exma

un assemblatore per VIC-20

di Andrea de Prisco

Da un po' di mesi il buon Valter Di Dio ci sta parlando di Assembler: una magica parola che identifica un linguaggio di programmazione dalla incredibile velocità, che vive in letargo nel personal in attesa di essere scoperto dall'utente. Purtroppo col VIC-20, anche dotato di VIC-MON, programmare in Assembler non è proprio facile-facile; si ha sempre a che fare con una marea di indirizzi Hex (guai a scrivere 3 al posto di $03 o $2D1 al posto di $02D1!!). Per cui si finisce spesso per fare confusione. Se, inoltre, dopo aver digitato un lungo programma si vuol inserire al centro una nuova linea ... Buonanotte: bisogna correggere tutti gli indirizzi dei Branch (salti) che facevano riferimento al di là della linea aggiunta e tutti gli eventuali riferimenti a tavole (= lavoraccio da cani).

Nel corso di quest'articolo non spiegheremo come si programma in Assembler; presenteremo invece un assemblatore per il 6502 scritto in Basic e linguaggio macchina, che ci permetterà di scrivere programmi senza l'assillo degli Hex, usando etichette, variabili, vettori, notazione decimale, binaria, ottale e, per i patiti, esadecimale.

Come per il Basal (MC n. 22) si potrà facilmente editare un programma, salvarlo su nastro, su disco, listarlo su stampante, e in più: curiosare nella memoria con un potente disassembler e con la funzione di Hex-Dump, e trasformare pezzi di memoria in linee Basic del tipo DATA 123, 12, 9, 44, 122, 0, 32 ... ecc.

Exma è organizzato in due puntate: in questa prima puntata vi consegneremo tutta la parte in linguaggio macchina (listato 1, quello con tutti quei DATA) e, del programmone Basic, tutta la parte riguardante l'editing, il Saveload, l'assemblatore vero e proprio e il monitor.

Il prossimo mese sarà la volta della creazione di Macroistruzioni che permetterà a tutti di costruirsi un proprio set di istruzioni speciali tipo la moltiplicazione, la divisione o (!) il M.C.D. di due numeri, consentendo così una ancora più spedita programmazione in L.M.

2 Listati

L'inconsueta veste di questo programma, in forma di due distinti listati, nasce dalla necessità (costi quel che costi) di risparmiare byte. Dato che le routine in linguaggio macchina, tramite i DATA, sono caricate una volta per tutte (con READ e POKE) prima di usare l'Assembler, è inutile continuare a tenersele in memoria sotto forma di linee Basic.

Si dovrà così, una volta che i due programmi saranno su nastro o su disco, caricare dapprima il programma DATA, dare RUN, aspettare il READY e poi caricare l'Assembler. Inutile dirvi che se qualcuno si azzarda a dare RUN al secondo listato senza aver eseguito il primo, non solo non funzionerà ma c'è il rischio che il tutto si inceppi sino a spegnimento e riaccensione del malcapitato VIC.

Sembra quasi superfluo avvertire che è necessaria la cartuccia da 16K prima di iniziare a digitare!!

RUN

Appare il menu #1, e già, non è l'unico: premendo il tasto "←" (freccia a sinistra) i due menu si scambiano di posto consentendo di accedere a tutte le opzioni.

Ritornate al menu #1: la prima opzione è "fase di input" e serve appunto per inserire un programma in memoria. Con Shift "1" si va in fase di input dopo aver ripulito l'area di lavoro, senza lo shift si ha lo stesso effetto senza cancellare: in questo modo è possibile lasciare di battere un programma, ritornare al menu, eseguire qualche altra opzione e ritornare al programma lasciato in sospeso per continuare a inseri-

```
10 POKE55,0:POKE56,76
20 FORI=19456TO20307
30 READII:POKEI,II
40 NEXT:END
1000 DATA169,80,133,168,169,0,133,167,168,162,16,169,32,145,167,200,208,251
1010 DATA202,240,4,230,168,208,244,96,234,169,0,133,251,168,169,16,133,253,169
1020 DATA24,133,252,133,254,169,148,133,255,177,167,41,63,145,252,169,6,145
1030 DATA254,200,152,197,251,208,240,165,251,208,12,169,200,133,251,230,168
1040 DATA230,253,230,255,208,224,96,234,234,234,234,234,234,234,169,0,133,0
1050 DATA169,18,133,1,230,0,208,2,230,1,160,0,177,0,201,255,240,82,205,64,3
1060 DATA208,237,200,177,0,205,65,3,208,229,200,177,0,205,66,3,208,221,206,67
1070 DATA3,173,67,3,10,24,101,0,133,0,169,0,101,1,133,1,200,169,64,133,2,177
1080 DATA0,36,2,240,3,24,105,9,41,15,10,10,10,10,141,68,3,200,177,0,36,2,240
1090 DATA3,24,105,9,41,15,24,109,68,3,141,68,3,96,169,3,141,68,3,96,160,0,165
1100 DATA162,41,16,208,7,177,167,9,128,145,167,96,177,167,41,63,145,167,96,169
1110 DATA3,162,8,168,32,186,255,173,175,2,162,176,160,2,32,189,255,32,192,255
1120 DATA162,3,32,201,255,169,80,133,252,160,0,132,251,132,253,177,251,201,32
1130 DATA208,4,230,253,208,4,162,0,134,253,166,253,224,24,240,10,32,210,255
1140 DATA200,208,230,230,252,208,226,32,204,255,169,3,32,195,255,96,169,3,162
1150 DATA8,160,3,32,186,255,173,175,2,162,176,160,2,32,189,255,32,192,255,162
1160 DATA3,32,198,255,169,80,133,252,160,0,132,251,32,183,255,201,64,240,12
1170 DATA32,207,255,145,251,200,208,241,230,252,208,237,32,204,255,169,3,32
1180 DATA195,255,96,169,192,133,252,169,95,133,253,160,0,177,252,160,24,145
1190 DATA252,56,165,252,233,1,133,252,165,253,233,0,133,253,197,168,208,231
1200 DATA165,252,197,167,208,225,169,32,145,167,136,208,251,96,169,192,133,252
1210 DATA169,95,133,253,160,24,177,167,160,0,145,167,230,167,208,2,230,168,165
1220 DATA167,197,252,208,236,165,168,197,253,208,230,96,160,0,169,18,133,1,169
1230 DATA0,133,0,200,208,2,230,1,177,0,201,255,240,70,197,167,208,237,230,0
1240 DATA177,0,197,168,208,229,162,2,152,72,165,1,72,136,192,255,208,2,198,1
1250 DATA232,177,0,201,42,208,242,138,41,1,208,8,104,133,1,104,168,24,144,195
1260 DATA200,138,56,233,5,74,201,1,48,238,133,254,162,3,177,0,149,250,200,202
1270 DATA208,248,240,4,169,15,133,254,104,104,96,24,165,47,105,10,133,0,165
1280 DATA48,105,2,133,1,162,0,169,7,133,167,169,80,133,168,160,0,177,167,201
1290 DATA32,240,39,169,3,145,0,200,185,166,0,145,0,192,2,208,246,232,24,165
1300 DATA0,105,3,133,0,144,2,230,1,24,165,167,105,24,133,167,144,213,230,168
1310 DATA208,209,142,161,2,169,0,133,167,169,80,133,168,133,170,169,10,133,169
1320 DATA165,47,24,105,9,133,0,165,48,105,0,133,1,165,47,105,11,133,171,165
1330 DATA48,105,4,133,172,160,0,169,6,145,0,169,14,145,171,200,185,166,0,145
1340 DATA0,185,168,0,145,171,192,2,208,241,160,0,24,185,0,0,105,3,153,0,0,185
1350 DATA1,0,105,0,153,1,0,192,171,240,4,160,171,208,231,160,0,24,185,167,0
1360 DATA105,24,153,167,0,185,168,0,105,0,153,168,0,192,2,240,4,160,2,208,231
1370 DATA202,208,174,162,0,134,2,24,165,47,105,9,133,0,165,48,105,0,133,1,160
1380 DATA2,177,0,153,166,0,136,208,248,177,0,240,68,177,167,201,32,208,17,56
1390 DATA177,0,233,1,145,0,240,53,230,167,208,2,230,168,208,233,160,2,185,166
1400 DATA0,145,0,136,208,248,177,0,56,233,1,24,200,113,0,133,167,200,177,0,105
1410 DATA0,133,168,160,0,177,167,201,32,208,9,56,177,0,233,1,145,0,208,221,24
1420 DATA165,0,105,3,133,0,144,2,230,1,232,208,160,230,2,165,2,201,2,208,152,96
```

Listato 1

```
0 REM*ADC75FF6D65697D796171*BCC90*BCSB0*BEQF0*BMI30*BNED0*BPL10*BRKFF00*BVC50*BV
S70
10 REM*CLCFF18*CLDFFD8*CLIFF58*CLVFFB8*DEXFFCA*DEYFF88*INXFFE8*INYFFC8*NOPFFEA*P
HAFF48
20 REM*PHPFF08*PLAFF68*PLPFF28*RTIFF40*RTSFF60*SECFF38*SEDFFF8*SEIFF78*TAXFFAA*T
AYFFA8
30 REM*TSXFFBA*TXAFF8A*TXSFF9A*TYAFF98*AND35FF2D25293D392131*ASL160A0E06FF1E*BITFFFF2C24
40 REM*CMPD5FFCDC5C9DDD9C1D1*CPXFFFFECE4E0*CPYFFFFCCC4C0*DECD6FFCEC6FFDE
50 REM*EOR55FF4D45495D594151*INCF6FFEEE6FFFE*JMP6CFF4C*JSRFFFF20*LDAB5FFADA5A9BD
B9A1B1
60 REM*LDXB6FFAEA6A2FFBE*LDYB4FFACA4A0BC*LSR564A4E46FF5E*ORA15FF0D05091D190111
70 REM*ROL3E2A2E26FF3E*ROR766A6E66FF7E*SBCF5FFEDE5E9FDF9E1F1*STA95FF8D85FF9D9981
91
90 REM*STX96FF8E86*STY94FF8C84*TSXFFBA*
200 POKE36879,29:REM*
210 POKE36864,10:IFPEEK(21000)=245THENSYS19456
220 POKE55,0:POKE56,78
230 POKE36866,24
240 POKE36867,170
250 POKE36865,42:PRINT"♥"
260 GOSUB810
270 GETZ$:IFZ$=""THEN270
280 IFZ$="←"THEN1070
290 IFZ$="1"THENGOSUB750:GOTO370
300 IFZ$="L"THEN1230
310 IFZ$="2"THENPRINT"♥":CLR:GOTO1280
320 IFZ$="↑"THENZ$="1":SYS19456:GOTO290
330 IFZ$="3"THEN2030
340 IFZ$="4"THEN700
350 IFZ$="5"THEN1030
360 GOTO270
370 CU=7:LI=0:M1=20480:FL=2:GOTO660
380 SYS19652:GETZ$:IFZ$=""THEN380
390 Z=ASC(Z$):IFZ>32ANDZ<95THEN640
400 IFZ=17THENLI=LI+1:GOTO630
410 IFZ=145THENLI=LI-1:GOTO630
420 IFZ=29THEN650
430 IFZ=157THENCU=CU-2:GOTO650
440 IFZ=205THEN260
450 IFZ=95THENCU=0:GOTO660
460 IFZ=19THENL=M1+24*LI:POKE168,L/256:POKE167,(L/256-PEEK(168))*256:SYS19659:G
OTO660
470 IFZ=20THENL=M1+24*LI-1:POKE168,L/256:POKE167,(L/256-PEEK(168))*256:SYS19812
:GOTO660
480 IFZ=13THENLI=LI+1:CU=7:GOTO660
490 IFZ=32ANDCU<=6THENCU=6:Z=58:GOTO640
500 IFZ=32ANDCU<11ANDCU>6 THENCU=11:GOTO640
510 IFZ=19THENCU=7:LI=0:GOTO660
520 IFZ=147THEN380
530 IFZ<>20THEN560
540 IFCU=0THEN660
550 FORI=M1+LI*24+CU-1TOM1+(LI+1)*24-2:POKEI,PEEK(I+1):NEXT:CU=CU-1:POKEI,32:GOT
O660
560 IFZ<>148ORPEEK(M1+LI*24+23)<>32THEN620
570 FORI=M1+(LI+1)*24-2TOM1+LI*24+CU-1STEP-1:POKEI+1,PEEK(I):NEXT:POKEM1+LI*24+C
U,32
580 GOTO660
590 M1=20480
600 IFI>18THENI=I-18:M1=M1+24*18:GOTO600
610 CU=7:LI=I:FL=2:WAIT197,191:GETA$:GOSUB750:GOTO660
620 IFZ=148ANDPEEK(M1+LI*24+23)<>32THEN380
640 POKEM1+LI*24+CU,ZAND127
650 CU=CU+1+(CU>22)-(CU<-1)
660 IFLI<0THENLI=0:M1=M1+24*(M1>20480):GOTO680
670 IFLI>18THENLI=18:M1=M1-24*(M1<24074)
680 POKE168,M1/256:POKE167,(M1/256-PEEK(168))*256+.5:SYS19483
690 I=8120+LI*24+CU:POKE168,I/256:POKE167,(I/256-PEEK(168))*256:GOTO380
700 PRINT"♥▶▶■SAVE■ ":GOSUB780:IFX$=""THEN260
710 GOSUB800:C=0:OPEN1,1,1,X$:I=20480
720 A$=CHR$(PEEK(I)):PRINT#1,A$;:C=C-(A$=" "):C=-C*(A$=" "):I=I+1
730 IFC<24THEN720
740 CLOSE1:GOTO260
750 PRINT"♥■LABEL  OPR  ADDRESS     ";
760 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■■■ (C) 1983  ADP SOFTWARE ■▶▶■";
770 RETURN
780 X$="":INPUT"■▶▶▶▶▶▶▶▶NOME FILE";X$:RETURN
790 PRINT"■■■▶▶▶▶▶▶▶PRESS PLAY ON TAPE■":RETURN
800 PRINT"■■■▶▶▶▶▶▶▶PRESS REC & PLAY ON TAPE■":RETURN
810 PRINT"♥▶▶■■     ASSEMBLER 6502      ■";
820 PRINT"■▶▶▶▶■ M E N U    # 1          ";
830 PRINT"  ______________              ";
840 PRINT" ■1■ FASE DI INPUT            ";
850 PRINT"■▶▶■ ■2■ ASSEMBLY             ";
860 PRINT"■▶▶■ ■3■ DISASSEMBLY          ";
870 PRINT"■▶▶■ ■4■ SAVE PR. SORGENTE    ";
880 PRINT"■▶▶■ ■5■ LOAD PR. SORGENTE    ";
890 PRINT"■■■▶▶▶▶▶▶▶▶▶■ (C) 1983  ADP-SOFTWARE ■"
900 RETURN
910 PRINT"♥■ADDR  HEX-DUMP    CHR$":GOTO760
920 PRINT"♥■ADDR M E X  DISASSEMBLY":GOTO760
930 PRINT"♥▶▶■■     ASSEMBLER 6502      ■";
940 PRINT"■▶▶▶▶■ M E N U    # 2          ";
950 PRINT"  ______________              ";
960 PRINT" ■1■ HEX  DUMP                ";
970 PRINT"■▶▶■ ■2■ DATA CONV            ";
980 PRINT"■▶▶■ ■3■ RUN PROGRAMMA        ";
990 PRINT"■▶▶■ ■4■ SAVE COD. OGGETTO    ";
1000 PRINT"■▶▶■ ■5■ LOAD COD. OGGETTO   ";
1010 PRINT"■■■▶▶▶▶▶▶▶▶▶▶■ (C) 1983  ADP-SOFTWARE ■"
1020 RETURN
1030 PRINT"♥▶▶■LOAD■ ":GOSUB780:IFX$=""THEN260
1040 GOSUB790:SYS19456:OPEN1,1,0,X$:I=20480
1050 GET#1,A$:POKEI,ASC(A$):I=I+1:IFST<>64THEN1050
1060 CLOSE1:GOTO260
1070 GOSUB930
1080 GETZ$:IFZ$=""THEN1080
1090 IFZ$="←"THEN260
1100 IFZ$="1"THEN1940
1110 IFZ$="2"THEN2290
1120 IFZ$="3"ANDHJ<>0THENSYSEP:GOTO1070
1130 IFZ$="4"THEN1160
1140 IFZ$="5"THENCLR:GOTO1190
1150 GOTO1080
1160 PRINT"♥▶▶■SAVE■ ":GOSUB780:IFX$=""THEN1070
1170 GOSUB800:OPEN2,1,1,X$:PRINT#2,HJ:PRINT#2,T:PRINT#2,EP:FORI=0TOT+1:PRINT#2,A
%(I)
1180 FORJ=1TO2:PRINT#2,"*"+A$(I,J):NEXT:NEXT:CLOSE2:GOTO1070
1190 PRINT"♥▶▶■LOAD■ ":GOSUB780:IFX$=""THEN1070
1200 GOSUB790:OPEN2,1,0,X$:DIMA$(170,2)
1210 INPUT#2,HJ,T,EP:DIMA%(T+3):FORI=0TOT+1:INPUT#2,A%(I)
1220 FORJ=1TO2:INPUT#2,A$:A$(I,J)=MID$(A$,2):NEXT:NEXT:CLOSE2:GOTO1730
1230 J=0:C=0:OPEN1,4:I=20480
1240 A$=CHR$(PEEK(I)):PRINT#1,A$;:C=C-(A$=" "):C=-C*(A$=" "):I=I+1
1250 J=J+1:IFJ=24THENPRINT#1:J=0
1260 IFC<24THEN1240
1270 PRINT#1:CLOSE1:GOTO260
1280 DIMA$(170,2):PRINT"♥▶▶▶ATTENDERE PREGO ...":SYS19992:T=PEEK(673)-1
1290 GOSUB3500:DIMA%(T+3),I%(T+3):PRINT"▶▶▶▶▶▶PRIMA   PASSATA":HJ=0
1300 FORI=0TOT:AD$=A$(I,2):GOSUB1860:A$(I,2)=AD$:NEXT
1310 A%(0)=20480
1320 FORI=0TOT
1330 OP$=A$(I,1):LA$=A$(I,0):AD$=A$(I,2):LE=LEN(AD$):A$=LEFT$(OP$,1):IFA$<>"."TH
EN1390
1340 IFOP$=".*="THENA%(I+1)=VAL(A$(I,2)):GOTO1550
1350 IFOP$=".BP"THENA%(I+1)=A%(I)+VAL(A$(I,2)):GOTO1550
1360 IFOP$=".AD"THENA%(I+1)=A%(I):GOTO1550
1370 IFOP$=".EP"THENEP=A%(I):A%(I+1)=EP:HJ=1:GOTO1550
1390 IFLE=0THENI%(I)=2:A%(I+1)=A%(I)+1:GOTO1550
1400 IFLEFT$(AD$,1)="#"THENI%(I)=5:A%(I+1)=A%(I)+2:A$(I,2)=MID$(AD$,2):GOTO1550
1410 FORH=2TOLEN(AD$):IFMID$(AD$,H,1)=","THENHH=H:H=75
1420 NEXT
1430 IFH=76THEN1480
1440 IFLEFT$(AD$,1)="("THENI%(I)=1:A%(I+1)=A%(I)+3:A$(I,2)=MID$(AD$,2,LE-2):GOTO
1550
1450 IFLEFT$(OP$,1)="B"ANDOP$<>"BIT"THENI%(I)=10:A%(I+1)=A%(I)+2:GOTO1550
1460 IF(VAL(AD$)=0ANDLEFT$(AD$,1)<>"0")ORVAL(AD$)>255THENI%(I)=3:A%(I+1)=A%(I)+3
:GOTO1550
1470 I%(I)=4:A%(I+1)=A%(I)+2:GOTO1550
1480 IFRIGHT$(AD$,3)="),Y"THENI%(I)=8:A%(I+1)=A%(I)+2:A$(I,2)=MID$(AD$,2,LE-4):G
OTO1550
1490 IFRIGHT$(AD$,3)="),Y"THENI%(I)=9:A%(I+1)=A%(I)+2:A$(I,2)=MID$(AD$,2,LE-4):G
OTO1550
1500 IFRIGHT$(AD$,2)=",X"THENI%(I)=6:GOTO1520
1510 IFRIGHT$(AD$,2)=",Y"THENI%(I)=7
1520 A=VAL(LEFT$(AD$,HH-1)):A$(I,2)=LEFT$(AD$,LE-2)
1530 IF(A=0ANDLEFT$(AD$,1)<>"0")ORA>255THENA%(I+1)=A%(I)+3:GOTO1550
1540 A%(I+1)=A%(I)+2:I%(I)=1
1550 NEXT:PRINT"▶▶▶▶▶▶▶▶▶▶▶SECONDA PASSATA"
1560 FORI=0TOT:LA$=A$(I,0):IFLA$=""THEN1620
1570 IFA$(I,1)=".AD"THENOX$=A$(I,2):GOTO1590
1580 OX$=STR$(A%(I))
1590 LE=LEN(LA$):FORII=0TOT:AD$=A$(II,2):IFLEFT$(AD$,LE)<>LA$THEN1610
1600 A$(II,2)=OX$+MID$(AD$,LE+1)
1610 NEXT:A$(I,0)=""
1620 NEXT:FORI=0TOT
1630 AD$=A$(I,2):FORH=1TOLEN(AD$):H$=MID$(AD$,H,1)
1640 IFH$="+"THENA$(I,2)=STR$(VAL(LEFT$(AD$,H-1))+VAL(MID$(AD$,H+1))):GOTO1660
1650 IFH$="-"THENA$(I,2)=STR$(VAL(LEFT$(AD$,H-1))-VAL(MID$(AD$,H+1)))
1660 NEXT:OP$=A$(I,1):A$=LEFT$(OP$,1):IFA$="."THEN1720
1665 IFOP$=N$THEN1720
1670 IFA$="$"THENAD$=OP$:GOSUB1860:H=VAL(AD$):GOTO1710
1680 IFI%(I)<>10THEN1690
1683 IFLEFT$(AD$,1)=">"THENA$(I,2)=STR$(A%(I+VAL(MID$(AD$,2)))):GOTO1687
1685 IFLEFT$(AD$,1)="<"THENA$(I,2)=STR$(A%(I-VAL(MID$(AD$,2))))
1687 I%(I)=1:IFFF<>1THENA$(I,2)=STR$((VAL(A$(I,2))-A%(I)-2+256)AND255)
1690 FORH=1TO3:POKE831+H,ASC(MID$(OP$,H,1)):NEXT
1700 POKE825,I%(I):SYS19542:H=PEEK(826):IFH=3THENPRINT"♥NON RICONOSCO.. "+OP$:GO
TO590
1710 A$(I,1)=STR$(H)
1720 NEXT
1730 FORI=0TOT:IFLEFT$(A$(I,1),1)="."THEN1760
1740 A=VAL(A$(I,1)):C=VAL(A$(I,2)):B=INT(C/256):C=INT((C/256-B)*256)
1750 POKEA%(I),A:POKEA%(I)+2,-B*(B>0):POKEA%(I)+1,-C*(C>0)
1760 NEXT
1770 GOTO1070
1780 H$=MID$(AD$,J+1):FORHG=1TOLEN(H$):SA$=MID$(H$,HG,1)
1790 IFSA$="+"ORSA$=","ORSA$="-"ORSA$=")"THENG$=LEFT$(H$,HG-1):GOSUB1830:HL=HG:H
G=30
1800 NEXT
1810 IFHG<30THENG$=H$:GOSUB1830:H$=MID$(STR$(HT),2):AD$=LEFT$(AD$,J-1)+H$:RETURN
1820 H$=MID$(STR$(HT),2)+MID$(H$,HL):AD$=LEFT$(AD$,J-1)+H$:RETURN
1830 HT=0:HS=0:FORH=LEN(G$)TO1STEP-1:HH$=MID$(G$,H,1)
1840 HH=ASC(HH$)-48+7*(ASC(HH$)>64):HT=HT+BA↑HS*HH:HS=HS+1
1850 NEXT:RETURN
1860 FORJ=1TOLEN(AD$):H$=MID$(AD$,J,1)
1870 IFH$="$"THENBA=16:GOSUB1780:J=24:GOTO1900
1880 IFH$="&"THENBA=8:GOSUB1780:J=24:GOTO1900
1890 IFH$="%"THENBA=2:GOSUB1780:J=24
1900 NEXT:RETURN
1910 X$=""
1920 X=X/16:Y=(X-INT(X))*16:X=INT(X):X$=CHR$(Y+48-7*(Y>9))+X$:IFX<>0THEN1920
1930 X$=LEFT$("0000",4-LEN(X$))+X$:RETURN
1940 GOSUB2010:IFAD$=""THEN1070
1950 GOSUB910:FORII=1TO19:X=A:GOSUB1910:PRINT"$";X$;" ";:J$="":FORIJ=1TO4:X=PEEK
(A)
1960 B=((X>31ANDX<91)OR(X>160ANDX<224))
1970 J$=J$+CHR$(-X*B-46*(NOTB))+LEFT$(CHR$(34)+CHR$(20),-2*(X=34))
1980 GOSUB1910:PRINT" "+RIGHT$(X$,2);:A=A+1:NEXT:PRINT"  ■■";J$;"■■":NEXT
1990 WAIT197,191:GETZ$:IFZ$<>"\"THEN1950
2000 GOTO1070
2010 AD$="":INPUT"♥▶▶▶DAL BYTE ";AD$
2020 :GOSUB1860:A=VAL(AD$):RETURN
2030 GOSUB2010:IFAD$=""THEN260
2040 GOSUB920:FORII=1TO19:X=A:GOSUB1910:PRINT"$";X$;" ";
2050 X=PEEK(A):GOSUB1910:FORI=1TO2:POKE166+I,ASC(MID$(X$,2+I,1)):NEXT:M$=""
2060 SYS19894:H=PEEK(254):FORI=253TO251STEP-1:M$=M$+CHR$(PEEK(I)):NEXT
2070 M$=M$+" ":IFH=15ORPEEK(A)=255THENM$="??? ":H=3
2080 ON H GOSUB 2110,2170,2180,2190,2200,2210,2220,2230,2240:NEXT
2090 WAIT197,191:GETZ$:IFZ$="\"THEN260
2100 GOTO2040
2110 T$=LEFT$(M$,1):IFT$="J"THENGOSUB2270:PRINT"(";X$;")":RETURN
2120 IFT$<>"B"THEN2150
2130 X=PEEK(A)*256+PEEK(A+1):GOSUB1910:PRINTX$;"   ";M$
2140 X=A+2+PEEK(A+1)-256*(PEEK(A+1)>127):GOSUB1910:PRINT"$"+X$;SPC(2);:A=A+2:RET
URN
2150 IFRIGHT$(M$,1)="X"THENGOSUB2250:PRINTX$;",Y  ";:RETURN
2160 GOSUB2250:PRINTX$;",X  ";:RETURN
2170 PRINTRIGHT$(X$,2)+"     ";M$;SPC(7);:A=A+1:RETURN
2180 GOSUB2270:PRINTX$;SPC(2);:RETURN
2190 GOSUB2250:PRINTX$;SPC(4);:RETURN
2200 GOSUB2250:PRINT"#";X$;SPC(3);:RETURN
2210 GOSUB2270:PRINTX$;",X";:RETURN
2220 GOSUB2270:PRINTX$+",Y";:RETURN
2230 GOSUB2250:PRINT"(";X$;",X)";:RETURN
2240 GOSUB2250:PRINT"(";X$;"),Y";:RETURN
2250 X=PEEK(A)*256+PEEK(A+1):GOSUB1910:PRINTX$;"   ";:X$="$"+RIGHT$(X$,2):PRINTM
$;
2260 A=A+2:RETURN
2270 PRINTRIGHT$(X$,2);:X=PEEK(A+1)*256+PEEK(A+2):GOSUB1910:PRINTX$;" ";
2280 X$="$"+RIGHT$(X$,2)+LEFT$(X$,2):A=A+3:PRINTM$;:RETURN
2290 A(1)=0:INPUT"♥▶▶▶INIZIO  ";A(1):A(2)=0:IFA(1)=0THEN1070
2300 INPUT"▶▶▶▶FINE     ";A(2):IFA(2)=0THENINPUT"♥▶▶▶▶N. BYTE ";A(2):A(2)=A(2)+A(
1)-1
2310 INPUT"▶▶▶▶▶▶N.LINEA ";A(3):A(4)=0
2320 :INPUT"▶▶▶▶▶▶▶▶STEP    ";A(4):IFA(4)=0THENA(4)=10
2330 F$=MID$(STR$(A(3)),2)+" DATA"
2340 F$=F$+MID$(STR$(PEEK(A(1))),2):A(1)=A(1)+1:IFLEN(F$)>83THENGOTO2370
2350 IFA(1)<=A(2)THENF$=F$+",":GOTO2340
2360 PRINT"♥■■■"+F$:PRINT"GOTO1070■":POKE198,2:POKE631,13:POKE632,13:END
2370 G$=STR$(A(1))+"■"+STR$(A(2))+"■"+STR$(A(3))+"■"+STR$(A(4))+"■"
2380 FORI=1TOLEN(G$):POKE828+I,ASC(MID$(G$,I,1)):NEXT
2390 PRINT"♥■■■"+F$:PRINT"■GOTO2400■":POKE198,2:POKE631,13:POKE632,13:END
2400 I=828:G$="":FORJ=1TO4
2410 I=I+1:E=PEEK(I):IFE<>255THEN G$=G$+CHR$(E):GOTO2410
2420 A(J)=VAL(G$):G$="":NEXT
2430 A(3)=A(3)+A(4):GOTO2330
2440 GOTO1070
3500 RETURN
```

Listato 2

re le linee. Simpatico il cursore lampeggiante che non è quello originale, ma un "sosia" ottenuto controllando continuamente l'orologio interno (loc. 160-162) e "negativizzando" ad intervalli di tempo regolari. Se per giocherellare coi tasti si è sporcato lo schermo, ritornate al menu con shift "M" e con shift "1" ripulite l'area di lavoro. Inizieremo da questa: è semplicemente la zona di memoria da 20480 a 24566 corrispondente quindi agli ultimi 4k Ram disponibili dei 16. È come una grande paginona video che continuamente è visualizzata (un pezzo alla volta) sullo schermo da una routine in linguaggio macchina. Ma ciò non ha molta importanza: l'unica cosa da dire è che il [RETURN] non serve per inserire qualcosa in memoria, ma solo per andare a capo. In altre parole, ogni cambiamento sullo schermo corrisponde allo stesso cambiamento in memoria, cosa che non accade in Basic se, dopo aver listato un programma, vi scarabocchiamo sopra col cursore e con la tastiera, senza toccare il [RETURN].

I comandi di movimento cursore, cancellamento e inserimento carattere funzionano nel modo più istintivo possibile: come se stessimo in ambiente Basic. Shift "D" e shift "I" si usano rispettivamente per cancellare una linea (quella dove sta lampeggiando il cursore) o per inserirne una in bianco da riempire. Avrete notato che esiste una precisa suddivisione del campo: Etichette (LABEL), operazione (OPR) e indirizzo (ADDRESS). Senza dilungarci ulteriormente in parole, "schiaffiamo" in memoria questo miniprogramma:

```
        .*=  23000
PIO    :JSR  $FFE4
        CMP  #32
        BNE  PIO
        BRK
```

Da menu digitate shift "1": il cursore è pronto per accettare ".*=", per ora digitatelo: dopo vedremo cos'è. Con la pressione della barra spaziatrice il cursore schizza nel terzo campo (abituatevi a questi "schizzamenti"). Digitate 23000 [RETURN]. Il cursore si trova sulla seconda linea, sempre nel campo OPR. Bisogna inserire una label (PIO): premete la freccia a sinistra (schizzo contrario) digitate PIO e premete una volta lo spazio: la label è inserita, il cursore è in OPR, potete procedere come sopra. Così via fino all'ultima linea ricordandovi soltanto che ogni cosa va scritta nel suo campo, quindi allenatevi un po' con la barra e la freccia a sinistra prima di cimentarvi a scrivere programmi lunghi.

Non preoccupatevi se andando per sbaglio nel campo label vedrete apparire qualche ":" in più: funziona lo stesso. Un'ultima cosa: se ancora non vi siete sintonizzati perfettamente sul modo di fare del programma di editing ricordate che lo spazio alcune volte fa schizzare il cursore; shift spazio è molto più pacifico: cancella il carattere sotto il cursore sostituendolo con un blank.

Le direttive

Le direttive per l'assemblatore sono istruzioni inseribili nel programma da assemblare pur non essendo istruzioni eseguibili dal 6502. Tanto per intenderci, una direttiva potrebbe essere: "il seguente programma andrà posizionato a partire dal byte $4000" oppure "attento: dove incontri la label PRT è da intendere come l'indirizzo $FFD2", o roba simile. Nell'esempio del paragrafo precedente abbiamo già incontrato una direttiva: ".*=" serve per definire dove andrà locato il programma dopo l'assemblaggio. In quel caso, 23000 nel campo ADDRESS indicava l'indirizzo della prima istruzione del programma. Come già accennato, ogni numero all'interno del programma può essere espresso in differenti basi. Non fanno eccezione eventuali argomenti delle direttive: in quel caso, al posto di 23000 (decimale) si sarebbe potuto scrivere $59D8 (esadecimale) oppure &54730 per esprimerlo in ottale. Per indirizzi a 8 bit o poco più è possibile esprimersi anche in binario usando come prefisso il carattere "%". Ritornando a ".*=", non resta da dire che questa direttiva può anche essere inserita in più punti del programma. Per specificare varie porzioni di programma o vari programmi ognuno con un suo indirizzo d'inizio. Il suo valore di default coincide con l'inizio dell'area di lavoro: $5000 o 24800 in decimale.

La seconda direttiva ".AD" definisce un riferimento in memoria: per esempio, se spesso in un programma si fa riferimento a $FFD2 possiamo dichiarare all'inizio del programma:

```
PRT   :.AD $FFD2
```

e ogni volta digitare PRT al posto di $FFD2 che è più corto, più mnemonico e meno esposto a errori di battitura.

Terza direttiva è ".EP" dove EP non sta per "extra pedestre" ma per Entry Point e indica qual è la prima istruzione da eseguire (non necessariamente la prima del programma) nel caso che da menu #2 sia dato il RUN PROGRAMMA dopo l'assemblaggio. Se nessun Entry Point è stato dichiarato, non sarà possibile far partire l'esecuzione da menu, ma bisognerà schiacciare R/S e digitare SYS seguito dall'indirizzo d'inizio del programma.

Quarta ed ultima direttiva è ".BB" (block byte) e si usa per dichiarare un vettore di byte di lunghezza data. Ad esempio:

```
PIPPO  :.BB 100
```

indica un vettore di lunghezza 100 con gli elementi individuabili con PIPPO, PIPPO+1, PIPPO+2, ..., PIPPO+99 o variabilmente col modo indicato dal registro X o Y, ad esempio LDA PIPPO,X con X compreso fra 0 e 99.

Per ultimo (non si tratta di una vera e propria direttiva), la possibilità di definire nel programma sorgente il contenuto di determinati byte. Si ottiene specificando nel campo operatore il valore esadecimale del byte, preceduto naturalmente da "$". Quando l'assemblatore incontra quest'oggetto, non ricerca il suo codice operativo come vedremo più avanti che fa per le altre istruzioni, ma semplicemente assegna al byte il valore esadecimale specificato e va all'istruzione successiva. In questo modo è possibile inserire una tabella nel programma sul tipo del DATA in Basic. Esempio:

```
        LDX  #2
LOOP   :LDA  TABLE,X
        JSR  $FFD2
        DEX
        BPL  LOOP
        RTS
TABLE  :$43
        $4D
        $93
```

Il processo d'assemblaggio

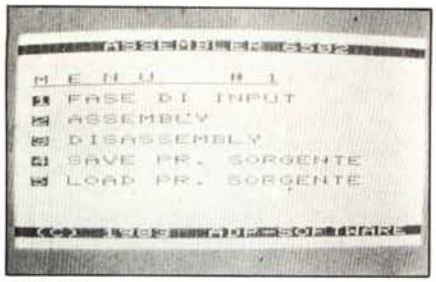
L'Assembler listato in queste pagine (a proposito: EXMA sta per Extended Mnemonic Assembler) ha alcune particolari features. Come dicevamo prima, è possibile per qualsiasi numero esprimersi in notazione decimale, esa, ottale e binaria; inserire etichette nel programma e fare ad esse riferimento per salti, subroutine e tabelle. Di conseguenza, la traduzione a codice numerico (l'unico effettivamente "digeribile" dal 6502) necessita di alcune passate. Dopo ogni passata, il nostro programma sorgente assomiglia sempre più al codice oggetto (fino, naturalmente, a diventarlo).

L'EXMA compie in tutto 4 passate anche se, dal punto di vista logico, possiamo assimilarle a due sole. Con la prima, linea per linea è riconosciuto il modo di indirizzamento e il numero di byte occupati da ogni istruzione, e sostituito ad essa il relativo codice operativo. Con la seconda passata, tutti i riferimenti mnemonici, le label, sono sostituiti con gli effettivi indirizzi dei byte cui si riferiscono.

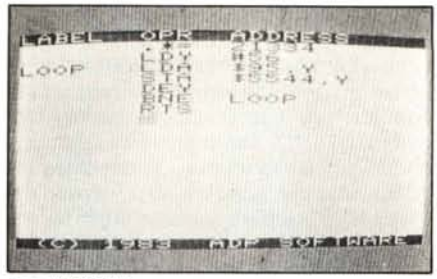
La necessità di eseguire almeno due pas-

Linee da modificare per salvare su disco i programmi EXMA.

```
710 X$=X$+",S,W":FORI=1TOLEN(X$)
720 POKE687+I,ASC(MID$(X$,I,1))
730 NEXT:POKE687,I-1
740 SYS19674:GOTO260
1040 X$=X$+",S,R":FORI=1TOLEN(X$)
1050 POKE687+I,ASC(MID$(X$,I,1))
1060 NEXT:POKE687,I-1:SYS19456:SYS19749:GOTO260
1170 OPEN2,8,2,X$+",S,W":PRINT#2,HJ:PRINT#2,T:P
RINT#2,EP:FORI=0TOT+1:PRINT#2,A%(I)
1200 OPEN2,8,2,X$+",S,R":DIMA$(170,2)
```



Menu # 1.



Fase di INPUT.

Hex-Dump delle locazioni $C000-C04B.

sate, sebbene con metodi molto contorti e perditempo sia possibile ridurla ad una sola, nasce dal fatto che un eventuale riferimento label ad un'istruzione qualche linea più avanti (es. un salto) non potrebbe essere direttamente sostituito col valore numerico, dato che di quella linea non si conosce ancora il suo effettivo indirizzo se non si assemblano tutte le linee precedenti. Sempre in merito alle label, per "questioni interne" non è possibile identificarne una con parte del nome di un'altra. Se, ad esempio, è stata usata la label PIOLO, non potremo usare PIO né P come etichetta.

Per riconoscere il modo d'indirizzamento, il programma si avvale di un metodo assai semplice: analizza alcuni particolari del campo address della linea da assemblare. Se ad esempio il suddetto campo termina con "),Y" siamo certamente nel caso del modo indiretto indicizzato; se termina con ")" il modo è indiretto e così via per gli altri modi. Chi è interessato ai dettagli dia uno sguardo alle linee 1320 - 1550.

Per accelerare un po' i tempi, la ricerca del codice operativo è affidata a una routine in L.M. Le REM a capo del programma costituiscono la tavola di tutte le istruzioni e dei relativi codici operativi espressi in Hex. La stessa, è sfruttata da un'altra routine in L.M. per disassemblare il contenuto della memoria.

Per trovare, ad esempio, l'opcode del modo 5 dell'istruzione LDA, i codici ASCII dei tre caratteri L, D e A sono immessi nei byte 832-834, il modo (5) nel byte 835 ed è eseguita una SYS 19542. A questo indirizzo, una routine in linguaggio macchina ricerca la stringa LDA fra le REM, pesca il quinto numero Hex dopo LDA, e lo "sbatte" nel byte 836. Ritornati in Basic, se PEEK(836) contiene il numero 3, vuol dire che l'istruzione non esiste e un messaggio d'errore è mostrato sul video. Ciò può capitare se si è scritto LDB al posto di LDA o giù di lì. In tal caso, dopo l'apparizione di "NON RICONOSCO ... LDB", basta premere qualsiasi tasto per ritrovarsi immediatamente "catapultati" in fase di input col cursore già bello e lampeggiante (!) sull'istruzione da correggere che ha generato l'errore (della serie: quando AdP fa spettacolo, n.d.a.d.p.).

Per ultimo, un miniminiquiz: chi starà un po' attento quando batterà il programma, noterà a un certo punto qualcosa di strano, molto strano, di cui volutamente non abbiamo parlato: potrebbe essere quasi la passata zero; grazie ad una particolare

Disassembly delle locazioni $DE45-$DE67.

SYS... pensateci, l'assassino, ops!, la soluzione nel prossimo numero.

Qualche esempio

Con i seguenti tre programmini, cercheremo di mettere in luce, più che con le solite parole, le effettive possibilità del linguaggio EXMA 6502. Il programma:

```
            .*=  22222
            LDA  #16
            STA  $A8
            LDY  #0
            STY  $A7
TITO        LDA  #32
PIPPO       STA  ($A7),Y
            INY
            BNE  PIPPO
            INC  $A8
            LDA  $A8
            CMP  #18
            BNE  TITO
            RTS
```

è un classico esempio di uso del modo indiretto indicizzato STA $A7),Y per il riempimento della memoria. Nella fattispecie, la routinetta sopra listata riempie con 512 "trentadue" la pagina video che, come si sa, ha inizio a $4000. Il codice di schermo 32 corrisponde allo spazio quindi equivale a cancellare il video. Per provarla, dopo averla naturalmente digitata e fatta assemblare, arrestate l'esecuzione e digitate SYS 22222 [RETURN]. Il video si cancellerà.

Il secondo programmino:

```
            .*=  21200
            .EP
PRT         .AD  $FFD2
GET         .AD  $FFE4
LOOP        JSR  GET
            BEQ  LOOP
            CMP  #205
            BEQ  FINE
            JSR  PRT
            BNE  LOOP
FINE        RTS
```

mostra l'uso delle direttive ".EP" e ".AD". $FFD2 e $FFE4 sono gli indirizzi di due routine del Sistema Operativo del VIC che implementano le istruzioni di PRINT e GET registro accumulatore. Assemblando e facendo partire l'esecuzione di questa routine (questa volta grazie a ".EP" da menu #2 con RUN PROGRAMMA), vedrete apparire sul video i tasti che schiaccerete. Shift "M" vi farà ritornare al menu.

L'ultimo programmino:

```
            .*=  1024
PIPPO       .BB  256
            .EP
            LDX  #0
LOOP        LDA  0,X
            STA  PIPPO,X
            DEX
            BNE  LOOP
            RTS
```

non fa altro che trasferire i 256 byte della pagina zero nel vettore PIPPO (linea 2) dichiarato appunto come Block Byte di 256 elementi. Si noti l'Entry Point a capo dell'effettivo programma e non, naturalmente, prima del vettore di 256 elementi.

Il monitor

Per curiosare all'interno della memoria del VIC, è possibile da menu richiedere l'Hex-Dump o il Disassembly di determinati byte. Sia l'uno che l'altro sono visualizzati una intera paginata per volta: con la pressione di qualsiasi tasto, una nuova paginata è mostrata; Shift "M", tanto per cambiare, riporta al menu.

L'opzione DATA.CONV, come già accennato in testa all'articolo, si usa per trasformare in DATA insiemi di byte della memoria. È richiesto il byte d'inizio, l'ultimo byte o il numero di byte (battendo solo [RETURN] alla seconda richiesta), il numero della prima linea DATA e lo step. Si raccomanda di non indicare numeri di linea già esistenti, pena la cancellazione di parte del programma in memoria. Questa opzione può essere utile a coloro i quali usano il L.M. per preparare veloci routine da accoppiare a programmi Basic. In questo caso, dopo aver trasformato in DATA, non resta che Deletare tutte le linee dell'EXMA con l'apposito comando presente sulla Toolkit, o sfruttando la routine dell'abile Tontini apparsa sul n. 17 di MC. Infine, l'esistenza di due comandi di SAVE e due di LOAD risponde alla eventuale necessità di salvare su nastro o su disco un programma in L.M. prima o dopo la fase di assemblaggio. Per il SAVE COD. OGGETTO, si raccomanda di usarlo appena terminata la seconda passata, prima di richiedere qualsiasi altra opzione. Prima del consueto "Tutto qui", un cordiale arrivederci al prossimo numero. MC