100 nomi in ordine alfabetico in 7 secondi

Shell-Metzer Sort

in linguaggio macchina per CBM 8032

di Pierluigi Panunzi

Il problema dell'ordinamento di dati, che gli inglesi chiamano "Sort", è un assillo quasi quotidiano del povero programmatore, specialmente se ha a che fare con archivi contenenti montagne di dati.

Affidare al computer l'ingrato compito di ordinare questa "montagna", comporta un fondamentale problema: il tempo di ordinamento.

Provate infatti a scrivere un programma di Sort in BASIC: impostato il fatidico "run", dovremo fare i conti con la durata dell'elaborazione. Mentre per qualche decina di "parole" il risultato si ha in tempi dell'ordine del secondo, già per due-trecento parole si hanno tempi molto lunghi: tra l'altro, anche un minuto può sembrare un'eternità (provare per credere!), tanto siamo abituati alle risposte per lo più istantanee del nostro computer.

Perciò, se abbiamo ad esempio da ordinare mille cognomi contenuti in un archivio, o dobbiamo rassegnarci ad aspettare oppure possiamo rivolgere la domanda al computer in un linguaggio a lui più familiare: il linguaggio macchina.

È proprio quello che abbiamo fatto, realizzando un programma di Sort in assembler per il computer CBM 8032 della Commodore.

Sul n. 3 di MC è già apparso un programma di Sort per l'Apple II, utilizzante però un differente algoritmo di ordinamento, l'"heapsort".

Nel nostro caso ci siamo rivolti all'algoritmo di Shell-Metzer, in quanto è direttamente applicabile ad un vettore già residente in memoria e viceversa non richiede ulteriori occupazioni di memoria se non per alcune variabili rappresentanti puntatori o contatori: ciò è un vantaggio specialmente quando tutta la zona di memoria (nel nostro caso i 32K di RAM) è occupata dal nostro programma e dal vettore da ordinare. Tra l'altro, le celle di memoria di cui si parlava prima non vanno minimamente ad intaccare tale zona "preziosa".

Prima di passare alla descrizione del programma, fissiamone in dettaglio le caratteristiche: innanzitutto consente di ordinare un vettore, formato da stringhe di caratteri alfanumerici, di "grandezza" qualsiasi compatibilmente con la memoria a disposizione; inoltre tale programma è direttamente richiamabile dal BASIC tramite l'istruzione "usr", come meglio vedremo in seguito.

"Last but not least", come dicono quelli che chiamano "Sort" l'ordinamento, il nostro programma, analogamente al già citato programma di Bo Arnklit, richiede tempi di elaborazione veramente fantastici: 10 elementi vengono ordinati in un "batter d'occhio" (e forse anche meno!), 100 elementi in meno di un secondo, 1000 elementi in 7 secondi, 2000 elementi in una ventina di secondi e così via. Il tutto in un tempo "ridicolmente" più basso di quello richiesto per la sola introduzione di tali elementi nel computer o per la loro generazione da parte del computer stesso (ad esempio nel programma di prova che descriveremo più avanti).

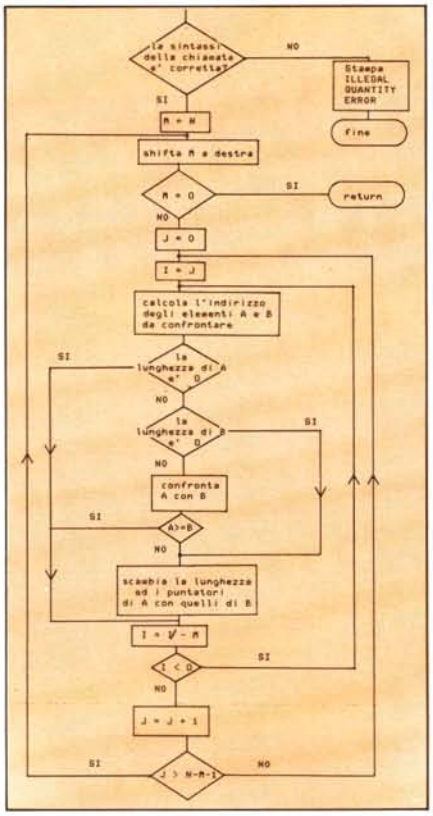

Figura 1 - Flow chart del programma "Shell-Metzer Sort".

La gestione delle stringhe nell'8032

Così come nell'Apple II, anche nel PET la gestione dei vettori (nonché delle matrici) formati da stringhe alfanumeriche avviene in maniera molto semplice e a dir poco geniale: a partire da un certo indirizzo di memoria (in RAM), dipendente dalla lunghezza del programma BASIC e dalle variabili che si incontrano all'atto dell'elaborazione, comparirà l'"array header", cioè l'"intestazione" (o meglio le caratteristiche) dell'array desiderato.

In particolare, supponendo che il nostro vettore si chiami "a$(i)", troveremo sette locazioni di memoria contenenti appunto le caratteristiche del vettore: facendo riferimento alla figura 2, vediamo che i primi due byte contengono la codifica da parte del Sistema Operativo del nome della variabile, poi abbiamo il puntatore all'array successivo e quindi tre byte che ci interessano direttamente: il numero di "dimensioni" dell'array (che può arrivare a 255 e che nel nostro caso dovrà essere uguale ad 1) ed il numero di elementi, (al massimo 65535, compatibilmente, al solito, con la memoria disponibile) posto in due byte, il primo dei quali, contrariamente alla consuetudine, rappresenta il "byte alto" ed il successivo il "byte basso".

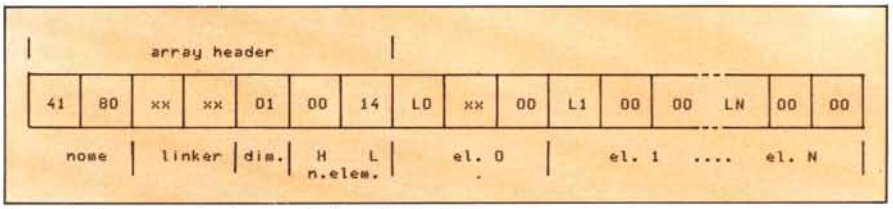

Figura 2 - Rappresentazione in memoria di un vettore-stringa.

Successivamente, se N è la dimensione del vettore, impostata da programma con un'istruzione "dim a$(N)", saranno presenti N+1 terne di celle: il perché dell'N+1 risiede nel fatto che in realtà si deve contare anche l'"elemento 0".

Invece il significato dei tre byte è quello già visto nel caso dell'Apple: il primo rappresenta la lunghezza della stringa (al massimo può arrivare a 255 caratteri) e i due successivi rappresentano l'indirizzo della cella di memoria a partire dalla quale è presente la stringa stessa. In questo modo

```
5 poke52,0:poke53,120:clr:print"J";
6 input"n=";n
10 poke1,0:poke2,120:dima$(n)
30 forj=0ton:fori=0to4+5*rnd(1)
40 a$=a$+chr$(rnd(1)*26+65):next
50 a$(j)=a$:a$="":print"@ ";:next
70 print"J";:ti$="000000":t=ti
80 x=usr(a$(0))
90 p=ti-t:t$=ti$:print"B";
110 printp,p/60"secondi",ti$
1000 forj=0ton:printa$(j):next
```

Esempio di applicazione del programma di sort pubblicato.
I "poke" nella prima linea (5) servono per modificare il "top of memory"; la seconda chiede il numero di elementi da inventare e riordinare. La terza linea (10) contiene i due poke che indicano l'indirizzo della subroutine in linguaggio macchina. Le linee 30, 40 e 50 inventano gli n elementi, la 70 fa partire il "cronometro", la 80 finalmente provoca l'esecuzione del sort. Infine, le linee 90 e 110 visualizzano il tempo impiegato per l'ordinamento e l'ultima, la 1000, stampa il vettore ordinato. Per impieghi diversi, ovviamente, basta modificare le routine di introduzione e stampa dei dati.

(ed analogamente si ha per un array ad "n" dimensioni) ogni elemento di un vettore è a lunghezza fissa, lasciando però la più completa variabilità alla lunghezza della stringa stessa: in gergo, il particolare vettore formato da puntatori ad elementi sparsi qua e là nella memoria si chiama "vettore di Iliffe".

Ora dovendo andare a confrontare coppie di elementi del vettore da ordinare, da quanto detto potremo ricavare i due indirizzi degli elementi e, se per caso tali elementi dovessero essere scambiati di posto, ecco che basterà scambiare soltanto le relative terne del vettore di Iliffe: in parole povere basterà scambiare i puntatori, lasciando al loro posto le due stringhe, a tutto vantaggio dei tempi di elaborazione.

Sulla carta tali tempi dipendono dalla lunghezza delle stringhe volta per volta da confrontare, in quanto il confronto tra i due elementi deve essere eseguito componente per componente: in realtà invece tali tempi risultano praticamente indipendenti dalla lunghezza delle due stringhe, dal momento che i confronti avvengono comunque in tempi dell'ordine del micro-secondo.

I tempi riportati all'inizio si riferiscono ad elementi formati da stringhe di caratteri casuali lunghe da 4 a 9 caratteri (tanto per avere un paragone con il già citato programma di heapsort).

Se si ha invece a che fare con elementi di lunghezza fissa (ad esempio appena 2 caratteri) avremo i seguenti tempi di elaborazione: per 10 elementi i soliti... "gnomo-secondi", per 100 elementi mezzo secondo, per 1000 elementi 8 secondi e per 2000 elementi 17 secondi.

Come si vede, per 1000 elementi abbiamo addirittura un tempo maggiore (!), dovuto al "grado di disordine" del vettore generato casualmente in questo caso, mentre per 2000 elementi si ha una diminuzione dell'ordine di due secondi. A parte il fatto che in questo caso il programma andrebbe ancor più vantaggiosamente modificato, eliminando i controlli sulla lunghezza dei due elementi volta per volta confrontati, ottenendo forse tempi ancora più brevi...

Il programma

Facendo riferimento al flow-chart di figura 1 ed al listing disassemblato del programma, possiamo notare che i primi 25 byte contengono alcuni test sulla correttezza della sintassi della chiamata del programma.

La sintassi corretta è la seguente:
<variabile numerica> = usr (<elemento k-esimo del vettore stringa>) dove < variabile numerica> può anche essere l'elemento di un vettore o di una matrice: basta che non sia una stringa; <elemento k-esimo del vettore stringa> indica che il "vettore stringa" da ordinare non deve avere più di una dimensione (cioè non può essere una matrice), mentre "elemento k-esimo" indica che dobbiamo "far capire" al Sistema Operativo (S.O.) che si tratta di un vettore e non di una singola variabile, che ovviamente verrebbe trattata in maniera completamente differente. Va da sé che il valore di k deve essere non maggiore del valore con cui è stato dimensionato il vettore, altrimenti si ha l'interruzione del programma chiamente con la segnalazione di "?bad subscript error".

Se <variabile numerica> è invece una stringa, si avrà la segnalazione di "?type mismatch error"; infine se il <vettore stringa> è in realtà una variabile, si avrà un "?illegal quantity error". Una ricca diagnostica fa sempre bene!....

Riuscire a condensare una così sofisticata "reiezione" ad errori voluti o meno in appena 25 byte, è stato possibile solo andando ad esaminare approfonditamente quali sono le operazioni compiute dal S.O. all'atto della chiamata "usr" ed in particolare cercando tra le varie locazioni di memoria qualcosa che ci potesse tornare utile.

Scendiamo nei dettagli, utili se non altro per far capire a quale "grado di contorcimento mentale" devono arrivare i progettisti di sistemi operativi (e c'è ancora qual-

```
c*

    pc  irq  sr ac xr yr sp
.; b780 e455 34 33 38 36 f6

.- 7800 a5 07     lda $07
.- 7802 d0 03     bne $7807
.- 7804 4c 73 c3  jmp $c373
.- 7807 38        sec
.- 7808 a5 5c     lda $5c
.- 780a e5 2c     sbc $2c
.- 780c a5 5d     lda $5d
.- 780e e5 2d     sbc $2d
.- 7810 90 f2     bcc $7804
.- 7812 a0 04     ldy #$04
.- 7814 b1 5c     lda ($5c),y
.- 7816 c9 01     cmp #$01
.- 7818 d0 ea     bne $7804
.- 781a c8        iny
.- 781b b1 5c     lda ($5c),y
.- 781d 85 4c     sta $4c
.- 781f 85 f5     sta $f5
.- 7821 c8        iny
.- 7822 b1 5c     lda ($5c),y
.- 7824 85 4b     sta $4b
.- 7826 85 f4     sta $f4
.- 7828 ea        nop
.- 7829 ea        nop
.- 782a ea        nop
.- 782b ea        nop
.- 782c ea        nop
.- 782d ea        nop
.- 782e ea        nop
.- 782f 46 4c     lsr $4c
.- 7831 66 4b     ror $4b
.- 7833 a5 4b     lda $4b
.- 7835 05 4c     ora $4c
.- 7837 d0 01     bne $783a
.- 7839 60        rts
.- 783a a0 00     ldy #$00
.- 783c 84 4f     sty $4f
.- 783e ea        nop
.- 783f 84 50     sty $50
.- 7841 a5 4f     lda $4f
.- 7843 85 4d     sta $4d
.- 7845 a5 50     lda $50
.- 7847 85 4e     sta $4e
.- 7849 a5 55     lda $55
.- 784b 85 57     sta $57
.- 784d 85 59     sta $59
.- 784f a5 56     lda $56
.- 7851 85 58     sta $58
.- 7853 85 5a     sta $5a
.- 7855 a2 03     ldx #$03
.- 7857 18        clc
.- 7858 a5 4d     lda $4d
.- 785a 65 57     adc $57
.- 785c 85 57     sta $57
.- 785e a5 4e     lda $4e
.- 7860 65 58     adc $58
.- 7862 85 58     sta $58
.- 7864 18        clc
.- 7865 a5 4d     lda $4d
.- 7867 65 59     adc $59
.- 7869 85 59     sta $59
.- 786b a5 4e     lda $4e
.- 786d 65 5a     adc $5a
.- 786f 85 5a     sta $5a
.- 7871 18        clc
.- 7872 a5 4b     lda $4b
.- 7874 65 57     adc $57
.- 7876 85 57     sta $57
.- 7878 a5 4c     lda $4c
.- 787a 65 58     adc $58
.- 787c 85 58     sta $58
.- 787e ca        dex
.- 787f d0 d6     bne $7857
.- 7881 a9 00     lda #$00
.- 7883 85 07     sta $07
.- 7885 ea        nop
.- 7886 ea        nop
.- 7887 ea        nop
.- 7888 a0 00     ldy #$00
.- 788a b1 57     lda ($57),y
.- 788c f0 68     beq $78f6
.- 788e 85 54     sta $54
.- 7890 b1 59     lda ($59),y
.- 7892 f0 35     beq $78c9
.- 7894 85 5b     sta $5b
.- 7896 c8        iny
.- 7897 b1 57     lda ($57),y
.- 7899 85 f0     sta $f0
.- 789b c8        iny
.- 789c b1 57     lda ($57),y
.- 789e 85 f1     sta $f1
.- 78a0 88        dey
.- 78a1 b1 59     lda ($59),y
.- 78a3 85 f2     sta $f2
.- 78a5 ea        nop
.- 78a6 c8        iny
.- 78a7 b1 59     lda ($59),y
.- 78a9 85 f3     sta $f3
.- 78ab ea        nop
.- 78ac ea        nop
.- 78ad ea        nop
.- 78ae a0 00     ldy #$00
.- 78b0 b1 f0     lda ($f0),y
.- 78b2 d1 f2     cmp ($f2),y
.- 78b4 f0 04     beq $78ba
.- 78b6 b0 3e     bcs $78f6
.- 78b8 90 0f     bcc $78c9
.- 78ba c8        iny
.- 78bb c4 54     cpy $54
.- 78bd b0 04     bcs $78c3
.- 78bf c4 5b     cpy $5b
.- 78c1 90 ed     bcc $78b0
.- 78c3 a5 54     lda $54
.- 78c5 c5 5b     cmp $5b
.- 78c7 b0 2d     bcs $78f6
.- 78c9 a0 00     ldy #$00
.- 78cb a2 03     ldx #$03
.- 78cd b1 57     lda ($57),y
.- 78cf 48        pha
.- 78d0 b1 59     lda ($59),y
.- 78d2 91 57     sta ($57),y
.- 78d4 68        pla
.- 78d5 91 59     sta ($59),y
.- 78d7 c8        iny
.- 78d8 ca        dex
.- 78d9 d0 f2     bne $78cd
.- 78db 38        sec
.- 78dc a5 4d     lda $4d
.- 78de e5 4b     sbc $4b
.- 78e0 85 4d     sta $4d
.- 78e2 a5 4e     lda $4e
.- 78e4 e5 4c     sbc $4c
.- 78e6 85 4e     sta $4e
.- 78e8 ea        nop
.- 78e9 ea        nop
.- 78ea ea        nop
.- 78eb ea        nop
.- 78ec ea        nop
.- 78ed ea        nop
.- 78ee ea        nop
.- 78ef ea        nop
.- 78f0 ea        nop
.- 78f1 90 03     bcc $78f6
.- 78f3 4c 49 78  jmp $7849
.- 78f6 e6 4f     inc $4f
.- 78f8 d0 02     bne $78fc
.- 78fa e6 50     inc $50
.- 78fc ea        nop
.- 78fd ea        nop
.- 78fe a5 f4     lda $f4
.- 7900 38        sec
.- 7901 e5 4b     sbc $4b
.- 7903 85 5c     sta $5c
.- 7905 a5 f5     lda $f5
.- 7907 e5 4c     sbc $4c
.- 7909 85 5d     sta $5d
.- 790b c6 5c     dec $5c
.- 790d a5 5c     lda $5c
.- 790f c9 ff     cmp #$ff
.- 7911 d0 02     bne $7915
.- 7913 c6 5d     dec $5d
.- 7915 38        sec
.- 7916 a5 5c     lda $5c
.- 7918 e5 4f     sbc $4f
.- 791a a5 5d     lda $5d
.- 791c e5 50     sbc $50
.- 791e ea        nop
.- 791f ea        nop
.- 7920 ea        nop
.- 7921 90 03     bcc $7926
.- 7923 4c 41 78  jmp $7841
.- 7926 4c 2f 78  jmp $782f
.- 7929 ea        nop
.- 792a ea        nop
.- 792b ea        nop
.- 792c ea        nop
.- 792d ea        nop
.- 792e ea        nop
.- 792f ea        nop
.- 7930 aa        tax
.- 7931 aa        tax
.- 7932 aa        tax
.- 7933 aa        tax
.- 7934 aa        tax
.- 7935 aa        tax
.- 7936 aa        tax
```

List disassemblato del programma di sort.

cuno che parlando di computer gli "affibbia" il nominativo di Cervello Elettronico!!).

Orbene la cella di memoria $07 conterrà un valore nullo se il parametro della funzione "usr" è una quantità numerica (variabile, costante, floating o fixed point) e viceversa contiene "ff" se invece si tratta di variabile di tipo alfanumerico (stringa): dovrebbe ora essere chiaro il primo test del programma.

Al momento dell'esecuzione, il S.O. pone in alcune celle di memoria gli indirizzi, "dinamici", nel senso che possono variare nel corso dell'elaborazione, delle "frontiere" tra le zone riservate al BASIC, alle variabili semplici, agli array ed alle stringhe. Inoltre pone nelle celle $42,43 il nome della variabile posta come parametro della "usr", nonché il suo indirizzo nelle celle $44,45.

Tale indirizzo però risente del valore di k impostato e perciò non è molto utile in quanto vogliamo puntare all'elemento 0 del vettore e non al k-esimo.

A questo punto abbiamo allora ricercato tra le variabili in pagina 0 ed abbiamo "scoperto" che le locazioni $55,56 contengono l'indirizzo desiderato, mentre le celle $59,5a contengono l'indirizzo dell'"array header", utilissimo per andare a verificare i valori caratteristici del vettore stringa da ordinare.

Ecco che perciò basta controllare il quinto elemento di tale "header" per verificare se è pari ad 1: nel qual caso si tratta di array monodimensionale e perciò di vettore.

A questo punto, se tutto è andato bene, inizia il programma di sort vero e proprio, che ricalca fedelmente il flow-chart e di conseguenza l'algoritmo di Shell-Metzer, leggermente modificato per prevedere l'esistenza del fatidico "elemento 0", altrimenti ignorato (inspiegabilmente!) dall'algoritmo stesso.

```
      PC   IRQ   SR AC XR YR SP
.;  B780 E455 34 33 38 36 FA
.
.:  7800 A5 07 D0 03 4C 73 C3 38
.:  7808 A5 5C E5 2C A5 5D E5 2D
.:  7810 90 F2 A0 04 B1 5C C9 01
.:  7818 D0 EA C8 B1 5C 85 4C 85
.:  7820 F5 C8 B1 5C 85 4B 85 F4
.:  7828 EA EA EA EA EA EA EA 46
.:  7830 4C 66 4B A5 4B 05 4C D0
.:  7838 01 60 A0 00 84 4F EA 84
.:  7840 50 A5 4F 85 4D A5 50 85
.:  7848 4E A5 55 85 57 85 59 A5
.:  7850 56 85 58 85 5A A2 03 18
.:  7858 A5 4D 65 57 85 57 A5 4E
.:  7860 65 58 85 58 18 A5 4D 65
.:  7868 59 85 59 A5 4E 65 5A 85
.:  7870 5A 18 A5 4B 65 57 85 57
.:  7878 A5 4C 65 58 85 58 CA D0
.:  7880 D6 A9 00 85 07 EA EA EA
.:  7888 A0 00 B1 57 F0 68 85 54
.:  7890 B1 59 F0 35 85 5B C8 B1
.:  7898 57 85 F0 C8 B1 57 85 F1
.:  78A0 88 B1 59 85 F2 EA C8 B1
.:  78A8 59 85 F3 EA EA EA A0 00
.:  78B0 B1 F0 D1 F2 F0 04 B0 3E
.:  78B8 90 0F C8 C4 54 B0 04 C4
.:  78C0 5B 90 ED A5 54 C5 5B B0
.:  78C8 2D A0 00 A2 03 B1 57 48
.:  78D0 B1 59 91 57 68 91 59 C8
.:  78D8 CA D0 F2 38 A5 4D E5 4B
.:  78E0 85 4D A5 4E E5 4C 85 4E
.:  78E8 EA EA EA EA EA EA EA EA
.:  78F0 EA 90 03 4C 49 78 E6 4F
.:  78F8 D0 02 E6 50 EA EA A5 F4
.:  7900 38 E5 4B 85 5C A5 F5 E5
.:  7908 4C 85 5D C6 5C A5 5C C9
.:  7910 FF D0 02 C6 5D 38 A5 5C
.:  7918 E5 4F A5 5D E5 50 EA EA
.:  7920 EA 90 03 4C 41 78 4C 2F
.:  7928 78 EA EA EA EA EA EA EA
.:  7930 AA AA AA AA AA AA AA AA
```

List in codice oggetto del programma di sort.

Non ci dilungheremo più di tanto su tale algoritmo, in quanto non prevede soverchie difficoltà: qualche parola la spenderemo invece sul calcolo dell'indirizzo dei due elementi volta per volta da confrontare e sul confronto vero e proprio. I due indirizzi che ci servono sono rappresentati dalle quantità I ed I+M: considerato che l'elemento iniziale ha indirizzo (che chiameremo IND) posto nelle celle $55,56, e che ogni elemento è formato, come visto da tre byte, gli indirizzi diverranno rispettivamente:

IND+3*I e IND+3*(I+M)

Ora perR realizzare tale (semplicissima!) moltiplicazione abbiamo altrettanto semplicemente effettuato tre volte un ciclo che, ogni volta, somma rispettivamente le quantità I ed I+M al contenuto di due coppie di celle ($57,58 e $59,5a) che fungono da puntatori veri e propri.

Più delicato è il discorso riguardante il confronto tra due elementi del vettore: in BASIC basta semplicemente porre "if a$(i)>a$(j)", senza ulteriori preoccupazioni. Invece qui dobbiamo andare a confrontare a due a due i corrispondenti caratteri costituenti le due stringhe, fermandoci se eventualmente una delle due stringhe avesse lunghezza nulla, oppure quando troviamo una disuguaglianza, tra i due caratteri, disuguaglianza che stabilisce così l'ordinamento tra i due elementi.

A seconda poi di quale delle due stringhe "finisce prima", si potrà stabilire un ulteriore criterio di "precedenza".

Tutto ciò è realizzato usando il registro "y" del 6502 come puntatore alle coppie di caratteri e sfruttando lo stato dei flag subito dopo ogni confronto.

Nel caso che i due elementi debbano essere scambiati, è previsto un classico "scambio con cella di salvataggio" rappresentata in questo caso dallo "stack".

L'uso del programma

Trattandosi di un programma in linguaggio macchina, conviene caricarlo da disco prima del programma BASIC.

Dato che è posto nella parte "alta" della memoria RAM, deve essere protetto da "invasioni" da parte delle stringhe: a tale scopo si deve modificare il valore di "top of memory" (contenuto nelle celle $34,35) con le istruzioni "poke 52,0: poke 53,120" e ciò deve essere fatto come prima istruzione.

Quindi, siccome per la chiamata è stata usata la funzione "usr", bisogna porre nelle celle di memoria $01,02 l'indirizzo della routine in linguaggio macchina, cosa che si fa ponendo "poke 1,0: poke 2,120".

Ora, generato in qualche modo il vettore alfanumerico da ordinare, lo si ordina impostando l'istruzione

x = usr (a$(k))

dove al posto di "x", "a$" e "k" ci vanno i nomi delle variabili usate nel programma BASIC: abbiamo visto che "k" può essere anche un valore numerico (p.es. 0).

Fatto ciò non resta altro che eseguire il programma.... MC

Il linguaggio macchina del CBM 8032

Dal momento che abbiamo parlato di linguaggio macchina, pensiamo di facilitare i nostri lettori indicando le modalità di inserimento di un programma in l.m. in memoria: per chi lo sa già si tratterà di un semplice pro-memoria.

Dal BASIC si passa al Machine Language Monitor (M.L.M.) con il comando "sys4": ora si hanno a disposizione varie possibilità di operazioni, che ora andiamo ad analizzare.

— *m XXXX YYYY* mostra i contenuti esadecimali delle locazioni di memoria comprese tra XXXX ed YYYY (valori espressi in esadecimale), per gruppi di 8 byte. Portando il cursore su uno di tali valori, si può introdurre il valore desiderato e premendo RETURN tale valore sarà immesso in memoria. Così per blocchi di 8 byte alla volta, è possibile introdurre un programma in l.m.

— *g XXXX* esegue la routine in l.m. presente all'indirizzo XXXX.

— *r* mostra (e permette di alterare) il contenuto dei registri interni del 6502 (accumulatore, x, y, ecc.).

— *s "nomefile", 08, XXXX, YYYY* salva su disco le locazioni contenute tra XXXX ed YYYY, creando un file chiamato "nomefile"; se si vuole salvare su di un disco posto nel drive n. 1, "nomefile" deve diventare "1: nomefile".

— *l "nomefile"* caricą in memoria il file "nomefile" ed equivale al comando "dload" del BASIC.

— *x* permette il ritorno al BASIC.

NEWS-NEWS-NEWS-NEWS

L&L computers

Via Galvani, 6/M - 70125 Bari
Tel. 080/364855 (2 linee)
Telex 860284 LLBA-I

NEW

NEWS-NEWS-NEWS-NEWS-NEWS-NEWS-NEWS-NEWS-

C'è chi preferisce le mele,
chi i limoni,
noi proponiamo

ORANGE

Unità centrale 48 K RAM utente
Interfaccia per registratore a cassette
Dispositivi di input analogici
12 K ROM contenenti Basic esteso,
Monitor e Disassembler

MICROCOMPUTER ORANGE 48 K RAM L. 1.350.000

Disk drive 5" con controller	**L. 890.000**
Disk Drive 5" aggiuntivo	**L. 720.000**
Monitor 9" fosfori verdi	**L. 175.000**
Monitor 12" fosfori verdi antiriflesso	**L. 210.000**
Stampante Epson MX 80F/T III°	**L. 1.100.000**
Stampante Epson MX 82F/T III°	**L. 1.250.000**
Stampante Epson MX 100F/T III°	**L. 1.440.000**

E in più è disponibile tutta la gamma di interfacce, monitor, stampanti e accessori, naturalmente compatibili con arance, mele e limoni.

Scheda esp. memoria 16 K RAM	**L. 160.000**
Scheda esp. memoria 128 K RAM	**L. 375.000**
Scheda Z 80 per CP/M	**L. 200.000**
Scheda video 80 colonne	**L. 260.000**
Scheda buffer 8 K per stamp. Epson	**L. 270.000**
Scheda buffer 8 K con contenitore per tutte le stampanti con ingresso Centronics	**L. 290.000**
Interfaccia parallela Centronics	**L. 120.000**
Interfaccia parallela Centronics con grafica per Epson	**L. 180.000**
Interfaccia seriale RS-232 (50-9.600 baud programmabile)	**L. 170.000**
Scheda buffer per tastiera	**L. 180.000**

Prezzi IVA 18% esclusa - Pagamento 1/3 all'ordine saldo contrassegno.
Indirizzare le richieste a: L&L Computers - Via Galvani, 6/M - 70125 Bari
Pronta consegna - Garanzia 3 mesi.