COMMODORE 64

16 K di ROM

Il Basic e il Sistema Operativo: come funzionano

di Andrea de Prisco

Dopo aver parlato di grafica, suono e disco, questo mese daremo un'occhiata all'interno del 64, ponendo particolare attenzione all'architettura del sistema e al funzionamento dal punto di vista software. Il tutto come introduzione ai prossimi articoli, in cui presenteremo un tool (ADP BASIC), che con le sue 50 e più istruzioni permetterà un uso molto semplificato delle periferiche Commodore: Disco, Plotter e le stampanti MPS 801 e MPS 802.

Smanettomania

Il Commodore 64 potrebbe essere eletto, senza troppi sforzi, il computer da smanettatore per eccellenza. Grazie alla sua architettura, e per come è stato partorito da mamma Commodore, si presta molto bene a modifiche di vario genere, da quelle hardware a quelle di livello più morbido, software. È una macchina aperta, come si suol dire. Lo testimonia anche il fatto che tutte le sue effettive caratteristiche non sono direttamente utilizzabili da tastiera o da Basic, ma in alcuni casi è necessaria una vera e propria cultura sugli interrupt e esperienza di programmazione in linguaggio macchina per sfruttarle appieno.

Basta citare lo scrolling fine, l'alta risoluzione, i caratteri multicolor, nemmeno citati sul manuale di istruzione dato col computer. Fortunatamente i libri sul 64 si contano a decine, non ultima la Programmers Reference Guide, della Commodore stessa, che dà una descrizione abbastanza precisa del 70% della macchina.

E il rimanente 30%?

Questo è il problema! Andiamo dunque a incominciare.

L'architettura

Il sistema 64, macchina e relative periferiche collegate, per come è organizzato ha molto dei computer più seri, i "veri" computer. Ciò che manca è una maggiore velocità di esecuzione, specialmente per quanto riguarda i trasferimenti di dati tra le varie unità. Probabilmente questa esasperante lentezza è dovuta all'interfacciamento tra device secondo un protocollo di comunicazione misto di RS-232 e IEEE-488. Facevano meglio a lasciare il protocollo usato dai Pet: IEEE-488 e basta!

Punto forte del 64 è invece l'architettura interna, in particolar modo la gestione del video da parte dell'integrato 6567 (o VIC II) di cui abbiamo ampiamente parlato nei numeri scorsi. La tendenza attuale nella costruzione di un computer è di far fare meno roba possibile al microprocessore, per lasciarlo dedicare maggiormente ai programmi dell'utente. Si provvede cioè a organizzare un computer in tante unità interagenti più che vedere un solo capo (il microprocessore) che fa tutto. Repubblica con consiglio dei ministri più che dittatura monarchica, in altre parole.

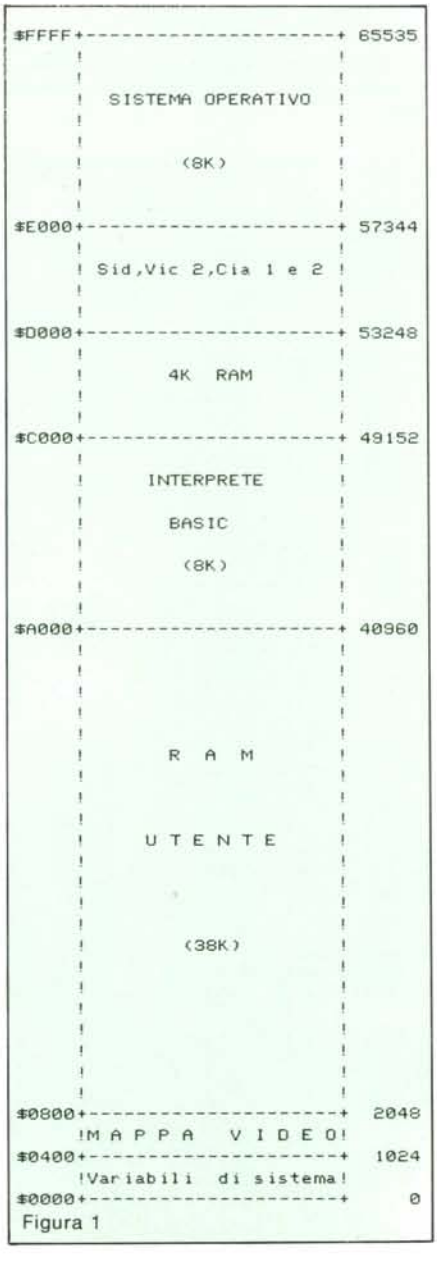

Figura 1

Il ministro della pubblica visualizzazione è il VIC II, quello degli esteri l'integrato 6526 (si occupa dell'interfacciamento con l'esterno). Inutile dire che il presidente del consiglio è il 6510, il microprocessore.

In figura 1 è mostrata la mappa della memoria del 64, al momento dell'accensione. Agendo opportunamente su alcuni bit del byte 1 e a livello hard su alcuni contatti della porta espansioni, è possibile manipolare tale configurazione. Tutta la memoria è lunga in tutto 64 K byte, quanti ne può indirizzare il 6510. Il primo K contiene le variabili di sistema, un insieme di informazioni sullo stato di tutta la macchina. Segue un altro K di mappa video, la zona di memoria immagine di quanto visualizzato sullo schermo. Poi abbiamo 38 K byte di Ram per mantenere programmi utente e relative variabili. Di seguito 8 K byte di memoria a sola lettura (Rom) contenenti l'interprete del linguaggio Basic. Troviamo poi 4 K Ram liberi, dove è possibile inserire programmini in linguaggio macchina. Di seguito a questi sono locati i registri dei processori ausiliari del 64: il VIC II, il SID (l'integrato del suono) e i due CIA (Complex Interface Adapter) che gestiscono l'interfacciamento col "resto del mondo" e la tastiera. In fondo, altri 8 K byte di Rom contenenti il sistema operativo, un insieme di routine in linguaggio macchina. Ci occuperemo per l'appunto di questi 16 K byte di Rom, dopo aver parlato un po' di interrupt e di programmazione parallela.

Gli Interrupt

Tutti (o quasi) gli integrati del Commodore 64 lavorano in sincronia tra loro gra-

zie ad un orologio interno detto clock che ad intervalli di tempo regolari, dell'ordine di un milionesimo di secondo, emette un impulso. Grazie al clock, ad esempio, è possibile che più processori accedano alla memoria (che può esaudire una sola richiesta di dato per volta) scongiurando pericolose collisioni. Un esempio tipico è quello del VIC II: per visualizzare la schermata su video, deve continuamente accedere alla mappa video per sapere appunto cosa mostrare. Questo avviene continuamente. Se "Printiamo" qualcosa, il 6510, tra le tante cose che fa (le analizzeremo meglio in seguito) riversa nella mappa video i dati stampati. La sincronia tra VIC II e 6510 è ottenuta proprio riferendosi continuamente al clock: il 6567 accede alla memoria quando il microprocessore non lo fa, e viceversa. Come dire che si sono spartiti il tempo. Questo è un caso di programmazione parallela; in uno stesso istante, due processi sono in opera: quello relativo alla visualizzazione da parte del VIC II e ciò che il microprocessore sta, dal canto suo, svolgendo. Due processori, due processi.

Almeno da quanto risulta dal discorso appena fatto. Per la precisione, le cose non vanno proprio così: c'è un ulteriore livello, volutamente non considerato. Il 6510, sebbene "un" (quantitativamente) processore, generalmente esegue in parallelo due programmi. Parallelamente, non nel vero senso della parola: questo ulteriore livello è simulato grazie al meccanismo degli interrupt.

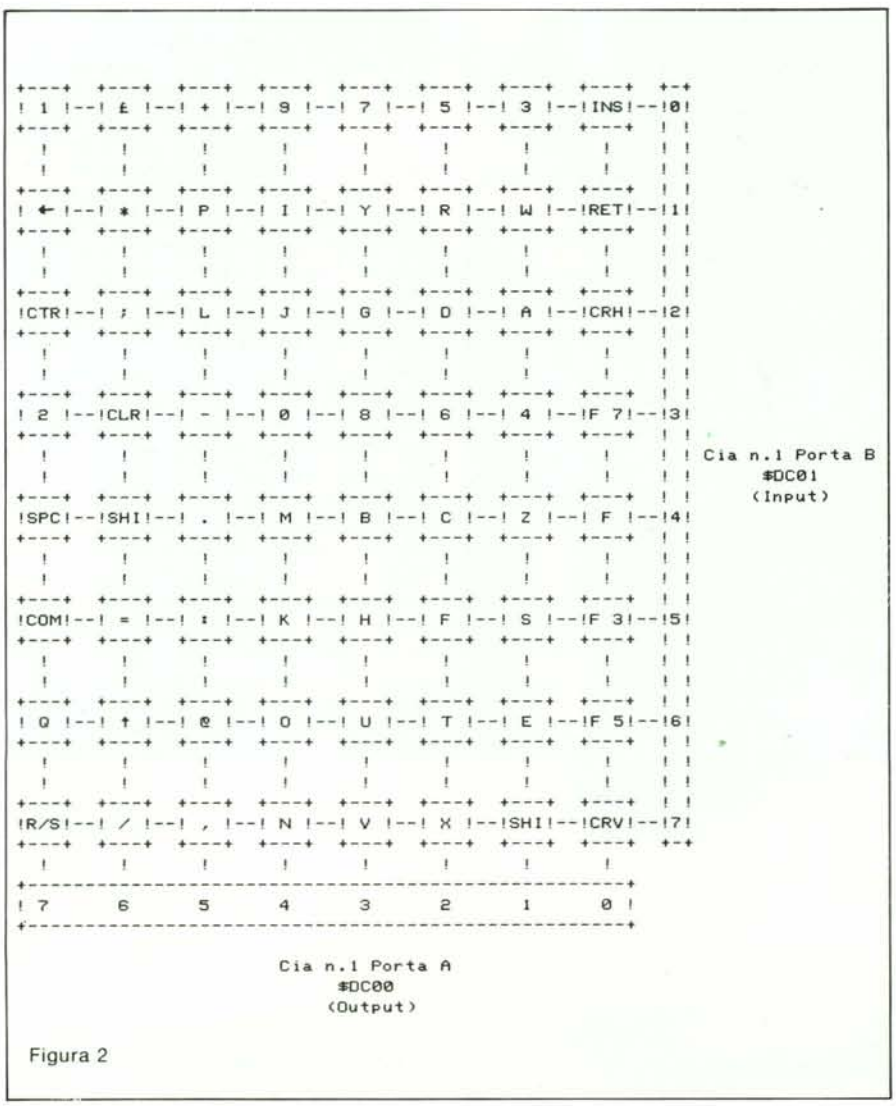

Figura 2

Ogni 60-esimo di secondo, al 6510 giunge un altro segnale: un'interruzione da parte del CIA n. 1. A seguito di questa, il microprocessore molla tutto ed esegue il programma di manipolazione dell'interruzione, per poi ritornare al programma interrotto, al termine. Passato un altro 60-esimo di secondo, il tutto si ripete. Solo in alcuni casi l'interruzione può essere ignorata, ad esempio quando si sta dialogando con periferiche esterne e non si vuole perdere tempo. Il sistema operativo questo lo fa automaticamente: ogni volta che è invocata una qualsiasi routine che riguarda l'Input/Output, il comando SEI (SEt Interrupt disable) rende sordo il 6510 alle interruzioni mascherabili. Al termine della routine, il comando CLI (CLear Interrupt disable) ripristina le interruzioni.

Il programma di manipolazione delle interruzioni serve essenzialmente per due scopi: scandire la tastiera se è stato premuto qualche tasto e aggiornare l'orologio interno (variabili TI e TI$).

Tutti sanno che il Commodore 64 ha l'orologio. La variabile TI, qualsiasi cosa si stia facendo (tranne l'I/O) è automaticamente incrementata ogni 60-esimo di secondo. Anche se è in esecuzione un programma Basic. Ciò dimostra che il 6510, oltre ad eseguire (tramite interprete) programmi Basic, "parallelamente" incrementa l'orologio.

In figura 2 è mostrata la matrice dei tasti, 64 in tutto. Per la scansione della tastiera sono utilizzati due registri del CIA n. 1: uno come Input e uno come Output. Un registro Output può essere visto come un insieme di 8 fili elettrici (numerati da 0 a 7) sui quali ci può essere o no presente tensione. Un registro di input può riconoscere su quali degli 8 fili è presente una tensione. È anche possibile usare un registro in parte come input e in parte come output.

Dando uno sguardo alla figura 2, la scansione avviene a grandi linee così:

1) Sia A = 7.
2) Si manda tensione al filo A del registro Output.
3) Se nel registro Input è rilevata una tensione è stato premuto un tasto della colonna A (la riga si stabilisce controllando nel registro Input quale degli 8 fili porta corrente). Si esce dalla routine.
4) Sia A = A-1.
5) Se A $\geq$ =0 si torna al punto 2.
6) Fine.

Anche la scansione della tastiera è tangibilmente all'opera, anche se non ce ne accorgiamo. Prova ne dà la pressione del tasto Run/Stop, durante l'esecuzione di un

programma Basic. Si ferma l'esecuzione, quindi il processore se n'è accorto!

La locazione di memoria 203, contiene costantemente il codice del tasto premuto, secondo la matrice di figura 2:

codice = riga + colonna * 8

ad esempio, il tasto 4 sta sulla colonna 1, riga 3: il suo codice è 3+1*8=11.

Digitate il seguente programmino e pigiate qualche tasto:

```
10 PRINT PEEK (203)
20 GOTO 10
```

vedrete i codici dei tasti premuti. Per risalire dal codice tasto premuto all'effettivo codice ASCII, il sistema operativo si avvale di 4 tabelle (codice tastiera, codice ASCII), rispettivamente per i tasti usati normalmente, shiftati, preceduti dal logo Commodore o da CTRL. La pressione di questi tre tasti "prefisso" è segnalata dal byte 653.

L'unico tasto non compreso nella matrice è il Restore: logicamente non fa parte della tastiera, è direttamente collegato al piedino NMI del 6510. Vedremo più avanti il suo funzionamento.

L'accensione e il Reset

Al momento dell'accensione, prima del fantomatico READY, si susseguono all'interno del 64 una serie di operazioni di inizializzazione. Lo stesso avviene se si resetta il sistema ponendo per qualche attimo un piccolo ponticello elettrico tra i contatti 1 e 3 della User Port.

L'inizializzazione riguarda essenzialmente le variabili di sistema e i vari processori ausiliari. Quando il 6510 riceve un Reset, blocca tutto ed esegue un apposito programma, sullo stile delle interruzioni. Le sostanziali (uniche) differenze sono che non è possibile mascherare un Reset e che al termine non si ritorna al punto interrotto.

Il programma di Reset essenzialmente compie queste operazioni:

1) Disabilita le interruzioni (quando resetta non vuol essere disturbato ...).

2) Controlla che non siano presenti Cartridge (in caso affermativo il controllo passa a questa e ... generalmente appare la scritta PRESS FIRE BUTTON!).

3) Inizializza i registri dei 2 CIA.

4) Inizializza le variabili di sistema.

5) Inizializza il timer delle interruzioni.

6) Abilita le interruzioni.

7) Stampa Commodore Basic v2 ecc.

8) Stampa Ready.

9) Attende la pressione del [RETURN] per eseguire linee Basic.

Notare che dal passo 6 in poi, è già attiva la scansione della tastiera: solo per questo è lecito il passo 9 che controlla la pressione del [RETURN].

Il tasto Restore, direttamente collegato al piedino NMI del microprocessore, corrisponde anch'esso ad una richiesta di interruzione, anche se nel 64 è assimilabile di più a un "semi-reset". NMI sta per Interrupt Non Mascherabile, e il nome sottolinea l'obbligo da parte del 6510 ad accoglierlo.

Ogni volta che tocchiamo il Restore, il microprocessore molla tutto ed esegue il programma dell'NMI.

Per questo motivo è sconsigliabile premerlo mentre è in corso un'operazione di I/O, può far perdere tempo e far rompere la sincronia di scambio di segnali tra unità e periferica.

Teniamo a evidenziare che basta premere solo il Restore per provocare l'interruzione.

È all'interno del programma di manipolazione NMI che si controlla che sia premuto anche il Run/Stop: in caso negativo, si ritorna al programma interrotto senza riinizializzare le variabili di sistema.

END	128	SPC(	166
FOR	129	THEN	167
NEXT	130	NOT	168
DATA	131	STEP	169
INPUT#	132	+	170
INPUT	133	-	171
DIM	134	*	172
READ	135	/	173
LET	136	↑	174
GOTO	137	AND	175
RUN	138	OR	176
IF	139	>	177
RESTORE	140	=	178
GOSUB	141	<	179
RETURN	142	SGN	180
REM	143	INT	181
STOP	144	ABS	182
ON	145	USR	183
WAIT	146	FRE	184
LOAD	147	POS	185
SAVE	148	SQR	186
VERIFY	149	RND	187
DEF	150	LOG	188
POKE	151	EXP	189
PRINT#	152	COS	190
PRINT	153	SIN	191
CONT	154	TAN	192
LIST	155	ATN	193
CLR	156	PEEK	194
CMD	157	LEN	195
SYS	158	STR$	196
OPEN	159	VAL	197
CLOSE	160	ASC	198
GET	161	CHR$	199
NEW	162	LEFT$	200
TAB(	163	RIGHT$	201
TO	164	MID$	202
FN	165	GO	203

Figura 3

Cursore lampeggiante

Immaginiamo ora di trovarci davanti al computer, subito dopo l'accensione. Sotto la scritta Ready, il cursore sta lampeggiando.

Sappiamo che il sistema è in grado di riconoscere i tasti premuti, grazie alla scansione della tastiera, e pazientemente attende un [RETURN] per eseguire linee Basic. Digitiamo:

```
10 PRINT "CIAO"
```

chiusi gli apici battiamo [RETURN]. Come arcinoto abbiamo inserito una linea Basic in memoria. Analizziamo cosa è avvenuto, passo-passo, all'interno del 64.

Oltre che su video, quando scriviamo qualcosa, automaticamente "carichiamo" il buffer di Input, un insieme di 80 byte locati a partire da $0200, 512 in decimale. Battendo [RETURN], il 64 sa già che abbiamo a che fare con una linea Basic e non con un comando diretto. Ciò è importante, essendo diverso il comportamento della macchina nei due casi. Ritorniamo alla nostra linea 10.

Inizia la fase di tokenizzazione: per risparmiare spazio in memoria le parole chiave del Basic sono trasformate in codici a un solo byte.

In figura 3 è riportata la lista delle parole riservate del Basic con relativi codici Token. Chiaramente, tutto ciò che non è statement Basic, occupa un byte per ogni carattere, secondo la codifica ASCII.

Per la tokenizzazione il sistema operativo usa una tabella locata a partire dall'indirizzo hex $A09E, dove sono elencate in ordine tutte le parole chiave.

Trovato l'opportuno spazio, la linea (tokenizzata) viene trasferita in memoria. Supposto che era la prima linea battuta, verrà inserita a partire dall'indirizzo decimale 2049. Se avevamo già in memoria qualcosa, il problema era un po' più grave, essendo necessario creare lo spazio per inserire la nuova linea nella posizione corretta.

Il tutto si conclude saltando al pezzettino di interprete Basic che implementa l'istruzione CLR. Infatti, inserendo una linea, si perde il contenuto di tutte le variabili.

Proviamo ora a digitare RUN, e a battere nuovamente [RETURN].

Siamo nel caso di comando diretto, anch'esso posto nel Basic input buffer a partire dalla locazione 512. Per uniformità di comportamento, sebbene sarebbe superfluo dal punto di vista logico, anche i comandi diretti vengono prima tokenizzati e poi eseguiti. A tokenizzazione avvenuta, è invocata un'altra routine, detta Execute Statements: non fa altro che prelevare un codice token e passare il controllo al pezzettino di interprete Basic che implementa l'istruzione.

Anche per questo scopo è usata una tabella, al codice token è sottratto 128 e sommato 40972 e 40973 per ottenere i due byte dove è conservato l'indirizzo di partenza del pezzettino di interprete che implementa l'istruzione da eseguire.

Quando si esegue un list, bisogna riconvertire i codici token nelle relative stringhe-comando.

Per far questo è nuovamente utilizzata la tabella delle istruzioni locata a $A09E: dato il codice token, sottratto ad esso 128, si ottiene la posizione relativa nella tabella.

Tutte le parole chiave sono separate tra loro usando l'ultimo carattere del nome shiftato.

In figura 4 è mostrato come un programma Basic occupa la memoria. La locazione 2048, la prima disponibile, deve contenere sempre 0, pena messaggi di errore al primo Run o New dato. Segue la prima linea inserita, a partire quindi dalla locazione 2049.

I primi due byte di ogni linea, sono il puntatore alla linea successiva, e contengono una coppia di zeri se la linea è vuota. Sono usati dal sistema per ripescare più o meno velocemente una linea Basic diversa dalla prima o dalla successiva (molto frequente quando avviene un salto).

I successivi due byte contengono il numero linea, codificato in binario, parte bassa nel primo dei due byte e parte alta nell'altro. Di seguito è posta la linea Basic tokenizzata. Come terminatore è usato lo 0.

Adiacenti al programma Basic, sono poste le variabili usate. Per questo motivo, quando si inserisce una nuova linea, per evitare di perderne qualcuna è meglio perderle tutte.

Sembrerebbe un discorso strano, ma dal punto di vista informatico regge perfettamente.

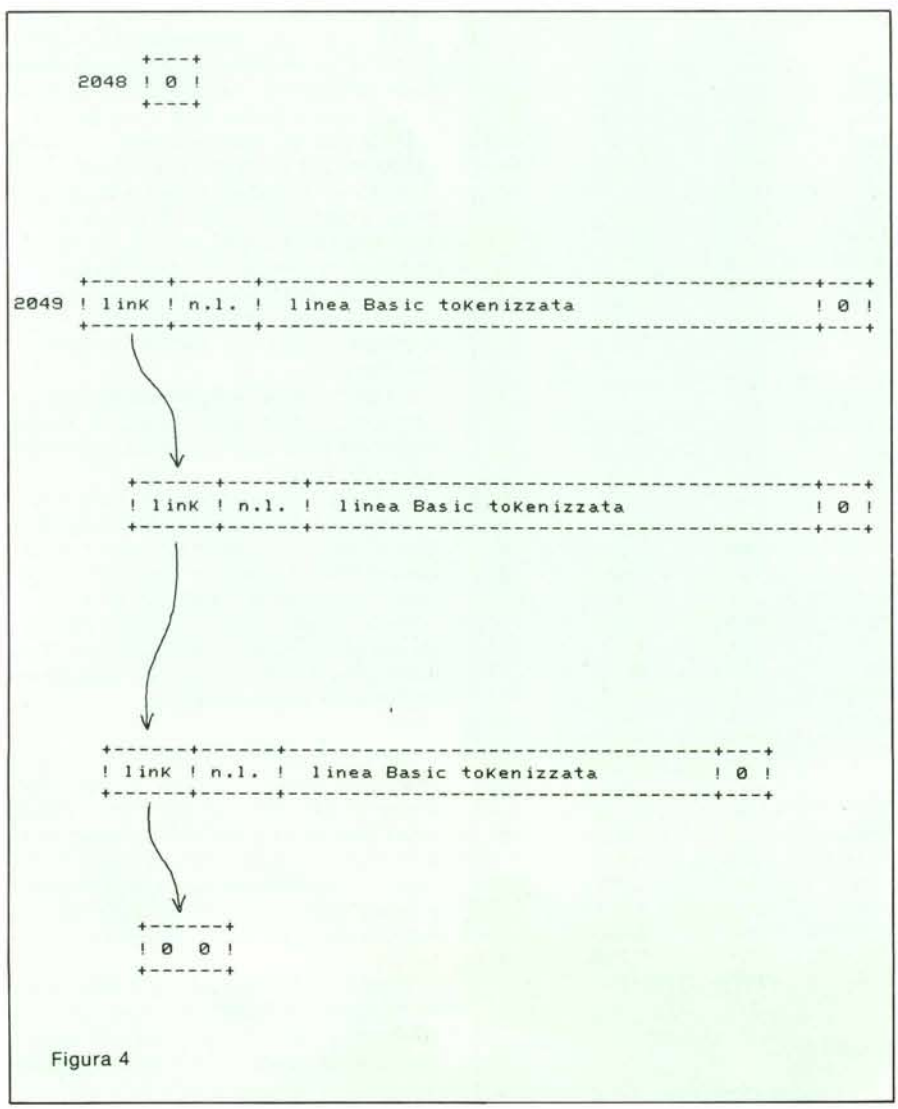

Figura 4

L'ADP Basic

Nei prossimi numeri, vedremo come è possibile manipolare l'interprete Basic per aggiungere nuovi comandi direttamente eseguibili da tastiera o da programma.

Come esempio verrà presentato un potente Tool di istruzioni per facilitare l'uso delle periferiche Commodore: Disco, Plotter, MPS 801 e MPS 802. Qualche nome in anteprima?

CAT
DSAVE
DLOAD
BSAVE
BLOAD
DISKNAME
FLASH
RANGE
PAGING
USING
OUT
LPRINT
HCOPY
MOVE
DRAW
CHARSIZE
COLOR
e tante altre

A proposito: ADP Basic sta per Advanced Disk Printers Basic e non per Andrea de Prisco Basic, come (... e forse hanno ragione) i maligni potrebbero pensare (!).

MC