MSORT 64:

un sort in lm per il 64

di Dario Accornero - Roma

MSORT 64 esegue ordinamenti ascendenti su vettori con velocità molto elevata poiché è scritto completamente in linguaggio macchina ed utilizza un algoritmo di SORT non «a bolla».
L'operazione di SORT è svolta direttamente in memoria; gli scambi tra le coppie di elementi operano, per il SORT numerico, sul numero vero e proprio e non sulla sua posizione nell'array, mentre, per il SORT alfabetico, sull'«header» delle stringhe e non sulla posizione. Questa tecnica implementata in linguaggio macchina, insieme all'algoritmo non tradizionale adottato nel programma, rendono MSORT 64 piuttosto veloce e facile da usare.

L'algoritmo di SORT

Si basa su un algoritmo pubblicato da MCmicrocomputer nel numero 38 (febbraio 1985) rubrica MCmicrofacile. L'idea di base è quella di individuare il numero più basso (o la stringa alfabeticamente inferiore) rispetto a parti dell'array via via più piccole fino al suo completamento. In altre parole, si isola prima il numero più piccolo di tutto l'array e lo si mette al primo posto. Poi si individua il numero più pic-

Nota

I codici di controllo nei listati sono riportati in forma «esplicita», in conseguenza dell'impiego della stampante Star NL-10 e relativa interfaccia per Commodore. Ovviamente, nella digitazione del programma è necessario usare i consueti tasti che corrispondono alle indicazioni fra parentesi: ad esempio cursore destro per (RGHT), CTRL-3 per (RED) eccetera.

SORTS LOADER

```
5 PRINT"(CLR)(SWLC)(DISH)(LBLU)";:POKE53281,6:POKE53280,14:PRINT"(RGHT)(DOW
N)(RVS)↘●┌┘(OFF) PROGRAM";
6 PRINTSPC(13)"┐┌\└┘┌┘─ 64":PRINT"(RGHT)──── ──────":PRINT"(─) 1986 BY ─
ARIO●OFT
8 PRINT"(DOWN)(DOWN)(RVS)┐└─♠●─ WAIT(OFF) - LOADING M.C. IN MEMORY"
10 FORI=49152TO49364:READA:POKEI,A:NEXT
20 FORI=49408TO49635:READA:POKEI,A:NEXT
25 PRINT"(DOWN)SYS49152,ARRAY(1),ELEMENTS: NUMBER SORT"
26 PRINT"SYS49408,ARRAY$(1),ELEMENTS: STRING SORT"
30 PRINT"(DOWN)(RGHT)1. ●♠×─ SORT PER NUMERI":PRINT"(RGHT)2. ●♠×─ SORT PER
STRINGHE"
35 PRINT"(RGHT)3. ●♠×─ ENTRAMBI INSIEME":PRINT"(RGHT)4. ─/─":POKE198,0
40 WAIT198,1:GETA$:IFA$=""THEN40
45 IFA$="1"THEN60
47 IFA$="2"THEN70
49 IFA$="3"THEN80
50 IFA$="4"THENPRINT"(CLR)(DOWN)(RGHT)↘●┌┘ PROGRAM:(DOWN)(DOWN)!HE ─ND":CL
R:END
60 PRINT"(DOWN)●♠×↘/! FLPSORT ┌/ ─↘●/":NA$="@0:FLPSORT,P,W"
62 RESTORE:OPEN4,8,4,NA$:PRINT#4,CHR$(0)CHR$(192);
65 FORI=49152TO49364:READA:PRINT#4,CHR$(A);:NEXT:CLOSE4:PRINT"(CLR)":GOTO30
70 PRINT"(DOWN)●♠×↘/! STRSORT ┌/ ─↘●/":NA$="@0:STRSORT,P,W"
72 RESTORE:FORI=49152TO49364:READA:NEXT:OPEN4,8,4,NA$:PRINT#4,CHR$(0)CHR$(1
93);
75 FORI=49408TO49635:READA:PRINT#4,CHR$(A);:NEXT:CLOSE4:PRINT"(CLR)":GOTO30
80 PRINT"(CLR)POKE43,0:POKE44,192:POKE45,227:POKE46,193:SAVE"CHR$(34)"@:MSO
RTS-MC";
85 PRINTCHR$(34)",8":POKE254,PEEK(45):POKE255,PEEK(46)
86 PRINT"(HOME)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)POKE43,1:POKE44,8:POKE45,PEEK(
254):POKE46,P─(255):PRINT"CHR$(34);
87 PRINT"(CLR)"CHR$(34)":GOTO30"
90 POKE198,3:POKE631,19:POKE632,13:POKE633,13:END
49151 REM *** SORT FOR NUMBERS ***
49152 DATA032,253,174,032,139,176,133,252,132,253,032,253,174,032,138
49167 DATA173,032,247,183,132,176,133,177,200,132,166,133,167,208,002
49182 DATA230,167,162,001,134,163,202,134,164,165,252,164,253,133,168
49197 DATA132,169,166,163,164,164,232,134,172,208,001,200,132,173,024
49212 DATA165,252,105,005,133,254,165,253,105,000,133,255,165,254,164
49227 DATA255,032,162,187,165,168,164,169,032,091,188,201,001,240,008
49242 DATA165,254,164,255,133,168,132,169,024,165,254,105,005,133,254
49257 DATA144,002,230,255,230,172,208,002,230,173,165,173,197,167,208
49272 DATA207,165,172,197,166,208,201,164,253,196,169,208,006,165,252
49287 DATA197,168,240,040,165,252,032,162,187,162,080,160,003,032,212
49302 DATA187,165,168,164,169,032,162,187,166,252,164,253,032,212,187
49317 DATA169,080,160,003,032,162,187,166,168,164,169,032,212,187,024
49332 DATA165,252,105,005,133,252,144,002,230,253,230,163,208,002,230
49347 DATA164,165,164,197,177,208,007,165,163,197,176,208,001,096,076
49362 DATA039,192,000
49407 REM *** SORT FOR STRINGS ***
49408 DATA032,253,174,032,139,176,133,252,132,253,032,253,174,032,138
49423 DATA173,032,247,183,132,176,133,177,200,132,166,133,167,208,002
49438 DATA230,167,162,001,134,163,202,134,164,165,252,164,253,133,170
49453 DATA132,171,166,163,164,164,232,134,172,208,001,200,132,173,024
49468 DATA165,252,105,003,133,254,165,253,105,000,133,255,160,000,177
49483 DATA254,209,170,144,002,177,170,133,002,200,177,254,153,102,193
49498 DATA177,170,153,105,193,200,192,003,208,241,162,000,189,000,000
49513 DATA221,000,000,240,013,176,024,165,254,164,255,133,170,132,171
49528 DATA076,136,193,232,228,002,208,230,160,000,177,170,197,002,208
49543 DATA232,024,165,254,105,003,133,254,144,002,230,255,230,172,208
49558 DATA002,230,173,165,173,197,167,208,169,165,172,197,166,208,163
49573 DATA165,171,197,253,208,008,165,170,197,252,208,002,240,015,160
49588 DATA002,177,252,072,177,170,145,252,104,145,170,136,016,243,024
49603 DATA165,252,105,003,133,252,144,002,230,253,230,163,208,002,230
49618 DATA164,165,164,197,177,208,007,165,163,197,176,208,001,096,076
49633 DATA039,193,000
```

colo dell'array dal secondo elemento in poi, cosicché, non esaminando il primo che sarebbe per forza più basso poiché il più piccolo dell'array, si trova il secondo numero più basso del vettore. In maniera ancora più semplice, si trova il numero più basso, poi quello successivo, un po' più grande, poi il terzo, un po' più grande del secondo e così via, fino all'ultimo elemento.
L'algoritmo ha quindi un numero di passaggi che è fisso per ogni quantità di elementi: esso dipende allora direttamente dal numero degli elementi e non dalla loro posizione. Non c'è bisogno di fare ripetuti passaggi su tutto l'archivio o tornare sui propri passi come in alcuni algoritmi «a bolla», che operano troppi confronti e passaggi inutili per essere utilizzabili seriamente.

L'algoritmo utilizzato è quasi identico a quello pubblicato sul numero 38 di MC, con una differenza. Ho notato un passaggio inutile nell'algoritmo in questione, o almeno inutile per il linguaggio macchina: viene memorizzato il numero più piccolo fino a un dato momento trovato insieme alla sua posizione. È più intuitivo memorizzarne soltanto la posizione, poiché nel confronto sarà ciò che verrà usato. Infatti, il minimo memorizzato è come uno stack che può essere usato per il confronto, ma non per lo scambio.
Eliminato questo passaggio, l'algoritmo acquista ulteriore velocità e divie-

```
FSORT DEMO
PROGRAMMA DIMOSTRATIVO PER L'ORDINAMENTO DI NUMERI


10 INPUT"(CLR)(DOWN)●UANTI ELEMENTI:";A:DIMA(A):PRINT"(CLR)─┘↑└┐ "A"ELEMENT-AR
RAY..."
20 FORI=1TOA:A(I)=INT(RND(0)*10000):PRINTI"(UP)":NEXT
30 PRINT"(DOWN)↗N TASTO PER IL SORT":POKE198,0:WAIT198,1:POKE198,0
40 PRINT"●┌─┘└┐↗ ...":TI$="000000":SYS49152,A(1),A:T=TI:T$=TI$
50 PRINT"(DOWN)↗N TASTO PER VEDERE I NUMERI":POKE198,0:WAIT198,1:POKE198,0
60 PRINT"(CLR)":FORI=1TOA:PRINTA(I):NEXT
70 PRINT"└─┐ RICHIESTO:"RIGHT$(T$,2)" SECS"T"JIFFIES"
```

```
MSORT Per numeri (floating-Point)
assembler listing


C000  20 FD AE JSR $AEFD ; Prendi Parametri: nome dell'array
C003  20 8B B0 JSR $B08B
C006  85 FC    STA $FC
C008  84 FD    STY $FD
C00A  20 FD AE JSR $AEFD ; Prendi Parametri: elementi da "sort"are
C00D  20 8A AD JSR $AD8A
C010  20 F7 B7 JSR $B7F7
C013  84 B0    STY $B0
C015  85 B1    STA $B1
C017  C8       INY       ; FOR co = 1 TO n-1
C018  84 A6    STY $A6
C01A  85 A7    STA $A7
C01C  D0 02    BNE $C020
C01E  E6 A7    INC $A7
C020  A2 01    LDX #$01
C022  86 A3    STX $A3
C024  CA       DEX
C025  86 A4    STX $A4
C027  A5 FC    LDA $FC   ; t1 = co
C029  A4 FD    LDY $FD
C02B  85 A8    STA $A8
C02D  84 A9    STY $A9
C02F  A6 A3    LDX $A3   ; FOR co2 = co+1 TO n
C031  A4 A4    LDY $A4
C033  E8       INX
C034  86 AC    STX $AC
C036  D0 01    BNE $C039
C038  C8       INY
C039  84 AD    STY $AD
C03B  18       CLC
C03C  A5 FC    LDA $FC
C03E  69 05    ADC #$05
C040  85 FE    STA $FE
C042  A5 FD    LDA $FD
C044  69 00    ADC #$00
C046  85 FF    STA $FF
C048  A5 FE    LDA $FE   ; IF ar(co2)>ar(t1) THEN GOTO xxx
C04A  A4 FF    LDY $FF
C04C  20 A2 BB JSR $BBA2
C04F  A5 A8    LDA $A8
C051  A4 A9    LDY $A9
C053  20 5B BC JSR $BC5B
C056  C9 01    CMP #$01
C058  F0 08    BEQ $C062
C05A  A5 FE    LDA $FE   ; t1 = co2
C05C  A4 FF    LDY $FF
C05E  85 A8    STA $A8
C060  84 A9    STY $A9
C062  18       CLC       ; .xxx NEXT co2
C063  A5 FE    LDA $FE
C065  69 05    ADC #$05
C067  85 FE    STA $FE
C069  90 02    BCC $C06D
C06B  E6 FF    INC $FF
C06D  E6 AC    INC $AC
C06F  D0 02    BNE $C073
C071  E6 AD    INC $AD
C073  A5 AD    LDA $AD
C075  C5 A7    CMP $A7
C077  D0 CF    BNE $C048
C079  A5 AC    LDA $AC
C07B  C5 A6    CMP $A6
C07D  D0 C9    BNE $C048
C07F  A4 FD    LDY $FD   ; IF ar(t1)=ar(co) THEN GOTO xxxx
C081  C4 A9    CPY $A9
C083  D0 06    BNE $C08B
C085  A5 FC    LDA $FC
C087  C5 A8    CMP $A8
C089  F0 28    BEQ $C0B3
C08B  A5 FC    LDA $FC   ; swap ar(t1),ar(co)
C08D  20 A2 BB JSR $BBA2
C090  A2 50    LDX #$50
C092  A0 03    LDY #$03
C094  20 D4 BB JSR $BBD4
C097  A5 A8    LDA $A8
C099  A4 A9    LDY $A9
C09B  20 A2 BB JSR $BBA2
C09E  A6 FC    LDX $FC
C0A0  A4 FD    LDY $FD
C0A2  20 D4 BB JSR $BBD4
C0A5  A9 50    LDA #$50
C0A7  A0 03    LDY #$03
C0A9  20 A2 BB JSR $BBA2
C0AC  A6 A8    LDX $A8
C0AE  A4 A9    LDY $A9
C0B0  20 D4 BB JSR $BBD4
C0B3  18       CLC       ; .xxxx NEXT co
C0B4  A5 FC    LDA $FC
C0B6  69 05    ADC #$05
C0B8  85 FC    STA $FC
C0BA  90 02    BCC $C0BE
C0BC  E6 FD    INC $FD
C0BE  E6 A3    INC $A3
C0C0  D0 02    BNE $C0C4
C0C2  E6 A4    INC $A4
C0C4  A5 A4    LDA $A4
C0C6  C5 B1    CMP $B1
C0C8  D0 07    BNE $C0D1
C0CA  A5 A3    LDA $A3
C0CC  C5 B0    CMP $B0
C0CE  D0 01    BNE $C0D1
C0D0  60       RTS       ; END
C0D1  4C 27 C0 JMP $C027 ; loop
```

ne anche più snello.
La realizzazione in linguaggio macchina non comporta grandi difficoltà: è sufficiente ricopiare passo per passo le operazioni compiute dalla versione Basic. Le routine del sistema operativo, infatti, sono utilissime e compiono tutte le operazioni del Basic (essendo a esso dedicate) anche manipolando numeri e stringhe in linguaggio macchina.

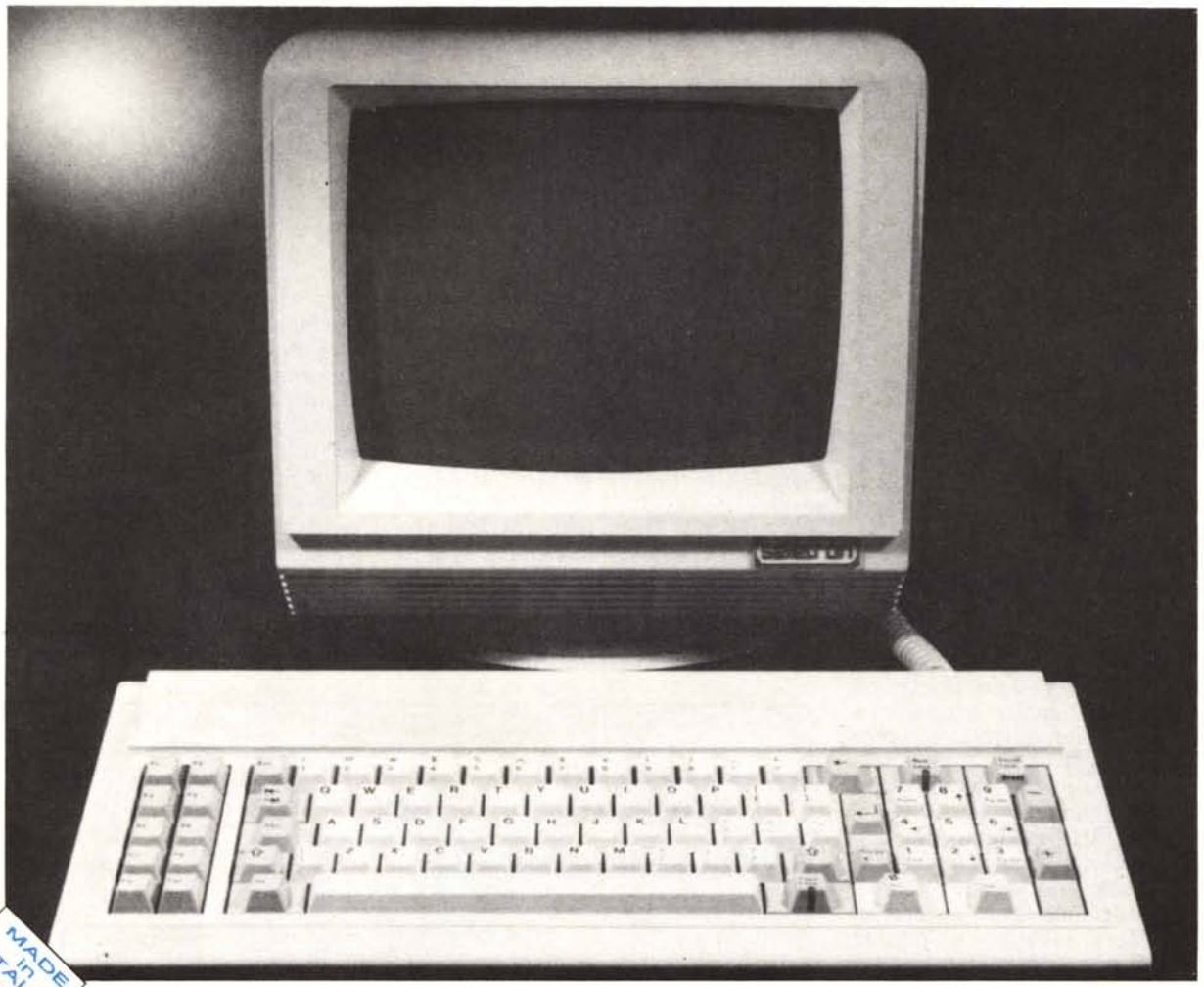
Utilizzo del programma

I due programmi in linguaggio macchina che svolgono il SORT per i numeri (non interi: ad esempio a%(300) NON può essere ordinato dal mio programma) vengono forniti con un caricatore Basic, che consente di salvarli su disco per poi essere richiamati da un programma che li utilizzi con LOAD «xxx», 8, 1. Possono essere salvati separatamente o insieme. Il caricatore non prevede forme di checksum perché le due routine sono molto corte (anche se hanno una parte iniziale ripetuta per ciascuna in modo da poter essere salvate separatamente). Viene fornito anche il disassemblato commentato per ogni routine, insieme alla versione Basic dell'algoritmo presentato. Infine, le due routine si attivano come normali programmi in linguaggio macchina: con il comando SYS<indirizzo> (fornito nel programma caricatore) più due parametri:
SYS<indirizzo>, array(1) [l'indice iniziale, 1, è fisso e non può venire alterato], numero degli elementi.
Il tempo impiegato è:
per 100 numeri: 1 secondo (12 sessantesimi);
per 100 stringhe: 75 secondi (45 sessantesimi). MC

```
SSORT DEMO
PROGRAMMA DIMOSTRATIVO PER L'ORDINAMENTO DI STRINGHE

10 INPUT"(CLR)(DOWN)●UANTI ELEMENTI:";A:DIMA$(A):PRINT"(CLR)─┘↑↓ "A"ELEMENT-A
RRAY..."
20 FORI=1TOA:N1=INT(RND(0)*10+1):FORN=1TON1
25 A$(I)=A$(I)+CHR$(INT(RND(0)*26+65)):NEXT:PRINTI"(UP)":NEXT
30 PRINT"(DOWN)↗N TASTO PER IL SORT":POKE198,0:WAIT198,1:POKE198,0
40 PRINT"●┌─┘↓↗ ...":TI$="000000":SYS49408,A$(1),A:T=TI:T$=TI$
50 PRINT"(DOWN)↗N TASTO PER VEDERE LE STRINGHE":POKE198,0:WAIT198,1:POKE198,0
60 PRINT"(CLR)":FORI=1TOA:PRINTA$(I):NEXT
70 PRINT"┘─┬┬ RICHIESTO:"RIGHT$(T$,2)" SECS"T"JIFFIES"

ATTENZIONE:          (DISH) = CTRL + H
```

```
MSORT Per stringhe
assembler listing

C100  20 FD AE JSR $AEFD ; Prendi Parametri: nome dell'array$
C103  20 8B B0 JSR $B08B
C106  85 FC    STA $FC
C108  84 FD    STY $FD
C10A  20 FD AE JSR $AEFD ; Prendi Parametri: elementi da "sort"are
C10D  20 8A AD JSR $AD8A
C110  20 F7 B7 JSR $B7F7
C113  84 B0    STY $B0
C115  85 B1    STA $B1
C117  C8       INY       ; FOR co = 1 TO n-1
C118  84 A6    STY $A6
C11A  85 A7    STA $A7
C11C  D0 02    BNE $C120
C11E  E6 A7    INC $A7
C120  A2 01    LDX #$01
C122  86 A3    STX $A3
C124  CA       DEX
C125  86 A4    STX $A4
C127  A5 FC    LDA $FC   ; t1 = co
C129  A4 FD    LDY $FD
C12B  85 AA    STA $AA
C12D  84 AB    STY $AB
C12F  A6 A3    LDX $A3   ; FOR co2 = co+1 TO n
C131  A4 A4    LDY $A4
C133  E8       INX
C134  86 AC    STX $AC
C136  D0 01    BNE $C139
C138  C8       INY
C139  84 AD    STY $AD
C13B  18       CLC
C13C  A5 FC    LDA $FC
C13E  69 03    ADC #$03
C140  85 FE    STA $FE
C142  A5 FD    LDA $FD
C144  69 00    ADC #$00
C146  85 FF    STA $FF
C148  A0 00    LDY #$00  ; IF ar$(co2)>ar$(t1) THEN GOTO xxx
C14A  B1 FE    LDA ($FE),Y
C14C  D1 AA    CMP ($AA),Y
C14E  90 02    BCC $C152
C150  B1 AA    LDA ($AA),Y
C152  85 02    STA $02
C154  C8       INY
C155  B1 FE    LDA ($FE),Y
C157  99 66 C1 STA $C166,Y
C15A  B1 AA    LDA ($AA),Y
C15C  99 69 C1 STA $C169,Y
C15F  C8       INY
C160  C0 03    CPY #$03
C162  D0 F1    BNE $C155
C164  A2 00    LDX #$00
C166  BD 00 00 LDA $0000,X
C169  DD 00 00 CMP $0000,X
C16C  F0 0D    BEQ $C17B
C16E  B0 18    BCS $C188

C170  A5 FE    LDA $FE   ; t1 = co2
C172  A4 FF    LDY $FF
C174  85 AA    STA $AA
C176  84 AB    STY $AB
C178  4C 88 C1 JMP $C188
C17B  E8       INX
C17C  E4 02    CPX $02
C17E  D0 E6    BNE $C166
C180  A0 00    LDY #$00
C182  B1 AA    LDA ($AA),Y
C184  C5 02    CMP $02
C186  D0 E8    BNE $C170
C188  18       CLC       ; .xxx NEXT co2
C189  A5 FE    LDA $FE
C18B  69 03    ADC #$03
C18D  85 FE    STA $FE
C18F  90 02    BCC $C193
C191  E6 FF    INC $FF
C193  E6 AC    INC $AC
C195  D0 02    BNE $C199
C197  E6 AD    INC $AD
C199  A5 AD    LDA $AD
C19B  C5 A7    CMP $A7
C19D  D0 A9    BNE $C148
C19F  A5 AC    LDA $AC
C1A1  C5 A6    CMP $A6
C1A3  D0 A3    BNE $C148
C1A5  A5 AB    LDA $AB   ; IF ar$(t1)=ar$(co) THEN GOTO xxxx
C1A7  C5 FD    CMP $FD
C1A9  D0 03    BNE $C1B3
C1AB  A5 AA    LDA $AA
C1AD  C5 FC    CMP $FC
C1AF  D0 02    BNE $C1B3
C1B1  F0 0F    BEQ $C1C2
C1B3  A0 02    LDY #$02  ; swap a$(t1),ar$(co)
C1B5  B1 FC    LDA ($FC),Y
C1B7  48       PHA
C1B8  B1 AA    LDA ($AA),Y
C1BA  91 FC    STA ($FC),Y
C1BC  68       PLA
C1BD  91 AA    STA ($AA),Y
C1BF  88       DEY
C1C0  10 F3    BPL $C1B5
C1C2  18       CLC       ; .xxxx NEXT co
C1C3  A5 FC    LDA $FC
C1C5  69 03    ADC #$03
C1C7  85 FC    STA $FC
C1C9  90 02    BCC $C1CD
C1CB  E6 FD    INC $FD
C1CD  E6 A3    INC $A3
C1CF  D0 02    BNE $C1D3
C1D1  E6 A4    INC $A4
C1D3  A5 A4    LDA $A4
C1D5  C5 B1    CMP $B1
C1D7  D0 07    BNE $C1E0
C1D9  A5 A3    LDA $A3
C1DB  C5 B0    CMP $B0
C1DD  D0 01    BNE $C1E0
C1DF  60       RTS       ; END
C1E0  4C 27 C1 JMP $C127 ; loop
```

seletron S10 PC



MADE IN ITALY

PRESENTI ALLO SMAU '86
PAD. 12 - E 10 bis

TASTIERA E SET GRAFICO IBM®

3 MODI OPERATIVI

- PC — per ambiente MultiLink®
- LSI — per ambienti general purpose, UNIX®
- ANSI — per ambienti CONCURRENT DOS® , XENIX®

MultiLink è un marchio registrato della The Software Link Inc., distribuito in Italia da Channel s.r.l. - Milano

SELETRON S.R.L. Via Pontina Km 32.500 - 00040 POMEZIA (Roma) tel. (06) 9123281-2-3 (R.A.) - Tlx 612629 SELETR I