```
JH 1692 DATA 173,124,6,141,173,6
JP 1698 DATA 173,125,6,141,174,6
MK 1704 DATA 169,0,160,39,153,255
OO 1710 DATA 255,136,16,250,160,40
JO 1716 DATA 136,177,5,208,5,192
GA 1722 DATA 0,208,247,96,201,5
PM 1728 DATA 176,242,169,0,141,152
JD 1734 DATA 52,169,1,141,153,52
NP 1740 DATA 96,-1
PJ 13470 DATA 169,80,141,3,210,169
MG 13476 DATA 1,141,152,52,165,45
FE 13482 DATA 141,143,52,169,100,141
PP 13488 DATA 144,52,169,0,141,146
CG 13494 DATA 52,141,151,52,141,145
PI 13500 DATA 52,141,153,52,165,88
CE 13506 DATA 24,105,24,133,208,165
PI 13512 DATA 89,105,1,133,209,165
MC 13518 DATA 88,24,105,121,133,3
JF 13524 DATA 165,89,105,2,133,4
FG 13530 DATA 172,154,52,162,255,202
CI 13536 DATA 224,0,208,251,136,192
MF 13542 DATA 0,208,246,173,5,208
IG 13548 DATA 201,0,240,6,32,106
MM 13554 DATA 54,32,54,54,174,143
PC 13560 DATA 52,236,144,52,240,14
BM 13566 DATA 144,2,202,202,232,142
MF 13572 DATA 143,52,142,0,208,76
GM 13578 DATA 199,53,160,120,136,177
MF 13584 DATA 208,201,0,208,7,192
JK 13590 DATA 0,208,245,76,35,53
AG 13596 DATA 201,5,176,238,76,134
CH 13602 DATA 53,172,145,52,204,147
PK 13608 DATA 52,16,222,238,145,52
CF 13614 DATA 169,172,141,1,210,165
MF 13620 DATA 67,141,3,210,169,55
BM 13626 DATA 141,2,210,169,120,141
LN 13632 DATA 0,210,173,143,52,56
BC 13638 DATA 233,46,74,74,168,169
MB 13644 DATA 1,145,208,169,2,200
CJ 13650 DATA 145,208,152,24,105,39
DD 13656 DATA 168,169,3,145,208,200
NA 13662 DATA 169,4,145,208,173,5
IM 13668 DATA 208,201,0,240,6,32
MO 13674 DATA 106,54,32,54,54,165
LM 13680 DATA 20,105,2,24,141,149
AE 13686 DATA 52,166,20,236,149,52
ML 13692 DATA 208,249,169,0,141,1
OM 13698 DATA 210,141,3,210,173,10
PM 13704 DATA 210,109,148,52,74,74
EI 13710 DATA 170,173,10,210,205,143
JG 13716 DATA 52,144,13,208,3,76
CK 13722 DATA 134,53,138,24,109,144
AG 13728 DATA 52,76,175,53,138,141
AD 13734 DATA 149,52,173,144,52,56
CL 13740 DATA 237,149,52,201,54,144
JB 13746 DATA 211,201,198,176,207,14
     1
PF 13752 DATA 144,52,173,5,208,201
FP 13758 DATA 0,240,6,32,106,54
JM 13764 DATA 32,54,54,173,5,208
ID 13770 DATA 201,0,240,6,32,106
MN 13776 DATA 54,32,54,54,173,112
MF 13782 DATA 2,73,255,201,42,144
CD 13788 DATA 10,201,196,176,14,141
NI 13794 DATA 1,208,76,244,53,169
MF 13800 DATA 47,141,1,208,76,244
PN 13806 DATA 53,169,192,141,1,208
PC 13812 DATA 141,150,52,173,5,208
IG 13818 DATA 201,0,240,6,32,106
ML 13824 DATA 54,32,54,54,173,152
LK 13830 DATA 52,201,0,240,38,172
CP 13836 DATA 145,52,204,147,52,208
LK 13842 DATA 32,173,151,52,201,1
CN 13848 DATA 240,11,238,151,52,165
BP 13854 DATA 20,24,105,140,141,146
PG 13860 DATA 52,165,20,205,146,52
DB 13866 DATA 208,155,169,0,141,152
NF 13872 DATA 52,104,96,76,218,52
GI 13878 DATA 165,1,166,0,32,62
DA 13884 DATA 54,96,134,212,133,213
CF 13890 DATA 32,170,217,32,230,216
MI 13896 DATA 160,0,132,2,177,243
JH 13902 DATA 72,41,127,32,93,54
JA 13908 DATA 104,48,5,164,2,200
DJ 13914 DATA 208,238,96,170,173,71
CP 13920 DATA 3,72,173,70,3,72
AI 13926 DATA 138,160,146,96,169,0
CD 13932 DATA 141,30,208,141,155,52
KI 13938 DATA 169,6,133,85,169,0
MK 13944 DATA 133,84,165,0,24,109
JA 13950 DATA 148,52,133,0,165,1
LN 13956 DATA 105,0,133,1,173,150
JE 13962 DATA 52,162,0,56,233,42
AJ 13968 DATA 74,74,24,105,160,168
AI 13974 DATA 169,0,141,149,52,177
IE 13980 DATA 3,201,1,240,45,201
IM 13986 DATA 2,240,61,201,3,240
NF 13992 DATA 77,201,4,240,93,169
MC 13998 DATA 0,145,3,200,232,224
OP 14004 DATA 13,144,228,162,1,238
CE 14010 DATA 149,52,173,149,52,201
LJ 14016 DATA 5,240,10,152,56,233
MN 14022 DATA 52,168,169,0,76,155
GK 14028 DATA 54,96,169,0,145,3
OF 14034 DATA 200,145,3,152,24,105
JA 14040 DATA 39,152,169,0,145,3
JD 14046 DATA 200,145,3,96,169,0
LO 14052 DATA 145,3,136,145,3,152
MH 14058 DATA 24,105,40,168,169,0
JA 14064 DATA 145,3,200,145,3,96
LL 14070 DATA 169,0,145,3,200,145
MJ 14076 DATA 3,152,56,233,40,168
MG 14082 DATA 169,0,145,3,136,145
DK 14088 DATA 3,96,169,0,145,3
PK 14094 DATA 136,145,3,152,56,233
IM 14100 DATA 40,168,169,0,145,3
PC 14106 DATA 200,145,3,96,-1
```

Program 4: Atari Lightsaver

Version by Chris Poer, Editorial Programmer

Refer to the "Automatic Proofreader" article before typing this program in.

```
DL 2 POKE 13464,0:POKE 106,64:GRAPHI
    CS 0:OPEN #1,4,0,"K:":HIGH=0:Q=
    USR(1536):DIM A$(3),B$(1)
HK 5 GOSUB 700:GRAPHICS 0:GOSUB 800:
    B$=" "
KG 10 PUT #6,125:POKE 82,0
KB 13 POKE 752,1:SETCOLOR 2,0,0:GOSU
     B 600:POKE 87,0
PB 15 BULB=13459:MEN=3:LEV=13460:SC=
     0
IF 20 POKE BULB,20:POKE LEV,1:EX=200
     0:XX=0:AM=20
DG 100 IF SC>HIGH THEN HIGH=SC
KL 110 GOSUB 900
GM 130 Q=USR(13470)
KL 135 SC=PEEK(0)+PEEK(1)*256
```

```
DL 137 IF SC>EX THEN EX=EX+2000:MEN=
       MEN+1:FOR I=1 TO 100:SOUND Ø,
       INT(RND(1)*255),1Ø,14:NEXT I:
       SOUND Ø,Ø,Ø,Ø
EP 14Ø IF PEEK(13465)=1 THEN GOSUB 5
       ØØ
JH 145 IF XX=1 THEN XX=Ø:GOTO 1ØØ
KC 15Ø SOUND Ø,14Ø,1Ø,12:FOR I=1 TO
       12Ø:NEXT I:SOUND Ø,9Ø,1Ø,14
AJ 16Ø FOR I=1 TO 8Ø:NEXT I:SOUND Ø,
       Ø,Ø,Ø
JE 18Ø POKE LEV,PEEK(LEV)+1
PJ 19Ø AM=AM+4:POKE BULB,AM+4
KI 2ØØ POKE 13468,INT(PEEK(LEV)/5)+1
       :IF INT(PEEK(LEV)/5)+1=7 THEN
        POKE 13468,6
FN 21Ø GOTO 1ØØ
DA 5ØØ SOUND Ø,2ØØ,12,14:FOR I=1 TO
       8Ø:NEXT I:SOUND Ø,Ø,Ø,Ø:XX=1
CB 5Ø5 IF PEEK(LEV)>1 THEN POKE LEV,
       PEEK(LEV)-1
ND 51Ø MEN=MEN-1:IF MEN=Ø THEN 85Ø
CH 515 POKE BULB,AM:IF INT(PEEK(LEV)
       /5)+1<7 THEN POKE 13468,INT(P
       EEK(LEV)/5)+1
HK 55Ø RETURN
DJ 6ØØ A=56:POKE 54279,A:PMBASE=256*
       A:POKE 756,56
BJ 615 POKE Ø,Ø:POKE 1,Ø:POKE 13468,
       1
GO 627 POKE 53249,9Ø:POKE 53248,9Ø
HM 63Ø FOR I=PMBASE+512 TO PMBASE+76
       8:POKE I,Ø:NEXT I
PE 64Ø POKE 7Ø4,216:POKE 7Ø5,118
CD 65Ø RESTORE 67Ø:FOR I=PMBASE+55Ø+
       Y TO PMBASE+562+Y:READ A:POKE
        I,A:NEXT I
DB 66Ø FOR I=PMBASE+739+Y TO PMBASE+
       75Ø+Y:READ A:POKE I,A:NEXT I
FO 67Ø DATA 24,24,24,24,24,24,24,24,
       24,24,6Ø,126,255
EB 68Ø DATA 255,255,255,255,255,255,
       126,126,126,6Ø,6Ø,6Ø
HH 69Ø POKE 53256,1:POKE 53257,1:POK
       E 623,1:RETURN
HN 7ØØ GRAPHICS 18:POSITION 4,3:? #6
       ;"LiGHTsaVEr"
PI 71Ø FOR I=1 TO 12Ø:X=INT(RND(1)*2
       55):SOUND Ø,X,1Ø,12:NEXT I
NC 72Ø SOUND Ø,8Ø,1Ø,14:FOR I=1 TO 1
       ØØ:NEXT I
LK 73Ø SOUND Ø,Ø,Ø,Ø:GRAPHICS 18:POK
       E 53248,22Ø:POKE 53249,22Ø
JD 74Ø POSITION 1,4:? #6;"enter leve
       l of play"
PK 75Ø POSITION 3,6:? #6;"(1/9) 1=HA
       RDEST"
PO 76Ø GET #1,DIF:IF DIF>57 OR DIF<4
       9 THEN 75Ø
PB 77Ø DIF=(DIF-48):POKE 13466,DIF:R
       ETURN
NJ 8ØØ DL=PEEK(56Ø)+4+PEEK(561)*256
PA 8Ø1 FOR I=2 TO 6:POKE DL+I,6:NEXT
        I:POKE DL-1,6+64
FJ 81Ø FOR I=7 TO 24:POKE DL+I,36:NE
       XT I:POKE 87,1:RETURN
JE 85Ø IF PEEK(Ø)+PEEK(1)*256>HIGH T
       HEN HIGH=PEEK(Ø)+PEEK(1)*256
MH 855 POKE 53248,22Ø:POKE 53249,22Ø
MC 86Ø POKE 53277,Ø:POSITION 1,2:? #
       6;"(E) TO END PROGRAM  (P) TO
        PLAY AGAIN"
DH 87Ø GET #1,W:IF W=69 THEN Q=USR(5
       8484)
DG 88Ø IF W<>8Ø THEN 87Ø
JJ 89Ø GOSUB 7ØØ:GRAPHICS Ø:GOSUB 8Ø
       Ø:GOTO 1Ø
DI 9ØØ POSITION Ø,Ø:? #6;"SCORE ";SC
ED 91Ø A$=STR$(PEEK(LEV)):IF PEEK(LE
       V)<1Ø THEN A$(LEN(A$)+1)=B$
CN 92Ø POSITION Ø,1:? #6;"HI SCORE "
       ;HIGH:POSITION 12,Ø:? #6;"LEV
       EL ";A$
HP 93Ø POSITION 15,1:? #6;"MEN ";MEN
       :POKE 53248,22Ø:POKE 53249,22
       Ø
IO 94Ø POKE 53277,Ø:FOR I=1 TO 2ØØ:P
       OKE 13464,1:NEXT I:POKE 13464
       ,Ø
EN 95Ø POSITION Ø,2:? #6;" hit paddl
       e button{3 SPACES}to begin ro
       und"
HB 96Ø IF PTRIG(Ø)=1 THEN 96Ø
PE 97Ø POSITION Ø,2:? #6;"
       {35 SPACES}"
CG 98Ø POKE 559,46:POKE 53277,3:POKE
        77,Ø:RETURN
```

Program 5: IBM PC/PCjr Lightsaver

Version by Tim Victor, Editorial Programmer

```
5 CLEAR ,&HDØØØ
1Ø ON ERROR GOTO 2ØØØØ:GOSUB 8ØØØ
65 NP=1ØØ:DF=15:LEVEL=1:MISSES=Ø
68 SC=Ø:C=Ø
7Ø CLS:GOSUB 4ØØØ
8Ø BP=JSF*(STICK(Ø)-3)
1ØØ CALL BLANK:PUT (BP,183),C%
11Ø LP=8Ø:PUT (LP,Ø),L%
114 FOR X%=Ø TO 6:XP(X%)=Ø:NEXT
115 GOSUB 5ØØØ
12Ø X%=5:CF=1:Z=STRIG(Ø)
13Ø BNUM=INT(1Ø*RND(1))+1Ø:BN=1
135 GOSUB 2ØØØ:IF CF=Ø THEN 31Ø
137 GOSUB 3ØØØ
138 IF BN<BNUM THEN GOSUB 1ØØØ ELSE XP(X
%)=Ø:X%=FNDEC(X%)
139 IF BN<BNUM+5 THEN BN=BN+1:X%=FNDEC(X
%):GOTO 135
14Ø DF=DF*1.1:GOTO 114
31Ø PUT (BB,153),B%:PUT (BB,185),B%
32Ø FOR I=1 TO 2Ø:SOUND 2ØØØ,.2:SOUND 32
767,.2:NEXT
325 DF=DF/1.1:MISSES=MISSES+1
33Ø IF MISSES=4 THEN GOSUB 6ØØØ:GOTO 65
34Ø GOSUB 7ØØØ:GOTO 7Ø
999 'move lamp and make new bulb
1ØØØ NP=NP+4*INT(DF*(RND(1)-.479))
1Ø1Ø IF NP>2ØØ THEN NP=2ØØ
1Ø2Ø IF NP<Ø THEN NP=Ø
1Ø3Ø CALL BLANK:PUT (LP,Ø),L%:PUT (NP,Ø)
,L%:LP=NP
1Ø4Ø XP(X%)=NP+12:PUT (XP(X%),28),B%
1Ø45 SOUND 37,.1
1Ø5Ø X%=FNDEC(X%)
1Ø6Ø RETURN
1999 'is bulb about to break?
2ØØØ BB=XP(X%)
2Ø1Ø IF BB<>Ø AND (BB<BP-3 OR BB>BP+26)
THEN CF=Ø:RETURN
```

```
2020 IF BB THEN PUT (BB,178),B%:C=C+1:SO
UND 2000,.1
2025 IF STRIG(Ø) THEN GOSUB 7000:WHILE S
TRIG(1):WEND:Z=STRIG(Ø)
2030 RETURN
2999 'drop all bulbs
3000 FOR I%=153 TO 23 STEP -25
3010 XP=XP(X%)
3020 IF XP THEN PUT (XP,I%),B%:PUT (XP,I
%+25),B%
3030 NBP=JSF*(STICK(Ø)-3)
3040 CALL BLANK:PUT (BP,183),C%:PUT (NBP
,183),C%:BP=NBP
3050 X%=FNDEC(X%):NEXT
3060 RETURN
3999 'draw scoreboard
4000 LINE (Ø,Ø)-(240,199),3,B
4005 LINE (240,Ø)-(319,199),1,B
4006 LINE (242,2)-(317,45),1,B
4007 LINE (242,47)-(317,86),1,B
4008 LINE (242,88)-(317,127),1,B
4009 LINE (242,129)-(317,168),1,B
4010 LOCATE 3,33:PRINT "LEVEL:"
4020 LOCATE 8,33:PRINT "SCORE:"
4030 LOCATE 13,33:PRINT "HIGH:"
4040 LOCATE 18,33:PRINT "BROKEN:"
4300 RETURN
4999 'update score
5000 SC=SC+C*LEVEL:C=Ø
5005 IF CF=Ø THEN LEVEL=LEVEL-1 ELSE LEV
EL=LEVEL+1
5006 IF LEVEL=Ø THEN LEVEL=1
5010 LOCATE 5,34:PRINT LEVEL
5020 LOCATE 10,34:PRINT FNFMT$(STR$(SC))
5025 LOCATE 15,34:PRINT FNFMT$(STR$(HI))
5030 LOCATE 20,34:PRINT MISSES
5200 RETURN
5999 'end of game
6000 LOCATE 20,12:PRINT "PRESS TRIGGER F
OR ANOTHER GAME"
6005 GOSUB 5000
6010 WHILE STRIG(1)=Ø:WEND
6020 IF SC>HI THEN HI=SC
6030 RETURN
6999 'wait for button press
7000 LOCATE 23,33:PRINT "PRESS";
7010 LOCATE 24,33:PRINT "BUTTON";
7020 WHILE STRIG(1)=Ø:IF INKEY$="e" OR I
NKEY$="E" THEN END
7030 WEND
7040 LINE (256,176)-(318,191),Ø,BF
7050 RETURN
7999 'initialize graphics
8000 SCREEN 1:COLOR Ø,1:KEY OFF:CLS
8005 STRIG ON:RANDOMIZE TIMER
8010 DIM B%(25),C%(47),L%(119)
8020 DEF FNDEC(X%)=X%-1-7*(X%=Ø)
8030 DEF FNFMT$(A$)=LEFT$("ØØØØ",5-LEN(A
$))+RIGHT$(A$,LEN(A$)-1)
8040 BLANK=&HDØØØ
8050 FOR I=BLANK TO BLANK+9:READ A
8060 POKE I,A:NEXT
8100 DRAW "bm117,10c2ta45d20taØ134"
8105 DRAW "ta-45u20bm117,15p2,2"
8110 LINE (116,Ø)-(118,11),3,BF
8115 LINE (100,24)-(134,25),3,BF
8120 GET (100,Ø)-(134,25),L%
8125 LOCATE 11,6:PRINT "THIS IS NO ORDIN
ARY LAMP."
8130 PRINT "ANGERED BY ITS BORING AND ME
NIAL JOB,"
8135 FOR I=1 TO 600:NEXT
8140 LINE (115,31)-(119,40),3,BF
8145 LINE (112,36)-(122,38),3,BF
8150 LINE (115,28)-(119,30),1,BF
8155 LINE (114,34)-(120,34),3
8160 LINE (113,35)-(121,35),3
8165 LINE (113,39)-(121,39),3
8170 PRESET (115,28):PRESET (119,28)
8175 LINE (117,38)-(119,38),1
8180 LINE (119,37)-(120,37),1
8185 PSET (120,36),1
8187 GET (112,28)-(122,40),B%
8190 LOCATE 14,4:PRINT "IT IS DROPPING F
RAGILE, HELPLESS"
8195 PRINT "LIGHTBULBS TO THEIR CERTAIN
DESTRUCTION."
8200 DRAW "C3BM103,183TA30D10TAØL12"
8205 DRAW "TA-30U10BM103,187P3,3"
8210 GET (97,183)-(109,192),C%
8215 PUT (97,183),C%,PRESET
8220 GET (97,183)-(109,192),C%
8225 FOR J=30 TO 63 STEP 4
8230 LINE (J,182)-(J+10,192)
8235 LINE (J,182)-(J-10,192)
8240 NEXT
8245 LINE (30,182)-(63,182),Ø
8250 LINE (30,183)-(63,183),3
8255 LINE (30,192)-(63,192),3
8260 DRAW "BM30,183TA30D10"
8265 DRAW "BM63,183TA-30D10"
8270 PUT (23,183),C%,AND
8275 PUT (58,183),C%,AND
8280 GET (30,183)-(63,192),C%
8285 LINE (Ø,182)-(120,192),Ø,BF
8290 LOCATE 17,1:PRINT "USING YOUR BASKE
T, YOU MUST SAVE THE"
8295 PRINT "BULBS FROM THIS PSYCHOPATHIC
 APPLIANCE."
8300 BP=2*(STICK(Ø)-3)
8305 IF BP>210 THEN BP=210
8310 CALL BLANK:PUT (BP,183),C%
8400 LOCATE 20,1:PRINT "TO BEGIN, MOVE T
HE BASKET ALL THE WAY"
8410 LOCATE 21,3:PRINT "TO THE RIGHT AND
 PRESS THE BUTTON."
8420 WHILE STRIG(1)=Ø
8425 NBP=2*(STICK(Ø)-3)
8430 IF NBP>210 THEN NBP=210
8432 CALL BLANK:PUT (BP,183),C%:PUT (NBP
,183),C%:BP=NBP:WEND
8435 JSF=210/(STICK(Ø)-3)
8440 RETURN
10000 DATA 186,218,3,237,37
10010 DATA 8,Ø,116,250,203
20000 IF (ERR=5 OR ERR=6) AND (ERL=3040
OR ERL=100) THEN BP=210:PUT (BP,183),C%
ELSE ON ERROR GOTO Ø
20010 RESUME NEXT
```

©

Build A Computer In Your Mind

Fred D'Ignazio, Associate Editor

In my recent column, "The Morning After," in the May and June 1984 issues of COMPUTE!, I wrote about a new kind of programming that I believe people are beginning to do on their computer. I called this "neoprogramming" to distinguish it from traditional programming in BASIC or Pascal and from "no programming" in which people treat the computer as a thinking machine and let it do their thinking for them.

In this month's column I'd like to explore neoprogramming and see how it can be related to computer activities that will help people develop thinking, learning, and communication skills that they can practice and refine using the computer, and that they can also take away from the computer and use, on their own, in all areas of their lives.

Neoprogramming

Neoprogramming can be defined as borrowing the most powerful ideas from programming languages and turning them into thinking skills that people can use, inside their head, in their daily life.

Another way to look at neoprogramming is as a toolbox that has three kinds of tools inside:

☐ Tools to Help You Think
☐ Tools to Help You Learn
☐ Tools to Help You Communicate

These are practical tools that will be valuable no matter what people's goals are. Mastering these tools is more worthwhile than simply learning how to operate a computer.

Thinking, learning, and communication tools can be found in many places—in textbooks, in courses, in jobs, etc. But they can also be found, in a concentrated form, in the computer. And through extensive use and familiarity with these tools on a computer, people can learn how to use the tools to think better without the computer.

How Not To Use A Computer

Learning how to operate a computer, on its own, will not automatically guarantee people a successful career, help them learn how to use more advanced computers of the future, or give them thinking skills they can apply to other areas of their lives.

Also, it is possible to have a relationship with computers that actually deadens or stifles the ability to think. Many people, for example, use computers mechanically and passively. They spend their time in front of a computer entering information, making trivial, routine queries, or typing other people's documents.

The Thinking Appliance

There is a strong assumption in many people's minds that computers are labor-saving appliances. People ask, "What can I do on a computer?" But what they mean is, "What can the computer do for me?" The labor that many people hope computers will save is not mechanical labor but thinking labor. For most of us, thinking is work—work that we would avoid if we had the chance.

Many people would be happy (though few would admit it) if computers would do their thinking for them. In the near future, with the

Fred D'Ignazio is a computer enthusiast and author of several books on computers for young people. His books include Katie and the Computer *(Creative Computing),* Chip Mitchell: The Case of the Stolen Computer Brains *(Dutton/Lodestar),* The Star Wars Question and Answer Book About Computers *(Random House), and* How To Get Intimate With Your Computer (A 10-Step Plan To Conquer Computer Anxiety) *(McGraw-Hill).*

As the father of two young children, Fred has become concerned with introducing the computer to children as a wonderful tool rather than as a forbidding electronic device. His column appears monthly in COMPUTE!.

advent of expert systems and friendlier computers, there is a great risk that computers will take over more and more of the thinking that people do. As a result, people and organizations will become increasingly dependent on computers.

Dumbo's Feather

For adults at work and at home, and for children in school, there is the risk that computers will become super calculators. When they want to do real work or thinking, they will, by habit, turn to the computer. The computer will become an adjunct to the person's mind. The computer will be like Dumbo's feather. Dumbo the elephant could fly because of his big ears, but he thought it was because of his magic feather. If he didn't hold on tight to his feather, he was afraid he couldn't fly. People may come to feel incapable of thought unless they do it using their computer.

The Computer Crutch

There is a real risk that many people will use computers as a crutch. They will expect computers to do their thinking for them, or they will be afraid that they cannot think without the aid of the computer. Either way, they will be tied to computers to help them carry on their daily affairs.

Also, if people use computers (or anticipate using computers) as a crutch, they will not get the most out of them. They will be using computers' powerful computational, communications, and information handling functions sloppily, indiscriminately, and inefficiently.

The Computer Lever

In fact, the computer is not a thinking machine, a magic feather, or a crutch. It is a complex lever. It amplifies our abilities to move information around, but we must position and guide it to get what we want.

In addition, we don't need to tie ourselves to the computer to use its lever. We can build the lever inside our head. The lever is, in fact, just an assortment of thinking skills embedded in general-purpose (BASIC, Logo, Pascal, Assembler, etc.) *procedural* languages and special-purpose (word processing, spreadsheet, file handling) *builder kit* languages. Once we have acquired these skills, we can employ them on the computer, or we can use them inside our heads. If we recognize and master these skills, we can get more out of using the computer, and we can become less dependent on it and more skilled, on our own, to think, learn, and communicate.

Building A Computer Inside Your Head

Burrell Smith, Apple's hardware wizard who helped create the Macintosh, has written that he never just goes into a workshop and builds a new computer. Instead he first spends considerable time building mental prototypes inside his head. Burrell's prototypes are like a writer's rough drafts. Using mental prototypes, he takes a rough, simple idea and turns it into a cluster of complex ideas, and eventually into an advanced concept or design. Then he begins building the computer.

Burrell can create mental prototypes because he has a computer inside his head. Burrell has built this computer from an array of thinking skills he has learned from programming real computers and from his other experiences in life. These skills aren't mysterious, nor are they Burrell's alone. They can be mastered by anyone.

Environments For Thinking

Programming languages offer an environment for thinking—a place in which these skills can be learned, practiced, mastered, and then used. Learning a programming language offers an opportunity to explore new avenues of thought.

For example, if taught properly, BASIC, Pascal, Logo, and other languages can help people learn algorithmic thinking, how to break complex problems into smaller, simpler problems, and how to organize large quantities of information.

A word processing program can give people a feeling for the fluidity and mobility of words, ideas, thoughts, and knowledge. It can help them learn how to create several rough drafts, in quick succession, that sharpen an image, refine a concept, or lead to new ideas.

A spreadsheet program can help break a complex situation down into lists and arrays of smaller parts. It can display the whole forest and the individual trees in the forest, all at the same time. It can also reveal the relationships between all the parts.

A file-handling (data base) program can teach how to organize thoughts, feelings, experiences, and information. It can show how to group facts according to categories of likeness, how to sort and prioritize, and how to cross-reference facts that have certain traits in common.

Graphing languages, word processing languages, and telecommunications languages, singly or together, can teach how to better communicate feelings, ideas, and desires. They can teach how to use visual images and symbols, page layout and design, and grammar and style to communicate more effectively.

Magnets For Thinking, Learning, And Communication

Computers, like other media, can have a push-pull effect, depending on how people use them.

If computers are used inefficiently or inappropriately, they have to be pushed just to get meager, mediocre results.

On the other hand, computers can also exert a powerful pulling effect. They can be so attractive, so elegant that they will pull at the mind, like a magnet. They can almost seduce a person into performing a task or solving a problem.

Magnets And Road Maps

Computer tools can pull you like a magnet to the computer, but they can also become magnets inside your head that draw related information and ideas toward them. They can help you make sense out of chaos. They can let you mentally map out individual facts in some kind of logical, coherent, and practical order.

For example, what happens if you think about two things: a paper route and a spreadsheet? What kind of associations can you make? How might you map the paper route onto a spreadsheet?

You don't need to use a computer to do this exercise. Instead, you can perform what Albert Einstein called a thought experiment. You can build a mental prototype of a paper-route spreadsheet inside your head.

Associating spreadsheets and paper routes is not a dull, artificial, or mechanical activity. If you have the proper image, appreciation, and passion for using spreadsheets as a thinking skill, you start mapping the paper route onto the spreadsheet even before you know it. The spreadsheet, as a thinking tool, or metaphor, will draw your thoughts playfully and automatically. When you begin thinking about the paper route, your mind will unconsciously make an association with spreadsheets and figure out how the two are related.

For example, you might start thinking of the different houses on the paper route as columns. You might think of the people's names, addresses, telephone numbers, amounts owed, and your last collection date as rows in the spreadsheet.

You might also think of mapping the spreadsheeted paper route into a data base in which you could quickly determine who owes you for the papers, who is the most overdue, and what might be the most effective collection route for you to follow on your bicycle or in your car.

In fact, you might never put all this information onto the computer. It might be too much trouble entering the information and keeping it up-to-date. But this doesn't matter as long as you have a model of the spreadsheet or the data base inside your head.

For many, many applications in life, building a mental prototype inside your head is enough. It's not practical to go any further. The value of the computer skills is not that you use them on the computer, but that you can organize information, perform tasks, and solve problems better inside your head. This helps you become a better thinker, learner, and communicator on your own. You don't need a real computer around. You can carry one inside your head.

Learning Through Play

One of our greatest joys in life comes when we play—or when we feel we are playing. We might be working, but if it feels like play, we will be more motivated, more intense, and do a better job.

Passion and joy are not attributes of work but of love. And when we love what we are doing, it is never work. No matter how difficult the activity is, it feels like play.

I think that people can use computers to think playfully, learn playfully, and communicate playfully. The real joy of computing doesn't come from getting a job done faster, easier, or cheaper; it comes from making the job more challenging and more fun while you're doing it.

Are You A Neoprogrammer?

How is your relationship with your computer? Does your computer challenge you to think, learn, and communicate better? Does it make work more fun and interesting? Have you been able to take your computer skills with you when you leave the computer? Can you think on your own when your computer is turned off?

If you can, congratulations. Maybe you are a neoprogrammer and you don't even know it.

Whether you think you are a neoprogrammer or not, I'd like to hear your thoughts. What do you think about building a computer inside your head? Please write to me:

Fred D'Ignazio
2117 Carter Road, SW
Roanoke, VA 24015

©

WE TAKE THE BYTE OUT OF HIGH COMPUTER SOFT & HARDWARE$!

CBS SOFTWARE

ATARI

IBM

SHARP

COMMODORE

apple

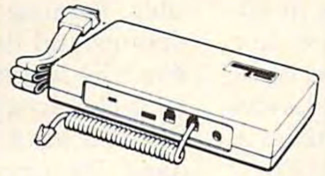
SANYO

COLECO

AND MANY, MANY MORE!

CALL 1-800-372-0214 AND SAVE!

MODEMS



Anchor Volksmodem **$59.00**

Anchor Mark VI	$159.00
Anchor Mark XII	$289.00
Hayes Smart Com 11	$ 75.00
Hayes 300	$209.00
Anchor Mark VII	$119.00
Auto Modem (Commodore)	$ 69.95
MPP 1000C (Atari)	$109.95

COMPUTERS

SANYO

COMMODORE 64

Commodore 64 **CALL**

Atari 800XL	CALL
SANYO 550, 550-2	CALL
SANYO 555, 555-2	CALL

SOFTWARE

Zork I, II or III **$29.95**

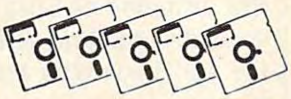
Super Base 64	$69.95
Paper Clip w/spell	$79.95
Home Accountant	$59.95
Bank Street Writer	$49.95

DISK DRIVES

Commodore 1541 **$235.00**

New Commodore Drive	CALL
Atari 1050	$329.00
Rana 1000	$289.00
Trak At-D2	$369.00
Indus GT	$369.00

DISKETTES



Computer Warehouse ss/dd **$15.95**

Elephant ss/sd	$16.95
Elephant ss/dd	$19.95
Maxell MDI	$24.95
Verbatim ds/dd	$24.95
Ultra Magnetics ss/sd	$15.95

PRINTERS

Commodore 801 **$219.00**

Gemini 10X	$269.00
Gemini 15X	$395.00
Delta 10	$489.00
Power Type	$399.00
AX10M AT 700 (color)	$499.00

AX10M AT 550 **$259.00**

Atari 1025	$249.00
Atari 1027	$289.00
Commodore 1526	$279.00
Okidata 82, 83, 84, 92, 93	CALL

MONITORS

Commodore Color **$249.00**

BMC 13" color	$229.00
Panasonic (composite & RGB)	$329.00
SAKATA 13 color	$235.00
BMC (green)	$ 69.00
BMC (yellow)	$ 99.00

COMPUTER WAREHOUSE

Computers and Software at warehouse prices.

Send for free monthly special sheet.
$3 for software, hardware & educational catalogs

8764 S.W. 133 STREET
MIAMI, FL 33156

FREE Computer Warehouse Diskette with any purchase.

Exodus: Ultima III For Commodore 64

David K. Peacock

Exodus: Ultima III ushers in an exciting new era of fantasy role playing. The combination of superb graphics, music, and excellent playability makes *Exodus* a modern-day masterpiece. The game presents challenges requiring clear, creative thinking plus the patience and determination to thwart hundreds of monsters during a quest to defeat the ultimate foe: Exodus.

An Adventure In The Box

Just opening the box is an adventure. Inside, you discover such magical items as a book of wizard spells, another full of incantations, a comprehensive playbook along with a quick reference guide, and a colorful cloth map of the realm to be explored. Also included is a key in the form of a black disk which, once booted, opens the way to the universe of Sosaria, where your dreams and fears materialize and your wits are your only hope.

After making a copy of the master side of the disk, you are ready to begin your journey. First you must create several characters to do your bidding. Up to 20 characters may reside per disk, and up to 4 may travel together at one time. Each character has a name, sex, race, profession, and the four attributes of strength, dexterity, intelligence, and wisdom. Take your time and choose wisely among the five possible races and eleven professions. Also, consider which attributes are important for different characters while using up as few points as possible. Because there are so many options and tradeoffs involved, don't be surprised if some of your characters just don't cut it and you have to create new ones. The opportunity for multiple characters, with varying personalities and abilities, enhances the playing environment over the single character allowed in *Ultima II*.

Sosaria Awaits You

Once your party is formed, the quest begins. The disk spins for a moment, and you find yourself in the magical realm of Sosaria where the waves lap the shores and banners atop towns flap in the breeze. Walking along, you notice open grassy plains, tall mountains, and dark forests. Your ears are treated to enchanting medieval tunes throughout. Suddenly, a band of nasty orcs appear heading straight for you. You duck behind a range of hills where the monsters can't find you.

Now is the time to seek a town and outfit your party with much-needed supplies such as weapons and armor. Even though all your characters begin with cloth armor and a dagger apiece, better equipment could be a lifesaver. Remember, at the beginning, your characters are weak in every respect and must be nurtured until they have grown strong in body and mind and have gained knowledge along with experience. Until then, on to the safety of a town.

Weapon Trading

When you enter a town you'll

find many citizens roaming the streets. These people are worth getting to know, for only by speaking to everyone will you learn secrets to help guide you along. Also, clues can be found only with extensive exploration.

One major improvement of *Ultima III* upon its precursor concerns the weapons and armor shops. In *Ultima II*, you were limited to buying; now, in this game, you can buy and sell. The variety of weapons and armor is better than ever. In fact, there's a rumor that some weapons are effective over a great distance—that might be worth even a steep price.

As in *Ultima II*, there are places to buy food and several pubs whose bartenders hear tales and could give you a tip or two. There are also stables with sturdy horses. Occasionally your party will come across an oracle, a man of wisdom and divine insight who might impart some of his knowledge for part of your gold. Two new and useful places to visit are the thieves' guild shops and the houses of healing. You'll find this and more in towns, not to mention a couple of castles and enough dungeons to make your head spin.

Dungeons. The word conjures up images of dark, twisting passages, sounds of funeral organ music, and thoughts of impending doom. This is the mood of the endless dungeons of *Ultima III*. These 3-D dungeons represent a significant improvement over the simple underground mazes in *Ultima II*. Exploring your first dungeon is thrilling as you attempt to overcome pesky gremlins, howling winds, foul traps, dozens of monster groups, and enough twists and turns to make getting lost no problem at all.

Reaching the lower depths—where the goodies are—requires careful planning and a working knowledge of the layout of each level. Once the treasures are located, it will take cunning to get your party back out alive. If all the treasures had been packed into one or two dungeons, the game would have been almost perfect. Instead, vital things were spread out among many dungeons, decreasing the enjoyment of each one. After you've conquered one dungeon, the rest can become tedious. Of course, a true dungeon lover might see things differently.

A Four-Player Battlefield

A new combat routine has been implemented to accommodate up to four players. When a monster group is encountered, the scene shifts to a battlefield where all the monsters and all the players can be seen. Each player gets a turn in which he may reposition himself, attack an oncoming monster, or cast a spell. Then each of the monsters performs a similar act. The battle rages on, turn by turn, round by round—gone are the days of instant destruction.

Though the combat sequence is well conceived, it is simply too slow considering the number of monster groups which must be dealt with. Granted, the pace does quicken once the characters' attributes have been raised, but most of the game is spent slugging it out. Then, for the effort, your party garners a single chest containing barely enough gold to sustain everyone. On rare occasions, a small weapon or cheap armor may be locked inside. If more items were found more of the time, agonizing money problems would diminish and the party could proceed with more interesting tasks.

Wizards And Clerics

One of the best aspects of *Ultima III* involves the extensive use of magic. Now wizards and clerics can demonstrate their true value as they cleverly choose just the right spell to save the party from a slew of poisonous balrons. At first, your spell casters will be limited and somewhat ineffective, but as time passes and they grow smarter and wiser, they will become indispensable. The wizards' spells mainly center on harming evil creatures, while the clerical spells are good for healing and resurrection. Both sets include very handy spells for maneuvering in dungeons. The two books of magic provide wonderful insights into the workings of each spell, making the game even more bewitching. Overall, the use of magic in *Ultima III* is well integrated with the obstacles to be overcome.

Moon Gates

Time affects many aspects of the game. If, for example, a member of the party is poisoned, the passage of time slowly brings about his death. Otherwise, wounds heal with time and spell points increase to their maximum. Also, if your party has fought pirates and gained control of their ship, only time will allow the winds to shift in your favor so that you may explore new lands. Perhaps the most important effect of time concerns the ever-present moons, Trammel and Felucca. As they pass through their cycles, strange events take place. Warps in space, called moon gates, appear only at certain times. Somehow, the moons and gates are thought to be connected, hence the name. There is a rumor of a city hidden in a vast forest. Not only hidden, but also not always there. Time, moons, cities, gates—all interwoven to challenge the best adventurers. Such is the spell *Exodus* weaves about its players.

Game designer Lord British has outdone himself with his latest work of art. *Ultima II* was a fantastic game, but *Exodus: Ultima III* makes it seem like child's play in comparison. *Exodus* has achieved an unparalleled blend of setting, multi-character development, magic, plus a strongly integrated plot. The animated graphics sparkle with speed and color, and the sound effects achieve nothing less than a complete, evocative sound track. Except for a few places that tend to drag, *Exodus* is a delight to play, and I eagerly await the perils and pleasures of the fourth installment in the ultimate series.

Exodus: Ultima III
Origin Systems
1545 Osgood St.
North Andover, MA 01845
(617) 681-0609
$59.95
Apple II family, Atari 400, 800, Commodore 64, IBM PC & PCjr versions

©

The Seven Cities Of Gold

James V. Trunzo

There is nothing quite as frightening—or exciting—as the unknown. Imagine, then, what it must have felt like to be with Pizarro, Cortez, Ponce de León, or Columbus and to sail away from the familiarity of Spain in search of discovery, gold, and fame.

The ocean was wide and uncharted, and the lands were filled with strangely painted natives who were often hostile. The storms were fierce and could easily blow the ship far off course. Starvation and a slow and painful death would follow if land was not sighted. Yet even in the face of such obstacles, the conquistadors were lured by the promise of gold and treasure. The ship's captain just had to be brave, smart, and lucky enough to discover a new world.

Gold For Spain

One after one, the would-be explorers visited the court and on bended knees requested a grant to buy ships and hire men. In return they promised to establish missions for the church, forts to prove Spain's sovereignty, and gold to fill Spain's coffers.

Now you can experience the thrill of sighting land after a long, arduous sea voyage. You can also experience the pride in returning to Spain after having explored the Mayan Peninsula and discovered rich and fertile lands; or the shame of returning home after having lost most of your crew, several ships, and having no gold to show for your efforts.

Graphics And Strategy

Ozark Softscape, in conjunction with Electronic Arts, has produced a riveting new adventure game entitled *The Seven Cities of Gold* that places you at the helm of a fleet of ships and allows you to venture forth from Spain in search of a new world, wealth, and fame.

As in their award-winning game *M.U.L.E.*, the Bunten brothers have designed a graphically enhanced strategy game that challenges and educates as well as entertains. Upon booting *The Seven Cities of Gold*, the player finds himself in front of a palace in Spain. He has just been given a commission by the Spanish court; and as captain of a fleet of four newly outfitted ships, he is ready for his first voyage.

Leaving The Old World

After scrolling past a pub, his home, and an outfitters building (all important places when returning home from an expedition), the player leaves the Old World and ventures forth in search of the new. Sailing is controlled by the joystick, as are all actions and options. While at sea, the player may navigate the ship, view the map, and keep track of how many days have elapsed. The latter is especially important for several reasons. For one thing, your food supply isn't unlimited.

Eventually you will sight land. At this point, you will have to decide how much of the on-board supplies, goods, and men you want to take to explore the uncharted mass into which you have just bumped. Now the real fun begins. There will be lush jungles, fertile plains, intimidating mountain ranges, dangerous swamps, major rivers, and natives.

Jungles And Swamps

Accomplishing all your objectives is no easy task. Ambushes in the thick jungles will take their toll as will sickness in the swamps. Food is a constant source of worry; men won't travel on an empty stomach, let alone fight on one. And as the land grows cold with the approach of winter, food becomes scarcer.

Once you decide that it is time to return home (a decision often made easy by the loss of men, goods, etc.), you must navigate back to Spain. Assuming that you make it back, thwarting the best efforts of nature's storms, a trip to your home will provide you with a tally of what areas you have discovered, what forts and missions have been established, and how much wealth has been obtained. A trip to the court will give you a rating based upon your successes or failures. More gold, a promotion, or chastisement awaits you in the court. Finally, a trip to the pub allows you to record (save to disk) maps for future voyages. The outfitter? Most assuredly, it will be your first stop before weighing anchor for the next excursion. There you will buy food and goods, hire more men, and perhaps even purchase more ships.

Historical Accuracy

The mechanics of *The Seven Cities of Gold* are easily implemented and well-done. All movement, both on land or at sea, is handled by use of the joystick, as are all option selections and even combat. The graphics are well-done, and *Cities* contains over 2800 screens that represent the lands you will explore. The computer literally draws the map as you move about North, South, and Central America, all accurately depicted.

Your expedition is represented by an arrow moving over a variety of easily identified terrain. Symbols are used in various places to represent hundreds of different types of settlements, ranging from farmers and hunters to wealthy Aztec strongholds. It is upon entering one of these settlements that another of *Seven Cities'* delights is discovered.

Once the player has moved the arrow onto a settlement symbol, the screen symbol begins to magnify, increasing in size until it is replaced by a detailed graphic screen. The arrow is replaced by a conquistador who represents the expedition, and you find yourself in the middle of the settlement, rapidly surrounded by natives. Find their chief and begin trading, or draw your sword.

An Enchanting Challenge

There are many more surprises in *The Seven Cities of Gold*. The program both challenges and enchants. It forces you to consider various strategies: What is the best way to outfit an expedition? Do you have enough men to establish forts? When should you return home for more supplies? Even the time of year can be an important factor.

And what happens after the player discovers the Mississippi or the Amazon, gold mines, the Fountain of Youth, and all the mysteries of the Americas? Is the game over? Not a chance. Aside from the fact that the game could be played again using different strategies and achieving higher rankings, *Seven Cities of Gold* provides a utility that randomly generates entire continents; no two are ever the same. Furthermore, all games can be played at one of three levels: novice, journeyman, or master.

Seven Cities of Gold
Electronic Arts
2755 Campus Drive
San Mateo, CA 94403
(415) 571-7171
$39.95
Available for Atari now and for Commodore 64 soon. ©

Word Flyer

Steve Hudson, Assistant Editor, COMPUTE! Books

Dozens of educational programs have been released—some good, some less than good—but one of the most interesting is *Word Flyer.*

Best known for dynamic and challenging games like *Archon, Pinball Construction Set,* and *Worms?*, Electronic Arts has developed a reputation for sophisticated programs. *Word Flyer* is no exception. Like most educational programs, it uses graphics and sound to reinforce learning, but uses them in a way that's both new and refreshing.

Word Flyer was developed by ChildWare, a programming group within Electronic Arts. Typically, ChildWare programs combine proven educational psychology with captivating programming, and *Word Flyer* is no exception.

The object of the game is straightforward: Use your joystick to maneuver word flyers and match zooming letters or words. It's a challenging and exciting game for young children. But there's learning amidst the laughter. Without realizing it, players are practicing valuable reading and vocabulary-building skills. On lower levels, the emphasis is on recognizing the letters of the alphabet; that makes the program valuable even for children who have not yet learned to read. Higher levels introduce words chosen from a built-in list of over 2000 entries. The approach is both original and nonviolent.

Booting The Birds

Two towers—built of the word *towers*—dominate the screen, one on the left and one on the right. Atop each tower sits a remarkably realistic-looking bird. A control panel runs across the bottom of the screen; it consists of flight level and speed indicators, a score bar, a timer, and a number-of-players indicator. On higher levels an alphabet bar appears too.

Play starts on flight 1, where emphasis is on the alphabet and on two-letter words. Flight 2 comes next, giving you the chance to match three-letter flyers. Subsequent flights introduce you to three- and four-letter flyers and faster speeds.

On flights 2 and above, you also gain access to the "alphabet bar." That allows you to select the first letter of your flyers. On levels 4 and 5 you can also change the color of your flyer to match the color of various zooming words.

If you're playing a two-player game, the hourglass timer will clock each player's turn. Need to take a break? At any time, on any level, you can move your flyer to the "rest nest" (an unmistakable mass of sticks and twigs) and press the joystick button to stop the timer. Also, at the beginning of each game (and at any point during play), you have the option of entering the "control panel" and changing any of the game parameters.

Although it takes a few minutes to get the hang of it, game play is fundamentally simple. Use your joystick to select a word from either word tower—the chosen word will be highlighted for you—and then press the button to send the chosen word flyer soaring into the air. Move it into position to match one of the soaring words, and press the joystick button again. If the match is correct, one of the birds will nod approval. If your match is incorrect, the bird will pronounce the avian equivalent of "uh-oh!"

Cooperative Scoring

In either case, your score will change appropriately. The score is increased when a player matches the flyer with the correct letter or word. On higher levels, additional points are awarded if the words' colors match too. Incorrect matches lower the score slightly and return you to the word tower. In two-player games, an incorrect match ends that player's turn.

Many parents will be pleased with this departure from the winner/loser approach of other multiplayer games. *Word Flyer* emphasizes constructive cooperation instead of conflict and destruction. The total score increases whenever either player correctly matches a letter or word. By working together, two players can move through the different levels more quickly than either could alone.

Parent and child can play together, working toward a common goal, and the child will learn to recognize letters, words, and colors. But he or she can learn the importance of cooperation too.

Where's The Word?

Word Flyer's graphics and sound are effective without being overpowering. Joystick control is responsive. The constantly changing list of letters or words holds interest, assuring many hours of satisfying and challenging play.

However, after several sessions, one odd quirk does become evident. In some cases, while exuberantly chasing down a zoomer, the flyer would fly off the top edge of the playing field. However, you can move the joystick to maneuver the flyer back onto the screen. Bothersome? A little, at first, and it might confuse very young children.

Also, at several points in the otherwise excellent manual, the reader is told that something will be described under a subsequent heading. It is mildly confusing (and occasionally annoying) to have to skip ahead to figure something out; in the case of instructions, at least, necessary redundancy is a feature that many software manuals still lack.

But once you figure it out—and it won't take long—control is simple and straightforward. Selecting flyers, colors, levels, and speeds quickly becomes second nature, allowing players to concentrate on the game itself. The educational goals underlying this game are pleasantly and effectively achieved. All in all, a deft piece of work. ©

Word Flyer
Electronic Arts
2755 Campus Drive
San Mateo, CA 94403
Atari 400/800/600XL/800XL/1200XL
$35 (48K disk)

BIG NAME, NO NAME DISKETTES: THE BEST BUY YOU CAN FIND!

LIFETIME WARRANTY!

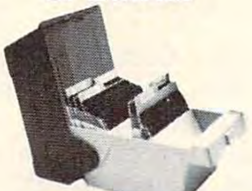
$1.19 ea. 5¼" SS-DD Qty. 50 — **$1.59** ea. 5¼" DS-DD Qty. 50

$1.89 ea. 5¼" SS-DD-96TPI Qty. 50 — **$2.14** ea. 5¼" DS-DD-96TPI Qty. 50

(These are bulk packed diskettes with reinforced hubs and separate white Tyvec sleeves.)

(Add $3.00 shipping per 100 diskettes. For C.O.D. orders, add another $3.00 handling charge.)

SPECIAL!



Act now and you can get a special bonus: an Amaray Media-Mate 50 disk storage unit. Extremely well designed with grooves for stacking. Normally $14.95. When ordered with 50 BIG NAME, NO NAME diskettes, you can get one for $9.99 (Shipping Included).

WELL, OUR SEARCH OF MORE THAN A YEAR HAS PAID OFF.

We found a diskette that you've used a hundred times...but have never heard of. It's the one used by a lot of major software houses for their copy-protected originals...the kind of diskettes that receive a lot of heavy use.

These BIG NAME, NO NAME diskettes *exceed* all ANSI and IBM standards. They are **individually** tested track by track, sector by sector and certified 100% error-free.

Moreover, the company that makes them has their own **proprietary** coating formulation which results in a super-smooth, anti-static surface.

In short, these are good diskettes.

If they weren't, we wouldn't be selling them.

LIFETIME WARRANTY!

As with every diskette we sell, BIG NAME, NO NAME diskettes are covered by a LIFETIME WARRANTY by us and the manufacturer.

If you have any problems with them at any time because of materials or workmanship, just call us to arrange a return.

DON'T SAY NO!

Sure, you've never heard of BIG NAME, NO NAME diskettes.

But a lot of software manufacturers who use hundreds of thousands of diskettes for their copy-protected originals have...and a lot of them use BIG NAME, NO NAME diskettes.

DISKETTE HEAVEN GUARANTEED!

Order your supply of BIG NAME, NO NAME diskettes today.

If you're not totally happy with them at the end of 30 days, we'll give you a refund, credit or exchange.

That's confidence.

That's DISK WORLD!...the nice people who sell almost a million good diskettes a month.

TO ORDER:

Call DISK WORLD between 9AM-5PM, Central Time, Monday through Friday at:

1-800-621-6827

or in Illinois, at:

1-312-944-2788

Or write us:

DISK WORLD!, Inc.

Suite 4806 • 30 East Huron Street • Chicago, Illinois 60611

DISK WORLD! Home of BIG NAME, NO NAME DISKETTES

RATED #1 FOR SERVICE & RELIABILITY

CALL TOLL FREE **800-221-8180** IN NEW YORK STATE CALL (212) 732-8600

J&R MUSIC WORLD — 23 PARK ROW, NEW YORK CITY, NEW YORK 10038

THIS MONTH'S SUPER SPECIALS

COMMODORE 64

- 64K RAM
- Microsoft Basic
- High Resolution Graphics

$199.95

COMMODORE 64 SOFTWARE

ENTERTAINMENT

- COMMODORE Gorf $14.95
- SYNAPSE Blue Max (C or D) $24.95
- SYNAPSE Zaxxon (C or D) $24.95
- SUBLOGIC Flight Simulator II $39.95
- ACCESS Beach Head (D) $24.95
- ACCESS Neutral Zone (D) $24.95
- ATARI SOFT Pole Position $29.95
- ATARI SOFT Ms. Pac-Man $29.95
- COMMODORE Music Machine $14.95
- AUTOMATED Summer Games (D) $24.95
- AUTOMATED Temple of Apshai (D) $24.95
- AUTOMATED Pitstop $24.95
- MUSE Castle Wolfstein (D) $19.95
- COMMODORE Kickman $14.95
- WAVEFORM MusicCalc ProPack (D) $119.95

BUSINESS

- COMMODORE EZSCRIPT (D) $39.95
- HES Multiplan (D) $59.95
- COMMODORE The Manager $34.95
- COMMODORE EZCALC (D) $54.95
- COMMODORE Simon's Basic $39.95
- COMMODORE CPM 2.2 $49.95
- COMMODORE Magic Desk (D) $49.95
- COMMODORE Logo $49.95
- COMMODORE Assembler/Edit (D) $14.95
- SCARBOROUGH Mastertype (Cart) $24.95
- COMMODORE Nevada Cobol (D) $39.95
- CONTINENTAL Home Acct (D) $44.95
- SIERRA Homeword (D) $39.95
- BATTERIES INCLUDED Oracle (D) $109.95
- BATTERIES INCL Paperclip w/spell $84.95
- ELECT. ARTS Cut and Paste (D) $34.95
- COMMODORE Nevada Fortran (D) $39.95
- COMMODORE Type Right (D) $17.95
- COMMODORE General Ledger (D) $34.95
- COMMODORE Inventory (D) $34.95

SPINNAKER each $24.95
Choose from: Fraction Fever, Aerobics, Story Machine, Up for Grabs, Trains, Snooper Trooper 1, Kidwriter Story Creator, Alf in the Color Caves, Cosmic Life, Bubble Burst, Rhymes and Riddles

MONITORS

- COMMODORE C1702 14" Color $239.95
- TAXAN 100G 12" Green Screen $99.95
- PANASONIC TR-120 12" Green $149.95
- PANASONIC DT1300D 13" RGB $299.95
- AMDEK 300A 12" Amber Screen $149.95
- ZENITH ZM122 12" Amber, 40/80 $119.95
- TAXAN 415 RGB Super Hi-Res $429.95

FLOPPY DISKS

SINGLE SIDED/DOUBLE DENSITY $24.99/10
Choose from: Verbatim MD525, Elephant Gold, BASF FD1D, TDK M1DS, Maxell MD1

DBL SIDED/DBL DENSITY $29.95/10
Choose from: Verbatim MD550, Elephant Orange, Scotch, Maxell MD2D, TDK M2DS

We Also Stock 8" Disks

APPLE COMPATIBLE DISK SOFTWARE

ENTERTAINMENT

- SIERRA Time Zone $54.95
- SIERRA Wizard and Princess $19.95
- INFOCOM Deadline $29.95
- INFOCOM Zork 1 or 2 each $24.95
- INFOCOM Infidel $29.95
- SUBLOGIC Flight Simulator II $34.95
- ELECT. ARTS One on One $29.95
- ELECT. ARTS Pinball $29.95
- ODESTA Chess $44.95
- AUTOMATED Temple of Apshai $24.95
- ATARI SOFT each $29.95
  Choose from: Ms. Pac-Man, Defender, Centipede, Donkey Kong, Stargate, Pac-Man

BUSINESS/EDUCATIONAL

- CONTINENTAL Home Accountant $39.95
- SCARBOROUGH Mastertype $24.95
- SIERRA Homeword $29.95
- HAYDEN Sargon III Chess $29.95
- HAYDEN S.A.T. Sample Test $14.95
- BARRONS S.A.T. Study Course $59.95
- BRODERBUND Bank Street Writer $44.95
- SIERRA Screenwriter $84.95
- ELECT. ARTS Music Construction $29.95
- HAYDEN Pie Writer 2.2 $84.95
- SIL. VALLEY Word/List/Spell Handlers $99.95
- VERSA Inventory Control $59.95

SPINNAKER

- Story Machine $24.95
- Snooper Troops 1 $29.95
- Search Most Amazing Things $24.95
- Alphabet Zoo $24.95
- Rhymes and Riddles $19.95

IBM PC COMPATIBLE DISK SOFTWARE

ENTERTAINMENT

- HAYDEN Sargon III Chess $39.95
- INFOCOM Planetfall $29.95
- CYBERIA Murder by the Dozen $24.95
- STRATEGIC Tigers in the Snow $29.95
- STRATEGIC Galactic Gladiator $29.95
- STRATEGIC Warp Factor $29.95
- ATARI SOFT each $29.95
  Choose from: Pac-Man, Centipede, Donkey Kong, Stargate, Defender, Robotron 2084

EDUCATIONAL

- BARRON'S S.A.T. Study Course $59.95
- SCARBOROUGH Mastertype $29.95
- S.E.I. S.A.T. Math Skills Pack $39.95
- S.E.I. S.A.T. Verbal Skills Pack $39.95
- S.E.I. Word Analogy $24.95
- S.E.I. Sentence Completion $24.95
- S.E.I. Foreign Language Skills: German, Spanish, French each $24.95

BUSINESS

- CBIS MultiMate 3.2 $279.95
- CBIS SalesEdge $229.95
- SORCIM Friday dbII Manager $229.95
- SORCIM Supercalc 3 $249.95
- ELECT. ARTS Cut & Paste $39.95
- SYDNEY DATA Telegram Program $39.95

MODEMS

- ANCHOR AUTOMATION: VOLKSMODEM Universal w/ Cable $64.95 (Specify computer with order)
- SIGNALMAN MARK XII RS-232, Hayes Compatible $329.95
- ATARI 1030 Direct Connect $119.95
- VIC V1600 For VIC 20 $54.95
- VIC V1650 Auto Dial for C64 & VIC 20 $79.95
- MPP 1000C Direct Con. for Atari $129.95

Also available and in stock Apple, Eagle, IBM, and Sanyo computer systems. Call for lowest prices!

FREE GIANT CATALOGS

CALL TOLL FREE 800-426-6027

- 288 PAGE AUDIO/VIDEO/COMPUTER CATALOG
- 64 PAGE RECORD AND CASSETTE CATALOG
- 80 PAGE VIDEO MOVIE CATALOG

23 PARK ROW, Dept. C9, NYC, NY 10038

HOW TO ORDER BY MAIL: FOR PROMPT AND COURTEOUS SHIPMENT SEND MONEY ORDER, CERTIFIED CHECK, CASHIER'S CHECK, MASTERCARD/VISA (Include card number, interbank No., expiration date and signature.) DO NOT SEND CASH! PERSONAL AND BUSINESS CHECKS MUST CLEAR OUR BANK BEFORE PROCESSING. $25 MINIMUM ORDER. Shipping, Handling & Insurance Charge is 5% of Total Order with a $3.95 minimum. WE SHIP TO CONTINENTAL U.S., ALASKA, HAWAII, PUERTO RICO AND CANADA ONLY. (Canadian Orders Add 15% Shipping with a $9.95 minimum charge.) For shipments by air please double these charges. SORRY, NO C.O.D.'S. NEW YORK RESIDENTS PLEASE ADD SALES TAX. ALL MERCHANDISE SHIPPED BRAND NEW, FACTORY FRESH, AND 100% GUARANTEED. WE ARE NOT RESPONSIBLE FOR ANY TYPOGRAPHICAL ERRORS.

ATARI 800XL AND 1050 DISK DRIVE SPECIAL!

- Full-Stroke Keyboard
- 64K RAM/256 Colors
- Built-in High Level Basic
- DOS III Included

$499.95

ATARI SOFTWARE

ENTERTAINMENT

- ATARI Joust $34.95
- ATARI Ms. Pac-Man $34.95
- ATARI Robotron 2084 $32.95
- ATARI Pac-Man $32.95
- ATARI Dig-Dug $32.95
- ATARI Football $34.95
- ATARI Pole Position $34.95
- ATARI Donkey Kong Jr. $34.95
- AUTOMATED Pitstop $24.95
- ACTIVISION River Raid $24.95
- ACTIVISION Megamania $24.95
- AUTOMATED Gateway Apshai (D) $24.95
- ATARI Touch Tablet $74.95
- SYNAPSE Dimension X (D) $24.95

BUSINESS/EDUCATIONAL

- ATARI Atariwriter $69.95
- ATARI Invite to Progr 1, 2 or 3 ea $19.95
- SCARBOROUGH Mastertype $24.95
- ATARI Microsoft Basic II (Cartridge) $59.95
- ATARI Touch Typing $19.95
- ATARI Visicalc (D) $139.95
- ATARI Pilot Package $59.95
- ATARI Home Filing Manager (D) $34.95
- MICRO PERIF. Microfiler $24.95
- ADVENTURE INTERNATIONAL Scott Adams Graphics Editor (C and D) $29.95
- BRODERBUND Bank Street Writer $39.95

SPINNAKER

- Kindercomp $19.95
- Story Machine (Cart) or (D) $24.95
- Facemaker $24.95
- Kids on Keys $24.95
- Alphabet Zoo $24.95
- Delta Drawing $24.95
- Fraction Fever $24.95
- Up For Grabs $24.95
- Rhymes and Riddles $19.95
- Snooper Troops 1 or 2 each $29.95
- Trains (D) $29.95

DISK DRIVES

- ATARI 1050 New with DOS III $299.95
- VIC V1541 For VIC 20 & 64 $229.95
- RANA 1000 For Atari $279.95
- RANA ELITE 1 For Apple $249.95
- RANA ELITE 1+ For Apple w/DOS $299.95
- RANA ELITE 2 Dual-Density Apple $349.95
- INDUS ATARI GT w/DOS and Case $349.95

PRINTERS

- VIC V1526 Dot Matrix, 80 col. $279.95
- ATARI 1027 Letter Quality $279.95
- VIC V1520 Printer/Plotter $149.95
- EPSON MX100 136-column $449.95
- JUKI 6100 Letter Qual. Daisy Wheel $449.95
- CARDCO LQ/1 Letter Qual for C64 $429.95
- ATARI 1020 4 Color Printer $249.95

ALL EPSON AND OKIDATA PRINTERS IN STOCK...CALL AND SAVE

WRITE or CALL for FREE 288 PAGE AUDIO/VIDEO/COMPUTER CATALOG

HARMONY

VIDEO & COMPUTERS

TO ORDER CALL TOLL FREE 800-221-8927 OR 800-441-1144

IBM PC $1549
OKIDATA 92 $374

Micromodem 2E	189	Banana Prtr	129	Epson FX100	689
Hayes 300 Modem	179	TTX 1014	409	Brother HR15	369
Hayes 1200 Modem	454	NEC 3550	1444	Brother HR25	624
Amdek Monitor	114	Radix 10	529	Prowriter 8510	339
Amdek Color 1	229	Okidata 92	374	Starwriter F10	929
Taxan Color Mntr	249	Okidata 93	584	Gemini 10X	249
Olympia Ro	379	Okidata 82	299	Gemini 15X	359
Juki 6100	364	Okidata 83	509	SilverReed	359
Pana KXP 1090	249	Epson RX80	279		

ATARI

Atari 800XL	224
1025 Printer	264
1027 Printer	249
1050 Drive	299
Indus Drive	CALL

IBM

PC	1549
PCXT	CALL
Mono Card	209
Quadboard	209
Hercules	299

COMMODORE

VIC 20	79
Commodore 64	189
Disk Drive	209
MPS 801	189
1526 Printer	229
1650 Modem	89
1702 Monitor	194

APPLE

2E	699
MAC	2099
RGB Card	129
Printer Card	89

SANYO

550 SD	729
555 SD	939
550 DD	849
555 DD	1209

KAYPRO

II	1139
IVe	1739
X	2419

EAGLE

PC2	1299
PCXLT	1899
File 10 Drive	999

FRANKLIN

1000 Pro	769
1200 DMS	1289

For Info Dial (212) 627-1000 No dealers on advertised specials, please! To order simply dial toll free 800-221-8927 or 800-441-1144 with your MasterCard or VISA and your order will arrive via UPS or send certified check or money order only to: HARMONY VIDEO AND ELECTRONICS, 2357 Coney Island Ave., Brooklyn, N.Y. 11223, and add approximate shipping, postage and insurance charges. Customer Service (212) 627-8888 Mon-Fri. 9-5. Credit cards for phone orders only. All prices and availability subject to change without notice. Dealer inquiries invited!! MC, VISA

FAMILY TRIVIA

NOW THE WHOLE FAMILY CAN PLAY TRIVIA AT THE SAME TIME! Adults, teenagers and children can all play — each one answering questions geared to their age level. No more parents getting bored with children's games, no more children getting frustrated at the difficulty of adult games.

BRING THE FAMILY TOGETHER TO PLAY — GET **FAMILY TRIVIA** TODAY!

SPANISH VERB TUTOR

- Helps Beginners or Advanced Students in conjugations and translations.
- Over 650 verbs (including those tricky irregular verbs).

If it isn't here, maybe you shouldn't say it. Fast dictionary listing, translation from English or Spanish. Conjugations in 8 tenses, includes examples. IDEAL FOR CLASSROOMS.

FILL OUT FORM BELOW TO ORDER OR FOR MORE INFORMATION

Comm. 64 ☐ Tape ☐ Disk — TI 99/4 ☐ Tape ☐ Disk (Req. 32K and Ext. Basic)
Family Trivia ☐ $29.95 — Family Trivia ☐ $29.95
Sp. Verb Tutor ☐ $29.95 — Sp. Verb Tutor ☐ $29.95

Name ______________________
Address ______________________
City__________ State____ Zip________

CT. RES. ADD 7½ SALES TAX

Make Checks Payable to:

PRONTO software

15 MEIKLEM ST., FORESTVILLE, CT 06010

Lightning Sort

Russ Gaspard

Last September COMPUTE! *published "Ultrasort," and we called it the fastest sorting program ever published for any home computer. It would sort a 1000-element array in less than eight seconds.*

It's been improved. Here's "Lightning Sort." It does the same thing in a breathtaking 2.1 seconds. Add this extraordinarily powerful subroutine to any of your BASIC programs where you need to alphabetize something. For the VIC, 64, and PC/PCjr. Atari users should refer to the accompanying sidebar and program "Bulldozer Sort."

The "Ultrasort" routine for Commodore computers (COMPUTE!, September 1983, p. 194) isn't as fast as it could be. After disassembling the code to study the algorithm, I found several opportunities to compact the code (mainly to reduce disk loading time) and to speed up the execution time. Using the "Sort Test" program from the original article as a benchmark, my "Lightning Sort" routine sorts a 1000-element array in an average of 2.1 seconds, versus 7.8 seconds for Ultrasort. That few seconds savings isn't much. But when I tried it on random 4000-element arrays the routine took an average of 10 seconds, versus 40 seconds for Ultrasort. A 400 percent speedup in execution time can be significant in applications where the sort routine is called repeatedly, or in sorting very large arrays.

The time for this type of algorithm to sort an N-element array is $T*N*Log_2N$ on the average, where T is about .21 milliseconds for the modified routine and .8 milliseconds for the original. Actual running time depends on the starting order of the array. Interestingly, whereas many sort algorithms run fastest when the original array is already in order, Hoare's Quicksort runs fastest on randomly ordered data. If you try it on an array which is already in correct order you'll find that it takes much longer (proportional to N^2).

Besides speeding up the execution, I was also able to reduce the amount of RAM needed from 908 bytes to 418 bytes. By storing the variables in RAM space above the actual sorting routine rather than within the routine, the actual program storage needed on disk is only 338 bytes. This means the saved program uses only two disk blocks, rather than the four required for the original.

Program 1 is a BASIC program which loads the machine language Lightning Sort routine for the Commodore 64. The routine is loaded into RAM from $C000 to $C152 (decimal 49152 to 49490), and writes variable data up to $C1A2 (decimal 49570). It is used in exactly the same way as Ultrasort. However, I prefer to define the call address 49152 as variable QS (either within the BASIC program or in direct mode) and then call the routine with:

SYS QS,N,AA$(K)

where K and N are the first element and the number of elements to sort, and AA$ is the array variable name, as in the Ultrasort article.

Program 2 is a BASIC loader for the VIC version of Lightning Sort. It automatically relocates the machine language to the top of available memory, regardless of the amount of expansion installed, and protects the sort routine from BASIC. The program also tells you the proper SYS to use to start the sorting.

Although Program 2 will run on an unexpanded VIC, we recommend that at least 8K expansion be used. With less than this, only a very few items can be sorted.

Program 3, the Sort Test program from the original Ultrasort article, can be used as a demonstration of Lightning Sort. The program creates an array, AA$, of 1000 random elements, then sorts them into order. If you are using a VIC with limited memory, you'll need to reduce the number of elements.

Program 1:
Lightning Sort Loader For The 64

Refer to the "Automatic Proofreader" article before typing this program in.

```
10 I=49152:SUM=0                        :rem 136
20 READ A:IF A=256 THEN 40              :rem 54
30 SUM=SUM+A:POKE I,A:I=I+1:GOTO 20
                                        :rem 79
40 IFSUM<>45295THENPRINT"ERROR IN DATA ST
   ATEMENTS":END                        :rem 191
50 PRINT"LIGHTNING SORT READY.":END
                                        :rem 214
49152 DATA 32,253,174,32,158,173  :rem 52
49158 DATA 32,247,183,165,20,133  :rem 52
49164 DATA 253,165,21,133,254,32  :rem 46
49170 DATA 253,174,32,158,173,162:rem 104
49176 DATA 1,165,71,157,85,193    :rem 221
49182 DATA 157,125,193,165,72,157:rem 114
```

Atari Bubble And Bulldozer Sorting

Christine C. Genet

While machine language data sorting is extremely fast, there still may be times you will want to insert a simple BASIC sorting routine into a program. When the list to be sorted is small, bubble sorting is a good method to use. For larger lists, a technique called bulldozer sorting may be better.

Using The Bulldozer Sort Program

The program is a demonstration of the bulldozer sort. It asks how many numbers you want to sort and the value of the highest number in the list. It then generates random numbers in the desired range. When finished sorting, it prints all nonzero values to the screen.

To use the bulldozer sort as a subroutine, delete lines 70 through 85 and add a line to the beginning of the program defining the number of data elements (RN) and the maximum value of the data (MV). Also, change line 111 so that it will input the data in the way that is needed for your program. For example, to input data from the keyboard, change the line to read:

```
111 INPUT DT:IF DT>MV THEN 111
```

If you would like the sorted list printed to the screen as part of your subroutine, change line 550 to read:

```
550 RETURN
```

If you don't want a screen print, delete lines 500 through 550 and add the following line:

```
200 RETURN
```

How Bubble Sorting Works

The bubble sort is a commonly used method of sorting small lists of data into numerical or alphabetical order. While bubble sorts are easy to understand and use in programs, they are often too slow to use for large sorting tasks—bubble sorting requires many comparisons.

A bubble sort compares each item against the other unsorted items. If the item tested is larger than the one it is tested against, their positions are switched. This way, after all of the values have been tested once, the first position in the array contains the lowest number in the list.

Sorting A Stack Of Cards

Suppose we have a small stack of index cards that are out of order. We have four cards (numbered 1 through 4) to sort, and they are in the following order: 3, 2, 4, 1. To begin, we compare the first card (3) with the second (2). Since 2 is less than 3, we swap the cards and the order becomes: 2, 3, 4, 1.

Next we compare the first and third cards in the deck, and since 2 is less than 4, no swap occurs. Comparing the first and fourth cards, we see that they should be swapped (since 2 is greater than 1) and our stack of cards reads 1, 3, 4, 2.

Now we have placed the lowest card in the first position, so we can start our second series of comparisons with the second card in the deck. We compare the second and third cards (3 and 4) and make no swap, then compare the second and fourth cards, swapping 3 with 2. At this point, the first two positions in the deck are set and the order is 1, 2, 4, 3. Testing the third card is easy, since there is only one comparison left, and we switch the positions of 4 and 3 to finish our bubble sort with the array filled as follows: 1, 2, 3, 4.

Our mental sort took only six comparisons, and was pretty quick. But with longer lists, bubble sorting slows down greatly. The reason for this is that in any array with N elements, the number of comparisons required will be $N(N-1)/2$. This means that while a bubble sort of 20 items will require 190 comparisons, a list only four times as long (80 items) will require over 16 times as many comparisons (3160). In order to speed things up, we need to reduce the number of comparisons as much as possible.

A Faster Sort

An alternative is bulldozer sorting, first described by Isaac and Singleton, in *JACM* 3 (1956): 169–174. Bulldozer sorting uses *address calculation* to roughly position items in the array before sorting them. We bulldozer sort every time we use an index card file—we look for the correct section of files first, then sort the card into the specific place it belongs. On a computer, this sort works well for up to around 500 items and is faster than bubble sorting, although it uses more memory for the array.

Another feature of the bulldozer sort that makes it faster than the bubble sort is that the bulldozer sort arranges the items one by one as the data is input—there is no long wait for the sort to finish after all of the data has been entered.

Address Calculation

To successfully predict where the data should be placed in the array before sorting, keep two requirements in mind:

1. The array used for sorting and storage of the data should be about 1.4 times as large as the data list, and
2. The formula for calculating the estimated address should be chosen to allow empty array spaces above, below, and between the sorted data elements.

The first requirement is easy to handle; just DIMension the data storage array to a value 1.4 times greater than the size of the data list.

Borrowing An Equation

To satisfy the second requirement—leaving extra space in the array—we need an equation that predicts a position for the lowest data element about 10 percent of the way into the array, and estimates the highest data element's position to be about 10 percent from the end of the array. Since the accuracy of the predicting equation is not critical, we'll use a simple one borrowed from geometry—the equation for a line—to put the data in the correct general area of the array. Then we'll sort the data into the exact location.

For example, if we had 200 job numbers (or other data elements) ranging in value from 0 to 500, we would DIMension the array to 280. We would also want the lowest value to be placed by the equation in the 28th array position and the highest value to be sent to the 252nd position.

The general equation for a line is $y=mx+b$, where m is the slope and b is the place where the line crosses the y-axis. The slope of a line is the rise (change in the value of y) divided by the run (change in the value of x). We want predicted points to be in the middle 80 percent of the array, so we multiply m in the above equation by 0.8. For the value of b, simply use 10 percent of the array size (28). The estimated array position for $x=250$ would be:

$$y=mx+b=0.8(280/500)x+28=0.448(250)+28=140$$

Note that of the 281 array positions created by DIMensioning, position 140 is very near the center. Using the same equation to predict a position for $x=251$, though, yields a value of 140.448, which rounds to 140.

Obviously one array element can hold only one data value, and this is where sorting becomes necessary. When an array location is already being used, the bulldozer sort compares the two values and rearranges the list. It is this readjusting feature of the bulldozer sort that requires the 40 percent extra array storage. The program slows down as it sorts near the end of the data list because more of the predicted locations are filled and more sorting is necessary.

Bulldozer Sort

```
EL 70 PRINT "{CLEAR}HOW MANY RANDOM
      DATA ELEMENTS";
JM 75 INPUT RN
PF 80 ? "WHAT MAXIMUM VALUE";
KA 85 INPUT MV
EL 90 AS=INT(0.5+RN*1.4):DIM JN(AS):
      DN=0:I=0
BP 95 PRINT "CLEARING THE ARRAY ";
OI 100 FOR A=0 TO AS:JN(A)=0:NEXT A
NG 105 PRINT "ARRAY CLEARED"
LN 110 I=I+1
GA 111 DT=(INT(100*MV*RND(0)+0.5)/10
       0)
FD 115 PRINT "DATA ELEMENT: ";I;"
       {4 SPACES}VALUE: ";DT
FN 130 APP=INT((0.8*AS*DT/MV)+0.1*AS
       +0.5)
EJ 135 C=0
FN 138 REM ****** Lines 140-160 dete
       rmine which subroutine to acc
       ess to sort data correctly **
       ****
DF 140 IF JN(APP)=0 THEN JN(APP)=DT:
       GOTO 180
IC 150 IF JN(APP)>=DT THEN GOSUB 500
       0:GOTO 180
EF 160 IF JN(APP)<DT THEN GOSUB 6000
       :GOTO 180
AD 180 IF I<RN THEN 110
CB 500 REM **** PRINTING SORTED NUMB
       ERS ****
BD 505 PRINT "NUMBERS SORTED. NOW PR
       INTING."
JM 508 DN=0
GD 510 FOR B=0 TO AS
EJ 515 REM *** Array positions witho
       ut numbers are not printed ou
       t ***
GF 519 REM **** Zeros are not printe
       d ****
IG 520 IF JN(B)=0 THEN 540
DO 530 DN=DN+1:? "ARRAY ELEMENT: ";D
       N;"{4 SPACES}VALUE: ";JN(B)
BK 540 NEXT B
HB 550 END
GE 5000 REM **** Placing numbers les
        s than job presently at loca
        tion****
CD 5010 APP=APP-1
OG 5020 C=C+1
```

```
HG 5Ø3Ø IF JN(APP)=Ø THEN JN(APP)=DT
        :RETURN
IF 5Ø4Ø IF JN(APP)>=DT THEN C=C-1
CH 5Ø5Ø APP=APP-1
OK 5Ø6Ø C=C+1
II 5Ø7Ø IF JN(APP)=Ø THEN 511Ø
MG 5Ø8Ø IF JN(APP)>=DT THEN C=C-1:GO
        TO 5Ø5Ø
NB 5Ø9Ø GOTO 5Ø5Ø
CK 51ØØ IF C<=1 THEN JN(APP)=DT:RETU
        RN
LB 51Ø5 REM **** Shifting other numb
        ers to make room for new num
        ber ****
HJ 511Ø D=1
AD 512Ø IF D=C THEN JN(APP)=DT:RETUR
        N
PG 513Ø JN(APP)=JN(APP+1)
OL 514Ø D=D+1
CG 515Ø APP=APP+1
MN 516Ø GOTO 512Ø
IA 6ØØØ REM **** Placing numbers gre
        ater than # presently at loc
        ation****
CC 6Ø1Ø APP=APP+1
OH 6Ø2Ø C=C+1
HH 6Ø3Ø IF JN(APP)=Ø THEN JN(APP)=DT
        :RETURN
EH 6Ø4Ø IF JN(APP)<DT THEN C=C-1
CG 6Ø5Ø APP=APP+1
OL 6Ø6Ø C=C+1
IK 6Ø7Ø IF JN(APP)=Ø THEN 611Ø
IJ 6Ø8Ø IF JN(APP)<DT THEN C=C-1:GOT
        O 6Ø5Ø
ND 6Ø9Ø GOTO 6Ø5Ø
CL 61ØØ IF C<=1 THEN JN(APP)=DT:RETU
        RN
LC 61Ø5 REM **** Shifting other numb
        ers to make room for new num
        ber ****
HK 611Ø D=1
AE 612Ø IF D=C THEN JN(APP)=DT:RETUR
        N
PJ 613Ø JN(APP)=JN(APP-1)
OM 614Ø D=D+1
CJ 615Ø APP=APP-1
MP 616Ø GOTO 612Ø
```

```
49188 DATA 1Ø5,193,157,145,193,165
                                :rem 167
49194 DATA 253,2Ø8,2,198,254,198 :rem 7Ø
492ØØ DATA 253,16Ø,3,24,189,125 :rem 249
492Ø6 DATA 193,1Ø1,253,157,125,193
                                :rem 15Ø
49212 DATA 189,145,193,1Ø1,254,157
                                :rem 155
49218 DATA 145,193,136,2Ø8,236,189
                                :rem 166
49224 DATA 85,193,133,8Ø,189,1Ø5 :rem 6Ø
4923Ø DATA 193,133,81,189,125,193:rem 1Ø8
49236 DATA 133,82,189,145,193,133:rem 111
49242 DATA 83,32,21,193,144,4   :rem 152
49248 DATA 2Ø2,2Ø8,228,96,165,82 :rem 64
49254 DATA 133,78,165,83,133,79  :rem 18
4926Ø DATA 16Ø,2,177,78,153,25Ø  :rem 2
49266 DATA Ø,136,16,248,48,11   :rem 158
49272 DATA 24,165,8Ø,1Ø5,3,133  :rem 2ØØ
49278 DATA 8Ø,144,2,23Ø,81,16Ø  :rem 2Ø4
49284 DATA 2,177,8Ø,153,247,Ø   :rem 16Ø
4929Ø DATA 136,16,248,32,32,193  :rem 4
49296 DATA 144,23Ø,56,165,82,233 :rem 59
493Ø2 DATA 3,133,82,176,2,198   :rem 158
493Ø8 DATA 83,32,21,193,176,31  :rem 2Ø8
49314 DATA 16Ø,2,177,82,153,247   :rem 3
4932Ø DATA Ø,136,16,248,32,32   :rem 145
49326 DATA 193,176,225,16Ø,2,177 :rem 58
49332 DATA 8Ø,145,82,185,247,Ø  :rem 21Ø
49338 DATA 145,8Ø,136,16,244,48  :rem 1Ø
49344 DATA 183,16Ø,2,177,8Ø,145   :rem 4
4935Ø DATA 78,185,25Ø,Ø,145,8Ø  :rem 2Ø9
49356 DATA 136,16,244,24,189,85  :rem 17
49362 DATA 193,125,125,193,133,82:rem 1Ø5
49368 DATA 189,1Ø5,193,125,145,193
                                :rem 168
49374 DATA 133,83,1Ø2,83,1Ø2,82 :rem 254
4938Ø DATA 32,21,193,176,22,189   :rem 7
49386 DATA 85,193,157,86,193,189 :rem 88
49392 DATA 1Ø5,193,157,1Ø6,193,32:rem 1Ø6
49398 DATA 53,193,232,32,69,193  :rem 2Ø
494Ø4 DATA 76,71,192,189,125,193 :rem 67
4941Ø DATA 157,126,193,189,145,193
                                :rem 164
49416 DATA 157,146,193,32,69,193 :rem 68
49422 DATA 232,32,53,193,76,71  :rem 2Ø9
49428 DATA 192,165,81,197,83,2Ø8 :rem 72
49434 DATA 4,165,8Ø,197,82,96   :rem 176
4944Ø DATA 16Ø,255,2ØØ,196,247,176
                                :rem 155
49446 DATA 11,196,25Ø,176,6,177  :rem 13
49452 DATA 248,2Ø9,251,24Ø,241,96:rem 1Ø7
49458 DATA 196,25Ø,96,24,165,8Ø  :rem 2Ø
49464 DATA 1Ø5,3,157,85,193,165  :rem 13
4947Ø DATA 81,1Ø5,Ø,157,1Ø5,193 :rem 253
49476 DATA 96,56,165,8Ø,233,3   :rem 173
49482 DATA 157,125,193,165,81,233:rem 112
49488 DATA Ø,157,145,193,96,256  :rem 23
```

Program 2:
Lightning Sort Loader For VIC

Refer to the "Automatic Proofreader" article before typing this program in.

```
5 HI=PEEK(56)-2:S=HI*256:S1=S     :rem 179
1Ø POKE 56,HI:POKE 55,Ø           :rem 231
2Ø READ A:IF A=256 THEN PRINT"TO RUN SORT
   , USE:{5 SPACES}SYS"S1:END     :rem 1Ø6
25 IF A<Ø THEN POKE S,ABS(A+2)+HI:S=S+1:G
   OTO 2Ø                         :rem 79
3Ø POKE S,A:S=S+1:GOTO 2Ø         :rem 16Ø
46Ø8 DATA 32,253,2Ø6,32,158,2Ø5   :rem 249
4614 DATA 32,247,215,165,2Ø,133   :rem 244
462Ø DATA 253,165,21,133,254,32   :rem 242
4626 DATA 253,2Ø6,32,158,2Ø5,162   :rem 45
4632 DATA 1,165,71,157,85,-3      :rem 1ØØ
4638 DATA 157,125,-3,165,72,157     :rem 2
4644 DATA 1Ø5,-3,157,145,-3,165   :rem 241
465Ø DATA 253,2Ø8,2,198,254,198    :rem 1Ø
4656 DATA 253,16Ø,3,24,189,125    :rem 2Ø7
4662 DATA -3,1Ø1,253,157,125,-3   :rem 233
4668 DATA 189,145,-3,1Ø1,254,157   :rem 52
4674 DATA 145,-3,136,2Ø8,236,189   :rem 54
468Ø DATA 85,-3,133,8Ø,189,1Ø5    :rem 2Ø4
4686 DATA -3,133,81,189,125,-3    :rem 2ØØ
4692 DATA 133,82,189,145,-3,133   :rem 255
4698 DATA 83,32,21,-3,144,4        :rem 49
47Ø4 DATA 2Ø2,2Ø8,228,96,165,82     :rem 4
471Ø DATA 133,78,165,83,133,79    :rem 214
4716 DATA 16Ø,2,177,78,153,25Ø    :rem 2Ø7
4722 DATA Ø,136,16,248,48,11       :rem 98
4728 DATA 24,165,8Ø,1Ø5,3,133     :rem 149
```

```
4734 DATA 80,144,2,230,81,160     :rem 144
4740 DATA 2,177,80,153,247,0      :rem 100
4746 DATA 136,16,248,32,32,-3     :rem 148
4752 DATA 144,230,56,165,82,233   :rem 255
4758 DATA 3,133,82,176,2,198      :rem 116
4764 DATA 83,32,21,-3,176,31       :rem 96
4770 DATA 160,2,177,82,153,247    :rem 208
4776 DATA 0,136,16,248,32,32      :rem 103
4782 DATA -3,176,225,160,2,177    :rem 202
4788 DATA 80,145,82,185,247,0     :rem 168
4794 DATA 145,80,136,16,244,48    :rem 215
4800 DATA 183,160,2,177,80,145    :rem 200
4806 DATA 78,185,250,0,145,80     :rem 158
4812 DATA 136,16,244,24,189,85    :rem 213
4818 DATA -3,125,125,-3,133,82    :rem 188
4824 DATA 189,105,-3,125,145,-3   :rem 242
4830 DATA 133,83,102,83,102,82    :rem 194
4836 DATA 32,21,-3,176,22,189     :rem 151
4842 DATA 85,-3,157,86,-3,189     :rem 162
4848 DATA 105,-3,157,106,-3,32    :rem 189
4854 DATA 53,-3,232,32,69,-3       :rem 94
4860 DATA 76,71,-2,189,125,-3     :rem 150
4866 DATA 157,126,-3,189,145,-3     :rem 0
4872 DATA 157,146,-3,32,69,-3     :rem 151
4878 DATA 232,32,53,-3,76,71      :rem 106
4884 DATA -2,165,81,197,83,208    :rem 216
4890 DATA 4,165,80,197,82,96      :rem 125
4896 DATA 160,255,200,196,247,176:rem 113
4902 DATA 11,196,250,176,6,177    :rem 209
4908 DATA 248,209,251,240,241,96   :rem 56
4914 DATA 196,250,96,24,165,80    :rem 216
4920 DATA 105,3,157,85,-3,165     :rem 148
4926 DATA 81,105,0,157,105,-3     :rem 141
4932 DATA 96,56,165,80,233,3      :rem 113
4938 DATA 157,125,-3,165,81,233     :rem 0
4944 DATA 0,157,145,-3,96,256     :rem 158
```

Program 3: Sort Test

```
100 PRINT"{CLR}"                     :rem 245
110 N=1000                           :rem 222
120 DIM AA$(N)                       :rem 178
130 PRINT"CREATING"N" RANDOM STRINGS"
                                      :rem 47
140 SD=-TI:A=RND(SD)                 :rem 183
150 FOR I=1 TO N                      :rem 37
160 PRINTI"{UP}"                      :rem 66
170 N1=INT(RND(1)*10+1)              :rem 221
180 A$=""                            :rem 127
190 FOR J=1 TO N1                     :rem 91
200 B$=CHR$(INT(RND(1)*26+65))        :rem 81
210 A$=A$+B$                          :rem 43
220 NEXT J                            :rem 29
230 AA$(I)=A$                        :rem 119
240 NEXT I                            :rem 30
250 PRINT"HIT ANY KEY TO START SORT"
                                     :rem 151
260 GET A$:IF A$="" THEN 260          :rem 83
270 PRINT"SORTING..."                 :rem 26
280 T1=TI                            :rem 249
291 REM SYS 49152,N,AA$(1) FOR 64:rem 163
292 REM USE SYS VALUE GENERATED BY THE LO
    ADER FOR VIC                     :rem 117
300 SYS 49152,N,AA$(I)               :rem 125
310 T2=TI                            :rem 244
320 PRINT"DONE"                      :rem 140
330 PRINT"HIT ANY KEY TO PRINT SORTED STR
    INGS"                             :rem 72
340 GET A$:IF A$="" THEN 340          :rem 81
350 FORI=1 TO N:PRINT I,AA$(I):NEXT
                                      :rem 28
360 PRINT:PRINT N" ELEMENTS SORTED IN"(T2
    -T1)/60"SECONDS"                 :rem 181
```

Programmer's Notes: PC And PCjr Version

Tim Victor, Editorial Programmer

The PC and PCjr version of "Lightning Sort" (Program 4) is based on the same algorithm as the version for Commodore computers, but runs in about one-third the time, due to the greater speed and power of the 8088 microprocessor used in the IBM computers. There are a couple of differences in the way that this program is loaded and used.

The BASIC loader program calculates a checksum from the DATA statements to help identify typing errors, then creates a disk file named "LSORT.BAS", containing the ML routine in binary form. The demonstration (Program 5) loads this file into memory using BLOAD and sets LSORT to the address of the sort routine. This variable is needed because IBM BASIC's CALL statement will only accept a variable name for the address of an ML routine.

Lightning Sort uses the first parameter in the CALL statement to find the array that it will sort. This is actually the address of the first string in the array, AA$(1) in the demonstration program, not the address of the array itself. The second parameter, N%, tells Lightning Sort how many strings are in the array. Variable names also have to be used for parameters, which is the reason for using N% instead of just plain 1000, and this version expects the length parameter to be an integer variable (a variable whose name ends with a percent sign).

Lightning Sort is loaded at address hex FF00 in BASIC's default segment. During a sort, the 256 bytes starting at hex FE00 are also used. To protect this memory, both programs start with the instruction CLEAR,&HFE00, which sets the top of BASIC's workspace to hex FE00.

Program 4: Lightning Sort Loader For PC/PCjr

```
100 CLEAR,&HFE00
110 ON ERROR GOTO 10000
120 DEF SEG
130 CHECKSUM = 0
140 ADDRESS = &HFF00
150 READ MLDATA
160 WHILE MLDATA <> -1
170 POKE ADDRESS,MLDATA
```

```
180 CHECKSUM = CHECKSUM + MLDATA
190 ADDRESS = ADDRESS + 1
200 READ MLDATA
210 WEND
220 IF CHECKSUM <> 22937 THEN ERROR 200
230 BSAVE "lsort",&HFF00,&HDC
240 END
1000 DATA 85,137,229,139,118,6,139,4
1010 DATA 72,185,3,0,247,225,139,86
1020 DATA 8,1,208,189,252,254,137,86
1030 DATA 2,137,70,0,252,41,192,80
1040 DATA 139,94,0,139,86,2,57,211
1050 DATA 127,3,233,129,0,135,211,232
1060 DATA 139,0,118,5,131,195,3,235
1070 DATA 246,135,211,57,211,126,31,131
1080 DATA 235,3,232,120,0,114,244,138
1090 DATA 15,139,71,1,135,211,134,15
1100 DATA 135,71,1,135,211,136,15,137
1110 DATA 71,1,135,211,235,214,139,118
1120 DATA 0,138,4,134,7,136,4,139
1130 DATA 68,1,135,71,1,137,68,1
1140 DATA 139,86,0,3,86,2,209,234
1150 DATA 57,218,114,23,139,70,2,131
1160 DATA 195,3,137,94,2,131,237,4
1170 DATA 131,235,6,137,94,0,137,70
1180 DATA 2,235,21,139,70,0,131,235
1190 DATA 3,137,94,0,131,237,4,131
1200 DATA 195,6,137,94,2,137,70,0
1210 DATA 88,64,80,233,114,255,88,72
1220 DATA 124,7,80,131,197,4,233,103
1230 DATA 255,93,202,4,0,139,118,0
1240 DATA 181,0,138,12,139,116,1,58
1250 DATA 15,118,2,138,15,139,127,1
1260 DATA 243,166,116,1,195,139,126,0
1270 DATA 138,13,58,15,195,-1
10000 IF ERR <> 200 THEN ON ERROR GOTO 0
10010 PRINT "Error in ML data: check for
 typo's"
10020 RESUME 240
```

Program 5: PC/PCjr Sorting Demonstration

```
10 CLEAR,&HFE00 : DEF SEG : CLS
20 BLOAD "lsort",&HFF00:LSORT=&HFF00
30 N%=1000
40 DIM AA$(N%)
50 LOCATE 2,16 : PRINT "Creating ";N%;"r
andom strings"
60 DEF SEG=&H40:RANDOMIZE PEEK(&H6C)
70 FOR I=1 TO N%:LOCATE 3,16:PRINT I
80 J%=RND(1)*10+1
90 A$="":FOR K=1 TO J%
100 A$=A$+CHR$(INT(RND(1)*26+65))
110 NEXT K
120 AA$(I)=A$
130 NEXT I
140 CLS:LOCATE 2,16:PRINT "Any key to st
art sort:"
150 A$="":WHILE A$="":A$=INKEY$:WEND
160 LOCATE 3,16:PRINT "sorting-  ";
170 SS=PEEK(&H6C)+256*PEEK(&H6D)
180 DEF SEG:CALL LSORT(AA$(1),N%)
190 DEF SEG=&H40:FS=PEEK(&H6C)+256*PEEK(
&H6D)
200 PRINT "done"
210 LOCATE 5,16:PRINT "Any key to print
sorted strings"
215 A$="":WHILE A$="":A$=INKEY$:WEND
220 FOR I=1 TO N%:PRINT AA$(I):NEXT
230 PRINT N%;"elements sorted in";(FS-SS
)/18;"seconds"
```

Notes For Apple Version Of Lightning Sort

Tim Victor, Editorial Programmer

The Apple version of "Lightning Sort," shown in Programs 6 and 7, requires an Apple II with at least 48K of random access memory and one disk drive. It has been tested on an Apple II Plus under DOS 3.3 and on an Apple IIc under ProDOS as well as DOS 3.3. The Applesoft demonstration program in Program 7 uses the BLOAD command to load the file LIGHTNING.SORT. This is a binary file containing the Lightning Sort program that is entered from Program 6 using the Apple II's built-in ML monitor.

Boot your computer, then type "CALL−151" to use the monitor. When you hit RETURN, the Applesoft input prompt will be replaced by an asterisk ("*"), the monitor's prompt. To enter a line from the listing, replace the hyphen after the first four-digit hexadecimal number with a colon. The first line in the listing would be entered as

9400: 20 B1 00 20 05 E1 A5 A0

Since no checksums are used in the listing, it's a good idea to make sure that the program in memory is correct. You can ask the monitor to display the contents of any memory location by typing its address as a hexadecimal number and hitting return. To examine a range of memory locations, type the address of the first location in the range, a period ("."), and then the address of the last location in the range. For example, Program 6 was made simply by entering "9400.9551" in response to the asterisk prompt.

When you're sure that the program is entered correctly, save it to disk using the BSAVE command. All DOS commands work in exactly the same way when entered from the monitor as when they are used in Applesoft. You can CATALOG, BLOAD, BSAVE, DELETE, and even LOAD and SAVE BASIC programs. To save the program you just entered, type "BSAVE LIGHTNING.SORT,A$9400,L$152" and hit RETURN. DOS will create a binary file named "LIGHTNING.SORT" and store in it $152 (338 in decimal notation) bytes beginning at memory location $9400 (decimal 37888).

Program 6: Lightning Sort For The Apple

```
9400- 20 B1 00 20 05 E1 A5 A0
9408- 85 FE A5 A1 85 FD 20 B1
```

```
9410- ØØ 2Ø E3 DF A2 Ø1 A5 83
9418- 9D 52 95 9D 7A 95 A5 84
942Ø- 9D 66 95 9D 8E 95 A5 FD
9428- DØ Ø2 C6 FE C6 FD AØ Ø3
943Ø- 18 BD 7A 95 65 FD 9D 7A
9438- 95 BD 8E 95 65 FE 9D 8E
944Ø- 95 88 DØ EC BD 52 95 85
9448- 1C BD 66 95 85 1D BD 7A
945Ø- 95 85 1E BD 8E 95 85 1F
9458- 2Ø 12 95 9Ø Ø4 CA DØ E4
946Ø- 6Ø A5 1E 85 1A A5 1F 85
9468- 1B AØ Ø2 B1 1A 99 FA ØØ
947Ø- 88 1Ø F8 3Ø ØB 18 A5 1C
9478- 69 Ø3 85 1C 9Ø Ø2 E6 1D
948Ø- AØ Ø2 B1 1C 99 ED ØØ 88
9488- 1Ø F8 2Ø 1D 95 9Ø E6 38
949Ø- A5 1E E9 Ø3 85 1E BØ Ø2
9498- C6 1F 2Ø 12 95 BØ 1F AØ
94AØ- Ø2 B1 1E 99 ED ØØ 88 1Ø
94A8- F8 2Ø 1D 95 BØ E1 AØ Ø2
94BØ- B1 1C 91 1E B9 ED ØØ 91
94B8- 1C 88 1Ø F4 3Ø B7 AØ Ø2
94CØ- B1 1C 91 1A B9 FA ØØ 91
94C8- 1C 88 1Ø F4 18 BD 52 95
94DØ- 7D 7A 95 85 1E BD 66 95
94D8- 7D 8E 95 85 1F 66 1F 66
94EØ- 1E 2Ø 12 95 BØ 16 BD 52
94E8- 95 9D 53 95 BD 66 95 9D
94FØ- 67 95 2Ø 32 95 E8 2Ø 42
94F8- 95 4C 44 94 BD 7A 95 9D
95ØØ- 7B 95 BD 8E 95 9D 8F 95
95Ø8- 2Ø 42 95 E8 2Ø 32 95 4C
951Ø- 44 94 A5 1D C5 1F DØ Ø4
9518- A5 1C C5 1E 6Ø AØ FF C8
952Ø- C4 ED BØ ØB C4 FA BØ Ø6
9528- B1 EE D1 FB FØ F1 6Ø C4
953Ø- FA 6Ø 18 A5 1C 69 Ø3 9D
9538- 52 95 A5 1D 69 ØØ 9D 66
954Ø- 95 6Ø 38 A5 1C E9 Ø3 9D
9548- 7A 95 A5 1D E9 ØØ 9D 8E
955Ø- 95 6Ø
```

Program 7: Lightning Sort Loader For The Apple

```
1Ø  HIMEM: 384ØØ: HOME : HTAB 8: PRINT
     "APPLE LIGHTNING SORT DEMO"
2Ø  HTAB 1Ø: PRINT "LOADING LIGHTNING.SORT"
3Ø  PRINT  CHR$ (4)"BLOAD LIGHTNING. SORT"
4Ø  HIMEM: 37887
5Ø N = 1ØØØ
6Ø  DIM AA$(N)
7Ø  HOME : PRINT "CREATING "N" RANDO
     M STRINGS"
8Ø  FOR I = 1 TO N
9Ø  VTAB 2: PRINT I
1ØØ N1 =  INT ( RND (1) * 1Ø + 1)
11Ø A$ = ""
12Ø  FOR J = 1 TO N1
13Ø B$ =  CHR$ ( INT ( RND (1) * 26 + 65))
14Ø A$ = A$ + B$
15Ø  NEXT J
16Ø AA$(I) = A$
17Ø  NEXT I
18Ø  PRINT "HIT ANY KEY TO START SORT"
19Ø  GET A$: IF A$ = "" THEN 19Ø
2ØØ  PRINT "SORTING..." CHR$ (7)
21Ø  CALL 37888,N,AA$(1)
22Ø  PRINT "DONE" CHR$ (7)
23Ø  PRINT "HIT ANY KEY TO PRINT SOR
     TED STRINGS"
24Ø  GET A$: IF A$ = "" THEN 24Ø
25Ø  FOR I = 1 TO N: PRINT I,AA$(I): NEXT ©
```

Glenn M. Kleiman

Aids For The Blind

Computers provide new and powerful aids for blind people. With special input and output devices and programs, computers enable blind people to more effectively substitute hearing and touch for sight and to use books, magazines, and newspapers that would otherwise be inaccessible to them. Computers can help blind people enjoy new opportunities for education, employment, social interaction, and recreation.

Much of this information about aids for the blind has been provided by the staff of the Sensory Aids Foundation of Palo Alto, California. They train blind people in job skills and help them find suitable jobs. They receive support from some of the major computer and electronics companies in Silicon Valley, and have placed workers at these companies. Other information has been provided by Telesensory Systems, Inc., the developers of Optacon and VersaBraille.

Computer Speech Synthesis

Speech synthesizers and text-to-speech conversion programs make it possible for computers to pronounce any word. The speech is not perfect, but people understand it easily after they get accustomed to it. During a visit to the Sensory Aids Foundation, I watched a demonstration of a talking terminal—a computer terminal combined with a speech synthesizer.

The blind user of the talking terminal has a control that lets him move a pointer to any line on the display screen. He can have the computer announce what line the pointer is on and speak the words on that line. He can have it repeat any words or read letter by letter. He can use the keyboard to edit the line.

Talking terminals make almost all the capabilities of a computer accessible to blind people. At Sensory Aids, blind people learn to use talking terminals for data entry, information retrieval, word processing, and programming.

Many educational programs could be used by blind people if the computer spoke what appears on the display screen. Staff members at Sensory Aids are revising some popular programs so that blind people can use them. During my visit, I saw a version of *MasterType* that was adapted for the blind. In the *MasterType* program, letters and words "attack" a central station. The user defends the station by typing the letters and words before they reach the station. In the adapted version of this program, the computer says the letters and words to be typed, and announces whether they have been typed correctly and quickly enough to defend the station.

Large Print Displays

Many people with impaired vision cannot read normal print, but can read large, high-contrast print. There are several ways to create large letters on the computer screen with standard equipment. One is to simply use a television or video monitor with a large display screen. Another is to use the computer's graphics capability to create large letters. In addition, many computer printers can produce large type on paper. With a suitable printer, any information stored in the computer can be printed in large letters.

A special large-print display processor, manufactured by Visualtek, magnifies letters on personal computer screens up to 16 times their

Dr. Glenn M. Kleiman is an educational psychologist and software developer. He is the author of Brave New Schools: How Computers Can Change Education *(Reston/Prentice-Hall) and the designer of* Square Pairs, *an educational game program (Scholastic, Inc.).*

usual size. A control panel lets the user set the scanning rate at which the letters move across the display screen.

Tactile Forms

Many people cannot see any letters, no matter how large. But these people can read when the letters are converted to a tactile form. One device which does that, Optacon, is already used by many blind people.

Optacon consists of a small camera, an electronics unit, and a stimulator array. The array is composed of 144 miniature rods. The electronics unit interprets the light pattern received by the camera and sends signals that cause certain rods to vibrate, thereby producing a tactile analogue to the light pattern. Some training is necessary to learn to read the vibrating patterns, but once this is mastered the blind person has access to all printed materials. Special adapters are available so that Optacon can be used to read computer screens and calculator displays.

Other devices use Braille, a system of writing in which each letter is represented by a pattern of raised dots in a 2 × 3 grid. Blind people read by feeling the dot patterns.

Although widely used, Braille has several disadvantages. Braille books are extremely bulky: A standard student dictionary fills a three-foot-square box. Braille typewriters are noisy and slow. Errors in Braille type cannot be corrected, since the raised dots cannot be erased. Braille books are therefore expensive, and most books, newspapers, and magazines are never made available in Braille.

Braille Word Processing

Special Braille printers can be interfaced to computers so that any information in the computer can be transformed to Braille. This provides a remedy for the problem of Braille not being correctable. A word processing program can be used to produce a Braille text after all corrections have been made on the computer screen.

Other Braille output devices can be interfaced to computers. One example is a device that contains sets of pins arranged in the 2 × 3 Braille grid. Each pin can be raised or lowered, thereby providing a mechanical Braille display. This device can be controlled by computer programs to produce instant Braille for a blind computer user.

A special device called VersaBraille incorporates recent advances in computer technology. VersaBraille is composed of a mechanical Braille display, a cassette information storage component, and a specially designed Braille keyboard, all under the control of a built-in computer. Information can be entered from the keyboard, revised and corrected (editing capabilities are built-in), stored on cassette, and transformed to Braille whenever needed.

VersaBraille provides a solution to the bulkiness of Braille. It is a self-contained unit that is easy to carry and can store 400 pages of Braille on a standard cassette tape.

A major advantage of VersaBraille is that it can be linked to a computer via a standard serial interface. A blind person can connect VersaBraille to a computer and quickly transfer information from the computer to VersaBraille's cassette storage system. The VersaBraille can then be taken away from the computer and read where and when convenient. A VersaBraille user can also take notes during class lectures, write reports, or enter any other information. He or she can then connect VersaBraille to a computer, transfer the information to the computer's memory, and use the computer to print the information, store it, or send it to others via an electronic mail system.

Computerized Letter Recognition

Speech synthesizers and text-to-speech programs can convert any words stored in a computer to speech. Other devices can convert information stored in a computer to large letter displays or to Braille or other tactile signals. However, much of the information people need is in books, not computers. To fully use the capability of computers to convert text to speech, Braille, or large print, we need efficient ways of transferring text from books to computers.

Special cameras and pattern recognition programs have been used for some time to recognize specially designed letters and numbers, such as the account numbers on checks. The camera converts the pattern of each letter into a binary code. A computer is programmed to process the binary code and determine which letter it represents.

In the last few years, devices and programs have been developed which make it possible for computers to recognize most typewritten characters and to adjust automatically for different type styles and sizes. In the next few years, this technology is likely to be perfected and become more widely available. (Only very limited success can be expected with handwritten letters, due to the large variations found in even one person's handwriting.)

Letter-recognition devices can be combined with appropriate output devices to produce large size displays, speech or Braille. Letter-recognition devices can also be combined with Braille printers to expedite the production of Braille books.

Converting Print To Speech

One impressive example of technology which serves the visually handicapped is the Kurzweil

Reading Machine that converts print to speech. This machine combines sophisticated pattern recognition, speech synthesis, and text-to-speech conversion capabilities. It lets blind users control how the material is read. They can set the speed of reading and adjust the tonality of the voice. They can stop the reading at any time, have the last few words or lines repeated, request the machine to spell out words or announce punctuation and capitalization, and mark certain words or phrases for later reference. This reading machine is currently a very expensive device. But as the technology advances and prices decrease, machines with these capabilities should become available to all blind people.

Technology For The Blind

Of 51 blind people who were assisted by the Sensory Aids Foundation during a one-year period, fifteen are now programmers, computer operators, or systems analysts. Other occupations include design engineer, word processor, medical transcriber, account clerk, attorney, cashier, clerk-typist, physicist, and college professor. Their employers include Apple, Hewlett-Packard, Pacific Telephone, Stanford Linear Accelerator, Department of Immigration, Internal Revenue Service, and other businesses, educational institutions, and government agencies.

Current technological aids include Optacon, VersaBraille, talking terminals, talking calculators, and closed circuit television systems that produce enlarged images of print on a television screen. These devices, and others now in development, can dramatically increase the opportunities available to blind people.

Kurzweil Computer Products, Inc.
33 Cambridge Parkway
Cambridge, MA 02142

Sensory Aids Foundation
399 Sherman Ave., Suite 12
Palo Alto, CA 94306

Telesensory Systems, Inc.
3408 Hillview Ave.
P.O. Box 10099
Palo Alto, CA 94303

Visualtek
1610 26th Street
Santa Monica, CA 90404

©

INSIGHT: Atari

Bill Wilkinson

Last month we discussed how to make programs designed for the Atari 400 and 800 load and run automatically on the new XL series without having to hold the option key down. We also looked at a way to make patches into Atari DOS 2.0s to enable it to work with the new enhanced density 1050 disk drive. The procedure is easy, but requires two disk drives. Just type in the source code (the portion printed last month and the continuation found in this issue) using an assembler capable of placing its object code directly in memory. Assemble it after LISTing or SAVEing the source code to disk. After assembling it once, change line number 1000 to read:

1000 .OPT NOLIST,OBJ

and assemble the code once more.

DOS should now be patched. Hit the SYSTEM RESET key and give the DOS command from your assembler. You should now be in the DOS menu (if you're not, something has gone wrong). Format a new disk using option I and then write the DOS files using option H. This will insure that everything is right and will give you a safe copy of your newly patched DOS.

The Tricky Part

There's one more step necessary to finish the procedure. Turn off your computer, put your BASIC (or BASIC XL) cartridge into your machine, and turn the power back on, thus booting the disk that was just formatted. Place a blank diskette into the 1050 drive that you are using as drive 2 and, from BASIC, type the following command:

XIO 254,#1,0,34,"D2:"

Drive 2 should now contain an enhanced-density diskette. Now hit the SYSTEM RESET key so that DOS will recognize the new density. Finally, go into DOS and write the DOS files to the new diskette (D2), using option H from the menu.

If everything has been done properly, drive 2 should now have an enhanced-density diskette containing the patched DOS. Once you have this master completed, creating others is simple and can be done with the I and H options in the DOS menu.

Patches To Atari DOS 2.0s

```
                1350 ;
                1360 ; BEGIN THE ACTUAL PATCHES
                1370 ;
                1380 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                1390 ;
                1400 ; This patch allows either $21 or $22 as
                1410 ;   a format command.
                1420 ;
0000            1430     *=  $07B5
07B5            1440 PATCHFORMAT
07B5 A920       1450     LDA #$20     ; format cmds are $21 or $22
07B7 2D0203     1460     AND DCBCMD   ; is this a format cmd?
07BA D002       1470     BNE $07BE    ; bit $20 is set, so yes...read
                1480 ;
                1490 ; This patch modifies the drive type
                1500 ;   reported by DINIT for use in DRVTBL
                1510 ;
07BC            1520     *=  $0819
0819            1530 PATCHINIT
0819 ADEA02     1540     LDA $02EA    ; get drive status
081C 0A         1550     ASL A        ; and this sequence...
081D 08         1560     PHP          ; ...will serve to
081E 0A         1570     ASL A        ; ...convert the status
081F 2A         1580     ROL A        ; ...$00, $20, and $80
0820 2A         1590     ROL A        ; ...to the more usable
0821 2A         1600     ROL A        ; ...$00, $01, and $80
0822 28         1610     PLP          ; (more usable because what we
0823 6A         1620     ROR A        ; want are $01,$02, and $81)
```

```
                    ;
                    ;
                    ; This patch allows SETUP to call the main
                    ;    patch-it-all-up routine
                    ;
Ø824                     *=  $1184
                    ; patch to SETUP code
                    ;
1184 BE1113              LDX DRVTBL,Y
1187 2ØØ115              JSR PATCHSETUP ; the real work
118A A8                  TAY
118B FØ4E                BEQ DERR1
                    ;
                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                    ;
                    ; The major patch:
                    ;    Here we determine the type of drive for
                    ;    the current operation and patch various
                    ;    locations (including LDA # instructions)
                    ;
118D                     *=  $15Ø1
15Ø1                PATCHSETUP
15Ø1 8E7213              STX TRUETYPE ; save true drive type
15Ø4 E8                  INX          ; convert Ø or 1 to 1 or 2
15Ø5 8A                  TXA
15Ø6 29Ø3                AND #$Ø3     ; mask off 1Ø5Ø bit
15Ø8 8DFE12              STA DRVTYP   ; ...and save it
15ØB 48                  PHA          ; and keep it for later return
                         ;           now we set up the tricky stuff
                    ;
                    ; we need different VTOC bases and sizes
                    ;    and different disk sizes
                    ;
15ØC AØØA                LDY #$ØA     ; 81Ø: start of vtoc
15ØE A964                LDA #9Ø+$ØA  ; 81Ø: end of vtoc bytes
151Ø A221                LDX #DCBCFD  ; 81Ø: format command
1512 2C7213              BIT TRUETYPE ; test drive type
1515 1ØØ5                BPL SGLDBLJOIN ; 81Ø, all ok
1517 AØØ6                LDY #$Ø6     ; 1Ø5Ø: start of vtoc
1519 A98Ø                LDA #122+$Ø6 ; 1Ø5Ø: end of vtoc bytes
151B E8                  INX          ; 1Ø5Ø: format command is '"'
                    ;
                    ; now store these values into code (shudder!)
                    ;
151C                SGLDBLJOIN
                    ;
151C 8E23ØD              STX $ØD23    ; where format command is loaded
                    ;
                    ; the various uses of START-OF-VTOC
                    ;
                    ;
151F 8C8ØØD              STY $ØD8Ø    ; in deallocation of boot
1522 8CEE1Ø              STY $1ØEE    ; in FRESECT
1525 8C4211              STY $1142    ; in GETSECTOR, displacement
1528 88                  DEY
1529 8CØ711              STY $11Ø7    ; at start of GETSECTOR
                    ;
                    ; and the uses of END-OF-VTOC
                    ;
152C 8DØA11              STA $11ØA    ; check end in GETSECTOR
152F 8D7AØD              STA $ØD7A    ; a CPY in format code
1532 98                  TYA
```

```
1533 18          2250      CLC
1534 692E        2260      ADC #46     ; adjust for ...
1536 8D84ØD      227Ø      STA $ØD84   ; the directory dealloc in fmt
1539 AEØ113      228Ø      LDX CURFCB  ; recover original value
153C 68          229Ø      PLA
153D 6Ø          23ØØ      RTS
                 231Ø ;
                 232Ø ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 233Ø ;
                 234Ø ;
                 235Ø ; This is another major patch...
                 236Ø ;    instead of using set values for VTOC
                 237Ø ;    info, we pick from one of two tables
                 238Ø ;
                 239Ø ;
153E             24ØØ      *=  $ØD37
ØD37             241Ø PATCHXFORMAT
ØD37 1Ø27        242Ø      BPL XFØ     ; same source, but XFØ has moved
                 243Ø ;
                 244Ø ;
ØD39             245Ø      *=  $ØD52
ØD52             246Ø TBL81Ø
ØD52 Ø2          247Ø      .BYTE 2
ØD53 C3Ø2        248Ø      .WORD 7Ø7,7Ø7
ØD55 C3Ø2
ØD57 ØØFF        249Ø      .BYTE Ø,$FF
ØD59             25ØØ TBL1Ø5Ø
ØD59 Ø2          251Ø      .BYTE 2
ØD5A C5Ø3        252Ø      .WORD 965,965
ØD5C C5Ø3
ØD5E ØØFF        253Ø      .BYTE Ø,$FF
                 254Ø ;
                 255Ø ;
                 256Ø ; we have moved the label xfØ
                 257Ø ; ...to make room for the tables
                 258Ø ;
ØD6Ø             259Ø XFØ
ØD6Ø AØØØ        26ØØ      LDY #Ø
ØD62 B952ØD      261Ø XFØ1 LDA TBL81Ø,Y
ØD65 2C7213      262Ø      BIT TRUETYPE
ØD68 1ØØ3        263Ø      BPL TYPEOK
ØD6A B959ØD      264Ø      LDA TBL1Ø5Ø,Y
ØD6D             265Ø TYPEOK
ØD6D 9145        266Ø      STA (ZDRVA),Y
ØD6F C8          267Ø      INY
ØD7Ø CØØ7        268Ø      CPY #7
ØD72 DØEE        269Ø      BNE XFØ1
                 27ØØ ;
ØD74             271Ø XFØ2
ØD74 9145        272Ø      STA (ZDRVA),Y
ØD76 C8          273Ø      INY
ØD77 1ØFB        274Ø      BPL XFØ2
ØD79 EA          275Ø      NOP
ØD7A EA          276Ø      NOP
ØD7B EA          277Ø      NOP
ØD7C EA          278Ø      NOP
                 279Ø ;
                 28ØØ ; This patch allows the user to choose
                 281Ø ;    diskette type for formatting via
                 282Ø ;    the 'XIO 254' command
                 283Ø ;
ØD7D             284Ø      *=  XFV     ; Where the format vector is
```

```
ØBD6 4C7313         JMP XFVPATCH
               ;
               ; The label 'Z' designates some unused
               ;    memory in the original DOS 2.Øs
               ;
ØBD9                *=  Z
1372 ØØ        TRUETYPE .BYTE Ø ; Where PATCHSETUP saves true disk type
               ;
               ; This code becomes the beginning of
               ;    the FORMAT code
               ;
1373           XFVPATCH
1373 BD4BØ3         LDA ICAUX2,X ; Get AUX2 value
1376 FØØ3           BEQ XFVP2    ; zero...don't do anything
1378 8D23ØD         STA $ØD23    ; non-zero...assume it is type of format
137B           XFVP2
137B 4C18ØD         JMP XFORMAT
               ;
               ;
               ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
               ;
               ; end of patches for 1Ø5Ø drive
               ;
               ;
               ; BEGIN patches for BURST I/O
               ;
               ; from COMPUTE!, July, 1982
               ;
               ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
               ;
137E                *=  $ØA1F
               ;
               ; first, patch the code where WTBUR used to be
               ;
ØA1F           WTBUR
ØA1F           BURSTIO
ØA1F BD8213         LDA FCBOTC,X ; open type code
ØA22 49ØC           EOR #$ØC     ; check for mode 12 (update)
ØA24 FØ24           BEQ NOBURST
ØA26 6A             ROR A        ; move carry to MSB of A-reg
ØA27 EA             NOP          ; filler only
ØA28           TBURST
               ;
               ; ... and STA BURTYP remains...but
               ;     BURTYP is negative if BURSTIO was
               ;     called from GET-BYTE and positive
               ;     if it was called from PUT-BYTE
               ;
ØA28                *=  $ØA41
               ; so we must patch here to count for the sense
               ;    of BURTYP being inverted from original
               ;
ØA41 1ØØ9           BPL WRBUR
               ;
ØA43                *=  $ØAD4
               ;
               ; finally, we must patch the GET-BYTE call
               ;    so that it JSR's to new location
               ;
ØAD4 2Ø1FØA         JSR BURSTIO
               ;
ØAD7                .END
```

©

Commodore Autoboot

David W. Martin

This utility makes loading and running programs quick and easy, and can also be used as a form of copy protection. For the VIC-20 and Commodore 64 with a disk drive.

Have you ever wondered how some commercial programs run automatically after they're loaded? "Autoboot" enables you to add this convenient feature to your own programs.

Type in and SAVE Autoboot. VIC users should substitute the following for lines 481 and 491 before saving:

```
481 DATA 165,175,133,46,165,174,133,45,32
    ,89,198,32                      :rem 234
491 DATA 142,198,76,174,199         :rem 77
```

To use Autoboot, first load the BASIC program that you want to make bootable. Then enter POKE 43,0:POKE 44,1 and SAVE the program using a different filename. This version of the program will be used by Autoboot. Now load and run Autoboot and enter the name of the modified version when prompted. Autoboot will then turn it into an autoboot program by directly changing certain disk sectors. The sector numbers are displayed on the screen as Autoboot runs.

Since the VIC and 64 automatically relocate programs when loading, all autobooted programs must be loaded using a nonrelocatable load as follows:

LOAD *"filename"*,8,1

Of course, any BASIC program can be made to load and run from disk just by typing:

LOAD *"filename"*,8:

and pressing SHIFT–RUN/STOP instead of RETURN. But the power of Autoboot lies in the copy protection it provides. To copy protect your autorun programs, add POKE 808,100 (VIC), or POKE 808,234 (64) as the first line in your program before saving the modified version to be used by Autoboot. This will disable the RUN/STOP key, the RESTORE key, and the LIST command as soon as the program runs. Since the autobooted program will run as soon as it's loaded, the user won't be able to break out of the program to SAVE it.

Autoboot

Refer to the "Automatic Proofreader" article before typing this program in.

```
100 PRINT"{CLR} AUTOBOOT ":T=18:S=1:D$="Ø
    ":OPEN15,8,15,"I"+D$            :rem 248
110 OPEN2,8,2,"#"+"Ø"               :rem 234
120 REM **** LOCATE TARGET          :rem 158
130 INPUT"FILENAME";NA$:LN=LEN(NA$)
                                    :rem 139
140 GOSUB210:GOSUB300               :rem 245
150 IFT=ØTHENPRINTNA$" NOT FOUND":GOTO540
                                    :rem 18
160 GOTO140                         :rem 101
170 GOTO540                         :rem 106
180 REM ***POINT TO BYTE AND GET IT INTO
    {SPACE}X.                       :rem 108
190 PRINT#15,"B-P:"2,L:GET#2,A$:IFA$=""TH
    ENA$=CHR$(Ø)                    :rem 197
200 X=ASC(A$):RETURN                :rem 206
210 PRINT"TRACK"T" SECTOR"S         :rem 148
220 PRINT#15,"U1:"2;D$;T;S          :rem 204
230 L=Ø:GOSUB180:T=X:L=1:GOSUB180:S=X:RET
    URN                             :rem 71
240 REM *** CHECK FOR FULL MATCH :rem 221
250 FORJ=ITOI+LN:L=J:GOSUB180:IFX=ØORX=16
    ØTHEN27Ø                        :rem 130
260 X$=X$+CHR$(X):NEXTJ             :rem 101
270 IFX$<>NA$THENX$="":RETURN       :rem 23
280 L=I-2:GOSUB180:TT=X:L=I-1:GOSUB180:SS
    =X:PRINT                        :rem 142
290 GOTO340                         :rem 107
300 REM *** CHECK THROUGH ONE BLOCK FOR N
    AME MATCH                       :rem 54
310 FORI=5TO23ØSTEP32               :rem 15
320 L=I:GOSUB180:IFCHR$(X)=LEFT$(NA$,1)TH
    ENGOSUB240                      :rem 95
```

```
330 NEXTI:RETURN                    :rem 56
340 REM *** ACCESS 1ST SECTOR OF TARGET P
    ROGRAM                          :rem 199
350 T=TT:S=SS:GOSUB210              :rem 142
360 L=2:GOSUB180:AL=X:L=3:GOSUB180:AH=X:S
    A=AL+AH*256                     :rem 183
370 IFSA<>256THENPRINT:PRINTNA$" IS NOT P
    REPARED FOR AUTOBOOT":GOTO540:rem 142
380 REM *** ESTABLISH FALSE STACK:rem 125
400 PRINT#15,"U1:"2;DR;TT;SS:PRINT
                                    :rem 104
410 FORPB=173TO254STEP2:PRINT#15,"B-P:"2;
    PB                              :rem 74
420 PRINT#2,CHR$(96);               :rem 160
430 PRINT#15,"B-P:"2;PB+1           :rem 113
440 PRINT#2,CHR$(3);:PRINT"*";:NEXT:PRINT
                                    :rem 22
450 PRINT#15,"U2:"2;DR;TT;SS        :rem 167
460 GOSUB210:PRINT                  :rem 116
470 REM ***PUT AUTOBOOT CODE ONTO PAGE 3
                                    :rem 14
481 DATA165,175,133,46,165,174,133,45,32,
    89,166,32                       :rem 229
491 DATA 142,166,76,174,167         :rem 67
500 PRINT#15,"U1:"2;DR;T;S          :rem 251
510 FORPB=105TO121:READBY:PRINT#15,"B-P:"
    2;PB                            :rem 194
520 PRINT#2,CHR$(BY);:PRINT". ";:NEXT:PRI
    NT:PRINTNA$" CAN NOW BOOT ITSELF"
                                    :rem 0
530 PRINT#15,"U2:"2;DR;T;S          :rem 255
540 CLOSE2:CLOSE15                  :rem 87
```
©

Atari Paddle Fixer

William Griner

Here's a quick fix for the Atari paddle jitters that still preserves the paddles' range.

The Atari paddles are so sensitive that the heat of a hand or any jarring can change their value. Some paddle-based games don't take the sensitivity into account, causing their characters to flicker annoyingly. Try this:

```
KN 1000 REM get the paddle value
IG 1010 PV=PADDLE(PN): IF ABS(PV-OPV
        )>1 THEN OPV=PV: RETURN
LF 1020 PV=OPV: RETURN
```

where:

PN is the paddle number (0–7)

PV is the value read from the paddle

OPV is the old paddle value (initialized to whatever value you wish)

Centered Values

The above subroutine keeps the paddle centered between the adjacent values. It takes a difference of two steps or more to change the paddle value. This is not to say that the paddle will use only all even or all odd values. For example, if the paddle is at value 77, it will not be allowed to move directly to 76 or 78. If you want to move from 77 to 78, you will have to move to 80 or 75, then to 78.

Better Than Brackets

This method is better than dividing the paddle range by a number since doing so creates fixed brackets of possible values and does nothing to keep the paddle value from straddling the bracket boundaries. This method could also be used to keep the paddle in a wide bracket, allowing only for coarse movement, yet giving access to the entire range of the paddle's values. ©

Apple Editing Hints

Patrick Moyer

Most computer owners develop a love-hate relationship with at least one feature of their machines. For Apple owners, this feature is often the editing functions. Here is a review of Apple editing controls and protocols and some tips on making the process easier and more effective.

The Apple uses a combination of screen editing and line editing. Changes are made by moving the cursor to a particular line which has been listed on the screen and retyping that line. This retyping is usually accomplished with the right arrow key. As the right arrow is pressed, the cursor moves to the right, reentering all it passes over. A change is made by typing over what is already there, or by inserting the correction through a combination of cursor moves.

Physical, Logical

Therefore, to make a change, we must specify the line to be changed. In this case, we are talking about a line of BASIC, not a line displayed on the screen. The BASIC line is called a *logical line*, as opposed to the *physical line* that is displayed on the screen. A logical line may contain multiple BASIC commands and may be up to 255 characters long. The physical display line is the 40-letter width of the screen.

Before a BASIC line can be changed, it must be listed. It is best to clear the screen with the HOME command initially. This eliminates confusion about what was changed and what wasn't.

When a line is listed, the computer puts one space between words or variables, two spaces after the line number, seven spaces at the end of the first physical line, and five spaces on the right and left sides of the remaining physical lines.

Most of the time, these extra spaces and lines are of little consequence. One can just merrily right-arrow over them with no harm. The one exception occurs in string information (characters in quotes). This causes a problem. If a string is broken between two or more physical lines during the listing process, and you right-arrow to retype, 12 additional spaces will be inserted between the last character on the first line and the first character on the next line. Certainly not what's wanted. The common solution is to avoid the right arrow and use the cursor with the <ESC>K sequence instead.

Simplified Cursor Control

There's an even simpler solution. Let's edit a line step by step to demonstrate this technique (<ESC> is the ESC KEY, <RET> is the RETURN KEY):

Here's the line as originally typed:

```
10PRINT"THIS IS A LONG LINE OF STRING
DATA"<RET>
```

List the line. It looks like this:

```
LIST10<RET>
10 PRINT "THIS IS A LONG LINE OF STR
     ING DATA"
```

We then type <ESC>I, repeating the I key until the cursor is over the second digit of the line number; J is pressed to move the cursor one space to the left. (This J keypress is important. If you forget it and continue the editing process, you will gain a line in your program. Line 0 will be created, but more about that later.)

Once you've moved left, leave <ESC> mode. This is done by pressing any key not having meaning in <ESC> mode. Because some keys not normally used for cursor movement do have special meaning, it's best to press the space bar. Remember, this will not move the cursor.

We can now use the right arrow to "retype" the line to the place of the change. The repeat key can be used to speed this process. Let's say you've used the right arrow until it appears after the last quote. The line on the screen looks no different. However, if we LIST the line, we now see this:

```
10 PRINT "THIS IS A LONG LINE OF STR
     ING DATA"
```

If we type RUN we get:

```
RUN<RET>
THIS IS A LONG LINE OF STR     ING DAT
A
```

Eliminating Problem Margins

The common solution, again, is to right-arrow to the R in STR, then type <ESC> and press K repeatedly to move the cursor until you reach the I in ING. Anyone who has done this often will know how easy it is to forget <ESC> K, and end up with a string of K's.

The solution is simply to eliminate those extra margins unless you need them. Let's start with the same original line:

```
10PRINT"THIS IS A LONG LINE OF STRING
DATA"<RET>
```

To edit the line we type:

```
HOME:POKE33,30:LIST10<RET>
```

The HOME gives us a clean screen to work with; the LIST puts the line to be edited on the screen. A POKE instruction places a single number into an "address" in the computer's memory. Address 33 controls the width of the screen display. Placing the number 30 in it reduces the size of the screen to 30 characters wide rather than 40.

Caution: The POKE must be done before the LIST for this method to work. The HOME is optional, but prevents a very confusing screen. (Try it. You'll see what I mean.) The screen will erase and display:

```
10  PRINT"THIS IS A LONG LINE OF S
TRING DATA"
```

As you can see, the line is 30 characters wide without the extra margin spaces. Move the cursor to the line number as usual. The right arrow may be used without ill effect. It will go directly from the S on the first display line to the T on the second line without inserting any blanks. This eliminates the need to use the <ESC> K sequence.

Once you have finished editing, you will need to type TEXT. This command will return you to normal 40-character screen mode.

Duplicating Lines

One strength of Apple II editing is the ability to duplicate lines. Let's try an example:

```
HOME: POKE33,30:LIST10<RET>
10  PRINT"THIS IS A LINE TO BE
DUPLICATED"
```

Next move the cursor up to the line using the normal <ESC>I. When the cursor arrives over the number, move it left until it is over the first digit of the number. Then press the space bar as before; but prior to using the right arrow, retype the line number, say, 20. Then use the right arrow to "retype" the line as described above until you reach the end of the logical line. At this point, press RETURN. If you LIST the program, you'll see:

```
HOME:POKE33,30:LIST<ret>
10  PRINT"THIS IS A LINE TO BE
DUPLICATED"
20  PRINT"THIS IS A LINE TO BE
DUPLICATED"
```

Once you have moved your cursor up to the number and changed it, you do not have to reuse the entire line. You can treat it like any line to be edited further if necessary.

Easy Program Merge

This technique can also be used on a limited scale to merge two programs. Let's say you have a favorite subroutine of three or four lines which you wish to add to a program. You could use the merge function of the Renumber program on the Systems Master, or the program that is part of the Programmer's Toolkit. If you don't have these programs or you don't have them handy, here is a simple procedure:

1. Save the program you are working on.
2. Load the program which contains the lines to be copied to your new program.
3. Clear the screen, change width, and list lines (using HOME:POKE33,30:LIST statements).
4. Now, load the program the lines are to be added to.
5. Using the normal <ESC> and right-arrow commands, edit each line without changes. It's best to edit the last line first and work up the screen, entering each line one at a time. This is because when multiple lines are listed and edited, once <RET> is pressed, the line number below it is partially destroyed and has to be retyped by hand. There's nothing wrong with changing the line numbers to fit your new program if the current line numbers are a problem.
6. Once all lines are edited, save the program. If you list it, you'll find the lines are now part of your program.

Finally, if you want to cancel a particular change as long as you have not pressed <RET> yet, cancel the editing of the line by typing <CTL> X. Be sure that you press the <CTL> key first, then X. The machine will answer with a backward slash. If you list the line, it will be unchanged. ©

Jim Butterfield, Associate Editor

Math And Tables

I'm frequently asked for addresses within ROM that do certain operations, usually mathematical functions. I do my best to talk programmers out of this approach if possible.

For one thing, the addresses of the ROM routines vary from machine to machine. I'd prefer to see a programmer borrow the code from the ROMs and include it in the program. At least that way, transportability is not a problem.

Using ROM math routines is often awkward. They often call for one or more values to be placed into floating point accumulators before calling, and return values in the same areas. A floating point number is often an inconvenient format and takes a fair-sized conversion routine to bring back to the more convenient "fixed point" notation used by most machine language programmers. The total effort can turn out to be greater than programming it yourself.

But the main reason that I try to discourage use of these routines is this: They are designed for a certain number of digits of accuracy, and your program usually wants either greater or less accuracy. If you need less, you're wasting processor time working out the extra places. If you need more, the built-in routine will not do the job for you.

A Question

I was recently asked by a user to supply the address of the logarithm routine within a certain computer. It would have been easy to just answer the question, but I balked. I asked the user to define his objective.

This makes an interesting case history, since the objectives were changed partway through the exercise. We have a chance to see a couple of approaches to avoiding the built-in routines.

My first thought was to replace the ROM log routine with a streamlined machine language version. There are several efficient ways of calculating a logarithm; any book on numerical analysis (or an encyclopedia) will supply information on this.

First Approach

After questioning the user closely, the objective appeared to be this: An eight-bit reading was being taken from a remote device. He desired to convert this reading to a base ten logarithm (with appropriate scaling) for display purposes, and the accuracy of the result was to be 16 bits.

My concept of the approach changed. The magic words, "eight bits," had been spoken. The objectives were still a bit fuzzy, since it's hard to get a full 16 bits of useful data when your original data was only 8 bits accurate; but not to worry on that score for the moment.

Here's the pitch: If you have an eight-bit value to work through any mathematical function, use a table. There are only 256 possible values to be worked out, 256 questions and 256 corresponding answers.

We'll need to have two tables—one for the low part of the answer and one for the high part—but that's no problem: 512 bytes of storage is usually not hard to come by.

Looking up things in a table of 256 values is the ultimate in simplicity. It's sometimes called a "list type lookup," and the principle is very simple. Put the original value into an index register, and read out the indexed answer. Our code might read something like the following:

```
LDX —input register—
LDA LOWTABLE,X
STA LOWRESULT
LDA HIGHTABLE,X
STA HIGHRESULT
```

No loops, no math, no complexity: Five instructions and it's done. We must be sure to prepare the table in advance, but that's a one-shot task. In fact, BASIC could do the job for us and POKE the values into the table.

Second Approach

When the requirement was examined more closely, the rules changed and the problem was inverted: Given a 16-bit reading, compute the base ten logarithm to 8 bits of accuracy. The eight bits, by the way, were to be used to draw a high-resolution graph; 256 points were quite sufficient for the resolution required.

This requirement makes a little more sense: Converting 16 bits into 8 involves a loss of accuracy, but that was compatible with the display objective.

We still have the magic words "eight bits" embedded in the problem, but this time they describe the result. We can still use our table approach if we invert the way we use the table.

Let's build our table this way: For each of the 256 entries, we'll put the corresponding "anti logarithm" in the table. When we search the table to find the closest match to our original value, the answer will turn out to be the number of the table entry.

An example might illustrate what I mean here. Suppose the 16-bit input number has a value of 2000. The desired result, allowing for the scale, will be 165. In slot 165 of the tables (high and low), I'll find a value that's quite close to 2000. My task: search the table to find the closest value.

Binary Splitting

This isn't hard to do. Most of us have learned to search a table by using a "binary split" method, splitting the table in half again and again until we find the value we want. And on a table of size 256, a computer can do a very efficient job of binary splitting. Eight comparisons and it's all over.

The code would follow these lines:

```
LDA  #$80
STA  MASK
```

This says, "we're going to split the table into chunks of 128 (hex 80) this time around."

```
LDX  #$00
STX  POINTER
```

We'll kick off starting at position zero in the table. Here comes the loop:

```
LOOP   LDA  POINTER
       ORA  MASK
       TAX
```

We've added our offset of 128 to the starting position of zero, so our first comparison will be at the midpoint of the 256 table.

```
COMPARE .....
```

Let's fudge the COMPARE coding for the moment. We'll need to load our high and low bytes into A, compare to the table high and low (indexed, of course) and decide whether our value is higher or lower than the table entry. If our value is LOW, we'll branch ahead to LOW; otherwise, we continue with HIGH:

```
HIGH   STX  POINTER
```

If our value is high, we store the index. If not, we skip this instruction and continue with the old value in POINTER.

```
LOW    LSR  MASK
```

Our mask contained 128, the size of the "split." Now we are dividing it by two so that it becomes 64, and 32 the next time, followed by 16, and so on. Eventually, we'll end up with zero as the bit rolls out of the end of the byte.

```
       BNE  LOOP
```

We go back to do another comparison. Let's see what has happened. POINTER started at zero. If our input value is lower than table item 128, POINTER will stay at zero and the next comparison will be with item 64. On the other hand, if our input value is higher than table entry 128, POINTER will be changed to 128, and the next comparison will be with item 192. In other words, we'll split the upper half or the lower half depending on how the previous comparison went.

It's not hard to see how the program zeros in on the answer after eight comparisons. Finally, MASK becomes zero, the program stops looping, and the answer may be found in POINTER.

The user started out looking for a logarithm routine in ROM, and ended up with something much better: faster, more compact, and well-suited to the application.

And there was a free bonus. After looking at this approach, the user discovered that he could do something he had previously thought impractical: switch to a new display scale—linear, split scale, or whatever—with no difficulty. It was just a matter of turning the tables. ©

Commodore Disk Pattern Matching

Part 1

Jim Butterfield, Associate Editor

*The flexible Commodore DOS allows the user to LOAD, Scratch, and obtain a directory of files using the symbols * and ? as pattern matchers. The quirks of these two symbols can, however, cause problems. For one thing, you might accidentally erase an entire diskette.*

Commodore disk drives are versatile; sometimes we don't realize how versatile they are. In this article, we'll discuss *pattern matching*: how it works, and how to use it to get rid of an annoying "comma" file that sometimes appears on your disk directory.

First, a recommendation: Unless you have 4.0 BASIC (in the PET/CBM series of computers), learn how to use the *Wedge* or *DOS Wedge* utility program. It's a great convenience. We'll refer to wedge commands within this article. The DOS Wedge has many handy features, but the two most important are these: You can find out about a disk error at any time by typing the @ key followed by a RETURN; and you can examine a disk directory without disturbing the program within your computer's memory by typing @$ followed by RETURN.

Pattern Matching

It's possible to identify one or more programs on disk without specifying their full names. Match the missing part of the filename by using a pattern. The two characters used for this are:

? – to match any single character;
* – to match any following characters.

If I have two files, one named DIG and the other, DOG, I can specify a name which matches both files with D?G—the question mark matches any character. If I have files named HOUSE, HO, HOTDOG, and HORRIBLE, I can match them all with HO*—the asterisk matches any group of characters, including no character.

This is good if you can't remember a filename exactly. If you have a file that might be called CATFOOD or might be called CAT FOOD, but you can't remember which, you can load it regardless of name with LOAD "CAT*",8. The first file whose name begins with CAT will be loaded. Unfortunately, you might discover that instead of the program you wanted, you have loaded something else, such as CATCH-MICE. The first name in the directory that matches will be the one loaded.

We can use pattern matching to get around this problem. If you load the directory using pattern matching, you'll see all programs that fit the pattern. To examine CAT programs, type:

```
LOAD "$0:CAT*",8
```

or, with the wedge program:

```
@$0:CAT*
```

You'll see a list of all programs (if any) whose names begin with the characters CAT, which allows you to select the one you want.

Command Variations

Note that LOAD picks the first program that matches, but the directory picks all programs that match.

It's probably obvious that SAVE must not allow pattern matching. You must save a real name, not an approximation. Thus, SAVE "CAT*",8 will produce a syntax error from the disk.

The Scratch command does accept pattern matching; all files that match will be removed from the disk. Use pattern matching with great care when using Scratch; you could remove more files than you planned.

To scratch all files from a disk that begin with the letter M, you would type the following:

```
OPEN 15,8,15
PRINT#15,"S0:M*"
```

or, using the wedge:

```
@S0:M*
```

Be careful. There might be more files starting with M than you expected. Take a directory listing first (using pattern matching, of course).

Here's another example. Suppose you've been writing a BASIC program called DIS. As you write code, you save the program from time to time, creating DIS1 and DIS2. Then you start testing and correcting, saving new versions as you go, and create DIS3, DIS4, and DIS5. Finally, you're satisfied, and you save your final version as DISK/EDIT. How can you get rid of your five development programs, named DIS1 to DIS5? Easy. Scratch pattern DIS? and they will all go. DISK/EDIT will stay, since the ? character matches only a single character. Do not scratch pattern DIS* since that would definitely clobber DISK/EDIT.

But be careful. Just before you give the command to scratch pattern DIS?, take a directory with the same pattern. You might have other files called DISK or DISH that match the same pattern. So you might code:

```
LOAD "$0:DIS?"
LIST
```

or, with the wedge:

```
@$0:DIS?
```

You'll see the programs that match the name pattern. If they are exactly the ones you want, type the Scratch command; or with the wedge, you can go back and type over the dollar sign with the letter S; pressing RETURN will scratch these files.

New Patterns

There are other patterns that are less well-known. For example, a filename is a pattern; it must be matched exactly. Thus, if I have a file named HOG and I want to see that it is in the directory, and perhaps check the number of blocks, I can type:

```
LOAD "$0:HOG",8
LIST
```

or,

```
@$0:HOG
```

The only item in the directory will be file HOG (if it exists).

Let's take this a step further. Suppose I don't want to see any file details. All I need is the title of the disk, its ID, and the number of blocks free. That's easy: Just specify a file that does not exist on the disk. The directory will then consist of the title line and the blocks free information. I often ask for a directory using a filename such as 0:#$&!%. This isn't an expletive; it's just a name that I know doesn't exist on the disk so that I'll get the blocks free count.

The Lone Asterisk

You would think that a pattern consisting of only a single asterisk would mean "any file." Thus, a command such as LOAD "*",8 would bring in the first file since anything will match. That's not quite correct: The asterisk often has a special meaning.

The single asterisk sometimes means "same name as last time." It may have been Commodore's intention to allow a user to load a program, and later save it with the same name with SAVE "*",8, the asterisk meaning "same name as before." This was never implemented fully, but you can see traces of this idea in the dual disk copy command. If you have a dual disk, type:

```
@C1:*=0:PROGNAME
```

We can see that this command asks to copy a file called PROGNAME to drive 1; but what name will the new file be given? The destination name is *—which in this case means "same name." Thus, the new file will be named PROGNAME, too. It seems that it was originally Commodore's intention to allow copying to take place with pattern matching, so that C1:*=0:RA* would copy all files whose names started with RA from drive 0 to drive 1 with the same name. If you have a dual drive, try it; it almost works correctly.

So it turns out that LOAD "*",8 does not always load the first file on the disk. Sometimes it loads the same file that was previously loaded.

Specifying Type

You may specify a file type by adding an equals sign to the pattern followed by the file designation: S for Sequential, P for Program, U for User, and R for Relative types. You may also type the three-letter designation such as SEQ or PRG if you wish. Thus, 0:*=S will reference all sequential files, 0:B*=P will reference all programs whose names start with B, and 0:?=P will reference all programs with one-letter names.

Next month we'll look at a common disk error and a way to fix it. ©

C. Regena

Writing An Educational Program

I'm sure you already know or have read what a "good" educational program should contain. I'd like to discuss how you actually program an educational program. I decided that the best way I could describe the process was to write a program, then provide a step-by-step explanation of what I did.

The hardest part of writing any program is deciding the topic and the type of program—drill and practice, tutorial, simulation, game, etc. I picked a very popular topic for computer programs, the Morse code, and decided to do a drill-and-practice program. Quite a few readers have requested programs for secondary school students, so next month I'll present a tutorial on a high school subject.

Memorization Quiz

A drill-and-practice program is useful for any subject that requires memorization. The usual procedure is to ask a question, then have the student input an answer. If you can avoid INPUT and use CALL KEY instead, there will be much less chance for errors or "crashing" the program. In the "Morse Code" program, the quiz will be to press the letter or number after the computer displays a code.

I decided to use the numbers from 0 to 9 and the whole alphabet in the quiz. Since each number and letter corresponds to a code, I set up the array M$ to contain the codes. M$(0) through M$(9) will hold the codes for the numbers in order from 0 through 9. The alphabet will be in M$(10) through M$(35). Since we need 36 elements for the array, line 160 dimensions M$. Lines 170–190 READ the codes for M$ from data in lines 200–250. The data items are in order—first the numbers then the alphabet—each item separated by a comma.

Dots And Dashes

I started out using periods for dots and minus signs for dashes, but decided it was too difficult to type periods with commas—too much chance for typing errors in the DATA statements. Also, the minus sign requires the SHIFT key and the period doesn't, so the typing was a little more complex. I looked on the ASCII character code chart to see what symbols I wouldn't be using in regular printing and decided to use the ampersand (&) to represent a dash and the percent sign (%) to represent a dot.

I borrowed my son's Morse code chart and converted the dots and dashes into % and & signs. These codes are in the DATA statements of lines 200–250. You may use longer DATA statements if you like (the TI accepts up to four screen lines for each numbered line), but I kept the statements shorter to make it a little easier to type and debug.

The next step was to design the graphics—the dots and dashes. The % sign represents a dot in the DATA statement codes and is redefined in line 140 using a CALL CHAR statement so that it will draw a dot on the screen. The & sign is redefined as a bar-shaped figure in line 150. When a dash is printed on the screen, it will actually be three & signs placed together.

The subroutine in lines 360–470 is the main section of coding that translates a code in M$ to the graphic representation on the screen. Looking at a code, if the symbol is % we need to draw a dot, and if the symbol is & we need to draw a dash. This process continues for the entire data, which can be from one to five dots and dashes. Line 360 instructs the computer to check from 1 to the length of the data (which will be from 1 to 5). Line 370 assigns a one-character value to A$ for every increment of the FOR-NEXT loop in line 360. This one-character value is the symbol in the Jth place of the string in the DATA statements. Lines 380–430 instruct the computer to print a dot if the symbol is % and a dash (which is &&&) if the symbol is &. I put a space after the dot or dash to separate them slightly on the screen. You could use CALL HCHAR instead if you wish, but I used PRINT. By printing with semicolons, everything will stay on the same line and be printed right after the previous printing.

Making Some Noise

Since the TI has sound, we can use sound in our Morse code program. Besides that, real Morse code transmission is by sounds. Line 390 plays a sound for a dash, and line 420 plays a different sound for a dot. I used a sound duration of 300 for the dash and 60 for the dot. As you learn the Morse code, you'll probably want to shorten those durations. You should also try different frequencies instead of the one I chose (131) or combinations of frequencies and noise numbers to get a sound you like. Line 440 stops the sound so that dots and dashes are distinct. If you don't have this statement, dashes would run together and you wouldn't be able to tell how many dashes there should be.

Line 450 forces the loop to go to the next symbol in the code. Line 460 PRINTs to get off the present line (colon means "go to the next line" in printing) and add an extra line between codes. Line 470 returns program execution from this subroutine.

Returning To The Menu

I thought it would be nice to review the numbers and letters before having to take the quiz, so there are three sections: Numbers, Alphabet, and Press a Key. Numbers will print each number and show the corresponding Morse code. Alphabet will go through the whole alphabet in order and print each letter with its code. In Press a Key the student can press any number or letter, and the computer will print the code. In any of these sections the student can at any time press ENTER, and the demonstration will stop and the program will return to the main menu screen.

The procedure to see the codes for the numbers is in lines 560–670. Line 570 begins the FOR-NEXT loop with the counter I varying from 0 to 9 for the numbers. The number is printed (by printing I), then the subroutine at 360 is called which deciphers the code M$(I) into the dots and dashes and prints the code on the screen while playing the tones. Line 600 calls subroutine 480, which is simply a delay loop to create a slight pause between numbers. Lines 520–530 check to see if the student has pressed ENTER to return to the main menu screen and stop the numbers section.

The Alphabet section, lines 680–790 is similar to the Numbers section. This time the loop counter I varies from 10 to 35, and the codes will go in order from M$(10) to M$(35), which are the letters from A to Z. To print the letters with the codes, line 700 uses the CHR$ function. The ASCII codes of the letters are from 65 to 90. Since the loop counter I varies from 10 to 35, the ASCII codes for CHR$ are 55+I.

In the Press a Key section, the student may press a letter or number and the computer will display the code. This section could be used as a quick review for students who want to study certain letters. The student may also spell words and phrases one letter at a time to see and hear the Morse code equivalent. Lines 840–920 detect which key is pressed. If the ENTER key (K=13) is pressed, the program branches back to the main menu screen. The IF-THEN statements make sure that only a number or a letter is pressed; all other keys are ignored. The variable K holds the ASCII value of the key pressed, and lines 900 and 930 relate K to the variable I which is used to print the code M$(I).

The instructions are in lines 970–1040, and the quiz is contained in lines 1050–1490. The quiz consists of all ten numbers and 26 letters. An array N() is set up so each of the 36 elements from 0 to 35 is equal to 1. This is in lines 1050–1070. Later as one of the numbers or letters is answered correctly, N(I) will be set to zero so it cannot be chosen again. Line 1080 initializes the number of guesses G to zero for the scoring.

The quiz loop first chooses a random number (I) from 0 to 35 (line 1140). If the number has previously been answered correctly, N(I) will be zero and another number I is chosen. Lines 1160–1190 determine the correct answer L for the number I, which will be the ASCII code of the number or letter chosen. Line 1200 calls the subroutine to print and sound out the code chosen, and line 1210 increments the number of guesses.

Lines 1220–1290 detect the key the student presses; makes sure it is ENTER, a number, or a letter; and then prints the key pressed. If the key pressed is ENTER, the program branches back to the main menu and the quiz ends. Lines 1300–1390 determine if the key pressed is the correct answer. If the answer is incorrect, an "uh-oh" sound is played and the program branches back to line 1200 to display and sound the code again and wait for another answer. If the answer is correct, an arpeggio is played. After the code is answered correctly, line 1400 sets N(I) to zero so that code cannot be chosen again, and line 1410 goes to the next problem. The student must get the right answer to continue the quiz.

Quiz Variations

You can change the program to give the right answer if the student misses. Instead of lines 1330 and 1340, print CHR$(L) or CALL HCHAR or CALL VCHAR and put L on the screen, then branch to line 1400. In this case you might want to keep a score of number correct and number incorrect. You might want to allow that missed letter or number to be shown again. Branch to line 1410 instead of 1400, and before you branch set Z=Z−1. Another way would be to GOTO 1140 instead of changing the loop counter Z and going to the NEXT Z.

If you prefer to let the student guess two or three times before the correct answer is given, set up a flag (FLAG=0) at line 1155 then at line 1340 increment the flag (FLAG=FLAG+1). You could then branch, depending on the value of FLAG, either back for another guess or to give the answer and branch to the next problem.

You might prefer to have a quiz of a certain number of codes, say 10, rather than all 10 numbers and 26 letters. Change line 1130 to FOR Z=1 TO 10. Using lines 1150 and 1400 will still prevent the quiz from choosing the same number or letter more than once.

Another idea would be to have an infinite quiz. Take off the FOR-NEXT loop, lines 1130 and 1410. Also, you won't need lines 1150 and 1400 (and 1050–1070) because the numbers and letters can keep being chosen. Now the quiz keeps going until the student presses ENTER to return to the main menu screen.

In this type of quiz you may want to make sure the code is not the same as the previous one. We can use a variable PI for previous I chosen, and add these two lines:

```
1150 IF PI=I THEN 1140
1155 PI=I
```

You can change the Numbers and Alphabet sections to fit your needs also. To change the delay time between codes, change the upper limit in line 480. Instead of 200, put your own number; a larger number will be a longer delay. Instead of using a delay between numbers and letters, you can have the student press any key to continue, or press the appropriate number or letter. You can change the following lines:

```
650 IF K<>I+48 THEN 610
655 NEXT I
770 IF K<>I+55 THEN 730
775 NEXT I
```

The program is flexible enough that you can change it to do exactly what you want it to do. You can even change the graphics and make it a quiz to learn Braille, or sign language, or some other type of code. You can use words instead of the alphabet and make a quiz for reviewing a foreign language, or perhaps vocabulary words.

Structuring Your Programs

A couple of readers have suggested that I include flowcharts with my programs. My secret is that I haven't touched a flowchart since it was required in my college FORTRAN class years ago. In answer to your questions of how I plan a program, I just sit down at the computer and start typing. With this program, I got to line 350 and typed

```
350 ON K-48 GOTO 1000,2000,3000,400
    0,5000
```

then worked on a section at a time, not necessarily in order. The Numbers section started with line 1000, Alphabet with line 2000, Press a Key with line 3000, the quiz with line 4000, and 5000 was END.

As I realized I needed subroutines, I numbered them 400, 600, and 700, making sure I didn't get to line 1000. On the TI it doesn't really matter where you put the subroutines; you can put them all at the end if you prefer. Anyway, after everything was running properly and each section was tested, I used the RES command to get all the line numbers to look nice. Each programmer has his or her own way of planning, and there's really no right way or wrong way. I say if it works, you're successful.

If you wish to save typing effort, you may obtain a copy of Morse Code by sending $3, a blank cassette or disk, and a stamped, self-addressed mailer to:

C. Regena
P.O. Box 1502
Cedar City, UT 84720

Be sure to specify the title and that you need the TI version.

Morse Code

```
100 CALL CLEAR
110 PRINT TAB(7);"**************"
120 PRINT TAB(7);"* MORSE CODE *"
130 PRINT TAB(7);"**************"::
    ::
140 CALL CHAR(37,"3C7EFFFFFFFF7E3C"
    )
150 CALL CHAR(38,"00FFFFFFFFFFFF")
160 DIM M$(35),N(35)
170 FOR A=0 TO 35
180 READ M$(A)
190 NEXT A
200 DATA &&&&&,%&&&&,%%&&&,%%%&&,%%
    %%&
210 DATA %%%%%,&%%%%,&&%%%,&&&%%,&&
    &&%
220 DATA %&,&%%%,&%&%,&%%,%,%%&%,&&
    %
230 DATA %%%%,%%,%&&&,&%&,%&%%,&&,&
    %
240 DATA &&&,%&&%,&&%&,%&%,%%%,&,%%
    &
250 DATA %%%&,%&&,&%%&,&%&&,&&%%
```

```
260 PRINT "CHOOSE:"
270 PRINT :TAB(5);"1  NUMBERS"
280 PRINT :TAB(5);"2  ALPHABET"
290 PRINT :TAB(5);"3  PRESS A KEY"
300 PRINT :TAB(5);"4  QUIZ"
310 PRINT :TAB(5);"5  END PROGRAM":
    ::::
320 CALL KEY(0,K,S)
330 IF (K<49)+(K>53)THEN 320
340 CALL CLEAR
350 ON K-48 GOTO 560,680,800,970,15
    00
360 FOR J=1 TO LEN(M$(I))
370 A$=SEG$(M$(I),J,1)
380 IF A$="%" THEN 420
390 CALL SOUND(300,131,0)
400 PRINT "&&& ";
410 GOTO 440
420 CALL SOUND(60,131,0)
430 PRINT "% ";
440 CALL SOUND(1,9999,30)
450 NEXT J
460 PRINT ::
470 RETURN
480 FOR D=1 TO 200
490 NEXT D
500 RETURN
510 PRINT :"PRESS <ENTER>";
520 CALL KEY(0,K,S)
530 IF K<>13 THEN 520
540 CALL CLEAR
550 RETURN
560 PRINT TAB(7);"** NUMBERS **"::
570 FOR I=0 TO 9
580 PRINT TAB(4);I;"   ";
590 GOSUB 360
600 GOSUB 480
610 CALL KEY(0,K,S)
620 IF K<>13 THEN 650
630 CALL CLEAR
640 GOTO 260
650 NEXT I
660 GOSUB 510
670 GOTO 260
680 PRINT TAB(6);"** ALPHABET **"::
    :
690 FOR I=10 TO 35
700 PRINT TAB(4);CHR$(55+I);"   ";
710 GOSUB 360
720 GOSUB 480
730 CALL KEY(0,K,S)
740 IF K<>13 THEN 770
750 CALL CLEAR
760 GOTO 260
770 NEXT I
780 GOSUB 510
790 GOTO 260
800 PRINT "PRESS A LETTER OR A NUMB
    ER."
810 PRINT :"ITS CODE WILL BE GIVEN.
    "
820 PRINT :"TO GET BACK TO THE MAIN
    "
830 PRINT :"MENU SCREEN, PRESS <ENT
    ER>."::::
840 CALL KEY(0,K,S)
850 IF K<>13 THEN 880
860 CALL CLEAR
870 GOTO 260
880 IF K<48 THEN 840
890 IF K>57 THEN 920
900 I=K-48
910 GOTO 940
920 IF (K<65)+(K>90)THEN 840
930 I=K-55
940 PRINT CHR$(K);"   ";
950 GOSUB 360
960 GOTO 840
970 PRINT "** MORSE CODE QUIZ **"
980 PRINT ::"YOU WILL HEAR AND SEE
    A"
990 PRINT :"MORSE CODE FOR ONE OF T
    HE"
1000 PRINT :"LETTERS OR NUMBERS."
1010 PRINT :"TYPE THE TRANSLATION."
1020 PRINT :"PRESS <ENTER> TO END T
     HE"
1030 PRINT :"QUIZ AND RETURN TO THE
     "
1040 PRINT :"MAIN MENU SCREEN."
1050 FOR I=0 TO 35
1060 N(I)=1
1070 NEXT I
1080 G=0
1090 PRINT ::"PRESS <ENTER> TO STAR
     T."::::
1100 CALL KEY(0,K,S)
1110 IF S<1 THEN 1100
1120 RANDOMIZE
1130 FOR Z=0 TO 35
1140 I=INT(36*RND)
1150 IF N(I)=0 THEN 1140
1160 IF I>9 THEN 1190
1170 L=I+48
1180 GOTO 1200
1190 L=I+55
1200 GOSUB 360
1210 G=G+1
1220 CALL KEY(0,K,S)
1230 IF K<>13 THEN 1260
1240 CALL CLEAR
1250 GOTO 260
1260 IF K<48 THEN 1220
1270 IF K<58 THEN 1290
1280 IF (K<65)+(K>90)THEN 1220
1290 CALL HCHAR(22,28,K)
1300 IF K=L THEN 1350
1310 CALL SOUND(80,330,2)
1320 CALL SOUND(80,262,2)
1330 GOSUB 480
1340 GOTO 1200
1350 CALL SOUND(100,262,2)
1360 CALL SOUND(100,330,2)
1370 CALL SOUND(100,392,2)
1380 CALL SOUND(200,524,2)
1390 CALL SOUND(1,9999,30)
1400 N(I)=0
1410 NEXT Z
1420 PRINT :::"OUT OF 36 NUMBERS AN
     D"
1430 PRINT "LETTERS, YOUR NUMBER OF
     "
1440 PRINT "GUESSES WAS";G:::
1450 FOR I=1 TO 25
1460 CALL SOUND(-99,INT(400*RND)+50
     0,2)
1470 NEXT I
1480 GOSUB 510
1490 GOTO 260
1500 END
```

©

64 EXPLORER

Larry Isaacs

This month let's discuss a few more things concerning the line-drawing and character-drawing routines presented in the last couple of columns. Some of you may have noted that the character-drawing routines did not support the multicolor mode. This could be done with some additional time and effort. However, because of the increased complexities of handling multicolor mode, there probably won't be room for the routines in the $C000 to $C7FF region of RAM where the other routines were located.

Multiuse Vector Bytes

There were some other things which were not implemented as well. First, vector byte strings were provided only for the uppercase character set. The remaining characters weren't implemented due to the space they would require. You could implement the remaining characters yourself, or even create an entire character set of your own design. Also, you are not restricted to drawing characters. The vector byte strings could be used to draw almost any design.

If you have studied the machine language listing for the character-drawing routines, you may have noticed there was some provision made for additional special function vector bytes. One I had in mind, but didn't get around to implementing, was a "clear character cell" special function code. This would clear a character cell of a specified size. The function would be useful if you wanted to draw characters on top of some other design. Another useful function would be contour fill function—that is, fill the area inside a boundary. With this, large solid characters could be made much more easily. Unfortunately, I doubt there is enough room in the code to have such a routine. Perhaps we can discuss contour filling in a future article.

As you might guess, there are lots of other things which could be implemented. Unfortunately, there isn't enough room to implement them all. This is where the machine language source code listing should come in handy. You can combine routines from various sources to construct the set of routines you require.

Easy To Understand

I hope the comments provided in the source code are sufficient to make most of the routines understandable. The thoroughness of the comments is not consistent throughout the source code. The variation is largely due to an effort to keep the source code from growing too large.

Having good comments in a program can be extremely useful. Unfortunately, there are a couple of factors which tend to discourage commenting. The first factor is that it makes the source code longer. With the speed of the 1541 disk drive, the extra size can noticeably affect the length of time it takes to edit or assemble the source file. The second factor is that it takes extra time to write the comments. Usually, writing the comments will be less interesting than writing the program.

However, if the machine language you plan to write will be of some importance, I highly recommend thoroughly commenting the program. You can use comments to understand how the program was intended to work after you've forgotten. You'd be surprised how fast you can forget.

Comment Fields

There are two basic places to put your comments. One is to the side of the machine language instructions, on the same line as the instructions. The other is between routines, where the comments would document the routine which follows. It is here that the extra effort commenting pays off the best. Ideally, the comments should include a description of what the routine is supposed to do, plus the entry and exit conditions that apply. This would allow you to use the routine, once it is written, without having to study the routine itself to determine what it does. In the long run, such comments can actually save a lot of time. Especially if someone else has to make use of your source code. In the source code I've provided so far, most of the time I've included the entry and exit conditions, but have omitted the description to conserve space.

Program 2 and Program 3 which follow are continuations of last month's column on drawing characters to the bitmap. They facilitate the drawing of letters to a hi-res screen.

Refer to the "Automatic Proofreader" article before typing these programs in.

Program 2: Data For Character Routines

```
1 READ LN,SA,EA:LN=LN+30                 :rem 146
10 FOR I=0 TO EA-SA                      :rem 232
20 READ BY:POKE SA+I,BY:SUM=SUM+BY
                                         :rem 120
30 IF INT((I+1)/8)*8<>(I+1) THEN 60
                                         :rem 242
40 READ CS:IF CS<>SUM THEN 90            :rem 124
50 SUM=0:LN=LN+10                        :rem 254
60 NEXT                                  :rem 165
70 IF INT(I/8)*8<>I THEN READ CS:IF CS<>S
   UM THEN 90                            :rem 78
80 PRINT "SUCCESSFUL LOAD":END           :rem 106
90 PRINT "ERROR IN LINE";LN:END          :rem 105
500 DATA 500                             :rem 68
510 DATA 50176                           :rem 179
520 DATA 51090                           :rem 176
530 DATA 76,220,197,76,230,197,76,99,1171
                                         :rem 0
540 DATA 199,76,109,199,76,138,199,76,107
    2                                    :rem 67
550 DATA 24,196,76,24,196,76,24,196,812
                                         :rem 155
560 DATA 96,0,208,0,0,0,0,0,304          :rem 213
570 DATA 0,0,0,0,0,0,0,0,0               :rem 198
580 DATA 0,0,0,0,0,0,0,0,0               :rem 199
590 DATA 0,0,0,0,0,0,0,32,32             :rem 50
600 DATA 253,174,32,138,173,32,247,183,12
    32                                   :rem 72
610 DATA 165,101,164,100,96,32,253,174,10
    85                                   :rem 67
620 DATA 32,158,173,36,13,48,3,76,539
                                         :rem 42
630 DATA 240,192,160,0,177,100,141,30,104
    0                                    :rem 250
640 DATA 196,200,177,100,133,20,200,177,1
    203                                  :rem 102
650 DATA 100,133,21,76,163,182,72,162,909
                                         :rem 230
660 DATA 0,201,32,144,5,233,32,232,879
                                         :rem 72
670 DATA 208,247,104,24,125,121,196,170,1
    195                                  :rem 125
680 DATA 96,128,0,192,224,192,192,128,115
    2                                    :rem 34
690 DATA 128,133,253,173,14,220,41,254,12
    16                                   :rem 67
700 DATA 141,14,220,165,1,41,251,133,966
                                         :rem 166
710 DATA 1,169,0,6,253,42,6,253,730
                                         :rem 179
720 DATA 42,6,253,42,133,254,24,173,927
                                         :rem 132
730 DATA 25,196,101,253,133,253,173,26,11
    60                                   :rem 67
740 DATA 196,101,254,133,254,162,0,160,12
    60                                   :rem 61
750 DATA 7,177,253,153,32,196,138,153,110
    9                                    :rem 36
760 DATA 41,196,136,16,244,165,1,9,808
                                         :rem 93
770 DATA 4,133,1,173,14,220,9,1,555
                                         :rem 177
780 DATA 141,14,220,96,160,7,162,7,807
                                         :rem 84
790 DATA 30,41,196,106,202,16,249,153,993
                                         :rem 241
800 DATA 32,196,136,16,241,96,169,7,893
                                         :rem 153
810 DATA 133,251,162,0,160,7,30,41,784
                                         :rem 68
820 DATA 196,106,136,16,249,164,251,153,1
    271                                  :rem 130
830 DATA 32,196,232,198,251,16,237,96,125
    8                                    :rem 45
840 DATA 160,7,162,7,94,41,196,42,709
                                         :rem 44
850 DATA 202,16,249,153,32,196,136,16,100
    0                                    :rem 17
860 DATA 241,96,172,29,196,208,1,96,1039
                                         :rem 202
870 DATA 162,7,189,32,196,157,41,196,980
                                         :rem 211
880 DATA 202,16,247,136,208,3,76,204,1092
                                         :rem 232
890 DATA 196,136,208,3,76,222,196,76,1113
                                         :rem 248
900 DATA 248,196,160,8,169,0,153,41,975
                                         :rem 147
910 DATA 196,136,16,250,169,255,141,40,12
    03                                   :rem 72
920 DATA 196,138,240,15,168,162,8,94,1021
                                         :rem 238
930 DATA 32,196,126,41,196,202,16,247,105
    6                                    :rem 29
940 DATA 136,208,242,96,32,97,192,173,117
    6                                    :rem 45
950 DATA 32,192,41,7,133,253,162,0,820
                                         :rem 73
960 DATA 160,0,177,251,45,49,196,29,907
                                         :rem 152
970 DATA 32,196,145,251,160,8,177,251,122
    0                                    :rem 28
980 DATA 45,40,196,29,41,196,145,251,943
                                         :rem 202
990 DATA 232,224,8,240,31,198,253,48,1234
                                         :rem 239
1000 DATA 8,230,251,208,219,230,252,208,1
     606                                 :rem 107
1010 DATA 215,169,7,133,253,24,165,251,12
     17                                  :rem 65
1020 DATA 105,57,133,251,165,252,105,1,10
     69                                  :rem 58
1030 DATA 133,252,208,196,76,114,192,140,
     1311                                :rem 162
1040 DATA 31,196,32,102,196,32,129,196,91
     4                                   :rem 26
1050 DATA 32,10,197,32,171,193,32,42,709
                                         :rem 173
1060 DATA 197,32,76,197,24,169,8,160,863
                                         :rem 203
1070 DATA 0,174,29,196,240,12,202,240,109
     3                                   :rem 9
1080 DATA 25,169,248,160,255,202,240,2,13
     01                                  :rem 60
1090 DATA 208,16,109,30,192,141,30,192,91
     8                                   :rem 21
1100 DATA 152,109,31,192,141,31,192,76,92
     4                                   :rem 16
```

```
1110 DATA 216,197,109,32,192,141,32,192,1
111                                 :rem 109
1120 DATA 172,31,196,96,32,55,196,141,919
                                    :rem 243
1130 DATA 25,196,140,26,196,96,32,69,780
                                    :rem 197
1140 DATA 196,36,13,48,3,76,151,197,720
                                    :rem 138
1150 DATA 173,30,196,240,13,160,0,177,989
                                    :rem 233
1160 DATA 20,32,151,197,200,204,30,196,10
30                                  :rem 48
1170 DATA 144,245,96,41,127,10,168,173,10
04                                  :rem 69
1180 DATA 27,196,133,251,173,28,196,133,1
137                                 :rem 132
1190 DATA 252,177,251,141,50,196,200,177,
1444                                :rem 174
1200 DATA 251,141,51,196,96,173,50,196,11
54                                  :rem 75
1210 DATA 133,251,173,51,196,133,252,160,
1349                                :rem 166
1220 DATA 0,177,251,72,238,50,196,208,119
2                                   :rem 24
1230 DATA 3,238,51,196,104,201,143,240,11
76                                  :rem 59
1240 DATA 1,24,96,72,41,15,201,8,458
                                    :rem 231
1250 DATA 144,2,9,240,141,54,196,169,955
                                    :rem 190
1260 DATA 0,141,53,196,104,74,74,74,716
                                    :rem 133
1270 DATA 74,201,8,144,7,9,240,162,845
                                    :rem 83
1280 DATA 255,142,53,196,141,52,196,96,11
31                                  :rem 82
1290 DATA 56,169,0,237,52,196,141,52,903
                                    :rem 189
1300 DATA 196,169,0,237,53,196,141,53,104
5                                   :rem 26
1310 DATA 196,96,56,169,0,237,54,196,1004
                                    :rem 243
1320 DATA 141,54,196,96,169,0,141,53,850
                                    :rem 185
1330 DATA 196,173,52,196,174,54,196,141,1
182                                 :rem 138
1340 DATA 54,196,142,52,196,16,5,169,830
                                    :rem 192
1350 DATA 255,141,53,196,96,174,29,196,11
40                                  :rem 89
1360 DATA 208,1,96,202,208,6,32,124,877
                                    :rem 132
1370 DATA 198,76,96,198,202,208,6,32,1016
                                    :rem 246
1380 DATA 96,198,76,114,198,32,124,198,10
36                                  :rem 100
1390 DATA 76,114,198,32,149,198,24,173,96
4                                   :rem 53
1400 DATA 32,192,109,54,196,141,36,192,95
2                                   :rem 30
1410 DATA 24,173,30,192,109,52,196,141,91
7                                   :rem 24
1420 DATA 34,192,173,31,192,109,53,196,98
0                                   :rem 34
1430 DATA 141,35,192,96,169,0,141,53,827
                                    :rem 186
1440 DATA 196,32,29,198,141,52,196,201,10
45                                  :rem 79
1450 DATA 0,16,5,169,255,141,53,196,835
                                    :rem 138
1460 DATA 32,29,198,141,54,196,76,179,905
                                    :rem 2
1470 DATA 198,140,31,196,32,3,198,32,830
                                    :rem 187
1480 DATA 29,198,176,99,32,59,198,201,992
                                    :rem 16
1490 DATA 248,240,9,32,179,198,32,195,113
3                                   :rem 41
1500 DATA 194,76,247,198,174,54,196,208,1
347                                 :rem 152
1510 DATA 15,32,29,198,32,59,198,32,595
                                    :rem 150
1520 DATA 179,198,32,159,193,76,247,198,1
282                                 :rem 159
1530 DATA 202,208,9,32,212,198,32,195,108
8                                   :rem 27
1540 DATA 194,76,247,198,202,208,9,32,116
6                                   :rem 40
1550 DATA 212,198,32,159,193,76,247,198,1
315                                 :rem 147
1560 DATA 202,208,6,32,24,196,76,247,991
                                    :rem 194
1570 DATA 198,202,208,6,32,24,196,76,942
                                    :rem 196
1580 DATA 247,198,202,208,6,32,24,196,111
3                                   :rem 28
1590 DATA 76,247,198,202,208,6,32,24,993
                                    :rem 201
1600 DATA 196,76,247,198,76,247,198,172,1
410                                 :rem 153
1610 DATA 31,196,96,32,55,196,141,27,774
                                    :rem 197
1620 DATA 196,140,28,196,96,32,69,196,953
                                    :rem 6
1630 DATA 36,13,48,3,76,241,198,173,788
                                    :rem 152
1640 DATA 30,196,240,13,160,0,177,20,836
                                    :rem 171
1650 DATA 32,241,198,200,204,30,196,144,1
245                                 :rem 116
1660 DATA 245,96,32,234,192,41,3,141,984
                                    :rem 190
1670 DATA 29,196,96,321             :rem 124
```

Program 3:
Illustration Of Character Routines

```
10 REM DRAW CHARACTERS IN BIT-MAP:rem 212
20 POKE 56,156:CLR                :rem 223
30 CT=PEEK(56)*256+PEEK(55):REM CHAR DATA
   PTR                            :rem 54
40 J1=49152:REM DRAWING JUMP TABLE
                                  :rem 239
50 J2=50176:REM CHAR. JUMP TABLE  :rem 47
60 GOTO 1000                      :rem 96
1000 REM MAIN ROUTINE             :rem 240
1010 GOSUB 10000:SYS J2+6,CT      :rem 12
1020 SYS J1:SYS J1+6,0:SYS J1+9,6,14
                                  :rem 185
1030 SYS J1+12,10,180:REM MOVE    :rem 115
1040 SYS J2+3,"EXAMPLE USE OF PUT "
                                  :rem 149
1050 SYS J2+3,"CHARACTER ROUTINE."
                                  :rem 240
1060 SYS J1+12,10,160:REM MOVE    :rem 116
1070 FOR CH=32 TO 63              :rem 232
1080 SYS J2+3,CH:NEXT             :rem 210
1090 SYS J1+12,257,140            :rem 93
```

```
1100 SYS J2+12,2:REM ROTATE 180 DEG.
                              :rem 173
1110 FOR CH=64 TO 95          :rem 237
1120 SYS J2+3,CH:NEXT         :rem 205
1130 SYS J2+12,0:REM NO ROTATION  :rem 85
2000 SYS J1+12,10,80:REM MOVE     :rem 64
2010 SYS J2+9,"EXAMPLE USE OF DRAW "
                              :rem 206
2020 SYS J2+9,"CHARACTER ROUTINE":rem 198
2030 SYS J1+12,10,60:REM MOVE     :rem 65
2040 FOR CH=64 TO 90          :rem 235
2050 SYS J2+9,CH:NEXT         :rem 214
2060 SYS J1+12,217,40:REM MOVE  :rem 123
2070 SYS J2+12,2:REM ROTATE 180 DEG.
                              :rem 180
2080 FOR CH=90 TO 64 STEP -1  :rem 137
2090 SYS J2+9,CH:NEXT         :rem 218
2100 SYS J2+12,0:REM NO ROTATION  :rem 83
9000 GET Z$:IF Z$="" THEN 9000  :rem 231
9010 SYS J1+3                 :rem 162
9020 END                      :rem 162
10000 REM LOAD CHAR. VB DATA  :rem 243
10010 C=0:PT=CT+256:REM INIT POINTER
                              :rem 143
10020 READ CH:IF CH<0 THEN RETURN:rem 105
10030 HB=INT(PT/256):LB=PT-HB*256:rem 142
10040 POKE CT+CH*2,LB:POKE CT+CH*2+1,HB
                              :rem 171
10050 GOSUB 10100:REM LOAD VB DATA:rem 88
10060 GOTO 10020              :rem 35
10100 REM LOAD CHAR. DATA AT PT  :rem 149
10110 READ VB                 :rem 167
10120 IF C>0 THEN C=C-1:GOTO 10180
                              :rem 241
10130 IF ABS(VB)>7 THEN 10160   :rem 223
10140 READ DY:VB=(VB*16+(DYAND15))
                              :rem 138
10150 GOTO 10180               :rem 42
10160 IF VB=143 THEN 10190     :rem 30
10170 IF VB<>128 THEN C=2      :rem 22
10180 POKE PT,VBAND255:PT=PT+1:GOTO 10110
                              :rem 129
10190 POKE PT,VBAND255:PT=PT+1:RETURN
                              :rem 54
11100 REM ADD CHARACTER DATA FROM PROGRAM
   1 IN LAST MONTH'S ISSUE     :rem 24
```

©

To receive additional information from advertisers in this issue, use the handy reader service cards in the back of the magazine.

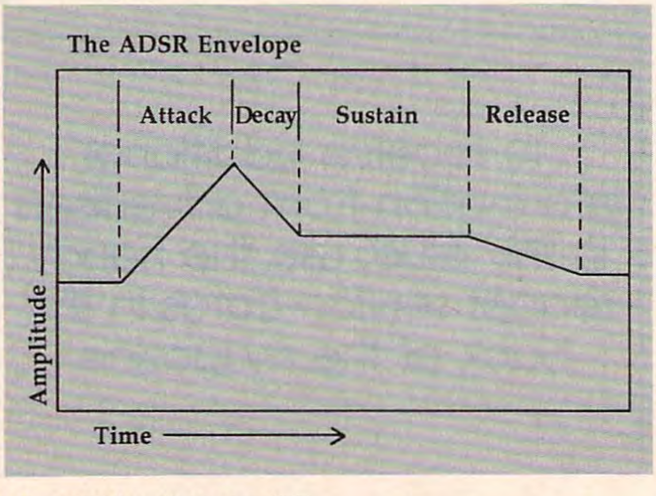
SYSound

Mike Steed

The Commodore 64 has an amazing sound chip, and anyone who has heard it knows this. However, anyone who has tried to program it may have been surprised or discouraged, because everything had to be done with POKEs. That is, until now. "SYSound" will make creating sounds much easier, using absolutely no POKEs at all. Also included is an example program to show how easy programming 64 music can be.

Type in Program 1 and be sure to save a copy before running it. Program 1 loads in SYSound, which is a machine language program, and one typing mistake can crash SYSound when you use it. You may wish to save a copy of just the machine language once it's loaded, if you have a machine language monitor. Program 1 will specify the start and end addresses.

To use SYSound, all you need to do is type SYS 49152 followed by any of several possible parameters, each separated by a comma. The number 49152 could (and probably should) be put into a variable, such as S or SOUND.



A list of possible parameters for the SYS statement and their meanings follows:

- V*x*, where *x* is the voice number used for the note (one, two, or three). More than one voice may be used at the same time.
- A*x*, where *x* is the attack rate of the note. This is the time it takes the sound to reach its highest volume. The value of *x* must be between 0 and 15; the larger the number, the more time it takes. (See the figure for a further description of attack, decay, sustain, and release.)
- D*x*, where *x* is the decay rate of the note (0–15). This is the time it takes the sound to soften to the sustain volume.
- S*x*, where *x* is the sustain level of the note (0–15). The sound remains at this volume until the release starts.
- R*x*, where *x* is the release rate of the note (0–15). The release rate is the time it takes the sound to drop from the sustain volume to silence.
- W*y*[*x*], where *y* is a letter representing the waveform used for the sound. This can be N (noise), S (sawtooth), T (triangle), or P (pulse). If the pulse waveform is chosen, then a pulse rate *x* (0–4095) must be entered after the waveform letter, such as WP2048 for a square wave.
- F*x*, where *x* is the frequency of the note (0–65535). Higher frequencies will produce higher notes.
- L*x*, where *x* is the volume (loudness) of the note (0–15). Note that this is the overall volume, so all the voices will be affected by this setting.
- C clears the sound chip. This is equivalent to the following in BASIC:

```
10 S=54272:FOR I=0 TO 24:POKE S+I,0:NEXT
```

Once a parameter has been entered, it need not be entered the next time the routine is used. For example, if all your sound effects are going to be done with voice 1, at volume 15, with the sawtooth waveform, attack 0, decay 9, and sustain and release 0, you could set all these at the beginning of your program:

10 S=49152:SYS S,C,V1,L15,WS,D9

(All parameters default to zero initially, so A, S, and R needn't be entered.) Then all that would need to be done to play a note would be:

20 SYS S,F5000

(Any valid numeric expression may be used after the parameter letter.) Also, if a parameter is entered more than once, only the last case will be considered. For example, SYS S,WS,WT,A0,A6 is effectively the same as SYS S,WT,A6.

Program 2 provides an example of SYSound in action, and shows how much simpler music programming can be accomplished.

If you would rather not type all those DATA statements, I will send you a copy of the program. Send a stamped, self-addressed mailer, a blank tape or disk (1540/1541), and $3 to:

Mike Steed
712 W. 1280 S.
Provo, UT 84601

Program 1: SYSound

Refer to the "Automatic Proofreader" article before typing this program in.

```
100 DATA 32,121,0,208,3,76            :rem 234
110 DATA 241,192,201,44,240,3         :rem 127
120 DATA 76,67,193,32,115,0            :rem 44
130 DATA 162,8,221,76,193,240         :rem 144
140 DATA 6,202,16,248,76,67            :rem 52
150 DATA 193,138,10,170,189,85        :rem 205
160 DATA 193,133,251,189,86,193         :rem 6
170 DATA 133,252,32,50,192,76         :rem 145
180 DATA 0,192,108,251,0,32            :rem 33
190 DATA 55,193,201,1,144,4            :rem 40
200 DATA 201,4,144,3,76,72            :rem 241
210 DATA 193,202,142,114,193,96       :rem 243
220 DATA 32,55,193,10,10,10            :rem 25
230 DATA 10,141,123,193,173,120       :rem 227
240 DATA 193,41,15,13,123,193         :rem 139
250 DATA 141,120,193,96,32,55         :rem 145
260 DATA 193,141,123,193,173,120       :rem 34
270 DATA 193,41,240,13,123,193        :rem 190
280 DATA 141,120,193,96,32,55         :rem 148
290 DATA 193,10,10,10,10,141           :rem 72
300 DATA 123,193,173,121,193,41       :rem 237
310 DATA 15,13,123,193,141,121        :rem 177
320 DATA 193,96,32,55,193,141         :rem 153
330 DATA 123,193,173,121,193,41       :rem 240
340 DATA 240,13,123,193,141,121       :rem 228
350 DATA 193,96,32,115,0,162           :rem 95
360 DATA 3,221,103,193,240,6           :rem 84
370 DATA 202,16,248,76,67,193         :rem 160
380 DATA 224,1,240,6,32,115            :rem 34
390 DATA 0,76,196,192,32,44            :rem 56
400 DATA 193,192,16,144,3,76           :rem 99
410 DATA 72,193,142,117,193,140       :rem 243
420 DATA 118,193,162,1,189,107        :rem 199
430 DATA 193,141,119,193,96,32        :rem 204
440 DATA 44,193,142,115,193,140       :rem 243
450 DATA 116,193,96,32,55,193         :rem 159
460 DATA 141,122,193,96,169,0         :rem 151
470 DATA 162,24,157,0,212,202         :rem 134
480 DATA 16,250,169,0,141,115         :rem 141
490 DATA 193,141,116,193,76,115       :rem 255
500 DATA 0,173,115,193,208,5           :rem 89
510 DATA 173,116,193,240,37,174       :rem 248
520 DATA 114,193,189,111,193,133       :rem 41
530 DATA 251,169,212,133,252,160       :rem 34
540 DATA 6,185,115,193,145,251        :rem 201
550 DATA 136,16,248,160,4,173         :rem 149
560 DATA 119,193,9,1,145,251          :rem 101
570 DATA 173,122,193,141,24,212       :rem 240
580 DATA 96,165,122,208,2,198         :rem 161
590 DATA 123,198,122,76,121,0         :rem 146
600 DATA 32,166,173,32,247,183        :rem 199
610 DATA 166,20,164,21,96,32           :rem 94
620 DATA 44,193,152,208,11,224        :rem 191
630 DATA 16,176,7,138,96,162          :rem 111
640 DATA 11,76,58,164,162,14          :rem 101
650 DATA 208,249,86,65,68,83          :rem 124
660 DATA 82,87,70,76,67,53             :rem 20
670 DATA 192,72,192,94,192,112        :rem 209
680 DATA 192,134,192,152,192,203       :rem 45
690 DATA 192,213,192,220,192,78         :rem 2
700 DATA 80,83,84,128,64,32            :rem 54
710 DATA 16,0,7,14,0,0                 :rem 33
720 DATA 0,0,0,0,0,0                  :rem 175
730 DATA 0,0                           :rem 64
740 FORI=49152TO49531:READJ:POKEI,J:K=K+J
    :NEXT                             :rem 121
750 IFK<>44621THENPRINT"ERROR IN DATA STA
    TEMENTS":STOP                     :rem 180
760 PRINT"{CLR}{3 DOWN}SYS SOUND{DOWN}
    {9 LEFT}E9 T3":Q$=CHR$(34)        :rem 178
770 PRINT"TO SAVE IN MONITOR:":PRINT"
    {DOWN}.S "Q$"SYS SOUND"Q$",01,C000,C1
    7C                                 :rem 85
780 PRINTSPC(15)"↑↑":PRINTSPC(15)"{DOWN}0
    1 FOR TAPE,":PRINTSPC(15)"08 FOR DISK
                                       :rem 32
```

Program 2: Sample Program Using SYSound

Refer to the "Automatic Proofreader" article before typing this program in.

```
120 S=49152:SYS S,C,L15:T=TIME       :rem 251
130 READ D:IF D=0 THEN SYS S,C:END
                                     :rem 111
140 READ F1,F2,F3                    :rem 113
150 SYS S,V1,F(F1),WT,A0,D9,S0, R0:rem 79
160 SYS S,V2,F(F2),WS,A2,D4,S2, R2:rem 82
170 SYS S,V3,F(F3),WT,A1,D2,S10,R10
                                     :rem 177
180 T=T+10*D                         :rem 120
190 IF T>TIME GOTO 190               :rem 189
200 GOTO 130                          :rem 95
300 DATA 1,13153,0,0                 :rem 191
310 DATA 1,11060,0,0                 :rem 187
320 DATA 2,8779,5530,2195            :rem 226
330 DATA 2,8779,6577,0                :rem 78
340 DATA 1,8779,4389,1644            :rem 236
350 DATA 1,9854,0,0                  :rem 161
360 DATA 1,11060,6577,0              :rem 105
370 DATA 1,11718,0,0                 :rem 203
400 DATA 2,13153,5530,2195           :rem 255
410 DATA 2,13153,6577,0              :rem 107
```

```
420 DATA 2,13153,4389,2463         :rem 10
430 DATA 2,11060,6577,2765         :rem 12
440 DATA 2,14764,5859,2930         :rem 23
450 DATA 2,14764,8779,0            :rem 126
460 DATA 2,14764,7382,2195         :rem 21
470 DATA 1,0,8779,0                :rem 169
480 DATA 1,13153,0,0               :rem 200
500 DATA 2,14764,5859,2930         :rem 20
510 DATA 1,0,8779,0                :rem 164
520 DATA 1,13153,0,0               :rem 195
530 DATA 1,14764,7382,2765         :rem 21
540 DATA 1,16572,0,0               :rem 205
550 DATA 1,17557,8779,2463         :rem 32
560 DATA 1,19708,0,0               :rem 211
600 DATA 2,22121,5530,2195         :rem 252
610 DATA 2,0,6577,0                :rem 160
620 DATA 2,0,4389,1644             :rem 63
630 DATA 1,17557,6577,0            :rem 122
640 DATA 1,13153,0,0               :rem 198
650 DATA 2,17557,5530,2195         :rem 18
660 DATA 2,0,6577,0                :rem 165
670 DATA 2,0,4389,2071             :rem 63
680 DATA 1,13153,6577,1845         :rem 21
690 DATA 1,11060,0,0               :rem 198
700 DATA 2,13153,5859,1644         :rem 14
710 DATA 2,0,6577,0                :rem 161
720 DATA 2,0,4927,2463             :rem 62
730 DATA 1,9854,6577,0             :rem 76
740 DATA 1,11108,0,0               :rem 197
750 DATA 2,8779,5530,2195          :rem 233
760 DATA 2,0,6577,1644             :rem 69
770 DATA 2,0,5530,1097             :rem 60
780 DATA 2,0,0,0                   :rem 255
790 DATA 0                         :rem 234 ©
```


Musical TI Keyboard

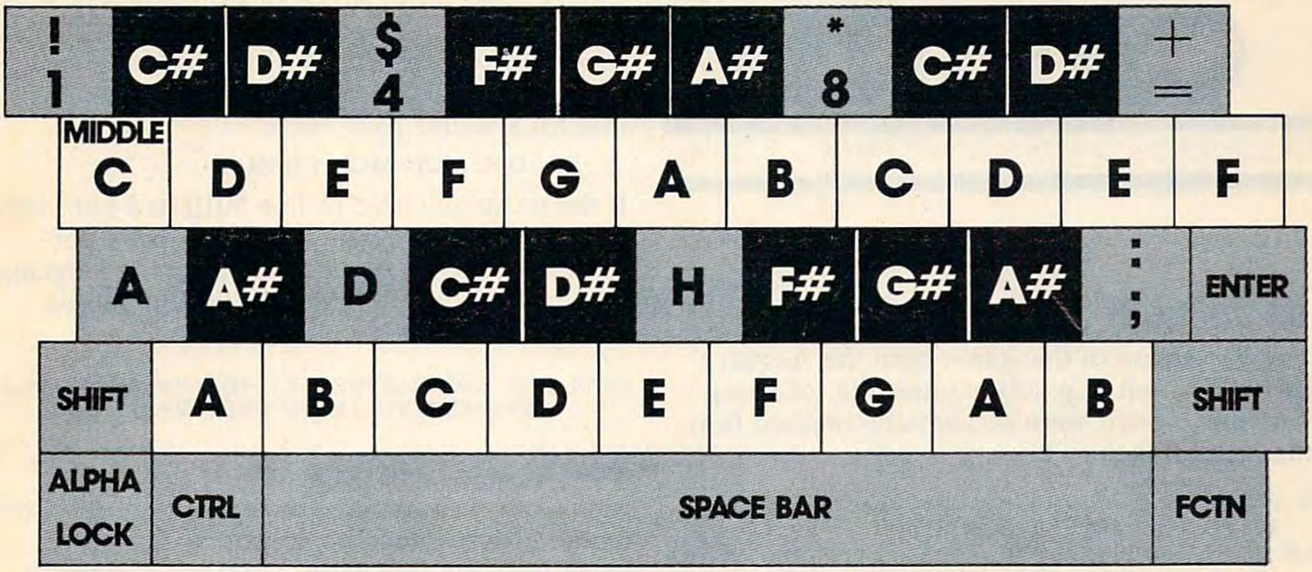
Randal J. Reifsnider

The TI music chip has long been regarded as an excellent sound chip, but few programs have yet demonstrated its capabilities. "Musical TI Keyboard" changes all that by turning your TI's keys into simulated piano keys.

In the book *Beginner's BASIC* that comes with the TI-99/4A computer, there is a short demonstration program illustrating how you can use the computer's keyboard to make musical tones. When you run this program and press the A key, the musical tone A will sound. The tone will continue as long as you hold down the key, with a slight gap of silence between repetitions of the tone. This sounds like a musical machine gun. It is an interesting program, but very limited. Since it uses only seven letters of the alphabet to represent musical notes, you could play only seven notes on the computer in this fashion (A through G, with no sharps or flats).

Also, if you play the piano and are familiar with its keyboard arrangement, you'll find that looking for letters feels unnatural and difficult. Hence, "Musical TI Keyboard," which makes the computer's keyboard more closely resemble that of a piano.

This program first READs frequency values from DATA statements into an array, then mathematically converts the ASCII code returned by the CALL KEY statement, and uses that value in the CALL SOUND statement to locate the corresponding frequency value within the array. The figure shows the arrangement of the keyboard. Since not all the keys are used, the program includes a check to silence any unwanted keys. ASCII code numbers of silenced keys which fall within the array are assigned a DATA value of 1 as a filler. This allows the array to be easily filled and insures that the ASCII code for a given key corresponds to the proper frequency.



Program Variations

One variation of this program you may want to try would be:

```
90 CALL SOUND (100,NOTE(Q),1,1.26*NOTE(Q),5,
   1.5*NOTE(Q),5)
```

This would produce a major chord for each key pressed. To create minor chords, try:

```
90 CALL SOUND (100, NOTE(Q),1,1.19*NOTE(Q),5,
   1.5*NOTE(Q),5)
```

If you change the duration from 100 to −150, the computer will play continuous tones. A value for a noise (−1 through −8) could be added to the CALL SOUND statement for an interesting effect. The space bar could be assigned a noise value for use as percussion. Since this program requires that the ALPHA LOCK be on, additional tones or noises could be assigned to what would be the lowercase letters.

Even though we do have a piano, our four-year-old daughter would rather play the computer. However, you can take the program further. You could include a routine within the program to print out the duration, frequency, and sequence of the notes you play on the computer's keyboard. This could be extremely helpful when tackling the laborious task of transposing sheet music so that it can be played by the computer. You could also try creating a routine that would play back any song played on the computer.

To make playing your computer/piano keyboard easier, you might want to buy two different colors of small gummed labels, like those sold in office supply stores. These may be placed on the computer keys to distinguish the white keys from the black keys. Novice musicians may also wish to write the name of the note on the label as an aid to playing. These labels can be easily removed when you are ready to let the computer go back to its regular keyboard functions.

Musical TI Keyboard

```
10 CALL CLEAR
20 DIM NOTE(47)
30 FOR C=1 TO 47
40 READ NOTE(C)
50 NEXT C
60 CALL KEY(0,N,S)
70 IF (N<44)+(N=45)+(N=49)+(N=52)+(
   N=56)+((N>57)*(N<66))+(N=68)+(N=
   72)+(N>90)THEN 60
80 Q=N-43
90 CALL SOUND(100,NOTE(Q),1)
100 GOTO 60
110 DATA 220,1,247,698,622,1,277,31
    1,1,370,415,466,1,554,1,1,1,1,1
    ,1,1,1,165,131,1,330,139,156,1
120 DATA 523,185,208,233,196,175,58
    7,659,262,349,117,392,494,147,2
    94,123,444,110
```

©

CAPUTE!

Modifications Or Corrections To Previous Articles

64 Jackpot

The 64 version of this game from the August issue (Program 3, p. 89) requires the following two lines, which were accidentally omitted from the original listing:

```
5 PRINT"{CLR}";:POKE51,Ø:POKE55,Ø:POKE52,
   48:POKE56,48:CLR:GOSUB 6Ø          :rem 61
1Ø TT=5Ø:S=54272:FORL=STOS+24:POKEL,Ø:NEX
   T                                  :rem 135
```

IBM PC/PCjr Blueberries

The IBM version (Program 3, p. 88) of this game in the July issue should work as published, but reader Michael Saletnik points out that the programmer used the VARPTR statement incorrectly in line 5000. VARPTR returns the starting address for the *descriptor* of the specified string variable. The descriptor is three bytes of data; the first byte tells the length of the string, and the other two hold the starting address within the current segment of memory where the characters that make up the string are stored. Thus, if you use a statement like V=VARPTR(ML$), then PRINT PEEK(V) will show the length of ML$, and PRINT PEEK(V+1)+256*PEEK(V+2) will give the starting address of the characters in ML$.

In line 5000, the calculated address ZZ does not point to the start of ML$ as intended, but rather off into some other part of the variable area. "Blueberries" works as printed because the programmer uses the computed address to POKE the machine language directly into memory in line 5010. A more standard way of transferring the machine language from DATA statements into ML$ would have been:

READ A: ML$=ML$+CHR$(A)

If the technique used in line 5010 had not been used, then the program would not have performed correctly. To place the machine language data properly into ML$, line 5000 should be changed to read:

```
5ØØØ DEF SEG:ML$=SPACE$(48):V=VARPTR(ML$)
    :ZZ=PEEK(V+1)+256*PEEK(V+2)
```

Bunny Hop For The 64

Characters were omitted in two lines of the Commodore 64 version (Program 1, p. 74) of this game from the July issue. The final number in line 35 should be 208 instead of just 2, and the final number in line 200 should be 33 instead of 3. The corrected lines should read as follows:

```
35 DATA4Ø,169,32,145,253,96,16Ø,41,177,25
   3,136,145,253,2ØØ,2ØØ,192,81,2Ø8
2ØØ POKEP,32:POKE37154,127:Y=PEEK(5632Ø)A
    NDPEEK(QQ):IF(YAND8)=ØTHENP=P+1:D=33
```

VIC Olympiad

There is an error in one of the PRINT statements which defines the arena in the VIC version (Program 2, p. 56) of this game from the June issue. Ed Eyerman notes that there should be two spaces following the five SHIFTed spaces in line 3080. The line should read as follows:

```
3Ø8Ø PRINT"-{2 SHIFT-SPACE}{5 SPACES}U{W}
    {2 SPACES}{Q}I{5 SHIFT-SPACE}
    {2 SPACES}-";
```

Also, line 1045 in the VIC version is an unintentional carryover from the original Commodore 64 version, and can be deleted. ©

Tiny MLX Machine Language Entry Program

For Unexpanded VIC-20

Charles Brannon, Program Editor

MLX is a labor-saving utility that allows almost fail-safe entry of machine language programs published in COMPUTE!. *You need to know nothing about machine language to use MLX—it was designed for everyone. "Tiny MLX" is a special version for the unexpanded VIC.*

MLX is a new way to enter long machine language (ML) programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to BASIC DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255 (forbidden in ML). It won't let you enter the wrong numbers on the wrong line. In addition, MLX creates a ready-to-use tape or disk file.

Using MLX

Type in and save "Tiny MLX" (you'll want to use it in the future). When you're ready to type in an ML program, run Tiny MLX. Unlike regular MLX, Tiny MLX does not ask for the starting and ending address of the program to be entered. Instead, this information must be included in line 210. The values currently shown in line 210 are for the "Lightsaver" program in this issue.

You'll see a prompt corresponding to the starting address. The prompt is the current line you are entering from the listing. It increases by six each time you enter a line. That's because each line has seven numbers—six actual data numbers plus a *checksum number*. The checksum verifies that you typed the previous six numbers correctly. If you enter any of the six numbers wrong, or enter the checksum wrong, the computer rings a buzzer and prompts you to reenter the line. If you enter it correctly, a bell tone sounds and you continue to the next line.

MLX accepts only numbers as input. If you make a typing error, press the INST/DEL key; the entire number is deleted. You can press it as many times as necessary back to the start of the line. If you enter three-digit numbers as listed, the computer automatically prints the comma and goes on to accept the next number. If you enter less than three digits, you can press either the comma, space bar, or RETURN key to advance to the next number. The checksum automatically appears in reverse video for emphasis.

MLX Commands

When you finish typing an ML listing, you can then save the completed program on tape or disk. Follow the screen instructions. If you get any errors while saving, you probably have a bad disk, or the disk is full, or you made a typo when entering the MLX program itself.

Since Tiny MLX has no provisions for reloading a partially completed program, you must enter the ML program all in one sitting.

Tiny MLX

```
100 POKE55,174:POKE56,23:CLR:POKE788,194    :rem 76
210 S=6063:E=7658                           :rem 136
300 PRINT"{CLR}";CHR$(14):AD=S              :rem 56
310 PRINTRIGHT$("0000"+MID$(STR$(AD),2),5);":";:FO
    RJ=1TO6                                 :rem 234
320 GOSUB570:IFN=-1THENJ=J+N:GOTO320        :rem 228
480 IFN<0THENPRINT:GOTO310                  :rem 168
490 A(J)=N:NEXTJ                            :rem 199
500 CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT                      :rem 200
510 PRINTCHR$(18);:GOSUB570:PRINTCHR$(20) :rem 234
515 IFN=CKSUMTHEN530                        :rem 255
520 PRINT:PRINT"LINE ENTERED WRONG":PRINT"RE-ENTER
    ":PRINT:GOSUB1000:GOTO310               :rem 129
530 GOSUB2000                               :rem 218
540 FORI=1TO6:POKEAD+I-1,A(I):NEXT          :rem 80
550 AD=AD+6:IFAD<ETHEN310                   :rem 212
560 GOTO710                                 :rem 108
570 N=0:Z=0                                 :rem 88
580 PRINT"Ɛ+Ʒ";                             :rem 79
581 GETA$:IFA$=""THEN581                    :rem 95
585 PRINTCHR$(20);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N670                                    :rem 229
590 IFA>128THENN=-A:RETURN                  :rem 137
600 IFA<>20 THEN 630                        :rem 10
610 GOSUB690:IFI=1ANDT=44THENN=-1:PRINT"{LEFT}
    {LEFT}";:GOTO690                        :rem 172
620 GOTO570                                 :rem 109
630 IFA<48ORA>57THEN580                     :rem 105
640 PRINTA$;:N=N*10+A-48                    :rem 106
650 IFN>255 THEN A=20:GOSUB1000:GOTO600     :rem 229
660 Z=Z+1:IFZ<3THEN580                      :rem 71
670 IFZ=0THENGOSUB1000:GOTO570              :rem 114
680 PRINT",";:RETURN                        :rem 240
690 S%=PEEK(209)+256*PEEK(210)+PEEK(211)    :rem 149
692 FORI=1TO3:T=PEEK(S%-I)                  :rem 68
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT     :rem 205
700 PRINTLEFT$("{3 LEFT}",I-1);:RETURN      :rem 7
710 PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}" :rem 236
720 INPUT"{DOWN} FILENAME";F$               :rem 228
730 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                        :rem 228
740 GETA$:IFA$<>"T"ANDA$<>"D"THEN740        :rem 36
750 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$    :rem 158
760 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                 :rem 3
762 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                     :rem 109
763 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 69
765 POKE254,S/256:POKE253,S-PEEK(254)*256:POKE780,
    253                                     :rem 12
766 POKE782,E/256:POKE781,E-PEEK(782)*256:SYS65496
                                            :rem 124
770 IF(PEEK(783)AND1)OR(ST AND191)THEN780 :rem 111
775 PRINT"{DOWN}DONE.":END                  :rem 106
780 PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN720                         :rem 171
781 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO720                             :rem 103
782 GOTO720                                 :rem 115
845 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 70
1000 REM BELL TONE                          :rem 250
1001 POKE36878,15:POKE36874,190             :rem 206
1002 FORW=1TO300:NEXTW                      :rem 117
1003 POKE36878,0:POKE36874,0:RETURN         :rem 74
2000 REM BELL SOUND                         :rem 78
2001 FORW=15TO0STEP-1:POKE36878,W:POKE36876,240:NE
     XTW                                    :rem 22
2002 POKE36876,0:RETURN                     :rem 119
```

MLX Machine Language Entry Program For VIC-20

Charles Brannon, Program Editor

MLX is a labor-saving utility that allows almost fail-safe entry of machine language programs published in COMPUTE!. *You need to know nothing about machine language to use MLX—it was designed for everyone.*

MLX is a new way to enter long machine language (ML) programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to BASIC DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255 (forbidden in ML). It won't let you enter the wrong numbers on the wrong line. In addition, MLX creates a ready-to-use tape or disk file. You can then use the LOAD command to read the program into the computer:

LOAD "filename",1,1 (for tape)
LOAD "filename",8,1 (for disk)

To start the program, you enter a SYS command that transfers control from BASIC to machine language. The starting SYS number appears in the article.

Using MLX

Type in and save MLX for your 64 (you'll want to use it in the future). When you're ready to type in an ML program, run MLX. MLX asks you for two numbers: the starting address and the ending address. These numbers are given in the article accompanying the ML program.

You'll see a prompt corresponding to the starting address. The prompt is the current line you are entering from the listing. It increases by six each time you enter a line. That's because each line has seven numbers—six actual data numbers plus a *checksum number*. The checksum verifies that you typed the previous six numbers correctly. If you enter any of the six numbers wrong, or enter the checksum wrong, the computer rings a buzzer and prompts you to reenter the line. If you enter it correctly, a bell tone sounds and you continue to the next line.

MLX accepts only numbers as input. If you make a typing error, press the INST/DEL key; the entire number is deleted. You can press it as many times as necessary back to the start of the line. If you enter three-digit numbers as listed, the computer automatically prints the comma and goes on to accept the next number. If you enter less than three digits, you can press either the SPACE bar, or RETURN key to advance to the next number. The checksum automatically appears in inverse video for emphasis.

To simplify your typing, MLX redefines part of the keyboard as a numeric keypad (lines 581–584):

	U	I	O			7	8	9
H	J	K	L	become	0	4	5	6
	M					1	2	3

MLX Commands

When you finish typing an ML listing (assuming you type it all in one session), you can then save the completed program on tape or disk. Follow the screen instructions. If you get any errors while saving, you probably have a bad disk, or the disk is full, or you've made a typo when entering the MLX program itself.

You don't have to enter the whole ML program in one sitting. MLX lets you enter as much as you want, save it, and then reload the file from tape or disk later. MLX recognizes these commands:

SHIFT-S: Save
SHIFT-L: Load
SHIFT-N: New Address
SHIFT-D: Display

When you enter a command, MLX jumps out of the line you've been typing, so we recommend you do it at a new prompt. Use the Save command to save what you've been working on. It will save on tape or disk as if you've finished, but the tape or disk won't work, of course, until you finish the typing. Remember what address you stop at. The next time you run MLX, answer all the prompts as you did before, then insert the disk or tape. When you get to the entry prompt, press SHIFT-L to reload the partly completed file into memory. Then use the New Address command to resume typing.

To use the New Address command, press SHIFT-N and enter the address where you previously stopped. The prompt will change, and you can then continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing, or else the checksum won't work. The Display command lets you display a section of your typing. After you press SHIFT-D, enter two addresses within the line number range of the listing. You can abort the listing by pressing any key.

What if you forgot where you stopped typing? Use the Display command to scan memory from the beginning to the end of the program. When you reach the end of your typing, the lines will contain a random pattern of numbers. When you see the end of your typing, press any key to stop the listing. Use the New Address command to continue typing from the proper location.

MLX: Machine Language Entry

```
100 PRINT"{CLR}{PUR}";CHR$(142);CHR$(8);
                                    :rem 181
101 POKE 788,194:REM DISABLE RUN/STOP
                                    :rem 174
110 PRINT"{RVS}{14 SPACES}"         :rem 117
120 PRINT"{RVS} {RIGHT}{OFF}E*3£{RVS}
    {RIGHT} {RIGHT}{2 SPACES}E*3{OFF}E*3
    £{RVS}£{RVS} "                  :rem 191
130 PRINT"{RVS} {RIGHT} EG3{RIGHT}
    {2 RIGHT} {OFF}£{RVS}£E*3{OFF}E*3
```

```
     {RVS} "                          :rem 232
140  PRINT"{RVS}{14 SPACES}"          :rem 120
200  PRINT"{2 DOWN}{PUR}{BLK}A FAILSAFE MA
     CHINE":PRINT"LANGUAGE EDITOR{5 DOWN}"
                                      :rem 141
210  PRINT"{BLK}{3 UP}STARTING ADDRESS":IN
     PUTS:F=1-F:C$=CHR$(31+119*F)   :rem 97
220  IFS<256ORS>32767THENGOSUB3ØØØ:GOTO21Ø
                                        :rem 2
225  PRINT:PRINT:PRINT:PRINT          :rem 123
230  PRINT"{BLK}{3 UP}ENDING ADDRESS":INPU
     TE:F=1-F:C$=CHR$(31+119*F)     :rem 158
240  IFE<256ORE>32767THENGOSUB3ØØØ:GOTO23Ø
                                      :rem 234
250  IFE<STHENPRINTC$;"{RVS}ENDING < START
     {2 SPACES}":GOSUB1ØØØ:GOTO 23Ø
                                      :rem 176
260  PRINT:PRINT:PRINT                :rem 179
300  PRINT"{CLR}";CHR$(14):AD=S       :rem 56
310  PRINTRIGHT$("ØØØØ"+MID$(STR$(AD),2),5
     );":";:FORJ=1TO6                 :rem 234
320  GOSUB57Ø:IFN=-1THENJ=J+N:GOTO32Ø
                                      :rem 228
390  IFN=-211THEN 71Ø                 :rem 62
400  IFN=-204THEN 79Ø                 :rem 64
410  IFN=-206THENPRINT:INPUT"{DOWN}ENTER N
     EW ADDRESS";ZZ                   :rem 44
415  IFN=-206THENIFZZ<SORZZ>ETHENPRINT"
     {RVS}OUT OF RANGE":GOSUB1ØØØ:GOTO41Ø
                                      :rem 225
417  IFN=-206THENAD=ZZ:PRINT:GOTO31Ø
                                      :rem 238
420  IF N<>-196 THEN 48Ø              :rem 133
430  PRINT:INPUT"DISPLAY:FROM";F:PRINT,"TO
     ";:INPUTT                        :rem 234
440  IFF<SORF>EORT<SORT>ETHENPRINT"AT LEAS
     T";S;"{LEFT}, NOT MORE THAN";E:GOTO43
     Ø                                :rem 159
450  FORI=FTOTSTEP6:PRINT:PRINTRIGHT$("ØØØ
     Ø"+MID$(STR$(I),2),5);":";       :rem 3Ø
455  FORK=ØTO5:N=PEEK(I+K):IFK=3THENPRINTS
     PC(1Ø);                          :rem 34
457  PRINTRIGHT$("ØØ"+MID$(STR$(N),2),3);"
     ,";                              :rem 157
460  GETA$:IFA$>""THENPRINT:PRINT:GOTO31Ø
                                      :rem 25
470  NEXTK:PRINTCHR$(2Ø);:NEXTI:PRINT:PRIN
     T:GOTO31Ø                        :rem 5Ø
480  IFN<Ø THEN PRINT:GOTO31Ø         :rem 168
490  A(J)=N:NEXTJ                     :rem 199
500  CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CK
     SUM=(CKSUM+A(I))AND255:NEXT      :rem 2ØØ
510  PRINTCHR$(18);:GOSUB57Ø:PRINTCHR$(2Ø)
                                      :rem 234
515  IFN=CKSUMTHEN53Ø                 :rem 255
520  PRINT:PRINT"LINE ENTERED WRONG":PRINT
     "RE-ENTER":PRINT:GOSUB1ØØØ:GOTO31Ø
                                      :rem 129
530  GOSUB2ØØØ                        :rem 218
540  FORI=1TO6:POKEAD+I-1,A(I):NEXT:rem 8Ø
550  AD=AD+6:IF AD<E THEN 31Ø         :rem 212
560  GOTO 71Ø                         :rem 1Ø8
570  N=Ø:Z=Ø                          :rem 88
580  PRINT"{+}";                      :rem 79
581  GETA$:IFA$=""THEN581             :rem 95
585  PRINTCHR$(2Ø);:A=ASC(A$):IFA=13ORA=44
     ORA=32THEN67Ø                    :rem 229
590  IFA>128THENN=-A:RETURN           :rem 137
600  IFA<>2Ø THEN 63Ø                 :rem 1Ø
610  GOSUB69Ø:IFI=1ANDT=44THENN=-1:PRINT"
     {LEFT} {LEFT}";:GOTO69Ø          :rem 172
620  GOTO57Ø                          :rem 1Ø9
630  IFA<48ORA>57THEN58Ø              :rem 1Ø5
640  PRINTA$;:N=N*1Ø+A-48             :rem 1Ø6
650  IFN>255 THEN A=2Ø:GOSUB1ØØØ:GOTO6ØØ
                                      :rem 229
660  Z=Z+1:IFZ<3THEN58Ø               :rem 71
670  IFZ=ØTHENGOSUB1ØØØ:GOTO57Ø       :rem 114
680  PRINT",";:RETURN                 :rem 24Ø
690  S%=PEEK(2Ø9)+256*PEEK(21Ø)+PEEK(211)
                                      :rem 149
692  FORI=1TO3:T=PEEK(S%-I)           :rem 68
695  IFT<>44ANDT<>58THENPOKES%-I,32:NEXT
                                      :rem 2Ø5
700  PRINTLEFT$("{3 LEFT}",I-1);:RETURN
                                        :rem 7
710  PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}"
                                      :rem 236
720  INPUT"{DOWN} FILENAME";F$        :rem 228
730  PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR
      {RVS}D{OFF}ISK: (T/D)"          :rem 228
740  GETA$:IFA$<>"T"ANDA$<>"D"THEN74Ø
                                      :rem 36
750  DV=1-7*(A$="D"):IFDV=8THENF$="Ø:"+F$
                                      :rem 158
760  T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$
     ):POKE782,ZK/256                   :rem 3
762  POKE781,ZK-PEEK(782)*256:POKE78Ø,LEN(
     T$):SYS65469                     :rem 1Ø9
763  POKE78Ø,1:POKE781,DV:POKE782,1:SYS654
     66                               :rem 69
765  POKE254,S/256:POKE253,S-PEEK(254)*256
     :POKE78Ø,253                     :rem 12
766  POKE782,E/256:POKE781,E-PEEK(782)*256
     :SYS65496                        :rem 124
770  IF(PEEK(783)AND1)OR(ST AND191)THEN78Ø
                                      :rem 111
775  PRINT"{DOWN}DONE.":END           :rem 1Ø6
780  PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}T
     RY AGAIN.":IFDV=1THEN72Ø         :rem 171
781  OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$
     ;E2$:CLOSE15:GOTO72Ø             :rem 1Ø3
782  GOTO72Ø                          :rem 115
790  PRINT"{CLR}{RVS}*** LOAD ***{2 DOWN}"
                                      :rem 212
800  INPUT"{2 DOWN} FILENAME";F$      :rem 244
810  PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR
      {RVS}D{OFF}ISK: (T/D)"          :rem 227
820  GETA$:IFA$<>"T"ANDA$<>"D"THEN82Ø
                                      :rem 34
830  DV=1-7*(A$="D"):IFDV=8THENF$="Ø:"+F$
                                      :rem 157
840  T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$
     ):POKE782,ZK/256                   :rem 2
841  POKE781,ZK-PEEK(782)*256:POKE78Ø,LEN(
     T$):SYS65469                     :rem 1Ø7
845  POKE78Ø,1:POKE781,DV:POKE782,1:SYS654
     66                               :rem 7Ø
850  POKE78Ø,Ø:SYS65493               :rem 11
860  IF(PEEK(783)AND1)OR(ST AND191)THEN87Ø
                                      :rem 111
865  PRINT"{DOWN}DONE.":GOTO31Ø       :rem 96
870  PRINT"{DOWN}ERROR ON LOAD.{2 SPACES}T
     RY AGAIN.{DOWN}":IFDV=1THEN8ØØ
                                      :rem 172
880  OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$
     ;E2$:CLOSE15:GOTO8ØØ             :rem 1Ø2
1000 REM BUZZER                       :rem 135
1001 POKE36878,15:POKE36874,19Ø       :rem 2Ø6
1002 FORW=1TO3ØØ:NEXTW                :rem 117
1003 POKE36878,Ø:POKE36874,Ø:RETURN
                                      :rem 74
2000 REM BELL SOUND                   :rem 78
2001 FORW=15TOØSTEP-1:POKE36878,W:POKE368
     76,24Ø:NEXTW                     :rem 22
2002 POKE36876,Ø:RETURN               :rem 119
3000 PRINTC$;"{RVS}NOT ZERO PAGE OR ROM":
     GOTO1ØØØ                         :rem 89
```

It all adds up...

HOME COMPUTERS

ATARI

WHILE SUPPLY LASTS
600XL....CALL
800XL....CALL

Call for new low pricing
1200XL....CALL
1450XL....CALL

850 Interface	$159.00
1010 Recorder	$71.99
1020 Color Printer	$219.00
1025 Dot Matrix Printer	$299.00
1027 Letter Quality Printer	$269.00
1030 Direct Connect Modem	$99.99
1050 Disk Drive	$349.00
1064 Memory Module	$125.00
Touch Tablet/Software	$64.99
Light Pen/Software	$72.99
CX22 Track Ball	$39.99
7097 Atari Logo	$74.99
4018 Pilot (Home)	$57.99
405 Pilot (Educ.)	$99.99
8036 Atari Pilot	$77.99
5049 VisiCalc	$149.99
488 Communicator II	$119.99
CX30 Paddles	$11.99
CX40 Joystick	$7.99
4011 Star Raiders	$31.99
4022 Pac Man	$31.99
4025 Defender	$31.99
8026 Dig Dug	$31.99
8031 Donkey Kong	$35.99
8034 Pole Position	$37.99
8040 Donkey Kong Jr.	$33.99
8043 Ms Pacman	$37.99
8044 Joust	$37.99
8045 Pengo	$33.99
8052 Moon Patrol	$33.99
4003 Assembler	$44.99
8126 Microsoft Basic I or II	$64.99

DISK DRIVES FOR ATARI

PERCOM

AT 88-S1	$279.00
AT 88-S1 PD	$299.00

INDUS

GT Drive	$379.00

RANA

1000	$299.00

TRAK

AT-D2	$389.00
AT-D4	$589.00

MEMORY BOARDS

ATARI

Axlon 32K	$59.99
Axlon 48K	$99.99
Axlon 128K	$299.00

APPLE/FRANKLIN

Axlon 128K	$299.00
Axlon 320K	$849.00

ALIEN VOICE BOX

Atari	$119.00
Apple	$149.00

DISKETTES

MAXELL

5¼" MD-1	$21.99
5¼" MD-2	$29.99
8" FD-1	$39.99
8" FD-2	$49.99

VERBATIM

5¼" SS/DD	$26.99
5¼" DS/DD	$34.99

BIB

5¼" Disk Head Cleaner	$14.99

ELEPHANT

5¼" SS/SD	$17.99
5¼" SS/DD	$21.99
5¼" DS/DD	$26.99

DISK HOLDERS
INNOVATIVE CONCEPTS

Flip-n-File 10	$3.99
Flip-n-File 50	$17.99
Flip-n-File 50 w/lock	$24.99
Flip-n-File (400/800 ROM)	$17.99

CONTROLLERS & JOYSTICKS

WICO

Joystick	$21.99
3-way Joystick	$22.99
Power Grip	$21.99
BOSS Joystick	$17.99
ATARI/VIC Trak Ball	$34.99
Apple Trak Ball	$54.99
Apple Analog	$37.99

KRAFT

Joystick	$41.99
Atari Single Fire	$12.99
Atari Switch Hitter	$15.99
Apple Paddles	$34.99
IBM Paddles	$34.99
IBM Joystick	$46.99

Koala

Atari (ROM)	$79.99
C-64 (ROM)	$79.99
IBM	$99.99
Apple/Franklin	$85.99

PRINTERS

AXIOM

AT-100 Atari Interface Printer	$199.00
GP-100 Parallel Interface	$189.00
GP-550 Atari Bidirectional	$319.00
GP-700 Atari Color Printer	$489.00
GP-550 Parallel Printer	$269.00

BMC

401 Letter Quality	$589.00
BX-80 Dot Matrix	$269.00

C.ITOH

Gorilla Banana	$149.00
Prowriter 8510P	$379.00
Prowriter 1550P	$599.00
A10 (18 cps) Son of Starwriter	$569.00
Hot Dot Matrix	CALL
F10-40 Starwriter	$999.00
F10-55 Printmaster	$1349.00

COMREX

ComWriter II Letter Quality	$449.00

DIABLO

620 Letter Quality	$949.00
630 Letter Quality	$1749.00

DAISYWRITER

2000	$999.00
Tractor Feed (Uni)	$109.00

EPSON

RX-80, RX-80FT, RX-100	CALL
FX-80, FX-100	CALL
LQ 1500	CALL

IDS

Prism 80...For Configuration	CALL
Prism 32...For Configuration	CALL

JUKI

6100	$469.00

MANNESMAN TALLY

160L	$589.00
180L	$799.00
Spirit 80	$289.00

NEC

8027 Dot Matrix	$389.00
Pinwriter	CALL
2010/15/30	$749.00
3510/15/30	$1369.00
7710/15/30	$1799.00

OKIDATA

82, 83, 84, 92, 93, 2350, 2410	CALL

OLYMPIA

Compact 2	$479.00
Compact RO	$509.00
ESW 3000	$1449.00

SMITH CORONA

TP-1000	$449.00
Tractor Feed	$119.00

SILVER REED

500 Letter Quality	$449.00
550 Letter Quality	$549.00
770 Letter Quality	$869.00

STAR

Gemini 10X	$279.00
Gemini 15X	$389.00
Serial Board	$75.00
Radix 10	$599.00
Radix 15	$699.00

TOSHIBA

1340	$829.00
1351	$1499.00

TRANSTAR

120P	$469.00
130P	$649.00
315 Color	$459.00

MODEMS

ANCHOR

Volksmodem	$59.99
Mark IL Serial	$79.99
Mark VII (Auto Ans/Auto Dial)	$99.99
Mark XII (1200 Baud)	$269.99
Mark TRS-80	$99.99
9 Volt Power Supply	$9.99

HAYES

Smartmodem 300	$199.00
Smartmodem 1200	$499.00
Smartmodem 1200B	$449.00
Micromodem IIe	$269.00
Micromodem 100	$299.00
Smart Com II	$89.99
Chronograph	$199.00

NOVATION

J-Cat	$99.99
Cat	$139.00
Smart Cat 103	$179.00
Smart Cat 103/212	$399.00
AutoCat	$219.00
212 AutoCat	$549.00
Apple Cat II	$249.00
212 Apple Cat	$449.00
Apple Cat 212 Upgrade	$259.00
Smart Cat Plus	$369.00

ZENITH

ZT-1	$339.00
ZT-10	$309.00
ZT-11	$369.00

MONITORS

AMDEK

300 Green	$149.00
300 Amber	$159.00
310 Amber	$169.00
Color 1	$269.00
Color 1 Plus	$289.00
Color 2 Plus	$399.00
New Color 300,400,500,600,700	Call
Color 4T IBM	$599.00

BMC

1201 (12" Green)	$88.99
1201 Plus (12" Green Hi-Res)	$98.99
9191 Plus	$249.00

GORILLA

12" Green	$88.99
12" Amber	$95.99

NEC

JB 1260 Green	$109.00
JB 1201 Green	$149.99
JB 1205 Amber	$159.99
JB 1215 Color	$259.00
JC 1216 RGB	$429.00
JC 1460 Color	$359.00

PRINCETON GRAPHICS

MAX-12 Amber	$199.00
HX-12 RGB	$499.00
SRT-2 RGB	$649.00

SAKATA

SC-100 Color	$269.00
SG-1000 Green	$129.00
SA-1000 Amber	$139.00

TAXAN

210 Color RGB	$269.00
400 Med-Res RGB	$319.00
415 Hi-Res RGB	$439.00
420 Hi-Res RGB (IBM)	$469.00
100 12" Green	$125.00
105 12" Amber	$135.00

USI

Pi 1, 9" Green	$99.99
Pi 2, 12" Green	$119.99
Pi 3, 12" Amber	$129.99
Pi 4, 9" Amber	$119.99
1400 Color	$269.99

QUADRAM

Quadchrome 8400 Color	$499.00

ZENITH

ZVM122 Amber	$99.99
ZVM123 Green	$89.99
ZVM124 IBM-Amber	$149.00
ZVM131 Color	$309.00
ZVM133 RGB	$429.00
ZVM135 RGB/Composite	$469.00

COMPUTER MAIL ORDER

west
MasterCard
800-648-3311
In NV call (702)588-5654, Dept. 105
Order Status Number: 588-5654
P.O.Box 6689
Stateline, NV 89449

canada
Ontario/Quebec 800-268-3974
Other Provinces 800-268-4559
In Toronto call (416)828-0866, Dept. 105
Order Status Number: 828-0866
2505 Dunwin Drive, Unit 3B
Mississauga, Ontario, Canada L5L1T1

east
VISA
800-233-8950
In PA call (717)327-9575, Dept. 105
Order Status Number: 327-9576
Customer Service Number: 327-1450
477 E. 3rd St., Williamsport, PA 17701

No risk, no deposit on C.O.D. orders and no waiting period for certified checks or money orders. Add 3% (minimum $5) shipping and handling on all orders. Larger shipments may require additional charges. NV and PA residents add sales tax. All items subject to availability and price change. Call today for our catalog.

the best prices

APPLE/FRANKLIN DISK DRIVES

MICRO-SCI

A2 .... $199.00
A20 .... Call
A40 .... $299.00
A70 .... $319.00

RANA

Elite 1 .... $279.00
Elite 2 .... $389.00
Elite 3 .... $569.00

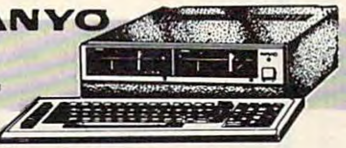
APPLE IIe STARTER PACK

64K Apple IIe, Disk Drive & Controller, 80 Column Card, Monitor II & DOS 3.3
COMPLETE .... CALL
Call on all other Apple Models

FRANKLIN

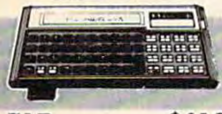
ACE 1000 Color Computer .... CALL
ACE PRO PLUS System .... CALL
ACE 1200 Office Mgmt. System .... CALL
ACE PORTABLES .... CALL

SANYO



MBC 550 .... CALL
MBC 550-2 .... CALL
MBC 555 .... CALL
MBC 555-2 .... CALL

HEWLETT PACKARD



HP 71B $419.99
41CV $189.99
41CX $249.99

HP 11C $62.99
HP 12C $92.99
HP 15C $92.99
HP 16C $92.99
HP 75D $879.99
HPIL Module $98.99
HPIL Cassette or Printer $359.99
Card Reader $143.99
Extended Function Module $63.99
Time Module $63.99

SHARP

PC-1500A $165.99
PC-1250A $88.99
CE-125 Printer/Cassette $128.99
CE-150 Color Printer/Cassette $171.99
CE-155 8K RAM $93.99
CE-161 16K RAM $134.99
CE-500 ROM Library $29.99

NEC

$469

NEC

PC-8221A Thermal Printer $149.99
PC-8281A Data Recorder $99.99
PC-8201-06 8K RAM Chips $105.00
PC-8206A 32K RAM Cartridge $329.00

commodore

CBM 8032 .... $639
CBM 8096 .... $869.00
CBM 9000 .... $999.00
B128-80 .... $769.00
8032 to 9000 Upgrade $499.00
2031 LP Disk Drive $299.00
8050 Disk Drive $999.00
8250 Disk Drive $1249.00
4023 Printer $399.00
8023 Printer $589.00
6400 Printer $1449.00
Z-RAM $369.00
Silicon Office $699.00
The Manager $199.00
SoftROM $125.00
VisiCalc $159.00

PROFESSIONAL SOFTWARE

Word Pro 2 Plus $159.00
Word Pro 3 Plus $189.00
Word Pro 4 Plus/5 Pluseach $279.00
InfoPro $179.00
Administrator $399.00
Power $79.99

SX-64Portable $839
CBM 64 .... $199
C1541 Disk Drive $249.00
C1530 Datasette $69.99
C1520 Color Printer/Plotter $129.00
M-801 Dot Matrix Printer $219.00
C1526 Dot Matrix/Serial $299.00
C1702 Color Monitor $259.00
C1311 Joystick $4.99
C1312 Paddles $11.99
C1600 VIC Modem $59.99
C1650 Auto Mdoem $89.99
Logo 64 .... $49.99
Pilot 64 .... $39.99
Word Pro 64 Plus $59.99
Calc Result 64 .... $65.99
Calc Result Easy $39.99
MCS 801 Color Printer $499.00
DPS 1101 Daisy Printer $459.00
Magic Voice Speech Module $54.99
Desk Organizer Lock $49.99
Vidtex Telecommunications $34.95

MSD

SD1 Disk Drive $349.00
SD2 Disk Drive $599.00

NEC PRINTERS

NEC 2050 .... $899.00
NEC 3550 .... $1599.00

PERCOM/TANDON DISK DRIVES

5¼" 320K Floppy $199.00
5 Meg Hard w/Controller $1049.00
10 Meg Hard w/Controller $1349.00
20 Meg Hard w/Controller $1899.00

VISICORP

VisiCalc IV $159.00
VisiWord + ... $249.00
Visi-on 'Series' .... CALL
Optical Mouse $189.99

IDEAssociates

5MB to 45MB Hard drives with removable Cart. backupas low as $1399

AST RESEARCH

Six Pak Plus .... from ... $249.00
Combo Plus II .... from ... $279.00
Mega Plus .... from ... $299.00
I/O Plus .... from ... $139.00

QUADRAM

Quadlink $479.00
Quadboard .... as low as ... $309.00
Quad 512 Plus ...... as low as ... $259.00
Quadcolor .... as low as ... $239.00
Chronograph $89.99
Parallel Interface Board $89.99
64K RAM Chips Kit $49.99

MICROPRO

WordStar Professional Pack $369.00

MICROMIM

R:Base 4000 .... CALL

MULTIMATE INT.

Multi Mate $289.00

MICROSTUF

Crosstalk $105.00

MICROSOFT

MultiPlan $159.00

ASHTON-TATE

dBASE II $299.00
dBASE III $429.00
Friday! $179.00

IUS

EasyWriter II $249.00
EasySpeller $119.00
EasyFiler $229.00

CONTINENTAL SOFTWARE

1st Class Mail/Form Letter $79.99
Home Accountant Plus $88.99

LOTUS

Symphony $519.00
1-2-3 .... $339.00

PROFESSIONAL SOFTWARE

PC Plus/The Boss $309.00

SYNAPSE

File Manager $89.99

ZENITH | data systems

PC COMPATIBLE
16 BIT SYSTEMS

Call for price and configurations

SOFTWARE

	C-64	Atari	IBM	Apple
Electronic Arts				
One on One	$29.99	$29.99	$29.99	$29.99
Music Construction	$29.99	$29.99	$29.99	$29.99
Pinball Construction	$29.99	$29.99	$29.99	$29.99
Cut & Paste	$39.99	$39.99	$39.99	$39.99
Hard Hat Mack	$27.99	$27.99	$27.99	$27.99
InfoCom				
Witness	$29.99	$29.99	$29.99	$29.99
Infidel	$29.99	$29.99	$29.99	$29.99
Deadline	$29.99	$29.99	$29.99	$29.99
Planetfall	$29.99	$29.99	$29.99	$29.99
Enchanter	$29.99	$29.99	$29.99	$29.99
Zork1,2,3 ea	$27.99	$27.99	$27.99	$27.99
Suspended	$29.99	$29.99	$29.99	$29.99
Sorcerer	$29.99	$29.99	$29.99	$29.99
AtariSoft				
Joust	$34.99	N/A	$28.99	$28.99
Moon Patrol	$34.99	N/A	$28.99	$28.99
Ms. PacMan	$34.99	N/A	$28.99	$28.99
PacMan	$34.99	N/A	$28.99	$28.99
Donkey Kong	$34.99	N/A	$28.99	$28.99
Pole Position	$34.99	N/A	$28.99	$28.99
Spinnaker				
Aerobics	$28.99	$28.99	$28.99	$28.99
Trans	$24.99	$24.99	$24.99	$24.99
Adventure Creature	$24.99	$24.99	$24.99	$24.99
Aegean Voyage	$24.99	$24.99	$24.99	$24.99
Snooper Troops 1,2 ea	$28.99	$28.99	$28.99	$28.99
Traction Fever	$22.99	$22.99	$22.99	$22.99
Alphabet Zoo	$22.99	$22.99	$22.99	$22.99
In Search of...	$24.99	$24.99	$24.99	$24.99
Facemaker	$22.99	$22.99	$22.99	$22.99
Kinder Comp	$17.99	$17.99	$17.99	$17.99
Dynatech				
Code Writer	$75.99	$79.99	$175.99	$155.99
VisiCorp				
VisiCalc	$159.99	$149.99	$159.99	$159.99
VisiCalc Advanced	N/A	N/A	N/A	$269.99
pfs:				
Write	N/A	N/A	$89.99	$89.99
Graph	N/A	N/A	$89.99	$79.99
Report	N/A	N/A	$79.99	$79.99
File	N/A	N/A	$89.99	$79.99
Solutions:* as low as	N/A	N/A	$16.99	$16.99

*Call on Titles

COMPUTER MAIL ORDER

west
800-648-3311
In NV call (702)588-5654, Dept. 105
Order Status Number: 588-5654
P.O.Box 6689
Stateline, NV 89449

canada
Ontario/Quebec 800-268-3974
Other Provinces 800-268-4559
In Toronto call (416)828-0866, Dept. 105
Order Status Number: 828-0866
2505 Dunwin Drive, Unit 3B
Mississauga, Ontario, Canada L5L1T1

east
800-233-8950
In PA call (717)327-9575, Dept. 105
Order Status Number: 327-9576
Customer Service Number: 327-1450
477 E. 3rd St., Williamsport, PA 17701

CANADIAN ORDERS: All prices are subject to shipping, tax and currency fluctuations. Call for exact pricing in Canada. INTERNATIONAL ORDERS: All orders placed with U.S. offices for delivery outside the Continental United States must be pre-paid by certified [illegible] only. Include 3% (minimum $5) shipping and handling. EDUCATIONAL DISCOUNTS: Additional discounts are available to qualified Educational Instit[illegible] add 3% (minimum $5) shipping and handling.

PRODUCT MART

Advertisers Index

The Editor's Feedback:

Computer: ☐ PET ☐ Apple ☐ Atari ☐ VIC-20 ☐ Commodore 64
☐ TI-99/4A ☐ Timex/Sinclair ☐ Radio Shack Color Computer
☐ Other ________________ ☐ Don't yet have one ...

Are you a COMPUTE! Subscriber? ☐ Yes ☐ No I would like to see:

More	Just Right	Fewer	
☐	☐	☐	Specific applications programs.
☐	☐	☐	BASIC programs.
☐	☐	☐	Machine language programs.
☐	☐	☐	Tutorials.
☐	☐	☐	Educational articles.
☐	☐	☐	Detailed explanations of programs.

More	Just Right	Fewer	
☐	☐	☐	Games.
☐	☐	☐	Reviews of game software.
☐	☐	☐	Reviews of business software.
☐	☐	☐	Reviews of educational software.
☐	☐	☐	Reviews of hardware.

What do you like best about COMPUTE! ?

What do you like least?

010118C

Place
Postage
Here

COMPUTE! Magazine
P.O. Box 5406
Greensboro, NC 27435-0406

COMPUTE!'s
FREE Reader Information Service

Use these cards to request FREE information about the products advertised in this issue. Clearly print or type your full name and address. Only one card should be used per person. Circle the numbers that correspond to the key number appearing in the advertisers index.

Send in the card and the advertisers will receive your inquiry. Although every effort is made to insure that only advertisers wishing to provide product information have reader service numbers, COMPUTE! cannot be responsible if advertisers do not provide literature to readers.

Please use these cards *only* for subscribing or for requesting product information. Editorial and customer service inquiries should be addressed to: COMPUTE!, P.O. Box 5406, Greensboro, NC 27403. Check the expiration date on the card to insure proper handling.

Use these cards and this address only for COMPUTE!'s Reader Information Service. Do not send with payment in any form.

COMPUTE!

101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122
123	124	125	126	127	128	129	130	131	132	133
134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155
156	157	158	159	160	161	162	163	164	165	166
167	168	169	170	171	172	173	174	175	176	177
178	179	180	181	182	183	184	185	186	187	188
189	190	191	192	193	194	195	196	197	198	199
200	201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220	221
222	223	224	225	226	227	228	229	230	231	232
233	234	235	236	237	238	239	240	241	242	243
244	245	246	247	248	249	250	251	252	253	254
255	256	257	258	259	260	261	262	263	264	265
266	267	268	269	270	271	272	273	274	275	276
277	278	279	280	281	282	283	284	285	286	287
288	289	290	291	292	293	294	295	296	297	298
299	300	301	302	303	304	305	306	307	308	309
310	311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330	331
332	333	334	335	336	337	338	339	340	341	342
343	344	345	346	347	348	349	350			

Circle 101 for a one year new U.S. subscription to COMPUTE!: you will be billed for $24.

Please print or type your full name and address. Limit one card per person.

Name ______________________

Address ______________________

City ______________________

State/Province ______________ Zip ______

Country ______________________

Please include zip code. Expiration 11/30/84. CO984

Place
Postage
Here

COMPUTE! Reader Service
P.O. Box 2141
Radnor, PA 19089

Turn your CRT Monitor into a television set with Cardco's new monitor tuners.

With a flick of the switch you can turn your computer monitor to provide the most vivid television pictures available. Receive a sharper, clearer television picture on your CRT monitor than you ever thought possible. Model MT/1 includes a full wireless remote control sixty-channel cable ready tuner with separate video and audio output. Completely compatible with your composite input computer monitor. For composite color or monochrome.

Also available to turn your CRT monitor into a television set is CARDCO Model MT/2, a full UHF/VHF TV tuner with separate video and audio output matched to your composite input computer monitor. For composite color or monochrome monitors.

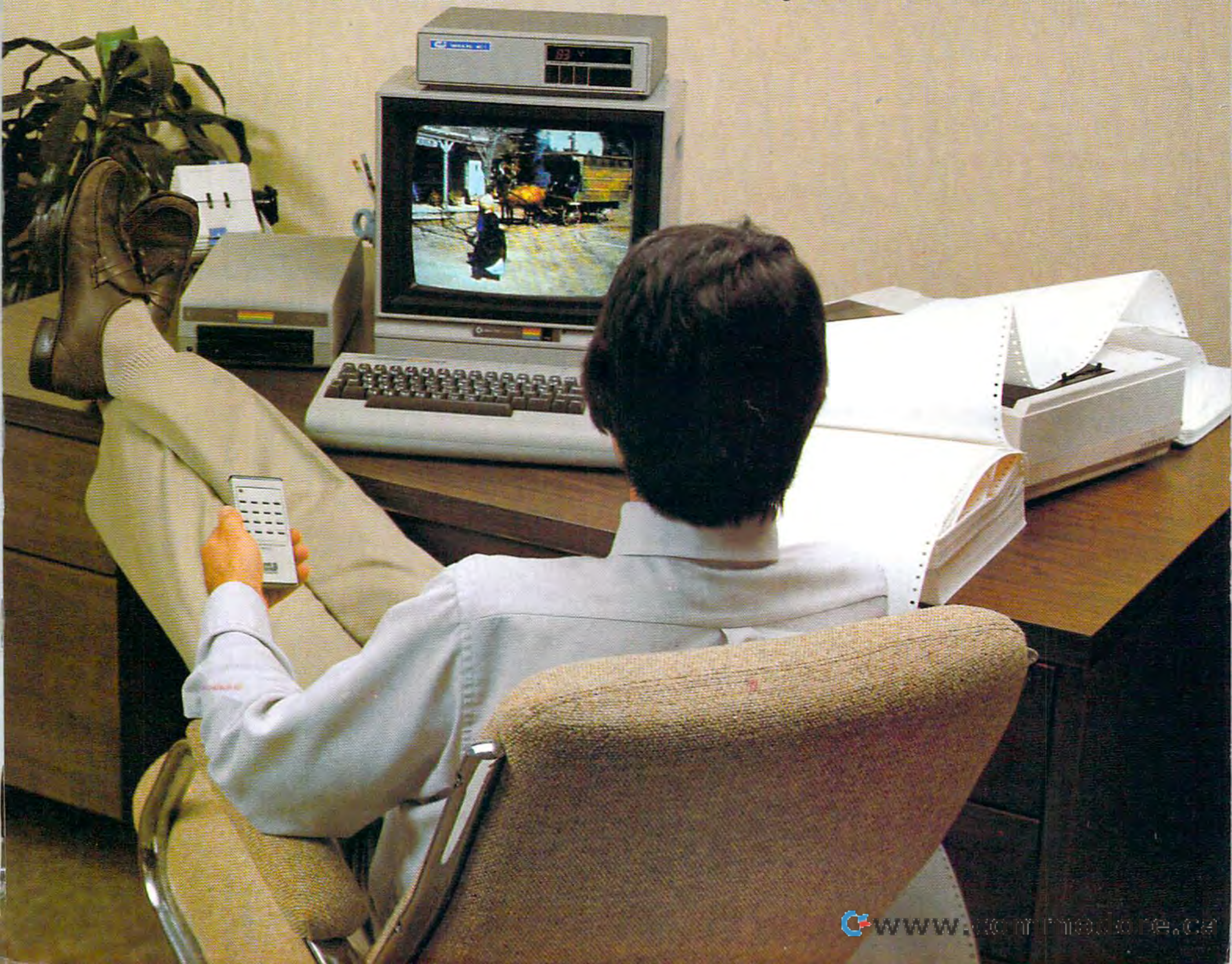
Backed by a full 90 day guarantee.

Write for illustrated literature and prices or see CARDCO Computer Accessories and Software wherever computers are sold.

cardco, inc.

300 S. Topeka Wichita, Kansas 67202
(316) 267-6525

"The world's largest manufacturer of Commodore accessories."



HOW TO BECOME COMPUTER LITERATE.

FEATURES	COMMODORE 64™	APPLE IIe™	IBM PC jr™	ATARI 800XL™
Price*	$219	$699	$669	$299
Built-in Memory	64K	64K	64K	64K
Typewriter Keyboard	YES (66 Keys)	YES (62 Keys)	"CHICKLET" (62 Keys)	YES (61 Keys)
Upper/Lower Case	YES	YES	YES	YES
Programmable Function Keys	YES	NO	YES	NO
AUDIO				
Polyphonic Tones	YES	NO	YES	YES
Music Synthesizer	YES	NO	NO	NO
Hi-Fi Output	YES	NO	YES	YES
VIDEO				
TV Output	YES	EXTRA COST	EXTRA COST	YES
Video Monitor Output	YES	YES	EXTRA COST	YES
INPUT/OUTPUT				
Intelligent I/O Bus	YES	NO	NO	YES
RS-232 Communications	YES**	EXTRA COST	YES**	EXTRA COST
"Smart" Peripherals	YES	NO	NO	YES

*Prices shown are common retail and may vary slightly in different markets.
**Requires an adapter to operate.

First you need the right input.

Like $219. That's what the Commodore 64™ costs. It's about one third the price of the Apple IIe™ or the IBM® PCjr.™

And 64K. That's how much memory the Commodore 64 has. It's also how much memory Apple IIe and the IBM PCjr have.

This computer lesson is brought to you as a public service by Commodore (certainly not by Apple or IBM), the only computer company that can afford to show you a chart like the one above.

But what you can't see above are the thousands of software programs that make the Commodore 64 fully capable of doing anything any "triple the price" computer can do; for fun or profit, for every member of the family; anything from soccer to spread sheets to space exploration.

Because the Commodore is so affordable, you can load up on Commodore peripherals. Like a disk drive, a printer or a telephone modem. All together they cost just a tad more than an IBM PCjr by itself. With no peripherals.

No wonder Commodore sells more computers than Apple and IBM combined.

commodore COMPUTERS

IT'S NOT HOW LITTLE IT COSTS, IT'S HOW MUCH YOU GET.