND	1	1	1
	0	1	0
	1	0	0
	0	0	0
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0
NOT	1	-	0
	0	-	1

Esempi: (In tutti i suddetti esempi, non appaiono gli zeri non significativi del numero binario).

63 AND 16=16	Poichè 63 è uguale al binario 111111 e 16 uguale al binario 10000, il risultato dell'AND sarà il numero binario 10000 equivalente al decimale 16
15 AND 14=14	15 è uguale al binario 1111 e 14 è uguale al binario 1110 per cui 15 e14 saranno uguali al binario 1110 o 14.
-1 AND 8=8	-1 è uguale al binario 111111111111111 e 8 uguale binario 1000 quindi il risultato sarà il binario 1000 o 8 decimale
4 AND 2=0	4 uguale al binario 100 e 2 uguale al Binario 10 Il risultato sarà quindi il binario 0 poichè nessun bit è pari a 1 in alcuna posizione in modo da dare 1 bit nel risultato non nullo.
4 OR 2=6	Il binario 100 OR con il binario 10 uguale al binario 110,o 6 decimale.

10 OR 10=10	Il binario 1010 OR con il binario 1010 è uguale al binario1010 o 10 decimale.
-1 OR -2=-1	Il binario 1111111111111111 (-1) OR con il binario 1111111111111110 (-2) è uguale al binario 1111111111111111, OR -1
NOT 0=-1	Il complemento del binario 0 a 16 bit è costituito da 16 uno (1111111111111111) 0 -1; allo stesso modo NOT -1=0
NOT X	NOT X è uguale a -(X+1). Questo succede perchè per ottenere il complemento a 2 di un numero si deve complementare ciascun bit e sommare 1.
NOT 1=-2	Il complemento a 16 bit di 1 è 1111111111110 che è uguale a -51+1) o -2.

Un tipico uso degli operatori a livello di bit è quello di controllare i gruppi di bit nelle locazioni I/O del computer che riflettono lo stato dei dispositivi esterni. La posizione bit 7è la più significativa di un byte, mentre la 0 è la meno significativa.
Per esempio supponiamo che bit 1 della locazione 5000 è 0 quando la porta della stanza X è chiusa, ed è 1 se la porta è aperta. Il programma seguente segnalerà "INTRUDER ALERT" se la porta è aperta:

10 IFNOT(PEEK(5000)AND2)THEN 10	Questa riga sarà ripetuta e ripetuta fintanto che il bit 1 (mascherato o selezionato dal 2) diventa un 1. Quando si verifica ciò passiamo alla riga 20.
20 PRINT "ITRUDER ALERT"	la linea 20 darà "INTRUDER ALERT"

Si può rimpiazzare lo statement 10 con lo statement WAIT che ha lo stesso identico effetto.

10 WAIT 5000,2	Questo comando ritarda l'esecuzione del prossimo ordine nel programma fino a che il bit 1 della locazione 5000 diventa 1. Il WAIT è molto più veloce dell'equivalente ordine IF ed impiega anche un minor numero di bytes per memorizzare il programma.

I commutatori di senso possono anche essere usati per rivelare lo stato di funzionamento di un organo di input.
Il programma che segue visualizza ogni cambiamento negli interruttori.

```
10 A=300: REM SET A TO A VALUE THAT WILL FORCE PRINTING
20 J=PEEK (sense switch Location) : IF J=A THEN 20
30 PRINT J; :A=J : GOTO20
```

Il seguente è un altro modo decisamente conveniente di usare gli operatori di relazione.

125 A=-(B>C)*B -(B<=C)*C Questo ordine porrà la variabile A al max (B,C) = la più grande delle 2 variabili B e C.

ISTRUZIONI

N.B. Nella seguente descrizione delle istruzioni, le lettere V o W stanno per variabili numeriche, X invece è un'espressione num., X$ denota una stringa e una I oJ denotanó un'espressione che è stata troncata a intero prima che l'ordine sia stato eseguito. "Troncata" significa che la parte frazionaria del numero è eliminata p.e. 3.9 diventa 3; 4.01 diventa 4.
Un'espressione è una serie di variabili, operatori, richiami di funzioni e costanti che dopo le operazioni e le chiamate funzione sono elaborate con le regole precedenti dando luogo ad una stringa o ad un valore numerico.
Una costnate può essere , ad esempio, sia il numero 2.71 o la stringa alfabetica "ABC".

NOME	ESEMPIO	UTILIZZO
CLOSE	10 CLOSE N	Chiude il file logico N (vedi file cassette).
DATA	20 DATA 1,-3,.04	Definisce dei dati che verranno letti da sinistra a destra. I dati dovranno essere inseriti nello statement DATA nello stesso ordine che verrà usato dal programma durante la lettura.
	30 DATA "ABC",PET	Dallo statement DATA possono essere lette anche stringhe. Se volete che la stringa contenga anche spazi (vuoti) iniziali, due punti (:) o virgole dovete includerla fra virgolette ("). Non si può usare in una stringa il carattere " .
DEF	40 DEF FNA(V)=B+C+V	Il programmatore può definire una funzione anche sulla base delle funzioni interne come (SQR,SIN,TAN,etc.) attraverso l'uso dello statement DEF. Il nome della funzione è "FN" seguito da una variabile legale p.e. FNX,FNJ7 .. Le funzioni definite dal programmatore hanno come lunghezza massima una linea. La funzione può essere definita da una qualsiasi espressione, ma deve avere un solo argomento. Nell'esempio B e C sono variabili usate nel programma.

		Eseguendo lo statement DEF si definisce la funzione. Le funzioni definite dal programmatore possono essere ridefinite eseguendo un altro statement DEF per la stessa funzione. Non è possibile definire funzioni stringa. "V" è chiamata variabile muta.
	50 Z=FNA(3)	L'esecuzione di questo statement deve seguire l'esecuzione dell'esempio precedente. Si otterrà B+C+3 ma il valore di V rimarrà inalterato.
DIM	80 DIM A(3),B(10)	Assegnazione degli spazi per le matrici. Tutti gli elementi delle matrici con lo statement DIM vengono inizializzati a Ø.
	75 A(5,5),D$(3,4,4)	Le matrici possono avere più di una dimensione.Non sono concesse più di 255 dimensioni. Considerando che non possiamo scrivere più di 80 caratteri per linea, in pratica le dimensioni si riducono a circa 34.
	35 DIM Q1(N),Z(2*1)	Le matrici possono essere dimensionate dinamicamente durante l'esecuzione del programma. Se una matrice non è esplicitamente dimensionata con uno statement DIM assume una singola dimensione che può variare da 0 a 10(11 elementi).Se questo statement viene prima dhèlo statement DIM per A venga incontrato dal programma esso sarà eseguito come se prima della linea 20 ci fosse un'istruzione DIM A(10). Tutti gli indici partono da zero intendendo che DIM (100) in realtà assegna 101 elementi alla matrice.
	20 A(8)=4.2	
END	999 END	Fine esecuzione di un programma senza messaggio di interruzione (vedere STOP). "CONT" dopo uno statement END riprende l'esecuzione di un programma dallo statement successivo a quello di END. END può essere usato in qualunque punto del programma ed è facoltativo.
FOR	20 FORV=1TO9.3STEP.6	(vedere lo statement NEXT). V è posto uguale al valore dell'espressione che segue l'uguale; in questo caso 1. Questo è chiamato valore iniziale. Ora vengono eseguiti gli statement fra FOR E NEXT. Il valore finale è il valore dell'espressione seguente il TO. L'incremento è il valore del-

l'espressione dopo STEP. Quando si incontra lo statement NEXT la variabile V viene aumentata del valore di STEP.

310 FORV=1TO9.3 — Se non vi è alcuno STEP l'incremento assume il valore di 1. Se STEP è positivo ed il nuovo valore della variabile è il valore finale (9.3 in questo caso), o il valore di STEP è negativo ed il nuovo valore della variabile è <= al valore finale, allora viene eseguito il primo statement dopo il FOR. Diversamente viene eseguito lo statement dopo NEXT. Tutti gli anelli FOR eseguono gli statement fra FOR e NEXT fino all'ultimo anche nei casi in cui si abbia FOR V=1 TO Ø.

315 FOR V=10*N TO3.4/Q STEP SQR(R) — Si noti che queste espressioni possono essere usate come valori iniziali,finali e per il valore di STEP in un ciclo di FOR. Il valore delle espressioni viene determinato una sola volta prima che il ciclo FOR.....NEXT sia eseguito.

320 FORV=9TO1STEP-1 — Quando è eseguito lo statement dopo il NEXT, la variabile V non è mai uguale a un valore qualsiasi in quanto il ciclo FOR...NEXT è terminato.
Gli statements fra i FOR e i corrispondenti NEXT in entrambi gli esempi visti sopra (310 e 320) saranno eseguiti nove volte.

330 FORW=1TO10:FORW=1TO :NEXTW:NEXTW — Errore: non usare annidamenti di cicli FOR...NEXT con la stessa variabile. L'annidamento del ciclo FOR è limitato soltanto dalla memoria disponibile.

GET — 10 GET C — accetta un singolo carattere da tastiera.

20 GET C$ — Accetta un singolo carattere stringa da tastiera.

30 GET#D,C — Accetta un singolo carattere da un file logico specifico.

40 GET D,C$ — Accetta un singolo carattere stringa da uno specifico file logico (vedere file cassette).

GOTO — 50 GOTO 100 — Trasferisce l'esecuzione alla linea specificata.

GOSUB	10 GOSUB 910	Trasferisce l'esecuzione alla linea specificata (910) fino a che si incontra uno statement RETURN; a questo punto l'esecuzione ritornerà allo statement successivo al GOSUB. Gli annidamenti GOSUB sono limitati solo dalla memoria disponibile.
IF..GOTO	32 IFX=Y+23.4GOTO92	Equivalente a IF..THEN; sola eccezione il fatto che al GOTO deve seguire un numero di linea mentre THEN può essere seguito indifferentemente da un numero di linea o da un altro statement
IF..THEN	10 IFX<1OTHEN 5	Se si verifica la condizione il programma passa alla linea 5
	20 IFX<O THEN PRINT"X LESS THAN O"	Se si verifica la condizione, visualizza il contenuto fra virgolette ed esegue tutti gli statements successivi alla linea del THEN.
	25 IFX=5THEN5O:Z=A	ATTENZIONE l'istruzione Z=A non sarà mai eseguita; se si verifica la condizione il programma passerà alla linea 50; in caso contrario procederà con la linea dopo la 25.
	26 IF X O THEN PRINT"ERROR,X NEGATIVE":GOTO 350	In questo caso se X è minore di O il comando PRINT verrà eseguito ed il programma passerà alla linea 350 se X è O o positivo il programma continua con l'esecuzione delle linee dopo la 26.
INPUT	3 INPUT V,W,W2,AB	Richiede dati dal terminale (devono essere inseriti). I valori devono essere separati fra loro da una virgola. L'ultimo valore deve essere seguito da un ritorno carrello"RETURN". Quando in ingresso viene richiesto un dato, sullo schermo apparirà "?". In uno statement INPUT possono essere inserite soltanto costanti come un 4.5E-3 o "CAT" . Se sono richiesti più dati di quelli immessi, un "?" rimarrà visualizzato e dovranno essere immessi gli altri dati. Se più dati di quelli richiesti saranno immessi i dati extra saranno ignorati. Le stringhe devono essere inserite nello stesso formato con cui sono specificate negli statements DATA. Se vengono inseriti dati alfabetici quando ci si aspetta numeri o viceversa, Il BASIC risponderà con ??"REDO FROM START".

	5 INPUT "VALUE";V	E' opzionale l'inserimento di una stringa ("VALUE") di richiamo prima di richiedere i dati del terminale . Se in risposta ad uno statement INPUT è immesso un RETURN il BASIC ritorna al modulo di comando. Scrivendo CONT dopo che un comando INPUT è stato interrotto l'esecuzione riprenderà dallo statement INPUT.
	40 INPUT#D,A	Legge il valore di A dal file logico D
	50 INPUT#D,A$	Legge una specifica stringa A dal file logico D.
	60 INPUT#D,A,A$,B,B$	Legge specifici valori e stringhe dal file logicoD, Le stringhe non devono essere racchiuse tra virgolette (vedere file cassette)
LET	300 LET W=X	Assegna un valore ad una variabile
	310 V=5.1	LET non è indispensabile.
LOAD	10 LOAD	Carica un programma o un file dal registratore nella memoria del PET.
	20 LOAD "NAME"	Carica un programma o un file di nome "NAME" dal registratore nella memoria del PET.
	30 LOAD "NAME",D	Carica uno specifico File di nome "NAME" dal periferico D (vedi file cassette)
OPEN	10 OPEN A	Apre un file logico A per leggere dal registratore.
	20 OPEN A,D	Apre un file logico A per leggere dal periferico D.
	30 OPEN A,D,C	Apre un file logico A per il comando C da un periferico D.
	40 OPEN A,D,C,"NAME"	Apre un file logico A dal periferico D. Se l'apparecchio D accetta file formattati il file "NAME" viene posizionato per il comando (vedi file cassette).
NEXT	340 NEXT V	Segna la fine di un ciclo FOR.
	345 NEXT	Se non è assegnata la variabile chiude l'ultimo ciclo FOR.
	350 NEXT V,W	Un solo NEXT può essere usato per chiudere più cicli FOR. Equivale a NEXTV: NEXT W.

ON..GOTO 100 ON I GOTO10,20,30,40 Salta alla linea indicata dal I-esimo numero dopo GOTO. Cioè:
IF I=1 THEN GOTO LINE 10
IF I=2 THEN GOTO LINE 20
IF I=3 THEN GOTO LINE 30
IF I≥4 THEN GOTO LINE 40
Se I=0 si tenta di selezionare una linea inesistente (in questo caso >=5) viene eseguito lo statement successivo a questo. Tuttavia se I è 255 o 0, avremo un messaggio di errore. A ON...GOTO possono seguire tanti numeri quanti possono stare su una linea.

105 ON SIGN(X)+2 GOTO 40,50,60 Questo statement farà saltare alla linea 40 se l'espressione X<0 alla 50 se =0 e alla 60 se >0.

ON..GOSUB 110 ON I GOSUB 50,60 Identica a ON..GOTO tranne che viene richiamata una subroutine (GOSUB); si ritorna allo statement successivo a ON...GOSUB.

POKE 357 POKE I,J Lo statement POKE immagazzina il valore specificato dal secondo argomento (J) nella posizione data dal primo argomento (I). Il valore che deve essere immagazzinato deve essere =>0 e <=255 o vi sarà una segnalazione di errore. L'indirizzo I deve essere =>0 e <65535 o vi sarà un errore. L'uso negligente dello statement POKE può probabilmente causare la distruzione nel BASIC. Il caricamento in una parte di memoria non utilizzata è innoquo. Uno degli usi principali del POKE è di passare gli argomenti ad una subroutine in linguaggio macchina. Potete usare i comandi PEEK e POKE per scrivere un programma diagnostico o un assemblatore utilizzando il BASIC.

PRINT 360 PRINT X,Y;Z
370 PRINT
380 PRINT X,Y;
390 PRINT"VALYE IS";A
400 PRINT,A2,3, Stampa il valore delle espressioni sul terminale. SE deve essere stampata una lista essa non deve finire con una virgola o punto e virgola ma con un ritorno carrello. Dopo che tutti i valori sono stati stampati viene eseguito un caporiga. Possono essere stampate anche stringhe racchiuse fra virgolette. Se in una linea due espressioni sono separate da un punto e virgola i loro valori saranno stampati uno di seguito all'altro.

	410 PRINT MID$(A$,2);	Possono essere stampate parti di stringhe
READ	490 READ V,W	Legge un dato in una variabile specifica da uno statement DATA. Il primo dato letto sarà il primo dato elencato nel primo statement DATA del programma. Il secondo letto sarà il secondo elencato e così via. Quando sono stati letti tutti i dati dal primo statement DATA, i dati successivi verranno letti dal secondo statement DATA del programma. Fate attenzione a non leggere più dati di quelli contenuti negli statements DATA del programma. Tale fatto darà luogo a un "DATA ERROR"
REM	500 REM NOW SETV=0	Il programmatore può mettere dei commenti nel suo programma. Lo statement REM non è esecutivo ma può essere utilizzato per eseguire un salto. Lo statement REM finisce con la fine della riga ma non con ":".
	505 REM SETV=Ø:V=Ø	In questo caso V=0 non sarà mai eseguito dal BASIC.
	506 V=0: REM SETV=0	In questo caso V=0 verrà eseguito.
RESTORE	510 RESTORE	Permette la rilettura degli statement DATA. Dopo un RESTORE il primo dato letto sarà il primo dato elencato nel primo statement DATA del programma. Il secondo dato letto sarà il secondo contenuto nel primo statement DATA e così come in una normale operazione di lettura.
RETURN	50 RETURN	In una subroutine causa il ritorno allo statement successivo all'ultimo GOSUB eseguito.
STOP	900 STOP	Causa lo stop dell'esecuzione di un programma e addende l'ingresso di un comando.
SYS	120 SYS (64824)	Causa il salto del computer alla locazione 64824 decimale della memoria e lavora in codice macchina da quel punto. Ritorna al comando o al BASIC con il codice macchina RTS.
TI	75 PRINT TI 85 TI $="HHMMSS"	La linea 75 stampa il numero di JIFFIES dopo che la macchina è stata accesa o il numero di JIFFIES equivalente al tempo in TI$. La linea 85 regola l'orologio interno del PET con

		il tempo reale . JIFFIES è 1/60 di secondo.
USR	95 USR(X)	Trasferisce il controllo al programma il cui indirizzo iniziale si trova alla locazione 1 e 2.X è un parametro passato nel o dal linguaggio di programmazione della macchina (vedere appendice).
VERIFY	10 VERIFY	Verifica il programma registrato in cassette leggendolo e confrontandolo con quello in memoria del PET.
	20 VERIFY"NAME"	Verifica il file specifico "NAME" registrato nella cassetta leggendolo e comparandolo con quello in memoria nel PET.
	30 VERIFY"NAME",D	Come sopra; in più è specificato il dispositivo da cui effettuare la lettura
WAIT	805 WAIT I,J,K	Questo statement legge lo stato della locazione I,esegue l'OR esclusivo con K e poi effettua l'AND con J finchè si ottiene un risultato diverso da Ø. L'esecuzione de programma continua con lo statement successivo a quello di WAIT. Se lo statement WAIT ha solo due argomenti K assume il valore di Ø. Se state aspettando che un bit diventi Ø vi deve essere un 1 nella corrispondente posizione diK. I,J èK devono essere =>0 e <=255.

FUNZIONI INTRINSECHE

ABS(X)	120 PRINT ABS(X)	Da il valore assoluto dell'espressione X. ABS restituisce -X se X<=0, altrimenti X.
INT(X)	140 PRINT INT(X)	Ritorna il più grande numero intero uguale o inferiore al suo argomento X . Per es. INT(.23)=0, INT(7)=7, INT(-.1)=1, INT(-2)=-2, INT(1.1)=1. Il seguente esempio sposta il valore di X di D posizioni decimali: INT (X*10↑D+.5)/10↑D
RND(X)	170 PRINT RND(X)	Genera un numero casuale tra Ø e 1. L'argomento dei numeri casuali è il seguente: X<Ø da inizio ad una nuova sequenza di numeri casuali che usano X . Chiamando RND con la stessa X inizia la stessa serie di numeri casuali. X=Ø dà l'ultimo numero casuale formato. Richiami ripetuti a RND (Ø) otterranno sempre lo stesso numero

		casuale . X>Ø crea un nuovo numero casuale tra Ø ed 1 . Notate che (B-A)*RND(1)+A avrà per risultato un numero casuale tra Ae B.
SGN(X)	230 PRINT SGN(X)	Da 1 se X>Ø, da Ø se X=Ø e -1 se X< Ø
SIN(X)	190 PRINT SIN(X)	Si ottiene il seno dell'espressione X X viene interpretato come se fosse in RADIANTI N.B. COS(X)=SIN(X+3.14159/2) e che 1 radiante =180/π gradi=57.2958 gradi; così che il seno di X gradi= SIN(X/57.2958).
SQR(X)	180 PRINT SQR(X)	Si ottiene la radice quadrata dell'argomento X. Si cadrà in errore se X<Ø.
TAB(I)	240 PRINT TAB(I)	Spazi per fissare la posizione di stampa sul terminal. Può essere usato solo negli statements PRINT. Zero è la colonna più a sinistra del terminal 39 invece la più a destra. Nel caso il carrello sia oltre la posizione I allora non vi sarà alcuna stampa. I deve essere =>0 e <=255.
ATN(X)	210 PRINT ATN(X)	Si ottiene l'arcotangente dell'argomento X. Il risultato è espresso in radianti e va da -π/2 a π/2 (π/2=1.5708).
COS(X)	200 PRINT COS(X)	Si ottiene il coseno dell'espressione X. X è interpretato come se fosse in radianti.
EXP(X)	150 PRINT EXP(X)	Si ottiene la costante "E" (2.71828) elevata all'esponente X (E+X).L'argomento più grande che può essere consegnato a EXP senza che si abbia overflow è 88.
FRE(X)	270 PRINT FRE(0)	Si ottiene il numero di bytes non usati in quel momento dal BASIC (stampa il numero di memorie libere).
LOG(X)	160 PRINT LOG(X)	Si ottiene il logarittmo naturale (base E) del suo argomento (ragione) X. Per ottenere il logaritmo in baseY di X si usi la formula LO(X)/LOG(Y). P.e. base 10 (comune) log di 7 = LOG(7)/LOG(10).
PEEK	356 PRINT PEEK(I)	La funzione PEEK dà di ritorno i contenuti dell'indirizzo di memoria I . Il valore restituito sarà =>Ø e<=255. Se I è > 65535 o<Ø si cadrà in errore. Il tentativo di leggere ad indirizzo di memoria non esistente avrà per risultato un valore sconosciuto.

(vedi l'ordine POKE).

SPC(I)	250 PRINT SPC(I)	Lascia I spazi bianchi sullo schermo. Può essere usato soltanto in uno statement PRINT. X deve essere =>Ø e <=255 o si cadrà in errore.
TAN(X)	200 PRINT TAN(X)	Si ottinene la tangente dell'espressione X . X viene espresso in radianti.

STRINGHE

La lunghezza di una stringa è compresa tra 0 e 255 caratteri. Il nome di tutte le variabili stringa finiscono con il segno $ (dollaro): p.e. A$,B9$,K$,HELLO$.
Matrici di stringhe possono essere dimensionate esattamente come le matrici numeriche. P.es. A$(10,10) crea una matrice di stringhe di 121 elementi, cioè di 11 righe per 11 colonne (righe e colonne da 0 a 10). Ogni elemento della matrice stringa è una stringa completa che può avere sino a 255 caratteri di lunghezza.

NOME	ESEMPIO	SCOPO / USO
DIM	25 DIM A(10,10)	Assegna gli spazi per l'indice e la lunghezza per ogni elemento della matrice stringa. Nessuna stringa di spazi viene allocata.
LET	27 LET A$="PET"+V$	Assegna i valori di una costante stringa ad una variabile stringa. LET è a discrezione
		Operatori di raffronto stringhe. Il confronto è fatto sulla base dei codici ASCII, un carattere per volta fino a che viene localizzata la differenza. Se durante il confronto di due stringhe viene raggiunta la fine di una, la serie più corta è considerata più piccola. Nota che "A " è più grande di "A" poichè gli spazi che seguono la lettera sono importanti.
+	30 LET A$=B$+C$	Concatenamento di stringhe. La stringa risultante deve avere meno di 256 caratteri o altrimenti si cadrà in errore. Il computer legge una stringa immessa da tastiera. La stringa non deve essere necessariamente messa fra virgolette, ma in tal caso gli spazi bianchi che precedono i caratteri saranno ignorati e la stringa terminerà o col carattere "," o ":" .

READ	50 READ X$	Legge la stringa dallo statement DATA durante l'esecuzione del programma. La stringa non deve essere messa fra virgolette; ma se non lo è essa termina col carattere "," o fine linea. Per il formato del DATA vedi DATA.
PRINT	60 PRINT X$ 70 PRINT "FOO"+A$	Visualizza le stringhe sullo schermo.

FUNZIONI STRINGA

ASC(X$)	300 PRINT ASC(X$)	Da di ritorno il valore numerico ASCII del primo carattere della stringa X$. Per una tavola di conversione ASCII/numeri vedi l'appendice. Si cadrà in errore se X$ è la stringa nulla
CHR$(I)	275 PRINT CHR$(I)	Si ottiene un carattere della stringa La posizione del carattere nella stringa è data da I il cui valore deve essere =>Ø e<=255.
LEFT$(X$,I)	310 PRINT LEFT$(X$,I)	Si ottengono gli I caratteri più a sinistra della stringa X$ Se I <=0 o >255 si cadrà in errore.
LEN(X$)	200 PRINT LEN(X$)	Si ottiene la lunghezza della stringa X$ in caratteri (bytes). I caratteri non stampati e gli spazi bianchi vengono conteggiati nella lunghezza totale.
MID$(X$,I,J)	340 PRINT MID$(X$,I,J)	MID$ chiamato con due argomenti restituisce i caratteri della stringa X$ ad iniziare dal carattere in posizione I. Se I>LEN(I$) allora MID$ è da una serie nulla (lunghezza 0). Se I<=0 o >255 si cadrà in errore.
MID$(X$,I,J)	340 PRINT MID$(X$,I,J)	MID$ chiamato con 3 argomenti da di ritorno una stringa iniziando con l'I-esimo carattere di X$ per j caratteri. Se I >LEN(X$),MID$ da una stringa nulla. Se I o J<=Ø o> 255 si cadrà in errore. Se J specifica più caratteri di quanti sono rimasti nella stringa, sono restituiti tutti i caratteri dal I in poi.
RIGHT$(X$,I)		Si ottengono gli I caratteri piu a destra della stringa X$. Quando I<=0 o> 255 sarà inevitabile l'errore. Se I=LEN(X=) allora RIGHT$ restituisce tutti i caratteri.

STR$(X)	290 PRINT STR$(X)	Da una stringa che è la rappresentazione in caratteri della espressione numerica X. P.es. STR$(3.1)="3.1"
VAL(X$)	280 PRINT VAL(X$)	Si ottiene l'espressione stringa X$ convertita in un numero. P.es.VAL("3.1") =3.1 . Se il primo carattere della stringa che differisca dallo spazio bianconon è un segno più (+) o meno (-), un numero o un punto decimale (.) allora sarà dato di ritorno lo Ø.

CARATTERI SPECIALI

CARATTERE	USO	
π	120 PRINT π 140 A=π* 2	Da 3.14159265
RETURN		Il tasto di "RETURN" deve essere premuto alla fine di ogni linea. Il RETURN Riporta il cursore all'inizio della riga. Dopo ciascun RETURN si passa automaticamente ad una nuova linea.
STOP		Interrompe l'esecuzione di un programma o la stampa di una linea. Lo STOP ha effetto quando l'istruzione in corso è stata completamente eseguita, o in caso di stampa di una lista, quando è terminata una riga. In entrambi i casi il controllo viene restituito al livello di controllo del BASIC e viene stampata la scritta READY. Viene altresì stampato "BREAK IN LINE XXXX" dove XXXX è il numero della linea che si sarebbe dovuto eseguire subito dopo.
:		I due punti vengono usati per separare istruzioni che si trovino sulla stessa linea. Essi possono venir usati sia nelle istruzioni dirette che indirette. Unico limite al numero di istruzioni che possono trovarsi sulla stessa linea è la lunghezza della linea. Non è possibile però inserire un GOTO o un GOSUB all'interno della Linea.
?		Il punto interrogativo è equivalente al codice PRINT. ?2+2 è identico a PRINT 2+2. I punti interrogativi possono essere utilizzati nelle istruzioni indirette. Quando si esegue una lista 10 ? X viene stampato nella forma 10 PRINT X . Non si può mai usare tuttavia la forma ?#, ma è neces-

sario scrivere sempre PRINT# .

% E' l'identificatore dei nomi di variabile per le variabili intere di ampiezza compresa fra -32767 e 32767.

AVVERTENZE PER IL MIGLIOR UTILIZZO DELLA MEMORIA

I seguenti avvisi possono essere utili per creare programmi più piccoli e risparmiare dello spazio di memoria.

1. Usate più statements per riga. C'è un piccolo numero di intestazioni (5 bytes) associate con ogni linea nel programma, 2 di questi 5 byte contengono il numero di linea in forma binaria. Ciò significa che non dovete preoccuparVi di quante cifre avete nel vostro numero di linea (il minimo numero di linea è 0, il massimo è 64000), occuperà sempre 2 byte. Inserendo nella linea più ordini è possibile ridurre il numero di byte necessari al programma.

2. Cancellate dal programma tutti gli spazi inutili. Per esempio:

```
10 PRINT X, Y, Z
```

   si serve di 3 byte in più di

```
10 PRINT X,Y,Z
```

   N.B. Tutti gli spazi tra il numero della linea ed il primo carattere scritto non sono contati.

3. Cancellate tutti gli statements REM. Ogni statement REM adopera almeno un byte in più del numero di byte del testo di commento.. Per esempio lo statement 130 REM THIS IS A COMMENT adopera 24 byte di memoria.. Nello statement 140 X=X+Y: REM UPDATE SUM, il REM utilizza 14 byte di memoria incluso il: prima del REM.

4. Usate variabili invece di costanti. Supponete di servirVi della costante 3.14159 per 10 volte nel vostro programma. Se voi inserite nel programma lo statement 10 P=3.14159 ed usate P invece di 3.14159 ogni volta che Vi serve, risparmierete 40 byte. Inoltre accellererete i tempi.

5. Un programma non esige la fine con END; così, un ordine END alla fine del programma può essere cancellato.

6. Riusate le stesse variabili. Se avete una variabile T che è usata per tenere un risultato temporaneo in una parte del programma e se più tardi avrete bisogno di una variabile temporanea nel vostro programma, servitevi nuovamente di T. Se state chiedendo all'utente di dare una risposta SI o NO a due domande differenti in due volte diverse durante l'esecuzione del programma, servitevi delle stesse variabili temporanee A$ per immagazzinare le risposte.

7. ServiteVi di GOSUB per eseguire parti di programma che compiono le stesse azioni.

8. Usate anche gli elementi zero delle matrici; per esempio A(0), B(0,X)..

INFORMAZIONI SULL' ALLOCAZIONE IN MEMORIA

BASIC	BYTES	UTILIZZI PER
BASIC	1028	Buffers di I/O Tabelle Scratch Pad
Numero di riga	4	
Codici BASIC	1	
Caratteri	1;....Incluso il RETURN	
Variabili	7.....e è assegnato un valore(per numeri interi e variabili in virgola mobile)	
	7 +	lunghezza della stringa per variabili stringa
Matrice	A + (S+1) + (2^{+}D)	
(La dimensione include l'elemento 0)	Dove:	
	A = 5	Per una matrice in virgola mobile
	A = 3	Per matrice alfa numerica
	A = 2	Matrice intera
e	S = ...	Dimensione della matrice
	D = ...	Numero delle dimensioni

Quando un programma viene eseguito, lo spazio è assegnato dinamicamente allo Stack nel seguente modo:

a) Ogni ciclo attivo FOR ... NEXT utilizza 22 byte

b) Ogni GOSUB attivo (uno che è ancora ritornato) adopera 6 byte

c) Ogni parentesi incontrata in un espressione usa 4 byte ed ogni risultato provvisorio calcolato in espressione si serve di I2 byte.

9) Servitevi di variabili o di matrici intere p.e. A%, HX% (I,J) ecc. dovunque sia possibile.

AVVISI PER LA VELOCITA' D'ESECUZIONE

Questi consigli dovrebbero accorciare i tempi d'esecuzione del vostro programma BASIC. Si osservi che alcuni di questi avvisi sono gli stessi usati per diminuire lo spazio del vostro programma. Ciò significa che spesso potete risparmiare dello spazio ed allo stesso tempo aumentare la velocità del vostro programma.

I. Eliminate tutti gli inutili e gli spazi di REM dal programma. Ciò può abbrevviare di un pò i tempi perchè altrimenti il BASIC dovrebbe ignorare e saltare gli spazi e gli ordini REM.

2. Probabilmente questo è il più importate e permette di aumentala velocità d'esecuzione di un fattore IO. servitevi di variabili al posto di costanti. Ci vuole più tempo per convertire una - costante nella sua rappresentazione in virgola mobile di quanto ce ne voglia per prendere il valore di una variabile o di una metrice.

Ciò è particolarmente importante con i cicli FOR....NEXT e con altri che sono eseguite ripetutamente.

3. Le variabili che sono state incontrate per prime durante l'esecuzione di un programma BASIC sono assegnate all'inizio delle tavole delle variabili. Ciò significa che uno statement come
5 A=O: B=A: C=a, piazzerà per prima B per seconda C per terza nelle tavole dei simboli (purchè la 5° riga sia la prima ad assegnare eseguita nel programma). In seguito nel programma, quando BASIC troverà un riferimento alla variabile A, esso cercherà solamentte una volta nella tavola dei simboli per trovare A, 2 per trovare B e 3 per trovare C;

4 Gli statements NEXT senza la variabile indice NEXT è più veloce di NEXT i perchè non è eseguito alcun controllo per vedere se ha variabili specificata nel NEXT è la stessa usata nel più vicino FOR.

AVVISI DI ERRORE

Quando vi è un errore il PET ritorna all'attesa di un comando esterno e visualizza READY sullo schermo. I valori delle variabili ed il testo del programma rimangono intatti, ma il programma nèn può essere continuato con il comando cont. Il GOSUB E TUTTO il ciclo FOR.. NEXT vanno persi, come tutto ciò che concerne l'esecuzione in atto.

Quando c'è un errore in uno statement del programma, l'avviso di errore che viene visualizzato indicherà il numero di linea in cui si trova l'errore. Quando l'errore è in uno statement diretto o a livello di comando, non viene visualizzato nell'avviso nessun numero di linea.

AVVISO DI ERRORE	CAUSE E CORREZIONI
CAN'T CONTINUE	L'errore è causato dal tentativo di continuare o un programma inesitente, o dopo che si è verificato un errore o dopo che una nuova linea è stata inserita nel programma o dopo che è stata fatta una correzione ad una linea esistente. Corregete l'errore, indi usate un GOTO diretto per riavviare il programma o battete RUN e ricominciate.
DIVISION BY ZERO	Dividere per zero è un errore. Controllate l'espressione usata nel denominatore nello statement aritemetico errato, indi corregetela in maniera tale che non possa mai essere valutata come Ø.
ILLEGAL DIRECT	Uso di uno statement INPUT, GET o DEF come comando diretto. Evitate di usare questi statement come comandi diretti.
ILLEGAL QUANTITY	Il parametro passato ad una funzione matematica od a una stringa era oltre la portata prevista. Questi errori possono essere dovuti a a. un indice di matrice negativo come LET A (-1) =Ø b. un indice di matrice troppo grande: 65535 c.. un argomento Ø o negativo per il logaritmo come LOG (-X) d. SQR con argomento negativo come SQR (-4)

AVVISO DI ERRORE	CAUSE E CORREZIONI
ILLEGAL QUANTITY (cont)	A B se A è una variabile negativa e B non è un intero. (Non da luogo a errore se viene usata una costante al posto di una variabile; p.e. -4 B perchè l'elevamento a potenza viene eseguito prima del meno). f. una chiamata a USR prima che l'indirizzo della subroutine in linguaggio macchina sia stato sitemato. Assicuratevi che l'argomento sia entro la portata della funzione che state usando. a. gli indici devono essere = Ø = 256 b. I log. richiedono un argomento non negativo.
NEXT WITHOUT FOR	La variabile in uno statement NEXT non corrisponde ad uno statement FOR eseguito precedentemente. Deve essere inserita la parte FOR di un ciclo FOR ...NEXT o deve esere cancellata la parte NEXT errata del ciclo. Assicuratevi che le variabili di indice siano le stesse ad entrambi i lati del ciclo. Esempio: FOR I=1 TO 10 . . . NEXT I
OUT OF DATA	E' stato eseguito uno statement READ ma sono già stati letti tutti gli statement DATA nel programma. Il programma ha cercato di leggere troppi dati oppure i dati inclusi nel programma sono insufficenti. Usate lo statement RESTORE per dati in modo che il PET possa rileggerli oppure aggiungete altri elementi DATA oppure in fine usate un contrassegno alla fine della lista di dati-controllate ciò prima della lettura.
OVERFLOW	Il risultato di un calcolo era troppo grande per essere rappresentato dalla capacità numerica del BASIC. (Se c'è un errore di underflow, il risultato viene posto a zero e l'esecuzione prosegue senza che sia stampato nessun messaggio di errore).

AVVISO D'ERRORE	CAUSE E CORREZIONI
OVERFLOW (cont)	Avete richiesto un numero più grande di quanto il PET possa ricordare. Provate a chiederne uno più piccolo. Il numero più grande possibile è 1.70141183E+38. Modificate l'ordine dei vostri calcoli.
REDIMENSIONED ARRAY	Dopo che una matrice è già stata dimensionata, è stato incontrato un altro statement di dimensionamento perla stessa matrice. Si cade spesso in questo errore se ad una matrice è stata data la dimensione implicita 10 perchè è stato incontrato prima uno statement come A(I)=3 e dopo nel programma è stato caricato DIM A (100). Assicuratevi di non aver usato un GOTO per saltare ad un'altra linea in uno satement che precede lo statement DIM oppure controllate che lo stesso statement DIM non sia dentro un ciclo FOR....NEXT o dentro una subroutine che sarà eseguita più volte o infine se vi siete serviti di un elemento della matrice prima di servirvi dello statement DIM. Fate del DIM una delle prime linee del vostro programma.
RETURN WITHOUT GOSUB	E' stato incontrato uno statement RETURN senza che sia stato eseguito un precedente statement GOSUB. Dovete inserire un GOSUB oppure cancellare il RETURN.
STRING FORMULA TOO COMPLEX	L'espressione stringa era troppo complessa. Dividete la stringa in due o più stringhe di minor lunghezza.
STRING TOO LONG	Avete cercato di creare una stringa di lunghezza superiore a 255 caratteri servendovi dell'operatore di concatenazione. Un numero viene stampato come SPACE-NEMBER-CURSOR RIGHT Dividete la stringa in due o più stringhe di minor lunghezza.
SUBSCRIPT OUT OF RANGE	Avete cercato di riferirvi ad un elemento della matrice che è al di fuori della dimensione della matrice.

AVVISO D'ERRORE	CAUSE E CORREZIONI
SUBSCRIPT OUT OF RANGE (cont)	Potete cadere in questo errore se nel riferimento ad una matrice viene usato un numero di indici sbagliato; LET A(1,1,1)=Z quando A è stato dimensionato DIM A(2,2). Dovete aumentare lo spazio che avete richiesto per la matrice (per es. Sostituite DIM A(10) con DIM A(20) oppure cambiere il numero di indici richiesti con l'istruzione DIM (p.e. da DIM A(10,10) a DIM A(10,10,10) o da DIM B(10,10,10) a DIM B(10,10).)
SYNTAX ERROR	Mancanza di parentesi in un'espressione, caratteri illegali in una linea, punteggiatura scorretta ecc. Questo tipo di errore è difficile da trovarsi ma facile da risolvere. Esaminate attentamente lo statement errato ed inserite o cancellate ciò che è necessario.
TYPE MISMATCH	La parte sinistra di uno statement d'assegnazione era una variabile numerica mentre la parte destra era una stringa o viceversa; una funzione che prevedeva un argomento stringa ne ha ricevuto uno numerico o viceversa. Non potete mescolare tipi diversi in un'istruzione così dovete cambiare una parte dello statement in modo che si accordi alla altra (le parti si incontrano al segno=). Controllate i tipi di argomento della funzione ed usate quello giusto (numerico o stringa).
UNDEFINID STATEMENT	Vi siete serviti come rimando di un GOTO, GOSUB o di un THEN di un numero di linea che non esiste. Inserite il numero di linea necessario o eseguite il salto ad un altro numero di linea.
UNDEFINED USER FUNCTION	Vi siete riferiti ad una specifica funzione dell'utente che però non è mai stata definita. DEFINITE LA FUNZIONE
FILE OPEN	Avete cercato di aprire un file precedentemente aperto. Controllate i numeri logici dei files (primo parametro dello statement OPEN) e assucuratevi di esservi serviti per ogni file di numeri non usati precedentemente.

AVVISO D'ERRORE	CAUSE E CORREZIONI
FILE NOT OPEN	Avete cercato di leggere o scrivere in un file non precedentemente aperto. APRITE IL FILE.
NOT INPUT FILE	Avete cercato di usare l'INPUT da un file aperto invece per scrivere. Leggere richiede uno Ø come terzo parametro dello statement OPEN. Leggere in questo caso è una scelta sbagliata.
NOT OUTPUT FILE	Avete cercato di usare il PRINT in un file aperto invece per leggere. Scrivere in un file richiede come terzo parametro dello statement OPEN un 1 (o un 2 se desiderate un EOT alla fine del file).
DEVICE NOT PRESENT	Avete cercato di aprire un file su di un dispositivo che "non è visibile" al PET. Controllate i numeri del dispositivo (secondo parametro nello statement OPEN) e assicuratevi che il dispositivo sia stato assegnato e collegato in maniera esatta e ovviamente che sia acceso.