

For your  
TANDY  
Color Computer

Registered by Australia Post — Publication No. QBG 4009

PNG K4.95 NZ \$5.20

\$4.50

AUSTRALIAN

# RAINBOW

March, 1986

No.57



Our  
**BIG**  
UTILITY  
ISSUE!

- ★ Programmes to help control robots
- ★ Assembly Language
- ★ Sound Experiments
- ★ Disk Helps

**PLUS!! HARDWARE COURSE**

TANDY ELECTRONICS DEALER.(No 9320)

# TANDY COMPUTERS & ACCESSORIES

best prices!

FREE DELIVERY THROUGH AUSTRALIA  
90 DAYS WARRANTY



DISK DRIVEN FOR CoCo  
40 TRACK DSD

DISKECB 1.4 AUTO LINE  
NUMBERING SUPPORTS FLEX & OS9.

6MS Access time  
inc Controller and Manual \$599

## BAYNE & TREMBATH

3 Boneo Rd., Rosebud, Victoria 3940  
Ph. (059) 86 8288. A/h (059) 85 4947

Bankcard &  
Cheque Orders  
accepted

### BLAXLAND COMPUTER SERVICES PTY. LTD.

### COCO SPECIALISTS

(047) 39-3903

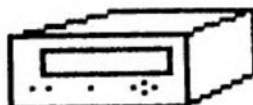
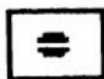
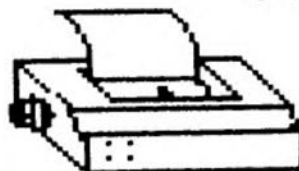
### £ 1000

LARGEST RANGE OF  
SOFTWARE - BOOKS  
AND ACCESSORIES.

(TANDY DEALER 9254)



### CHECK OUR \* PRICES



MAIL ORDER  
BANKCARD  
WELCOME

76A MURPHY ST. BLAXLAND 2774

# Computer Hut Software

Are proud to announce that we are now the sole Australian distributors for :-

MARK DATA PRODUCTS	* COGNITEC
COMPUTER ISLAND	* DERRINGER SOFTWARE
SPECTRAL ASSOCIATES	* TRIAD PICTURE CORP.
TOM MIX SOFTWARE	* VIP TECHNOLOGIES
SUGAR SOFTWARE	* COMPUTER STORY BOOKS
PRICKLY PEAR SOFTWARE	* PICOSOFT
SPECTRUM PROJECTS	* COASTAL COMPUTER SERVICES
PAL CREATIONS	* NOVASOFT
OWLS NEST SOFTWARE	* COCO CASSETTE (monthly)

We have about 300 programs with more on the way.

SEE YOUR LOCAL TANDY DEALER

or send \$1-00 for a full catalogue to:-

We accept :-

- \* BANKCARD
- \* MASTERCARD
- \* VISACARD

Computer Hut Software  
21, Williams Street.  
Bowen, Qld. 4805.  
Phone (077) 86-2220

**CHILD'S PLAY™**  
EDUCATIONAL LEVEL  
WORD PROCESSOR

**CHILD WRITER**

**CHILD'S PLAY™**  
FAMILY LEVEL  
WORD PROCESSOR

**MEMO WRITER**

WE ARE STOCKISTS OF 1000 & CO CO  
PROGRAMMES FOR A WIDE RANGE  
OF CATEGORIES

**AI:TYPIST**  
AI:AIRUS™

WORD PROCESSING WITH  
INSTANT SPELL CHECKING

Just **\$155.00**

Requires 256K T1000

**ENCOM**

AVAILABLE FROM —  
OR YOUR TANDY STORE

2nd Floor, 20 McDougall St.  
Milton, Qld. 4064  
Telephone: 369 8399

**EDUCATION  
SOFTWARE**

# RAINBOW

# Info

## How To Read Rainbow

Please note that all the BASIC program listings you find in THE RAINBOW are formatted for a 32-character screen — so they show up just as they do on your CoCo screen. One easy way to check on the accuracy of your typing is to compare what character "goes under" what. If the characters match — and your line endings come out the same — you have a pretty good way of knowing that your typing is accurate.

We also have "key boxes" to show you the *minimum* system a program needs. But, *do* read the text before you start typing.

Finally, the little cassette symbol on the table of contents and at the beginning of articles indicates that the program is available through our RAINBOW ON TAPE service. An order form for this service is on the insert card bound in the magazine.

## What's A CoCo

CoCo is an affectionate name that was first given to the Tandy Color Computer by its many fans, users and owners.

However, when we use the term CoCo, we refer to both the Tandy Color Computer and the TDP System-100 Computer. It is easier than using both of the "given" names throughout THE RAINBOW.

In most cases, when a specific computer is mentioned, the application is for that specific computer. However, since the TDP System-100 and Tandy Color are, for all purposes, the same computer in a different case, these terms are almost always interchangeable.

## The Rainbow Check Plus



The small box you see accompanying a program listing in THE RAINBOW is a "check sum" system, which is designed to help you type in programs accurately.

*Rainbow Check PLUS* counts the number and values of characters you type in. You can then compare the number you get to those printed in THE RAINBOW. On longer programs, some benchmark lines are given. When you reach the end of one of those lines with your typing, simply check to see if the numbers match.

To use *Rainbow Check PLUS*, type in the program and *CSAVE* it for later use, then type in the command *RUN* and press *ENTER*. Once the program has run, type *NEW* and *ENTER* to remove it from the area where the program you're typing in will go.

Now, while keying in a listing from THE RAINBOW, whenever you press the down-arrow key, your CoCo gives the check sum based on the length and content of the program in memory. This is to check against the numbers printed in THE RAINBOW. If your number is different, check the listing carefully to be sure you typed in the correct BASIC program code. For more details on this helpful utility, refer to H. Allen Curtis' article on Page 21 of the February 1984 RAINBOW.

Since *Rainbow Check PLUS* counts spaces and punctuation, be sure to type in the listing exactly the way it's given in the magazine.

```
10 CLS:X=256*PEEK(35)+178
20 CLEAR 25,X-1
30 X=256*PEEK(35)+178
40 FOR Z=X TO X+77
50 READ Y:W=W+Y:PRINT Z,Y:W
60 POKE Z,Y:NEXT
70 IF W=7985 THEN B0 ELSE PRINT
  "DATA ERROR":STOP
80 EXEC X:END
90 DATA 182, 1, 106, 167, 140, 60, 134
100 DATA 126, 183, 1, 106, 190, 1, 107
110 DATA 175, 140, 50, 48, 140, 4, 191
120 DATA 1, 107, 57, 129, 10, 38, 38
130 DATA 52, 22, 79, 158, 25, 230, 129
140 DATA 39, 12, 171, 128, 171, 128
150 DATA 230, 132, 38, 250, 48, 1, 32
160 DATA 240, 183, 2, 222, 48, 140, 14
170 DATA 159, 166, 166, 132, 28, 254
180 DATA 189, 173, 198, 53, 22, 126, 0
190 DATA 0, 135, 255, 134, 40, 55
200 DATA 51, 52, 41, 0
```

## Using Machine Language

Machine language programs are one of the features of THE RAINBOW. There are a number of ways to "get" these programs into memory so you can operate them.

The easiest way is by using an editor/assembler, a program you can purchase from a number of sources.

An editor/assembler allows you to enter mnemonics into your CoCo and then have the editor/assembler assemble them into specific instructions that are understood by the 6809 chip that controls your computer.

When you use an editor/assembler, all you have to do, essentially, is copy the relevant instructions from THE RAINBOW's listing into CoCo.

Another method of getting an assembly language listing into CoCo is called "hand assembly." As the name implies, you do the assembly by hand. This can *sometimes* cause problems when you have to set up an *ORIGIN* statement or an *EQUATE*. In short, you have to know something about assembly to hand-assemble some programs.

Use the following program if you wish to hand-assemble machine language listings:

```
10 CLEAR 200,&H3F00:I=&H3F80
20 PRINT "ADDRESS:";HEX$(I):
30 INPUT "BYTE":B$
40 POKE I,VAL("&H"+B$)
50 I=I+1:GOTO 20
```

This program assumes you have a 16K CoCo. If you have 32K, change the &H3F00 in Line 10 to &H7F00 and change the value of I to &H7F80.

## The Crew

**Founder** Greg Wilson  
**Publishers** Graham & Annette Morphett  
**Managing Editor** Graham Morphett  
**Accounts** Annette Morphett  
**Assistand Editor** Sonya Young  
**Advertising** Tracey Yapp

**Art** Jim Bertick  
**Sub Editors**

Assembly Language: Kevin Mischewski  
MC-10: Jim Rogers  
softgold: Barry Cawley  
Forth: John Poxon  
OS-9: Jack Fricker

**Special Thanks to**  
Brian Dougan, Paul Humphreys,  
Alex Hartmann, Michael Horn,  
Darcy O'Toole, Martha Gritwhistle,  
Geoff Fiala, John Redmond  
and Mike Turk.

Phones: (075) 51 0577 Voice  
(075) 32 6370 CoCoLink

**Deadlines:**  
7th of the preceding month.  
**Printed by:**

Australian Rainbow Magazine  
P.O. Box 1742  
Southport, Qld. 4215.  
Registered Publication QBG 4009.

This material is COPYRIGHT. Magazine owners may maintain a copy of each program plus two backups, but may NOT provide others with copies of this magazine in ANY form or media.

## index

DISSASSEMBLER  
 ..... by Rod Hoskinson P 4

DIRECTORY PRINT  
 ..... by Geoff Mackie P 6

YOUR CHILD'S SELF IMAGE  
 ..... by Steve Blyn P 8

Hardware Course Part 2  
 ..... by Tony DiStefano P 9

THE COMMANDOS WANT YOU  
 ..... by Anthony Frerking P 12

AN EASY WAY TO RUN PROGRAMS  
 ..... by Andrew Dater P 14

ADVENTURING INTO SOUND EXPERIMENTATION  
 ..... By Bill Bernico P 15

CoBBS Part 4 .... by Richard Duncan P 17

RENUM WITH A TWIST  
 ..... by Fredric Haberer P 18

ROBOCISE ..... by W.J. Moore P 19

GATHERING UP SCATTERED PROGRAMS  
 ..... by Pete Eichstaedt P 21

QUICK RESTORE ..... by John Galus P 22

Load Double-Speed Tapes  
 ..... by Craig Carmichael P 23

CROSS-REFERENCE with XREF  
 ..... by Douglas Van Dusen P 24

AUTO-EXECUTING TAPE PROGRAMS  
 ..... by Harold Nicket P 27

CoCo MERGE ..... by John Nicolettos P 30

CLS COMMAND .... by Gerry Schechter P 34

PIX FILES ..... by Joseph Kohn P 35

UTILPACK ..... by Rod Hoskinson P 37

BUBBLE WARS  
 ..... by Richard Ramella P 39

CRASH PROOF IT .... by Terry Wilson P 41

Assembly File  
 ..... by Kevin Mischewski P 42

Odds and Evens #1  
 ..... by Andrew Simpson P 43

Hard Facts about Assembly Language  
 ..... by William Barden Jr. P 45

RAM TEST ..... by Craig V. Bobbitt P 51

DEFEAT DE BUGS ..... by Mike Dean P 56

Forth Forum ..... by John Poxon P 58

OS-9  
 Frickers Follies  
 ..... by Jack Fricker P 60

DIR SORT ..... by Ross McKay P 62

..... Subscription Page ..... P 63



This month's mag is a potporri of utilities from both the USA and Australia.

Just so the games players will feel at ease, we've also included a couple of games!

There is a heap of new and very exciting software available this year, I hope you will take the time to get down to your local Tandy store and see some of it.

The major supplier of third party software in Australia is the Computer Hut in Bowen, Qld.

Tony from Computer Hut sent us a bundle of software, and this forms the basis of a fairly large review section in Australian CoCo this month - so if you are looking for something different, and there's plenty of different things there, have a read of that magazine.

The boys and girls at Liverpool users' group have been very busy of late, encouraging Leonie Duggan, their meet co ordinator, in her work with local unemployed kids.

Leonie leads a course in computing for the young people of the area and so far, after just 2 years (I think) she reports an 85% employment rate - ie 85% of the kids attending, get jobs soon after they graduate.

Currently Leonie is after computers for use in the course. As I said, the group there is already helping, and have supplied a number; but if you have an old CoCo that perhaps doesn't go, or even an MC-10 that you don't need, then how about donating it to a worthy cause?

Anyone who does donate a computer this month will receive from us, a six month sub to CoCoOz AND from The Computer Hut in Bowen, one of the new range of games / programs Computer Hut has just imported. (Thanks go to Computer Hut for donating so generously.)

Finally, CoCoConf '86 is on in August this year.

We are presenting some 10 tutorials this year, and these will be lead by some of the most talented people in Australia. Plan to come - we'd love to see you.

*John*

# DISSASSEMBLER

by Rod Hoskinson

(Greg Wilson's right hand man, Rod Hoskinson, has been busy with exams and courses since Greg passed away and the Mag moved to the Gold Coast.

We've all missed Rod's involvement and so it is very nice to be able to present the first of two excellent articles by Rod. G.)

I wrote the bulk of this program as far back as 1982, then it was put aside, not quite finished, and forgotten. The school holidays over the Christmas break gave me the spare time to dig it out, chop and change a little here and there, and send it up to Graham.

My interest in Machine Language programming on the CoCo with its great 6809 processor inspired me to begin writing this disassembler - what better way to learn the instruction set and addressing modes?

The program as presented requires a minimum system of 16K ECB, however it could probably be modified to run on a standard Basic without too much trouble.

You must POKE 25,6:NEW before CLOADing and RUNNING. Disk users will have to make some changes - DELETE lines 4-8, and delete the POKES in line 9. (And instead of POKEing 25,6, POKE25,14:NEW G.)

The program is in BASIC and incorporates a machine code utility by Michael Batley which appeared in May 1983 Australasian Rainbow.

This short utility reverses the screen colours and I heartily recommend it to anyone who wishes to keep their sight and still use CoCo! (Thanks Michael.)

After a short initialisation you will be presented with a sign on message and the main menu. To select an option, simply press the key corresponding to the first letter of that option (these letters are inversed on the screen). You can exit most options by simply holding the <SPACEBAR>,

which will return you to the menu.

The choices are explained below:

I. for Input, is used to input the start and end addresses for your disassembly. Hexadecimal address should be preceded by the '\$' sign or by &H.

D. for Disassemble 6809 machine code into assembly language. To exit before the end address is reached, hold the spacebar.

A. for ASCII dump, is used to disassemble FCC tables in the machine code, from your start address to the end address.

B. for Byte dump, is used to disassemble FCB tables in the machine code.

W. for Word dump, disassembles FDB tables.

CLOADM lets you do exactly that. If you get an IO error, then reRUN or type GOTO 10. You are free to load into memory from \$2700 up, which is close to decimal 10 000, a nice easy number to remember.

Therefore you should specify an appropriate offset that will load your program into this memory area. (For example if the ORG is 0, then an offset load by 10 000 will do the trick.)

P, for printer, will redirect all output to the printer, if it is connected.

S will switch output back to the screen. If you enter O for orange at the COLOUR? prompt, the screen will be that colour.

Finally E simply ends the program. Your start and end locations are always given at the Menu, in hex.

I have found this utility very useful in my assembly language programming endeavours. It gives you some of Z-Bug's major functions at virtually no cost. I believe the program is bug-free (sarcastic Ha-Ha) it has disassembled CoCo's ROMs with no trouble. I hope this can be of use to some readers.

The Listing:

```

1 '6809 DISASSEMBLER FOR COCO
  BY ROD HOSKINSON
  REVISION 1.1:31/1/1986
  PUBLIC DOMAIN PROGRAM.
2 GOTO4
3 POKE65494,0:CSAVE"DISASSEM":EN
D
4 CLS0:POKE65495,0:CLR300,9833
5 RESTORE:M=9834:FORI=1TO3:READS
,D:FORJ=S TOD:POKEM,PEEK(J):M=M+
1:NEXTJ,I
6 DATA33395,33413,38316,38350,41
738,41825
7 FORI=1TO25:READM$,D$:M=VAL("&H
26"+M$):D=VAL("&H"+D$):POKEM,D:N
EXT
8 DATA6E,0,6F,D,9A,F,9F,16,F4,16
,F5,32,F6,62,F7,39,AE,20,BA,20,A
F,8D,B0,48,CF,8,D2,22,D3,2,D4,88
,D5,40,D6,84,D7,DF,EF,20,F9,A7,F
A,84,FB,A7,CF,82,FD,39
9 S=0:E=0:POKE&H168,&H26:POKE&H1
69,&H6A:POKE359,126:SCREEN0:DN=0
:PRINT"MC6809E DISASSEMBLER FOR
COCO BY ROD HOSKINSON V1.1 198
6":PRINT
10 PRINT"DISASSEMBLE ASCII DUMP
BYTE DUMPWORD DUMP CLOADM
END":IFDN=-2THENPRINT"SCREEN
":ELSEPRINT"PRINTER
":
11 IFS<OORS>65535ORE<OORE>65535T
HENS=0:E=0ELSEIFS>E THENS=E
12 PRINT"INPUT":PRINT"START=":RI
GHT$( "0000"+HEX$(S),4):" END=":R
IGHT$( "0000"+HEX$(E),4):" *":
13 AS=INKEY$:IFAS="" THENEXEC4136
9:GOTO13
14 IFAS="D" THEN16ELSEIFAS="A" THE
N160ELSEIFAS="W" THEN167ELSEIFAS=
"C" THEN170ELSEIFAS="B" THEN172ELS
EIFAS="P" THEN179ELSEIFAS="S" THEN
180ELSEIFAS="I" THENGOSUB17:GOTO1
0
15 IFAS="E" THENCLS0:POKE65494,0:
END:ELSE13
16 PRINT"DISASSEMBLE":GOTO25
17 PRINT:LINEINPUT"START=":C$
18 LINEINPUT"END=":D$
19 IFLEFT$(C$,1)="$" THENS=VAL("&
H"+MID$(C$,2)) ELSE S=VAL(C$)
20 P0=S
21 IFLEFT$(D$,1)="$" THENE=VAL("&
H"+MID$(D$,2)) ELSE E=VAL(D$)
22 IFE<S THEN17
23 IFS<OORS>65535ORE<OORE>65535
THEN17
24 RETURN
25 Z=0:P0=S:BG=S:ST=S:G$="":B=PE
EK(S):F=0
26 IFB<&H10 AND B<&H11 THENS=S
+1:GOTO34
27 T=PEEK(S+1):IFT>&H20 ANDT<&H3
0 THENS=S+2:B=T:F=1:GOTO34
28 IFB=&H11 THENGOSUB206:C=50 EL
SEGOSUB205:C=155
29 FORI=1TOC STEP7:Y$=MID$(S$,I,
7)
30 IFVAL("&H"+LEFT$(Y$,2))>T TH
EN32ELSEM$=MID$(Y$,3,1):G$=RIGHT
$(Y$,4):IFRIGHT$(G$,1)=" " THENG$
=LEFT$(G$,3)
31 S=S+2:GOTO43
32 NEXT
33 S=S+2:G$="???":GOTO149
34 Q=INT(B/16)+1:ONQ GOSUB193,19
4,195,196,197,198,199,200,201,20
2,202,203,204,203,203,203:G$=MID
$(S$, (B-(Q-1)*16)*5+1,5)
35 IFB<&H90 OR (B)&HBF AND B<&HD
0) THENM$=RIGHT$(G$,1):G$=LEFT$(
G$,4):IFM$=" " THENM$=RIGHT$(G$,1
):G$=LEFT$(G$,3)
36 IFB<&H90 OR (B)&HBF ANDB<&HD0
) THEN40
37 G$=LEFT$(G$,4):IFRIGHT$(G$,1)
=" " THENG$=LEFT$(G$,3)
38 M=0:M=-68*(B)&H8F ANDB<&HA0)-
68*(B)&HCF ANDB<&HE0)-73*(B)&H9F
ANDB<&HB0)-73*(B)&HDF ANDB<&HF0
):IFM=0THENM$="E" ELSEM$=CHR$(M)
39 GOTO41
40 IFF=1THENG$="L"+G$
41 IFG$="****" THENG$="???":GOTO1
49
42 IFG$="----" THENG$="ANDCC":M$=
"1"
43 IFM$="H" THEN149
44 IFM$<"R" THEN55
45 G$=G$+" $"
46 IFLEFT$(G$,1)="L" THENB=16ELSE
B=8
47 IFB=16THENT=PEEK(S)*256+PEEK(
S+1):S=S+2:ELSET=PEEK(S):S=S+1
48 IFB=8THEN52
49 IFT>32767THENT=- (65536-T)
50 GOSUB182:T$=HEX$(S+T):GOSUB15
7
51 GOTO54
52 IFT>127THENT=- (256-T)
53 GOSUB182:T$=HEX$(S+T):GOSUB15
7
54 G$=G$+T$:GOTO149
55 IFM$<"D" THEN61
56 G$=G$+" <$"
57 T$=HEX$(PEEK(S)):GOSUB158
58 S=S+1
59 G$=G$+T$
60 GOTO149
61 IFM$<"E" THEN64
62 G$=G$+" $"
63 T$=HEX$(PEEK(S)*256+PEEK(S+1)
):GOSUB157:S=S+2:G$=G$+T$:GOTO14
9
64 IFM$="1" ORM$="2" THEN65ELSE71
65 G$=G$+" # $"
66 IFM$="1" THENT$=HEX$(PEEK(S)):
S=S+1:GOSUB158
67 IFM$="2" THENT$=HEX$(PEEK(S)*2
56+PEEK(S+1)):S=S+2:GOSUB157
68 G$=G$+T$
69 IFLEFT$(G$,4)="ORCC" OR LEFT$(
G$,5)="ANDCC" THENGOSUB184:GOTO1
49
70 IFM$="1" ANDVAL("&H"+T$)<&H7F
ANDVAL("&H"+T$)>&H1F THENG$=G$+"
"+CHR$(VAL("&H"+T$)):GOTO149EL
SE149
71 IFM$<"M" THEN102
72 G$=G$+" "
73 IFLEFT$(G$,3)!="PSH" ORLEFT$(G$
,3)!="PUL" THEN90
74 B=PEEK(S):S=S+1
75 T$=HEX$(B):GOSUB158
76 L$=LEFT$(T$,1):R$=RIGHT$(T$,1
)
77 L=VAL("&H"+L$):R=VAL("&H"+R$)
78 N=L:GOSUB79:G$=G$+T$+" ",N=R:
GOSUB79:G$=G$+T$:GOTO149
79 IFN=0THENT$="D":RETURN
80 IFN=1THENT$="X":RETURN
81 IFN=2THENT$="Y":RETURN
82 IFN=3THENT$="U":RETURN
83 IFN=4THENT$="S":RETURN
84 IFN=5THENT$="PC":RETURN
85 IFN=8THENT$="A":RETURN
86 IFN=9THENT$="B":RETURN
87 IFN=10THENT$="CC":RETURN
88 IFN=11THENT$="DP":RETURN
89 T$="???":RETURN
90 D=PEEK(S):S=S+1:GOTO91
91 IF(D AND128)=128THENG$=G$+"PC
,"
92 IF(D AND64)=64ANDLEFT$(G$,1)="
"U" THENG$=G$+"S",:GOTO94
93 IF(D AND64)=64THENG$=G$+"U,"
94 IF(D AND32)=32THENG$=G$+"Y,"
95 IF(D AND16)=16THENG$=G$+"X,"
96 IF(D AND8)=8THENG$=G$+"DP,"
97 IF(D AND4)=4THENG$=G$+"B,"
98 IF(D AND2)=2THENG$=G$+"A,"
99 IF(D AND1)=1THENG$=G$+"CC,"
100 IFRIGHT$(G$,1)=", " THENG$=LEF
T$(G$,LEN(G$)-1)
101 GOTO149
102 D=PEEK(S):S=S+1
103 PMS=""
104 O$="":E$="":F=0
105 IF(D AND16)=16 AND (D AND128
)=128 THENF=16:D=D-16
106 IFD=159THENG$=G$+" "+["$"+RI
GHT$( "0000"+HEX$(PEEK(S)*256+PEE
K(S+1)),4)+"J":S=S+2:GOTO149
107 Q=D AND31
108 IF(D AND128)<>0THEN112
109 T=-16*((D AND16)=16)-8*((D A
ND8)=8)-4*((D AND4)=4)-2*((D AND
2)=2)-1*((D AND1)=1)
110 IFT>15THENT=- (32-T)
111 O$=STR$(T):GOTO130
112 IFQ=4THEN130
113 IFQ<>8THEN116
114 T=PEEK(S):S=S+1:IFT>127THENT
=- (256-T)
115 O$=STR$(T):GOTO130
116 IFQ<>9THEN119
117 T=PEEK(S)*256+PEEK(S+1):S=S+
2:IFT>32767THENT=- (65536-T)
118 O$=STR$(T):GOTO130
119 IFQ=6THENO$="A":GOTO130
120 IFQ=5THENO$="B":GOTO130
121 IFQ=11THENO$="D":GOTO130
122 IFQ=0THENPMS="+":GOTO130
123 IFQ=1THENPMS="++":GOTO130
124 IFQ=2THENPMS="---":GOTO130
125 IFQ=3THENPMS="---":GOTO130
126 IFQ=12THENT=PEEK(S):E$="PCR"
:S=S+1:GOSUB182:IFT>127THENT=- (2
56-T)
127 IFQ=12THENO$=" $" +RIGHT$( "000
0"+HEX$(S+T),4):GOTO130
128 IFQ=13THENE$="PCR":T=PEEK(S)
*256+PEEK(S+1):S=S+2:GOSUB182:IF
T>32767THENT=- (65536-T)
129 IFQ=13THENQ=12:GOTO127
130 IFLEFT$(O$,1)=" " THENO$=MID$(
O$,2)
131 G$=G$+" "+O$
132 IFE$="PCR" THENG$=G$+" "+E$:G
OTO143
133 T=D AND96
134 IFT=0THENE$="X"
135 IFT=32THENE$="Y"
136 IFT=64THENE$="U"
137 IFT=96THENE$="S"
138 IFPMS="+" THENE$=E$+"+"
139 IFPMS="++" THENE$=E$+"++"
140 IFPMS="---" THENE$="---"+E$
141 IFPMS="---" THENE$="---"+E$
142 G$=G$+" "+E$

```

continued on Page 59

# DIRECTORY PRINT

by Geoff Mackie

## RAINBOW PROGRAMS

```
ALPHA1  BAS    ALPHA2  BAS    DJACKET  BAS
DJACKINS VIP  JACKET  BAS
```

This is a utility program designed to print out jackets for your disks. The idea arose when I was looking for a program that I knew was on one of 5 full disks. The question was which one? I thought it would be handy to have a record of each disk's contents right there on the cover.

"Disk Jacket" will read your disk's directory (single sided, double sided or floppy), alphabetize it if you wish, print a title for the disk then print the directory on the jacket in one of two sizes. You can have an inner sleeve to go inside the original jacket or a larger cover to replace the original jacket - the choice is yours.

The program is set up for a Gemini printer, but routines are provided for other printers. If you wish to make a permanent copy for your printer, the variables to change are as follows:

```
CN$ = double width on
CO$ = double width off
FF$ = form feed on in line 2080
and in 2070 PRINT#-2, 'double strike'.
```

The Listing:

```
0 '* DIRECTORY PRINT PROGRAM *
1 '* WRITTEN BY GEOFF MACKIE *
2 '* 6 PIALBA CRT NTH GOSFORD*
3 '*      N.S.W. 2250 *
4 '*      COPYRIGHT JAN'86 *
5 '*****
10 CLEAR2000:FL=1:GOSUB2500:GOSU
B2000
20 CLS:PRINT@128,"DO YOU WANT IN
STRUCTIONS (Y/N) ?";
30 GOSUB3000
40 IF I$="Y" THENGOSUB1500
50 CLS:PRINT@128,"WHAT SORT OF D
ISK JACKET      WOULD YOU LIKE
:
NER SLEEVE      OR      <o>U
```

```
TER COVER ?";
60 GOSUB3000:IFI$="I" THENA=60:
TC=67:TD=7ELSEIFI$="O" THENA=63
:TC=70:TD=9ELSEISOUND20,5:GOTO60
90 IFFL=0 THENGOSUB2510
100 CLS:PRINT@128,"PLACE THE DIS
K FOR WHICH YOU  WANT TO MAKE
A NEW JACKET FOR  IN DRIVE 0 AN
D PRESS <ENTER>";
110 GOSUB3000
120 IF I$<>CHR$(13) THEN110
125 CLS:PRINT@128,"IS THE DISK <
d>OUBLE SIDED,      <f>LIPPY, OR
<n>ORMAL ?"
126 GOSUB3000
127 IF I$="D" THENS=2:NS=2ELSEIFI
$="F" THENS=2:NS=0ELSEIFI$="N" TH
ENS=1:NS=0ELSE126
```

```
130 CLS:PRINT@0,"READING DIRECTO
RY .....      PLEASE WAIT
A MOMENT";
135 FORJ=1TO S:IFJ=2 THENSD$=" (B
)"ELSEIFS=1 THENSD$=" "ELSESD$=" (
A)
136 IFJ=1 THENSD=0ELSESD=NS
140 FORI=3TO11
150 DSKI$SD,17,1,A$,B$
160 C$=A$+LEFT$(B$,127)
170 NAM$(A)=LEFT$(C$,8)
180 EXT$(A)=MID$(C$,9,3)+SD$
190 IF LEFT$(NAM$(A),1)=CHR$(255
) THEN 290
200 IF LEFT$(NAM$(A),1)=CHR$(0)
THEN NAM$(A)="":EXT$(A)="":A=A-1
210 FORN=1TO7
220 NAM$(A+N)=MID$(C$,N*32+1,8)
230 EXT$(A+N)=MID$(C$,9+N*32,3)+
SD$
240 IFLEFT$(NAM$(A+N),1)=CHR$(25
5) THEN290
250 IFLEFT$(NAM$(A+N),1)=CHR$(0)
THENNAM$(A+N)="":EXT$(A+N)="":A
=A-1
260 NEXTN
270 A=A+8
280 NEXTI
285 IFS=2 ANDNS=0 THENPRINT@256,
"PLEASE TURN THE DISK IN DRIVE 0
OVER AND PRESS <enter>.";
287 GOSUB3000
288 IF I$<>CHR$(13) THEN287
290 NEXTJ:NF=A+N-1
300 IFNF>66 THENCLS:PRINT@128,"TO
O MANY FILES TO FIT ON THIS JA
CKET, DO YOU WANT TO :      <p
>RINT AS MANY AS POSSIBLE OR<a
>BORT THIS JACKET ?";ELSE310
303 GOSUB3000
305 IF I$="F" THENNF=66:GOTO310EL
SEIFI$="A" THEN840ELSE303
310 CLS:PRINT@128,"ALPHABETIZE D
IRECTORY FILES
(Y/N) ?";
320 GOSUB3000
330 IF I$="Y" THENPRINT@260,"ALPH
ABETIZING DIRECTORY ....":GOSUB
1000
340 CLS:PRINT@128,"TITLE OF DISK
(0-26 CHAR'S) ?";
350 LINEINPUT TL$
360 IFLEN(TL$)>26 THENPRINT@224,"
TITLE TOO LONG!":GOTO340
370 IFS=2 THENIN=IN-3ELSEIN=5
400 Z=0:FORI=1TO71
410 PRINT#-2,"-";
420 NEXTI
430 PRINT#-2," "
440 T=LEN(TL$):TB=(26-T)/2:IFTB<
>INT(TB) THENTB=INT(TB)+1
450 PRINT#-2,"":STRING$(TD," ")
:CN$:STRING$(TB," ");TL$:STRING$
(TB," ");CO$:STRING$(TD," ");":
460 PRINT#-2,"":TAB(TC);":
```



```

480 FORI=1TO22
490 Z=Z+3:PRINT#-2," ";
500 IF Z<=NF THENGOSUB700ELSEIF
NF-Z+3=1 THENGOSUB730ELSEIF NF-Z
+3=2 THENGOSUB750
510 PRINT#-2,TAB(TC);":":NEXTI
520 FORI=1TO2
530 FORJ=1TO8
540 PRINT#-2,"-";
550 NEXTJ:PRINT#-2,TAB(TA):NEXTI
560 PRINT#-2," "
570 FORI=1TO28
580 PRINT#-2,TAB(7)":":TAB(TA)":":
590 NEXTI
610 PRINT#-2,TAB(7);
620 FORI=1TO57
630 PRINT#-2,"-";
640 NEXTI
650 PRINT#-2,CHR$(10)
660 PRINT#-2,FF$
670 GOTO810
700 IFTA=60 THEN770ELSEPRINT#-2,
STRING$(7," ");:FORY=1TO3:PRINT#
-2,NAM$(Y+Z-3);" ";EXT$(Y+Z-3);S
TRINGS(IN," ");
710 NEXTY
720 RETURN
730 PRINT#-2,STRING$(7," ");NAM$(
NF);" ";EXT$(NF);
740 RETURN
750 IFTA=60THEN780ELSEPRINT#-2,S
TRINGS(7," ");NAM$(NF-1);" ";EXT
$(NF-1);STRINGS(IN," ");NAM$(NF)
;" ";EXT$(NF);
760 RETURN
770 PRINT#-2,STRING$(7," ");NAM$(
Z-2);" ";EXT$(Z-2);STRING$(IN,"
");NAM$(Z-1);" ";EXT$(Z-1);STR
INGS(IN," ");NAM$(Z);" ";EXT$(Z);
:RETURN
780 PRINT#-2,STRING$(7," ");NAM$(
NF-1);" ";EXT$(NF-1);STRINGS(IN
," ");NAM$(NF);" ";EXT$(NF);:RET
URN
810 CLS:PRINT@128,"WOULD YOU LIK
E ANOTHER COPY OF THIS JACKET (
Y/N) ?";
820 GOSUB3000
830 IFI$="Y" THENGOTO400
840 CLS:PRINT@224,"WOULD YOU LIK
E TO PRINT ANOTHER DISK JACKET (
Y/N) ?";
850 GOSUB3000
860 IF I$="Y" THENFL=0:GOTO90
900 CLS:PRINT@6,"YOU HAVE BEEN U
SING":PRINT:PRINT:" 'D
ISK JACKET MAKER'":PRINT:PRINT@1
94,"WRITTEN FOR YOUR CONVENIENCE
":PRINT@233,"BY GEOFF MACKIE":PR
INT@262,"OF COMPUTER WIZARDRY":P
RINT@333,"BYE FOR NOW"
910 END
1000 FORI=1TO NF
1010 TEMP$(I)=NAM$(I)+EXT$(I)
1020 NEXTI
1030 FORI=2TO NF
1040 J=I-1:TS=TEMP$(I)
1050 IFJ>0THENIFTEMP$(J)>TS THEN
TEMP$(J+1)=TEMP$(J):J=J-1:GOTO10
50
1060 TEMP$(J+1)=TS
1070 NEXTI
1075 IFS=2 THENG=G+4ELSEG=3
1080 FORI=1TO NF
1090 NAM$(I)=LEFT$(TEMP$(I),8):E
XT$(I)=RIGHT$(TEMP$(I),G)
1100 NEXTI
1110 RETURN

```

```

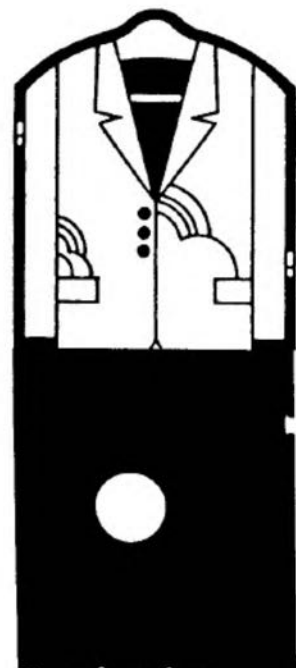
1500 CLS:PRINT@32,"THIS IS A PRO
GRAM THAT PRINTS OUT A DIRECTO
RY OF THE DISK IN DRIVE 0 IN TH
E FORMAT OF A DISK JACKET.
THAT IS, ONCE"
1510 PRINT"PRINTED, IT CAN BE CU
T AND GLUEDTO MAKE EITHER A SLEA
VE (LIKE ARECORD COVER) OR A NE
W DISK JACKET.";
1515 PRINT@480,"PRESS <enter> TO
CONTINUE";
1517 GOSUB3000:IFI$(<>CHR$(13))THE
N1517
1520 CLS:PRINT@32,"IF YOU MAKE O
NE OF THESE JACKETSFOR EACH OF Y
OUR DISKS YOU WILL HAVE AN INSTA
NT RECORD OF THEIR CONTENTS."
1530 PRINT@256,"THE PROGRAM WILL
PROMPT YOU AT EACH STEP FOR TH
E INFORMATION THAT IT NEEDS. J
UST ANSWER THE PROMPTS AS REQUI
RED."
1540 PRINT@480,"PRESS <enter> TO
CONTINUE";
1550 GOSUB3000:IFI$(<>CHR$(13))THE
N1550
1560 CLS:PRINT@32,"THE PROGRAM C
AN HANDLE FLIPPIS,DOUBLE-SIDED
AND ORDINARY DISKS.FLIPPIS AND
DOUBLE-SIDED ARE TREATED AS SI
DE 'A' AND SIDE 'B'AND THE FILES
ARE LABELED AS SUCH ON THE D
ISK JACKET."
1570 PRINT@480,"PRESS <enter> TO
CONTINUE";
1580 GOSUB3000
1590 IFI$(<>CHR$(13)) THEN1580
1600 CLS:PRINT@32,"THE PRINT ROU
TINES USE DOUBLE STRIKE, DOUBL
E WIDTH FOR THE HEADING AND A
UTOMATIC FORMFEED FOR TRACTOR F
EED PAPER."
1610 PRINT:PRINT"IF YOU DON'T WA
NT TO USE THIS FORMAT JUST ANS
WER THE 'EPSOM PRINTER' PROMPT
WITH 'n' AND DON'T INPUT ANY
CODES. (ENTER 500 AT EACH PRO
MPT)"
1620 PRINT@480,"PRESS <enter> TO
RETURN";:GOSUB3000
1630 GOSUB3000
1640 IFI$(<>CHR$(13))THEN1630
1650 RETURN
2000 IF (PEEK(&HFF22) AND 1)=1 T
HEN 2010 ELSE 2020
2010 CLS:PRINT@128,"PRINTER IS O
FFLINE!";:SOUND20,3:GOTO2000
2020 CLS:PRINT@128,"ENTER BAUD R
ATE CONSTANT (DEFAULT IS
600 BAUD)";:INPUTB$
2030 IFB$=CHR$(13)THEN2050
2040 POKE150,VAL(B$)
2050 CLS:PRINT@128,"DO YOU HAVE
AN EPSOM OR GEMINI TYPE PRINTER
(Y/N) ?";:GOSUB3000
2060 IF I$="N" THEN2100
2070 IF I$(<>"Y") THENGOSUB3000ELS
EPRINT#-2,CHR$(27);CHR$(71);
2080 CN$=CHR$(14):CO$=CHR$(20):F
F$=CHR$(12)
2090 RETURN
2100 CLS:PRINT@128,"ENTER YOUR P
RINTERS' CODE FOR DOUBLE STRIK
E (E.G. 27 <ENTER> 71 <ENTER> E
TC. ENTER 500 WH
EN FINISHED.":PRINT
2110 INPUTA:IFA=500THEN2140
2120 CO=CO+1:DS$(CO)=CHR$(A)

```

```

2130 GOTO2110
2140 FORI=1TO CO:PRINT#-2,DS$(I)
;:NEXTI
2150 CLS:PRINT@128,"ENTER YOUR P
RINTERS' CODE FOR DOUBLE WIDTH
PRINT. ENTER 500 WHEN FINISHE
D.":PRINT
2160 INPUTA:IFA=500THEN2180
2170 CU=CU+1:CN$(CU)=CHR$(A):GOT
O2160
2180 IFCU=0THENCN$=""ELSEIFCU=1T
HENCN$=CN$(1)ELSEIFCU=2THENCN$=C
N$(1)+CN$(2)ELSEIFCU=3THENCN$=CN
$(1)+CN$(2)+CN$(3)
2190 CLS:PRINT@128,"ENTER YOUR P
RINTERS' CODE TO TURN OFF DOU
BLE WIDTH PRINT. ENTER 500 WH
EN WHEN FINISHED.":PRINT
2200 INPUTA:IFA=500THEN2220
2210 CF=CF+1:CO$(CF)=CHR$(A):GOT
O2200
2220 IFCF=0THENCOS$=""ELSEIFCF=1T
HENCOS$=COS$(1)ELSEIFCF=2THENCOS$=C
OS$(1)+COS$(2)ELSEIFCF=3THENCOS$=CO
$(1)+COS$(2)+COS$(3)
2230 CLS:PRINT@128,"ENTER YOUR P
RINTERS' CODE FOR A FORMFEED. EN
TER 500 WHEN DONE.":PRINT
2240 INPUTA:IFA=500THEN2260
2250 F=F+1:FF$(F)=CHR$(A):GOTO22
40
2260 IFF=0THENFF$=""ELSEIFF=1THE
NFF$=FF$(1)ELSEIFF=2THENFF$=FF$(
1)+FF$(2)ELSEIFF=3THENFF$=FF$(1)
+FF$(2)+FF$(3)
2270 RETURN
2500 DIM NAM$(70),EXT$(70),TEMP$(
70)
2510 N=0:A=1:G=3:IN=5
2600 RETURN
3000 IS=INKEY$
3010 IFI$="" THEN3000ELSEReturn

```



# Gaining Insight Into Your Child's Self-Image

By Steve Blyn



Schools are in a unique position due to their obligation to reach and teach the entire population. Consequently, they exert a major force in the process of transferring fundamental values from one generation to another. A self-image, hopefully a positive one, is a necessary component of learning these social values.

This article presents a program that can become part of a guidance system at home or in your classroom.

This month's program, *Who Am I?*, deals with the topic of helping children develop a good self-image. The program is in the form of a questionnaire. This type of guidance goal deals with forming attitudes and developing reasoning processes rather than imparting any factual knowledge. There are, therefore, no right or wrong student responses to the questions. On the contrary, we should be prepared for a wide variety of student thoughts and feelings in response to the questions.

The giver of the questionnaire, either parent or teacher, must try to create a climate where empathy and trust are established before beginning this or any similar activity. We do not want children to give answers they think will make us happy. We want, instead, the children to give honest answers that you may explore with them afterward.

Children's answers to such questionnaires are often very revealing. It is not uncommon to find children with obviously very poor self-images. We all go through periods of life where we may have a relatively poor self-image. As adults, we have hopefully learned how to deal with and modify our feelings.

Your work really begins after the questions are answered. The answers

may expose problems that are bothering the child about his or her feelings. This is your chance to apply a little guidance. We want to help impart strategies for change or means of coping with these feelings. If done in a group setting, it is often helpful for children to hear their type of negative feelings expressed by others. There's a little bit of the "Charlie Brown" poor self-image in all of us.

Lines 90-130 set the tone of the program by drawing a silhouette of a large letter 'I'. Lines 140-340 print the 13 key questions and the child's responses. This comprises the bulk of the questionnaire. There is little room on the screen for the answers. It is best to instruct children to keep their answers short, although there is really no problem if any of the answers are longer than the space allotted since the entire answer is stored and reappears in the next section of the program.

There is additionally a final question. Lines 380-480 contain ample room for three answers to the statement "What I like about myself." This section appears on a new screen. Pressing the up-arrow key then the ENTER key allows you to utilize the questionnaire that was just completed. Each press of the ENTER key shows the next response in the series. The questions and the child's complete responses will reappear as a reminder or helper for him or her to answer the final question. This can also serve as a review for both of you. The items are recalled by lines 580-650.

An option for a permanent hard copy is provided as the final part of the program. The printout is useful as we tend to forget the responses. An alter-

native is for you to write the answers down on a sheet of paper as the child enters them into the computer. Line 490 asks you to press the 'E' key to end the program or the up-arrow key to get the printout. The printout is performed by lines 520-560.

The questions contained in this program were taken from a guidance bulletin published by the New York City Board of Education. Another activity I like very much from this bulletin presents a slightly different way of helping to look at a child's self-image. It is called the "Coat of Arms" game. The child or class is given a blank outline of a shield divided into four parts. In each of the parts, they draw response to the following four items.

- Draw two things you do well.
- Draw your greatest success in life.
- Draw two things you would do if you had only one more year to live.
- Draw two things you would like said about you.

The resulting picture represents the child's individual "coat of arms." The adult can glean valuable information about the child's values and self-image. This leads to similar follow-up discussions as with the questionnaire.

The "Coat of Arms" activity is just as good an activity as the questionnaire. Consider, however, the challenges of programming that it presents. Perhaps one of you readers would like to take up this challenge. We at Computer Island would love to see the results of any of your efforts in this direction. In any case, we always enjoy hearing from the readers of our column. □

continued on Page 11

# A Beginner's Hardware Course

## Part 2

By Tony DiStefano

Last month we took a look at binary bits and different numbering systems. So far, there doesn't seem to be any relation between these and computers. All we did is express numbers in different forms. But, we are a little closer to computers than you think. We know the computer is made up of a lot of chips that use bits of zeros and ones. In order to understand the ins and outs of these chips, I will go into detail of how chips use zeros and ones.

The heart of all digital computers is the logic circuit elements. They perform binary arithmetic operations, make logical decisions and perform operations such as counting and temporary storage. The basic type of logic element is called a "gate." Gates are circuits that look at two or more binary signals and produce a binary output, which depends upon the conditions of the input signals.

In order to comprehend this better, let's look at an equivalent circuit that is easier to understand, using conventional components you are likely to find around the house. If you want to build and test these circuits yourself, Radio Shack has all the parts. The switches are single pole, single throw. Any battery and bulb combination will do, just be sure the battery and the light bulb are the same voltage rating, otherwise you may end up burning out the bulb or get no results at all. Such a circuit is shown in Figure 2.

This circuit contains three components: a battery, a switch and a bulb. Here, the switch is considered the input and the bulb is considered the output. When the switch is on (a logical 1) the bulb is lit (this is also considered a logical 1). When the switch is off (logical 0) the bulb is off, also giving us a logical 0. In a logical element such as this, the

input (the switch) and the output (the bulb) follow each other, one to one or zero to zero.

The symbol used to represent this circuit or logical element in a logic (or computer) schematic is shown in Figure 1a. This gate is called a "buffer." The input is exactly the same as the output. Not very useful in a logical sense, in that it does nothing, but it is needed under certain circumstances. For instance, when the output of a gate (logic element) is connected to many other gates, it may not have enough power to drive all the gates properly. In this case a buffer is used. Whenever a gate is used there is always a small delay between when the input changes and the output changes; a buffer is sometimes used just for that delay.

To continue our understanding of gates, let's introduce another factor in our battery circuit. Now study the circuit in Figure 3. It has two switches. The two switches are in a series, that is, one after the other. Therefore, they must both be on before the bulb will turn on. This circuit or logical element is known as an AND gate. The definition of an AND gate is: "The AND gate is a logical element with two or more inputs and a single output. Both (or all in the case of more than two) inputs must be binary '1' to produce an output of binary '1'."

The symbol for an AND gate is shown in Figure 1b. The main value of the AND gate is its ability to detect when all inputs are binary '1'. For example, in a control system when all the motors are on, turn on the extra generator. A quick way to remember this gate is, when 'A' AND 'B' are '1', then 'Y' is '1'. Hence the term AND.

The next gate we will study is the OR gate. Again, we have two switches in our next diagram, Figure 3. The difference

is that now they are wired in parallel, one on top of the other. If either switch is on, then the bulb will be on. If both are on, the light is, of course, still on. This circuit or gate is known as an OR gate. The definition of an OR gate is as follows: "The OR gate is a logical element with two or more inputs and a single output. If any one input is a binary '1' then the output is binary '1'."

The symbol for an OR gate is shown in Figure 1c. The main value of the OR gate is its ability to detect when any input is binary '1'. An example of this use is when any door or window opens, an alarm sounds. A quick way to remember this gate is when 'A' OR 'B' is '1', then 'Y' is '1'. Hence the term OR.

If we look back to our first gate, the buffer, we notice the input matches the output. Since the input and the output are the same, it is called a "non-inverting" output. This gate, and most other gates, can also come in an "inverting" output. In the case of our buffer, it becomes an inverter, or better known as a NOT gate. Figure 1d shows the symbol of an inverter. The definition of an inverter is: "An inverter is a logic element whose output is always the complement (the opposite) of its input."

Notice the difference between a buffer symbol and an inverter symbol. The inverter symbol has a small circle on the output side. Any inverting output gate has a small circle on the output. This is true for the AND and the OR gate, too. If you take the output of an AND gate and tie it to the input of a NOT gate, the result (the output of the NOT gate) is an inverted AND gate (see Figure 5). This requires two gates and some wiring. It is so often used that the IC designers decided to put it all in one chip. This is called a NAND gate. The same thing goes with an OR gate — it becomes a NOR gate. These two gates

are defined as follows: "NAND and NOR gates are the complements of AND and OR gates, respectively."

The last gate we will look at is the EXCLUSIVE-OR gate. The symbol for the EXCLUSIVE-OR gate is shown in Figure 1e. For short, this gate is called XOR. It is a little different than the OR gate and is used mostly when a signal needs to be inverted in some cases and not in others. The definition of an XOR gate is: "The logical XOR is defined as a binary '1' output when either of the two inputs is a binary '1'. The other input being a binary '0'."

A quick way to remember the function of this gate is when the inputs are different, the output is '1'. Like the other gates, it, too, has the inverted version. It is called the EXCLUSIVE-NOR or XNOR for short. The definition of an XNOR gate is the same as the XOR, but has its output inverted to a binary '0' when either of the two inputs is a binary '1'.

The gates described so far are quite simple in structure. They have one or two inputs and one output. They are the fundamental elements in creating more complex chips, and even the basis of complete computer CPUs. In the case of the simple two-input AND gate, there are four discrete combinations of inputs. The two inputs are represented by a two-digit binary number. Remember last month? They are 00, 01, 10 and 11, and the output for each given condition is 0, 0, 0 and 1, respectively. Not so hard to remember or display. But, in other chips, where there might be five or six inputs and eight or 10 outputs, it can be too much to remember. Now is when the "truth table comes in. The definition of a truth table is: "A truth table is a graphic representation of all possible combinations of inputs versus outputs of a particular logic element."

The second column of Figure 1 represents the truth tables for the given gates. Notice that all possible combinations of inputs are given. Columns A and B are the inputs, as you can see from the gates in Column 1. Column Y is the output. Read the truth table as you read text, one line at a time. Each line is one condition. The condition is given for 'A' and 'B'. The output, 'Y', is the result for a given gate. Every line is different, and continues until all possible combinations for that gate are shown. This way, at a glance, you can tell what the output is for a given input of any gate. In these cases, it is not too difficult to follow or remember. Later on, when I show you the truth tables for some of the chips that make up our good ol' CoCo, you

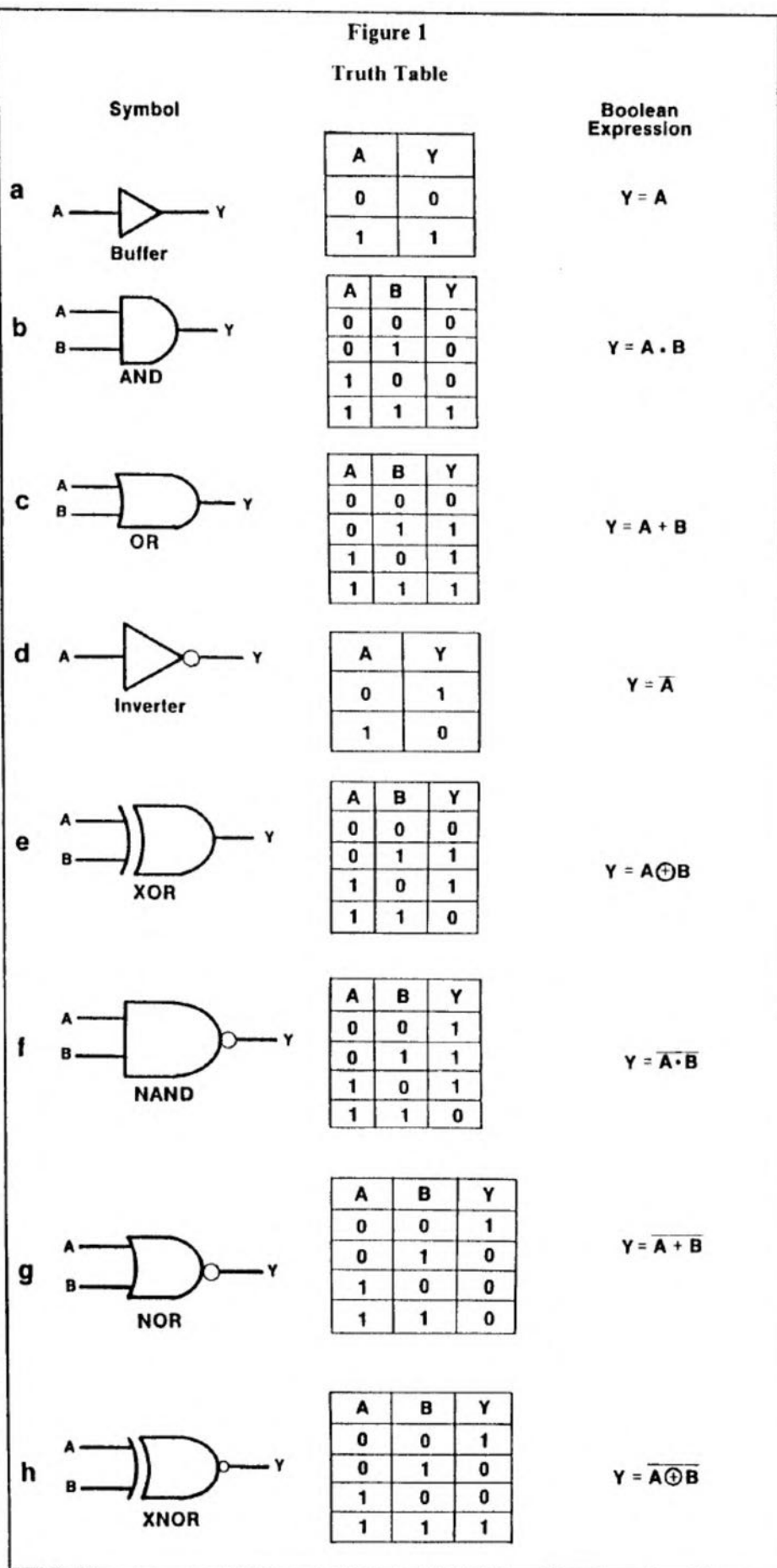


Figure 2

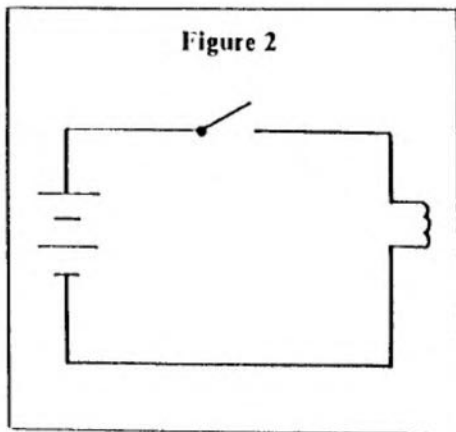


Figure 4

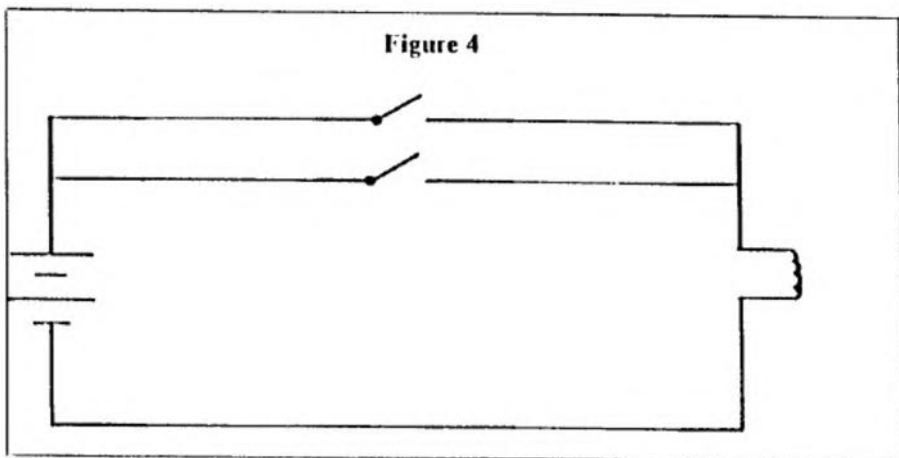


Figure 3

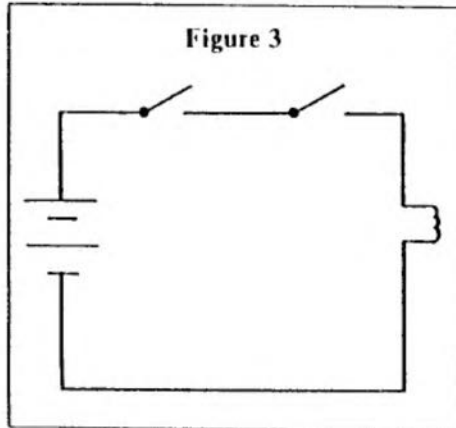
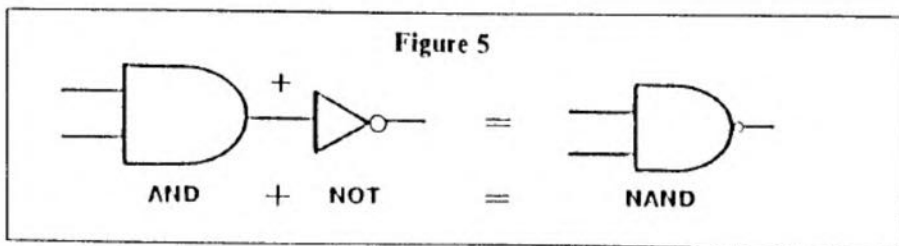


Figure 5



will be glad I introduced you to these tables.

Though I will not be getting into great detail in this series of articles, I feel it is necessary to talk a little about Boolean algebra. The definition of Boolean algebra is: "A system of mathematical logic used to represent digital logic signals and express the logic operations

performed by digital signals."

To put it into simple terms, Boolean algebra is an equation that represents the function of a logical element. Take, for instance, the buffer in Figure 1. The output is equal to the input. A Boolean equation would be:

$$Y = A$$

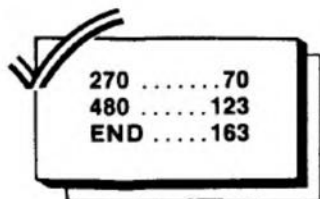
Now an inverter would look like this:

$$Y = \text{NOT } A \text{ or } Y = *A$$

The AND symbol in a Boolean expression is a dot in the middle of the

line, like the multiplication sign in regular math. Notice its occurrences in Figure 1. The OR symbol in a Boolean expression is a plus sign (+). Again, the Boolean OR symbol can be seen in Figure 1. The next Boolean symbol is the EXCLUSIVE-OR. This is no more than the plus symbol with a circle around it. Figure 1 also shows the XOR symbol. Any of the inverting symbols in Boolean algebra are represented by a small horizontal bar above the equation in question. You can see the inverting gates in Figure 1.

continued from Page 8



The listing: WHO AM I

```

10 REM"WHO AM I?"
20 REM"STEVE BLYN, COMPUTER ISLAN
D,NY,1986"
30 CLEAR 2000
40 DIM A$(13),B$(13),N$(3)
50 CLS
60 W$=STRING$(28,143)
70 Z$=CHR$(128)+CHR$(128)
80 PRINT@11," who am i?";
90 PRINT@32," ":PRINT
100 FOR Y=1 TO 9
110 PRINT@98+Z,W$;
120 Z=Z+32:NEXT Y
130 PRINT@384," ":PRINT
140 A$(1)="I AM "

```

```

150 PRINT@32,A$(1);:LINEINPUT B$
(1)
160 A$(2)="I FEEL GOOD WHEN "
170 PRINT@64,A$(2);:LINE INPUT B
$(2)
180 A$(3)="I FEEL BAD WHEN "
190 A$(4)="I LIKE PEOPLE WHO "
200 A$(5)="I LIKE TO PLAY "
210 A$(6)="I DON'T LIKE "
220 A$(7)="I GET ANGRY WHEN "
230 A$(8)="I AM BEST AT "
240 A$(9)="I AM PROUD WHEN "
250 A$(10)="I AM NERVOUS WHEN "
260 A$(11)="I AM AFRAID TO "
270 FOR T=3 TO 11
280 PRINT@98+L,A$(T);:LINEINPUT
B$(T)
290 PRINT@126+L,Z$;
300 L=L+32:PLAY"O2L30GCG":NEXT T
310 A$(12)="I AM BORED WHEN "
320 PRINT@384,A$(12);:LINEINPUT
B$(12)
330 A$(13)="I DO BEST WHEN I'M "
340 PRINT@416,A$(13);:LINEINPUT
B$(13)
350 PRINT@486,"PRESS ENTER TO GO
ON";
360 EN$=INKEY$
370 IF EN$=CHR$(13) THEN 380 ELS
E 360
380 CLS:PRINT@3,"WHAT I LIKE ABO
UT MYSELF";
390 PRINT@64,"1.";
400 PRINT@160,"2.";
410 PRINT@256,"3.";

```

```

420 PRINT@354,W$;
430 PRINT@384,"ENTER ^ TO REVIEW
YOUR ANSWERS.";
440 FOR K=1 TO 3
450 PRINT@66+V,"";:LINEINPUT N$(
K)
460 IF N$(K)="" OR N$(K)=" " THE
N 450
470 IF N$(K)=CHR$(94) THEN GOSUB
590:GOTO 450
480 V=V+96:NEXT K
490 PRINT@416," ":PRINT@384,"ENT
ER ^ FOR PRINTOUT OR E TO END";
500 EN$=INKEY$
510 IF EN$="" THEN 520 ELSE IF
EN$="E" THEN 660 ELSE 500
520 PRINT#-2,TAB(10)"WHO AM I?";
530 FOR T= 1 TO 13:PRINT#-2," ":
PRINT#-2,A$(T)B$(T):NEXT T
540 PRINT#-2," "
550 PRINT#-2,TAB(10)"WHAT I LIKE
ABOUT MYSELF":PRINT#-2," "
560 FOR T=1 TO 3:PRINT#-2," ":PR
INT#-2,N$(T):NEXT T
570 GOTO 660
580 REM"REVIEW THE ITEMS"
590 FOR T= 1 TO 13
600 PRINT@416," ":PRINT@448," "
610 PRINT@416,A$(T)+B$(T)
620 PLAY"O3L50CDEFG"
630 EN$=INKEY$
640 IF EN$=CHR$(13) THEN NEXT T
ELSE 630
650 RETURN
660 END

```

# The Commandos



## Want You!

By Anthony Frerking

**W**elcome recruit, you have just been assigned to Camp Ike, training camp of the Commandos. The Commandos are an elite army force of skilled pilots and athletes. They are able to get in and out of any situation. You, in your infinite wisdom, have chosen to join them. There's only one problem: You must survive basic training. You must complete a three-stage test six times to achieve the beloved rank of Commando First Class. Each test increases in difficulty as one progresses up in levels.

### Stage 1: Hundred-Yard Dash

A Commando must be fit, so to prove your agility you must run from the camp to the A.F.I spider jet on the other side of the compound. Sounds easy, but look up; as you run bombs will drop. You are gone if the explosion or radioactivity reaches you. Upon reaching the jet it takes you to the next stage of the test.

### Stage 2: Rescue

In this stage you control the jet in an attempt to rescue a helpless captive trapped in the valley. You must maneuver the jet down through the opening in the valley and avoid being shot. Once in the valley, you place the jet over the victim and press the firebutton to beam him up. Caution: On higher levels the tank moves toward the victim. If the tank runs over the victim, you both die. Once you have the victim, leave the valley and fly toward the left side of the screen to complete the second part of the test. Note: Hitting the valley walls will also kill you.

### Stage 3: Obstacle Course

The final phase of each test is to successfully travel through the obstacle course. This requires starting at the top of Snake Rock, avoiding cannon fire and entering the cave at the bottom left of the screen to complete the test. As

levels increase, moving walls are added to impede your progress. Should you get shot or crash into walls, you will die.

### End of Game

The game is over when you have lost all of your men (there are three of them) or you finish six complete tests. At the end of the game you receive your score, your rank, the level last completed and number of lives you saved. If you complete all six levels, you are also given the total time to complete the tests and receive the Commando Medal of Honor. After seeing your statistics, the screen clears and displays the high scores (up to 10).

The first two levels are meant to give you a chance to get familiar with the game and how to handle the joystick.

You are awarded 100 points multiplied by the level you are on for each stage completed, plus extra points for finishing each stage under the required time limit. Good luck, Cadet!

260	.....81	2320	.....102
430	.....222	3052	.....16
640	.....25	3220	.....224
1170	.....205	4090	.....151
1400	.....34	6070	.....108
1560	.....152	END	.....84
2120	.....102		

The listing: COMMANDO

```

100 'BEGIN
110 DIM C(15,15),D(15,15),N$(11)
,SC(11),LV(11)
150 'INTRO
160 CLS
170 PRINT@68,"ARTIFACTS RARELY F
OUND":PRINT@133,"PROUDLY PRESEN
TS ...";
180 PRINT@203,"COMMANDO":PRINT
@270,"BY":PRINT@396,"1985";
190 PRINT@327,"ANTHONY FRERKING"
:PRINT@455,"<PRESS ANY KEY>";
200 AS=INKEY$:IF AS="" THEN 200
210 CLS:INPUT"YOUR NAME";N$
213 GOSUB7000
215 IF N$="ARF" THEN 230
220 GOSUB 5000
230 PMODE 3,1:PCLS:SCREEN1,1
235 IF N$="ARF" THEN 265
240 FORI=1TO170:R=RND(8):CIRCLE(
127,96),I,R:NEXT
250 FORJ=1TO5:FORI=1 TO 8:I$=STR
$(I):DRAW"C"+I$+";BM10,60;NR30D4
0R30BRI0NR20U20R20D20BRI0U20F10E
10D20BRI0U20F10E10D20BRI0U10E10F
10NL20D10BRI0U20F20U20BRI0NR10D2
0R10E10H10BR20D20R20U20L20":NEXT
:NEXT
260 GOSUB5000
265 R$(1)="GARBAGE SCRUBBER":R$(
2)="COOK":R$(3)="FOOT SOLDIER":R
$(4)="MINER":R$(5)="COMMANDO 2ND
CLASS":R$(6)="COMMANDO 1ST CLAS
S"
270 TT=0:LV=1:PH=1:SC=0:LI=3
280 PLS(1)="";L2D2R2D8L2":PL$(2)=
";L2D2R2NG4NF4D4NG4F4"
290 C$="";G5ND5R5ND5R5ND5H5U5L5D2
R5":PL$(3)="";G6ND2BU4NU2F4NE4R4N
H4E4NU2BD4ND2H4NL4H2":B$(1)="";R4
G8NR8D2NR8F4NU4E4U2H8L4D8"
300 B$(2)="";BR4G2H2G2NH2D6E2F2NU
6R2G2D2NL4G2H2U2H2R2"
310 T$="";G2D1F2L3G2NR15D1F2R11E2
U1H2L3NL5E2U1NR3H2L5NU2"
320 BL$="";NU15ND15NR10NL10NE5NH5
NG5F5"
350 'LV1
360 PH=1:AR=0:H=230:V=148:V$=STR
$(V):E=1:H$=STR$(H):R=7:Q=LV*10+
20:R=R-L:TI=0
370 PCLS(1):DRAW"C2;BM0,140;R40G
5L15D15R220U10H10R25":PAINT(0,16
0),2,2
380 DRAW"C3;BM10,130"+PL$(3)
390 DRAW"C4;BM240,130"+C$
400 'MOVE
410 DRAW"C1;BM"+H$+"", "+V$+PL$(E)
420 J(1)=JOYSTK(0)
430 IF J(1)<=15 THEN H=H-5 ELSE
IF J(1)>=55 THEN H=H+5 ELSE 460
440 IF H>=230 THEN H=230 ELSE IF
H<=30 THEN 4010
450 IF E=1 THEN E=2 ELSE E=1
460 H$=STR$(H):DRAW"C3;BM"+H$+"",
 "+V$+PL$(E)
470 TI=TI+1
500 'ENEMY
510 IF AR=1 THEN500
520 Y=RND(2):IF Y<>1 THEN 400
530 Y=RND(2):ON Y GOTO 540,550
540 F=32:X=(RND(14)*10)+65:Y=1:G
OTO 560
550 F=20:X=(RND(16)+10)+50:Y=2
560 HBS=STR$(X):VBS=STR$(Q):DRAW
"C4;BM"+HBS+"", "+VBS+B$(Y)
570 Z=Q:AR=1
580 GET(X-5,Z)-(X+5,Z+15),C,G

```

```

590 PUT(X-5,Z)-(X+5,Z+15),D
600 Z=Z+10:SOUND 100-Z,1:IF Z<15
0 THEN 660
610 Z=8:PLAY"V3003T4L1C":ON Y GO
TO 620,640
620 COLOR Z,8:LINE(X-7,160)-(X+7
,135),PSET,BF
630 FORI=1TO20STEP3:CIRCLE(X-I,1
25),10,Z:CIRCLE(X+I,125),10,Z:NE
XT:Z=Z-3:IF Z<4 THEN 650 ELSE 62
0
640 FORI=1TO20STEP3:FORJ=4TO1 ST
EP-1:CIRCLE(X,160),I,J,1,.5,1:NE
XT:NEXT
650 IF H>=X-F AND H<=X+F THEN 30
0 ELSE AR=0:GOTO 400
660 PUT(X-5,Z)-(X+5,Z+15),C,PSET
:GOTO 400
1000 'LV2
1010 PH=2:N=10:H=225:U=103:V=30:
AR=0:TI=0:AA=115:AB=0
1015 Y=50:R=150
1020 PCLS(1)
1030 DRAW"C2;BM0,80;R100F20D10L1
0H10G10D30L40G10D10R200U60L70G10
L10U10E20R90":PAINT(0,100),2,2
1040 DRAW"C4;BM60,150"+T$
1050 FORI=1TOLV
1060 LINE(90,(I*5)+110)-(100,(I*
5)+110),PSET
1065 NEXT
1070 DRAW"C3;BM210,150"+PL$(2)
1080 DRAW"C4;BM90,70"+PL$(2)
1090 IF LV>2 THEN DRAW"C4;BM180
,70"+PL$(2)
1100 IF LV<3 THEN Q=1 ELSE IF LV
<4 THEN Q=2 ELSE Q=3
1110 GET(Y,R)-(Y+15,R+10),D,G
1150 'MOVE
1160 COLOR1,1:LINE(H,V)-(H+12,V+
9),PRESET,BF
1170 J(1)=JOYSTK(0):J(2)=JOYSTK(
1):FR=PEEK(65280)
1180 FORI=1TO2:IF J(I)<=15 THEN
X(I)=-5 ELSE IF J(I)>=55 THEN X(
I)=5 ELSE X(I)=0
1190 NEXT
1200 H=H+X(1):V=V+X(2)
1210 IF H=240 THEN H=240 ELSE I
F H<=30 AND AB=0 THEN H=30
1220 IF V<=20 THEN V=20
1230 PUT(H,V)-(H+15,V+10),C,PSET
1240 IF PPOINT(H-1,V-1)=6 OR PPO
INT(H+14,V)=6 OR PPOINT(H+14,V+1
0)=6 OR PPOINT(H-1,V+10)=6 THEN
3000
1250 IF H<=30 AND AB=1 THEN 4150
1260 IF FR=254 OR FR=126 THEN GO
SUB1300
1270 TI=TI+1:GOTO 1400
1300 'FIRE
1310 IF H<200 THEN SOUND1,1:RETU
RN
1320 IF V<110 THEN SOUND 1,1:RET
URN
1330 AB=1:COLOR4,1:LINE(H+6,V+5)
-(210,135),PSET:SOUND250,1:LINE(
H+6,V+5)-(210,135),PRESET
1340 DRAW"C5;BM210,150"+PL$(2)
1350 RETURN
1400 'ENEMY
1410 IF AR=1 THEN 1460 ELSE AR=1
1420 IF Q=1 THEN 1440 ELSE Z=RND
(2)
1430 N=170:IF Z=1 THEN X=0 ELSE
X=-5
1440 Z=RND(2):IF Z=1 THEN M=0 EL
SE M=5
1450 SOUND 100,1:Z=100:T=80
1460 IF Q=1 THEN 1470 ELSE PRESE
T(N,T):N=N+X
1470 PSET(2,T,1):Z=Z+M:T=T-5:PSE
T(Z,T,2):IF Q>1 THEN PSET(N,T,2)
1480 IF T<=25 THEN PSET(Z,T,1):P
SET(N,T,1):PSET(U,AA,1):AR=0:GOT
O1510
1490 IF T<=V+8 AND T>=V THEN 150
0 ELSE 1510
1500 IF (Z<=H+15 AND Z>=H)OR(Q>1
AND N>=H+15 AND N<=H) THEN 3000
1510 IF Q<3 OR AB=1 THEN 1540
1520 LINE(Y-5,R-5)-(Y+11,R+10),P
RESET,BF:V=Y+INT(LV/2):H$=STR$(Y
):V$=STR$(R):DRAW"C4;BM"+H$+"", "+

```

```

V$+T$
1530 IF Y+15>=210 THEN LINE(Y,R)
-(Y+16,R+10),PRESET:GET(Y,R)-(Y+
16,R+10),D,G:GOTO 3000
1535 PLAY"V30T801L16;C;D"
1540 IF AR=1 THEN 1560
1550 AA=RND(LV):AA=110+(AA*5):U=
100
1560 PSET(U,AA,1):U=U+5:PSET(U,A
A,4)
1570 IF AA>=V AND AA<=V+10 THEN
1580 ELSE 1150
1580 IF U>=H AND U<=H+15 THEN 3
000 ELSE 1150
2000 'LV3
2010 R2=20:S2=110:TI=0:PCLS(1):R
1=130:S1=170:HA=220:X=10:Y=160:H
=45:V=10:PH=3:AA=135:ZA=180:AR=0
:E=0
2015 R3=135:S3=130
2020 GET(20,20)-(35,35),D,G
2030 DRAW"C2;BM0,40;R100BG10G20R
20E10H10BE10F20R30F20D10L140G10D
10G10D20G10D10R30F10R170E20U130H
25"
2040 PAINT(10,100),2,2:PAINT(10,
180),2,2
2050 COLOR 4,1:FORI=1TOLV:LINE(2
0,15+(I*5))-(230,15+(I*5)),PSET
:LINE(220,85+(I*5))-(230,85+(I*5
)),PSET
2060 NEXT
2070 PUT(H,V)-(H+16,V+10),C,PSET
2080 DRAW"C2;BM50,110;D20R100E20
L120":PAINT(60,125),2,2
2100 'MOVE
2110 PUT(H,V)-(H+16,V+10),D
2120 J(1)=JOYSTK(0):J(2)=JOYSTK(
1)
2130 FORI=1TO2
2140 IF J(I)<=15 THEN X(I)=-5 EL
SE IF J(I)>=55 THEN X(I)=5 ELSE
X(I)=0
2145 NEXT
2150 H=H+X(1):V=V+X(2)
2160 IF V>=140 AND H+16 <=35 THE
N 4000
2170 IF PPOINT(H-1,V)=6 OR PPOIN
T(H-1,V+10)=6 OR PPOINT(H+14,V)=
6 OR PPOINT(H+14,V+10)=6 THEN 30
00
2180 IF V<=0 THEN V=5
2190 IF H<=10 THEN H=10
2200 PUT(H,V)-(H+15,V+10),C,PSET
2210 TI=TI+1
2250 'ENEMY
2260 IF AR=1 THEN 2300
2270 AR=1:IF V<=30 THEN 2280 EL
SE 2290
2280 T=218:M=(RND(LV)*5)+15:N=15
0:GOTO 2300
2290 T=218:M=(RND(LV)*5)+85:N=17
0
2300 PSET(T,M,1):T=T-3:PSET(T,M,
3)
2310 IF (H<=T AND H+15>=T) AND (
V<=M AND V+10>=M) THEN 3000
2320 IF T<=N THEN AR=0:PSET(T,M,
1)
2330 IF LV>=3 THEN 2340 ELSE 21
00
2340 PSET(R1,S1,2):R1=R1+1:IF R1
>135 THEN R1=130:S1=S1-1
2350 IF LV>=4 THEN 2360 ELSE 210
0
2360 PSET(R2,S2,2):S2=S2+1:IF S2
>115 THEN S2=110:R2=R2+1
2370 IF LV>=5 THEN 2380 ELSE 21
00
2380 PSET(R3,S3,2):R3=R3+1:IF R3
>140 THEN S3=S3+1:R3=135
2390 GOTO 2100
3000 'DEATH
3001 II=0
3005 IF PH >1 THEN 3040
3010 'LV1
3020 DRAW"C5;BM"+H$+"", "+V$+PL$(E
)
3030 FORI=150 TO 50 STEP-5:CIRCL
E(H,I),5,3:CIRCLE(H,I),5,5:NEXT:
V=45:H=H-5
3040 'LV1,LV2,LV3
3050 FORJ=1TO10:FORI=8 TO5 STEP-
1

```

continued on Page 40

# An Easy Way to Run Your Programs

By Andrew Dater

**H**ow many times have you typed in DIR and frantically hit the SHIFT-@ keys trying to find a program you wanted to run? Well, your troubles are over. *Disk Menu* takes all of your programs on a disk and displays them in a menu. Move the arrows over the program you want to run, press ENTER and away you go!

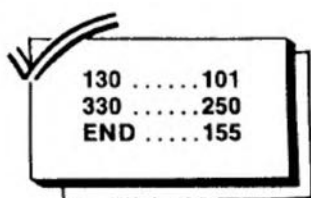
Combined with Roger Schrag's "A Special Use for the DOS Command" (November 1984, Page 140), *Disk Menu* is a very easy way of running your programs. Simply type in the program and save it as *MENU*, then run *Dosstart* and enter RUN"MENU" as the command

to be executed upon typing DOS. It sure is a keystroke saver! I put it on all of my disks, especially my RAINBOW ON TAPE disks. I just type in DOS and get a menu of the programs from THE RAINBOW all ready to run. *Disk Menu* only takes up one granule on the disk, so with it and the DOS command, you can save yourself a lot of typing with the sacrifice of only two granules.

When you run the program, you are prompted to enter the drive number. Just press 0-3 without pressing ENTER and *Disk Menu* loads the directory from the disk to memory. You are then presented with a menu of the programs

on your disk. Use the arrow keys to move the "> <" symbols over the program you want to run and press ENTER. The arrow keys repeat, so if you just want to move the pointers one space, be sure to release the arrow key quickly. If there are more than 30 programs on your disk, they will not fit on one screen, so press 'M' (for "more") to switch between screens. To switch disks or drives, press CLEAR to restart the program.

You can also get the free granules on your disk very easily: press 'F' and it displays how many are free. Press any key to get back to the menu.



The listing: DISKMENU

```

130 ..... 101
330 ..... 250
END ..... 155

```

```

12,1)) :N=N+1:IFN=69THEN100
90 NEXTI,X
100 N=N-1:FORX=1TON:N$(X)=LEFT$(
N$(X),8)+". "+MID$(N$(X),9,3):NEX
T
110 CLS
120 IFN<=30THENPP=1:N1=N
130 IFN>30THENPP=2:N1=30
140 IFN>60THENPP=3:N1=30
150 FORX=1TON1:PRINT "N$(X),:NE
XTX
160 IFPP=2ORPP=3THENPRINT@480,"P
RESS <M> FOR MORE...";
170 L$=">":R$="<":P=Y=1
180 PRINT@P,L$;:PRINT@P+13,R$;
190 FORX=338TO345:POKEX,255:NEX
T
200 IFPEEK(338)=191THEN390
210 IFPEEK(339)=191THENCLEAR:GOT
O40
220 IFPEEK(341)=247THEN350
230 IFPEEK(342)=247THEN360
240 IFPEEK(343)=247THEN370
250 IFPEEK(344)=247THEN380
260 IFPEEK(344)=254THEN290
270 IFPEEK(343)=253THEN300
280 GOTO200
290 CLS:PRINTFREE(DR)"FREE GRANU
LES ON DRIVE"DR:PRINT:PRINT"PRE
S ANY KEY TO CONTINUE":EXEC44539
:GOTO100
300 IFPP=2ANDY=1THENCLS:FORX=31T
ON:PRINT "N$(X),:NEXT:PRINT@480
,"PRESS <M> FOR MORE...";:Y=2:P=
0:N1=N-30:GOTO100
310 IFPP=2ANDY=2THENY=1:P=0:N1=3
0:GOTO110
320 IFPP=3ANDY=1THENCLS:FORX=31T
ON:PRINT "N$(X),:NEXT:PRINT@480
,"PRESS <M> FOR MORE...";:Y=2:P=
0:N1=N-59:GOTO100
330 IFPP=3ANDY=2THENCLS:FORX=62T
ON:PRINT "N$(X),:NEXT:PRINT@480
,"PRESS <M> FOR MORE...";:Y=3:P=
0:N1=N-61:GOTO100
340 IFPP=3ANDY=3THENY=1:P=0:N1=3
0:GOTO110
350 IFP<17THEN180ELSEP=P-32:PRIN
T@P+32,"":PRINT@P+45,"":GOTO
180
360 IF P/16=>(N1-2)THEN190ELSEP=
P-32:PRINT@P-32,"":PRINT@P-19,
" ":GOTO180
370 P=P-16:IFP<0THENP=0:GOTO180E
LSEPRINT@P+16,"":PRINT@P+29,"
":GOTO180
380 IF P/16=>(N1-1)THEN190ELSEP=
P+16:PRINT@P-16,"":PRINT@P-3,"
":GOTO180
390 F=P/16+(Y-1)*30+1:F$=N$(F)+
"+DN$:TP=TP(F)
400 CLS
410 IFTP=0THENLOADF$,R
420 IFTP=2THENLOADMF$:CLEAR200:P
OKE&HFF40,"":EXEC:END
430 CLS:PRINT"IS FILE: "F$" A BA
SIC":PRINT"PROGRAM? (Y/N)"
440 I$=INKEY$:IFI$="Y"THENLOADF$,
R ELSEIFI$="N"THENCLEAR200:ENDE
LSE440

```



*Producing sounds without the PLAY or SOUND commands*

# Further Adventuring Into Sound Experimentation

By Bill Bernico

(Editor's Note: This article is an addendum to "An Adventure Into Sound Experimentation," which appeared in the December 1985 edition, Page 35. For more complete information, please refer to this previous article.)

**H**alf the fun of programming is trying to discover the unusual, the bizarre and the humorous. I think I've combined all three elements in Sound Story 2, a continuation of my original Sound Story program. The significant difference with this installment is that it doesn't use any SOUND or PLAY commands. That's right, there's another way to generate sounds - one I stumbled on purely by accident. Remember, experimentation often leads to some of the best ideas.

In order to generate the odd sounds contained in this program I had to POKE values into memory. By trial and error, I found the combination of values that were right for each sound. I had some help finding these values, though, in the form of the second program listing, Random Sound Generator. With it, you can hear randomly created sounds and see the values that went into making up the sound.

From there you can jot down the values and insert them into the proper slots in the main program.

If you'd like to make up your own sounds, be it for part of a program or a menu-driven sound selection such as SS2, first run RANDOM SOUND GENERATOR. Immediately you'll hear an odd sound, followed by a listing like this:

```
Number of times (N)=2
Duration value (D)=88
Start Address (S)=1327
Ending Address (E)=2784
```

The number of times is fixed at '2', but the duration, start address and end address are random. The values for 'N' and 'D' can be from 255. The start and end addresses can be any value up to 65,536. Keep this in mind when you use values from here for your program. The difference between the 'S' and 'E' values is restricted to a maximum of 3,000 simply because it might randomly generate a start address of, for example, 123, and an end address of 65,000 with a duration of 255. You could wait a long time to hear that kind of sound played twice (the value of 'N').

## THE MAIN PROGRAM

I purposely stayed away from sounds you might hear in any run-of-the-mill program and instead created some unusual sounds from which to pick. When the menu page appears, simply press any number key from 1-9 to hear a sound or press 'E' to end the program. Each sound selection features a short description of what you're hearing.

The idea behind selection number '9' is just what it says. It's randomly selected sound that can be named whatever you like. Unless you select number '9' 3,000 times, chances are you won't hear the same sound twice. Sometimes it's short and barely audible and sometimes it's 10 seconds long, so listen closely for it.

Listing 1: SOUND 2

```

10 'SOUND STORY 2 (THE SEQUEL)
20 'BY BILL BERNICO
30 '708 MICHIGAN AVE.
40 'SHEBOYGAN, WI 53081
50 '(414) 459-7350
60 '
70 DATA 16,190,63,0,190,63,3,166
,128,72,72,183,255,32,141,12,31,
16,179,63,5,38,240,49,63,38,233,
57,182,63,2,74,38,253,57
80 FOR I=16135 TO 16169
90 READ A
100 POKE I,A
110 NEXT I
120 DEFUSR0=16135
130 POKE 65281,(PEEK(65281) AND
247)
140 POKE 65283,(PEEK(65283) AND
247)
150 POKE 65315,(PEEK(65315) OR 8
)
160 CLS:PRINT@3,"SOUND STORY 2 (
THE SEQUEL)
170 PRINT@75,"selections:
180 N=0:D=0:S=0:E=0
190 PRINT@128,"1. 78 RPM RECORD
SCRATCHING
200 PRINT@160,"2. WILD AMINAL MA
TING CALL
210 PRINT@192,"3. SOMEONE DIALIN
G '555'
220 PRINT@224,"4. FRONT DOOR BUZ
ZER
230 PRINT@256,"5. WILLIE WONKA'S
FACTORY
240 PRINT@288,"6. A PEG-LEG CENT
IPEDE
250 PRINT@320,"7. DEATH RAY SPAC
E GUN
260 PRINT@352,"8. NUCLEAR POWERE
D HUMMINGBIRD
270 PRINT@384,"9. (RANDOM) YOU N
AME IT
280 PRINT@448,"SELECT (1-9) OR e
ND
290 A$=INKEY$:IF A$=""THEN 290
300 IF A$="E"THEN CLS:END
310 M=VAL(A$):ON M GOTO 330,340,
350,360,370,380,390,400,410
320 GOTO 290
330 CLS:PRINT@132,"A 78 RPM PHON
OGRAPH NEEDLE AT THE END OF
THE RECORD.":N=6:D=33:S=56789:E
=60000:GOTO420
340 CLS:PRINT@132,"IT'S THE MATI
NG CALL OF PURPLE-BELLIE
D, FRILLY-CROWNED, 3-TO
OTH SNIPE.":N=6:D=143:S=44:E=999
:GOTO420

```

```

350 CLS:PRINT@130,"THIS IS WHAT
IT SOUNDS LIKE FROM YOUR END
WHEN YOU DIAL '555'.":N=3:D
=190:S=888:E=2222:GOTO420
360 CLS:PRINT@132,"IT COULD ALSO
BE THE BACK DOOR BUZZER."
:N=255:D=2:S=11:E=111:GOTO420
370 CLS:PRINT@132,"YOU'RE INSIDE
THE FACTORY WHERE THEY MA
NUFACTURE THE EVERLASTING G
OB STOPPERS!":N=10:D=33:S=333:E=
3333:GOTO420
380 CLS:PRINT@132," READY...MA
RCH. LEFT,RIGHT,RI
GHT,RIGHT, LEFT,RIGHT,RI
GHT,RIGHT, LEFT,RIGHT,RI
GHT,RIGHT...":N=8:D=108:S=809:E=
2334:GOTO420
390 CLS:PRINT@132,"YOU JUST GOT
ZAPPED BY COMMANDER COM
MOTION OF THE 33RD STAR
FLEET!":N=35:D=44:S=66:E=444:GO
TO420
400 CLS:PRINT@132,"IF THERE WAS
SUCH A THING, THIS IS WHAT
IT WOULD SOUND LIKE...OR WOU
LD IT?":N=77:D=7:S=77:E=777:GOTO
420
410 CLS:PRINT@132,"WHAT THE HECK
WAS THAT?":N=2:D=RND(255):S=RND
(3000):E=RND(3000)+S
420 POKE 16128,INT(N/256):POKE 1
6129,N-INT(N/256)*256
430 POKE 16130,D
440 POKE 16131,INT(S/256):POKE 1
6132,S-INT(S/256)*256
450 POKE 16133,INT(E/256):POKE 1
6134,E-INT(E/256)*256
460 A=USR0(0)
470 GOTO 160

```

Listing 2: SOUNDGEN

```

10 'RANDOM SOUND GENERATOR
20 'USED TO PRODUCE SOUNDS IN
SOUND STORY 2 (THE SEQUEL)
30 '
40 DATA 16,190,63,0,190,63,3,166
,128,72,72,183,255,32,141,12,31,
16,179,63,5,38,240,49,63,38,233,
57,182,63,2,74,38,253,57
50 FOR I=16135 TO 16169
60 READ A
70 POKE I,A
80 NEXT I
90 DEFUSR0=16135
100 POKE 65281,(PEEK(65281) AND
247)
110 POKE 65283,(PEEK(65283) AND
247)
120 POKE 65315,(PEEK(65315) OR 8
)

```

continued on Page 18

The last of a four-part series on operating  
with this BBS software

# CoBBS:

## How to Modify the Program to Use the CoCo 'Serial Port'

By Richard Duncan

Written for use with the RS-232 Pak, *CoBBS* takes advantage of it being a true serial port. The "serial port" on the back of the CoCo was designed as a printer port, but through the miracle of software can be used as a communications port (but still not a true RS-232 port, just voltage compatible). *CoBBS* can be modified to use the serial port, but it loses a lot of its features, including advanced key input, no pausing or stopping while a message or file is being displayed, no uploads, no 1200 Baud, slower operation and awkward termination of a call.

If I sound pessimistic about serial operation using the serial printer port, I am! We will discuss briefly some of the modifications required to convert *CoBBS* and its operation. You will have to do the installation depending on your needs. I strongly suggest obtaining an RS-232 Pak if you are serious about running this BBS software, but for those who want to experiment... here we go.

Changing *CoBBS* over to the serial port requires modification of the serial driver and all the BASIC routines. The main difference is that the serial version pauses anytime it is polled and waits for a character, where the Pak returns a

CHR\$(0) and returns to BASIC. Throughout the programs, the system jumps to the single key input routine of the driver to strip any extra character waiting to be received, making sure no extraneous character is in the buffer.

The following lines of *USER/SYS* have the statement EXEC4314, or EXEC&H10DA, in them and should be removed: 40, 68, 70, 150, 180, D266, 345, D1205, 1225 and D7035. If there is a 'D' preceding the number, delete the whole line and replace it with a REM statement. The following lines in *COBBS/SYS* to change are: 410, 440, 960, D975, 1005, 1270, 1345, 1420, 1465, 2085, 2410, D7050 and 7057.

The carrier detect routine must also be changed. The way to do this is to check the CD flag set via the serial port. The port should first be reset by the command K=PEEK(&HFF20). Then, by monitoring the location of \$FF21 for a change in state, you will know when a carrier is coming in. The command CD=PEEK(&HFF21) checks the flag. If the value of 'K' is greater than 100, the system has detected a carrier. After detecting a carrier, again issue the command K=PEEK(&HFF20). From this point on the value of 'K' should be less than 100. If not, it means the last user

has dropped his carrier, this is a new caller and the system needs to be re-booted. The carrier detect subroutine for *USER/SYS* should read: 9700 '- CD CHECK-9705 IFPEEK(&HFF21)=180 THEN CLOSE:RUN 9710 RETURN. In *COBBS/SYS* replace the RUN with LOAD"USER/SYS".R.

Next, you must devise a way to hang up your modem when you want to terminate a call. This might be done with the "+++" and "ATH" with the Hayes modem, or through use of the cassette relay and the MOTOR ON/OFF command. This is done in the 9800's subroutine.

The listing provided is used to load in *COTERM/BIN* for the RS-232 Pak, convert it for the serial port and save it back out *under the same name*. You cannot use the C/R modification with the serial port version. It is hard to modify a big program to be used a different way from the way it was originally written.

With some effort and patience you will be able to get a basic version of *CoBBS* running with the printer port. If you want the Pak version along with a documentation disk, send \$25 to me at 2504 N. Gathings Drive, West Memphis, AR 72301. □

### The listing: LOADER

```

10 'THIS ROUTINE WILL LOAD IN
20 'COTERM/BIN, MODIFY IT FOR
30 'THE SERIAL PORT AND SAVE
40 'THE MODIFIED VERSION OUT TO
50 'DISK. this routine will
60 'overwrite the original versi
on!
70 LOADM"COTERM"
80 A=4240
90 READ D$:IF D$="END" THEN 110
100 POKE A,VAL(D$):A=A+1:GOTO90
110 SAVEM"COTERM/BIN",&H0E00,&H1
2BF,&H1090
120 CLS:PRINT@260,"COTERM/BIN MO
DIFIED":END
130 DATA 67,48,141,0,81,188,1,10
4,39,50
140 DATA 182,1,103,167,141,0,218
,190,1,104
150 DATA 175,141,0,212,182,1,106
,167,141,0
160 DATA 207,190,1,107,175,141,0
,201,134,126
170 DATA 183,1,106,183,1,103,48,
141,0,36
180 DATA 191,1,104,48,141,0,4,19
1,1,107
190 DATA 57,15,112,13,111,16,38,
0,169,127
200 DATA 255,64,50,98,141,44,129
,3,38,2
210 DATA 134,42,183,17,129,57,52
,2,18,18
220 DATA 18,18,18,18,18,18,18,18
,150,111
230 DATA 53,2,16,38,0,129,141,62
,129,13
240 DATA 38,6,134,10,141,54,134,
13,32,115
250 DATA 52,21,26,80,173,159,160
,0,39,2
260 DATA 32,36,182,255,34,71,37,
242,141,84
270 DATA 182,255,34,71,37,242,79
,52,2,198
280 DATA 7,141,69,182,255,34,18,
71,102,96
290 DATA 90,38,244,141,55,53,2,6
8,53,149
300 DATA 52,23,26,80,246,255,33,
193,180,38
310 DATA 2,32,31,127,255,32,141,
34,52,2
320 DATA 198,8,100,96,73,73,183,
255,32,18
330 DATA 141,20,90,38,243,134,2,
183,255,32
340 DATA 141,8,50,97,53,151,141,
0,141,0
350 DATA 141,0,141,0,174,141,0,5
,48,31
360 DATA 38,252,57,0,182,126,203
,74,126,197
370 DATA 143,13
380 DATA END

```

# RENUM

with a Twist

By Fredric M. Haberer

Ordinarily, using RENUM is simple and straightforward: You have been working on a BASIC program for some time, and additions and revisions have filled all the gaps between line numbers, yet another line needs to be inserted. A little work space at the beginning of the program would be nice, so your new start line becomes 100. You want to renumber from the start, and the present first line is '7'. Line increments of 10 keep things simple and leave room for new lines. So, you type RENUM 100,7,10, and ENTER.

If the program is just a few lines long, OK appears on the screen. If the program contains 300 lines, the processing takes a few seconds. In either case, the lines are renumbered. And, significantly, every GOSUB and GOTO is renumbered to its new target line number.

There's nothing so unusual about that, but the CoCo's method of renumbering GOSUBS and GOTOS makes possible quite a different use for the RENUM function. Suppose you have been developing a program for some time. It has numerous branches; even the branches have branches. As you revise, reorganize and consolidate program lines, you lose track of the GOSUBS, the GOTOS and their target lines. As you run the pro-

gram, UL (Undefined Line) Errors come up in frustrating profusion. RENUM is the answer to your problem — if you employ a special twist.

When you enter RENUM 100,7,10, as in the example, the CoCo attempts to reconcile all GOSUBS and GOTOS with their target lines. If, in editing, you have deleted target lines (REM statements, for example, which never should have been GOSUB targets in the first place), a statement such as the following appears on the screen: UL 3766 in 550. Roughly translated, this means: "In Line 550, there is an instruction to go to Line 3766, but no such line is in your program."

Now you know that newly designated Line 550 contains a GOSUB or GOTO targeted to Line 3766. Unfortunately, Line 3766 did not exist in the first place, and you haven't the slightest idea where it would be in the newly numbered sequence. If the program is a long one, you're better off reloading the original program and starting over. If only the CoCo could have identified those ULs before renumbering, you would have been saved hours of tracing and decoding.

As it turns out, the CoCo can do just that, but you will have to do some minor subterfuge. Just ask the CoCo to do the impossible: tell it to renumber using a starting line number that doesn't exist — a line beyond the range of your program. For example, your program

starts on Line 10 and ends on Line 15277. You tell CoCo to renumber starting at Line 16000; enter RENUM 16000,16000,1. (It's important that both the start line and the new first line be numbered higher than the highest line in the program. Otherwise, you'll get an FC Error.)

The CoCo first searches for line numbers that can't be reconciled, then attempts to renumber. There is no Line 16000 from which to start renumbering. Therefore, it gives up, dutifully lists the unreconciled lines and says, "OK." There's no error message, no cough and no sputter. CoCo has done its best to renumber as asked, and has instantly done a heap of work for you. On your screen appears the number of each line containing an unreconciled GOSUB or GOTO and the number of its target line.

Your original line numbering remains intact. The tedious job of finding the errors is done. You now know which lines to list and edit. After you've made your corrections, if you still want to renumber, you may go ahead and do it.

You might not have wanted to renumber in the first place. In this case, renumbering is a debugging tool that locates your UL problems, and keeps its "fingers" off of your numbering system.

Of course, this procedure cannot identify incorrect target lines if the lines actually exist. However, it is a real headache-reliever in the case of a long program with holes in it.

*(Fred Haberer teaches junior and senior high school English at WACO High School in Olds, Iowa.)*

continued from Page 16

```

130 CLS
140 N=2:D=RND(255):S=RND(3000):E
=RND(3000)+S
150 POKE 16128,INT(N/256):POKE 1
6129,N-INT(N/256)*256
160 POKE 16130,D
170 POKE 16131,INT(S/256):POKE 1
6132,S-INT(S/256)*256
180 POKE 16133,INT(E/256):POKE 1
6134,E-INT(E/256)*256
190 A=USR0()
200 PRINT@6,"random sound genera

```

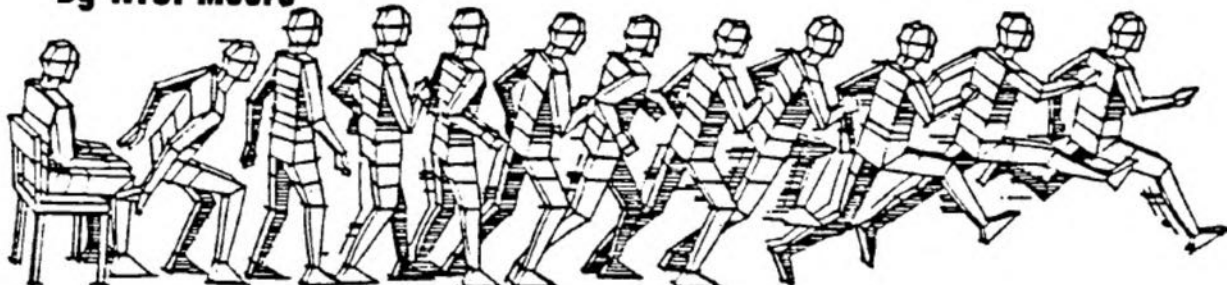
```

tor"
210 PRINT:PRINT
220 PRINT"NUMBER OF TIMES (N)=";
N
230 PRINT"DURATION VALUE (D)=";
D
240 PRINT"START ADDRESS (S)=";
S
250 PRINT"ENDING ADDRESS (E)=";
E
260 PRINT@484,"HIT ANY KEY TO DO
ANOTHER";:EXEC44539
270 GOTO 130

```

# Robocise

By W.J. Moore



The next time you exercise and would like to have a companion, try "Robert the Robot." He can exercise at any speed and never gets tired. You can adjust Robert's speed by pressing the 'F' key to go faster or by pressing the 'S' key to go slower. Holding the key down does not work; press the key repeatedly.

After typing in this program, use the RAINBOW Check Plus program (see "Rainbow Info" for an explanation on how to use the Check Plus). You may also type in the following in the direct command mode to check if all DATA statements are entered correctly.

CLEAR (ENTER)

FOR I=1 TO 610:READ A:T=T+A:NEXT (ENTER)

PRINT T (ENTER)

The value of 'T' should equal 15177. If it does not, then something is wrong in the DATA statements. It is important to have the correct data since the program would be wiped out. It is always wise to save what you have while debugging a program or else you may have to retype the entire program.

It is recommended that all users enter PCLEAR 8 before running the program. This should resolve any differences between systems and/or ROM sets. Also, if you have a 32K 'D' board CoCo, the program may not run the first time. If this occurs, simply try it a second time and it should run.

```
240 .....49
500 .....139
710 .....170
890 .....129
END .....173
```

The listing: ROBOCISE

```
10 REM ROBOCISE BY W.J. MOORE -
PITTSBURG CALF
20 ' DISPLAY PICTURE
30 PCLEAR8
40 L=PEEK(186)*256+PEEK(187)
50 PMODE4:POKE179,32:PCLS
60 CLS:PRINT@264,"BUILDING A ROB
OT"
70 PRINT@327,"ROBERT IS HIS NAME
"
80 N=L+19*32:GOSUB630
90 N=L+21*32:GOSUB630
100 N=L+24*32:GOSUB630
110 N=L+28*32:GOSUB630
120 N=L+33*32:GOSUB630
130 N=L+39*32:GOSUB630
```

```
140 N=L+46*32:GOSUB630
150 C=0:X=L+11:N=X:GOSUB490
160 PCOPY1TO3
170 C=0:X=L+7:N=X:GOSUB490
180 PCOPY1TO4
190 PCOPY3TO1
200 C=0:X=L+16:N=X:GOSUB490
210 PCOPY1TO5
220 C=0:X=L+1924:N=X:GOSUB490
230 A$=" PRESS: F=FASTER S=SLOWE
R "
240 FORI=1TO LEN(A$):T$=MID$(A$,
I,1):T=ASC(T$)
250 IF T<64 THEN T=T+64:MID$(A$,
I,1)=CHR$(T)
260 NEXT
270 C=0:N=L+2884
280 FORI=1TO LEN(A$):T=ASC(MID$(
A$,I,1))
290 FOR C=0TO5:POKEN+C*32,T:NEXT
300 N=N+1
310 NEXT
320 L=PEEK(186)/2:A=65478
330 FORP=0TO6:N=INT(2^P)
340 IF L AND N THEN POKE A+P*2+1
```

```

,Ø ELSE POKE A+P*2,Ø
35Ø NEXT
36Ø N=PEEK(65314):POKE65314,(N A
ND 7)
37Ø POKE65472,Ø:POKE65474,Ø:POKE
65477,Ø
38Ø X=5Ø:PLAY"ØIT2Ø"
39Ø PCOPY4TO1:PLAY"C+":GOSUB44Ø
4ØØ PCOPY3TO1:PLAY"C":GOSUB44Ø
41Ø PCOPY5TO1:PLAY"C+":GOSUB44Ø
42Ø PCOPY3TO1:PLAY"C":GOSUB44Ø
43Ø GOTO39Ø
44Ø FORI=1TOX
45Ø SP$=INKEY$
46Ø IF SP$="F" THEN X=X-5
47Ø IF SP$="S" THEN X=X+5
48Ø NEXT:RETURN
49Ø READ A,B
5ØØ IF A=-1 THEN F=1 ELSE F=Ø
51Ø IF A=-9 THEN 6ØØ
52Ø IF A=-99 THEN 62Ø
53Ø A=A+127
54Ø FOR I=1 TO B
55Ø IF F=1 THEN 57Ø
56Ø POKE N,A
57Ø N=N+32
58Ø NEXT
59Ø GOTO49Ø
6ØØ C=C+1:N=X+C
61Ø GOTO49Ø
62Ø RETURN
63Ø FORI=ØTO31:POKE N+I,223:NEXT
:RETURN
64Ø ' DATA FOR PICTURE
65Ø DATA-1,12,8Ø,14,128,2,123,2,
-9,
66Ø DATA-1,12,8Ø,4,75,1Ø,123,4,-
1,12,118,7,-9,
67Ø DATA8Ø,9,-1,3,8Ø,12,128,1,8Ø
,11,128,1,8Ø,6,128,6,-9,
68Ø DATA8Ø,1,48,2,8Ø,3,64,2,8Ø,1
,7Ø,3,8Ø,12,128,1,8Ø,11,128,1,8Ø
,6,128,6,-9,
69Ø DATA8Ø,4,-1,,8Ø,2,64,1,8Ø,16
,128,1,8Ø,3,-9,
7ØØ DATA8Ø,1,48,2,8Ø,3,64,2,8Ø,1
,75,3,8Ø,12,128,1,8Ø,11,128,1,8Ø
,6,128,6,-9,
71Ø DATA8Ø,9,-1,3,8Ø,12,128,1,8Ø
,11,128,1,8Ø,6,128,6,-9,
72Ø DATA-1,12,8Ø,4,7Ø,1Ø,118,4,-
1,12,123,7,-9,
73Ø DATA-1,12,8Ø,14,128,2,118,2,
-9,
74Ø FORI=ØTO31:POKEN+I,223:NEXT:
RETURN
75Ø DATA-99,
76Ø DATA-1,12,118,1,-1,2,118,1,-
9,
77Ø DATA-1,12,128,4,-9,
78Ø DATA-1,12,8Ø,4,-9,
79Ø DATA-1,12,8Ø,4,-9,
8ØØ DATA-1,16,1,3,96,1,1,1,96,1,
1,2,96,1,1,3,96,1,1,1,-9,
81Ø DATA-1,16,1,3,96,1,1,1,96,1,
1,2,96,1,1,3,96,1,1,4,96,1
82Ø DATA1,1,118,6,1,5,96,1,1,2,-
9,
83Ø DATA-1,31,128,1,8Ø,4,128,5,1
,5,96,1,1,2,-9,
84Ø DATA-1,31,128,1,8Ø,4,128,5,1
,5,96,1,1,2,-9,
85Ø DATA-99,
86Ø DATA-1,31,128,1,8Ø,4,128,5,1
,5,96,1,1,2,-9,
87Ø DATA-1,31,128,1,8Ø,4,128,5,1
,5,96,1,1,2,-9,
88Ø DATA-1,16,1,3,96,1,1,1,96,1,
1,2,96,1,1,3,96,1,1,4,96,1
89Ø DATA1,1,123,6,1,5,96,1,1,2,-
9,
9ØØ DATA-1,16,1,3,96,1,1,1,96,1,
1,2,96,1,1,3,96,1,1,4,96,1,-9,
91Ø DATA-1,12,8Ø,4,-9,
92Ø DATA-1,12,128,4,-9,
93Ø DATA-1,12,123,1,1,2,123,1,-9
,
94Ø DATA-99,
95Ø DATA32,1,27,2,32,1,27,3,-9,
96Ø DATA32,1,-1,2,32,1,27,1,22,1
,-9,
97Ø DATA27,4,-1,2,27,1,-9,
98Ø DATA7Ø,1,8Ø,1,75,3,8Ø,1,7Ø,1
,-9,
99Ø DATA8Ø,2,-1,3,8Ø,2,-9,
1ØØØ DATA-1,1,75,5,-9,
1Ø1Ø DATA64,1,59,2,64,1,59,2,64,
1,-9,
1Ø2Ø DATA64,1,54,2,64,1,54,2,64,
1,-9,
1Ø3Ø DATA-1,1,59,2,-1,1,59,2,-9,
1Ø4Ø DATA6,1,16,1,11,3,16,1,6,1,
-9,
1Ø5Ø DATA16,2,-1,3,16,2,-9,
1Ø6Ø DATA-1,1,11,5,-9,
1Ø7Ø DATA38,1,48,5,38,1,-9,
1Ø8Ø DATA48,1,-1,5,48,1,-9,
1Ø9Ø DATA43,1,-1,5,43,1,-9,
11ØØ DATA1Ø2,1,-1,5,1Ø2,1,-9,
111Ø DATA112,1,1Ø7,5,112,1,-9,
112Ø DATA-9,
113Ø DATA7Ø,1,8Ø,2,7Ø,1,-1,1,8Ø,
1,-9,
114Ø DATA8Ø,1,-1,2,8Ø,1,7Ø,2,8Ø,
1,-9,
115Ø DATA75,1,-1,3,75,2,-9,
116Ø DATA128,1,123,2,128,1,123,2
,128,1,-9,
117Ø DATA128,1,-1,2,128,1,-1,2,1
28,1,-9,
118Ø DATA123,1,-1,5,123,1,-9,
119Ø DATA-99,

```

# Gathering Up Scattered Programs

By Pete Eichstaedt

When I found I had several disks with only a few programs on each, I wrote *File Search*, a disk file "search and copy" utility. It seemed that every time I had a new idea for a program, I used a new disk. Although I still use the programs on the varied disks, most of them don't require their own disk, especially the machine language programs. *File Search* allowed me to put them on disks sorted by program type (BASIC, machine language and data). Those with 16K and only one drive will be happy to know that the program works fine on your machine. If you have two drives, the program will work faster because you don't have to keep swapping disks.

Here's how the program works: On startup, the program asks which is the source drive and which is the destination. If you only have one drive, answer '0' to both prompts. If you have more than one drive, you can use any single valid drive in your system, or any two drives from '0' to '3'. Next, CoCo asks which type of file you want to copy or if you want to copy all files from the source disk. If you want to copy all your BASIC files, but have some of them saved as ASCII files, don't worry. They're still

identified as BASIC programs. Once the questions have been answered, CoCo takes off and does the rest. The only interaction required is if only one drive is being used and a disk swap is required.

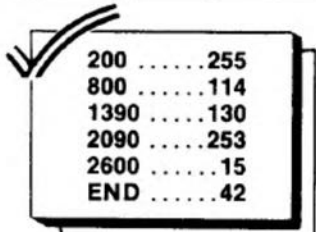
As the program runs, it reads the source disk directory, then checks the target disk directory to see if the program is there already. This saves the dreaded AE Errors common in copying. A message is displayed to show which file is being checked. You might see a comparison check being made on a file that doesn't look right. This is probably from a killed file, but don't worry — if the file isn't there, it can't be copied. If a file of the same name and format exists on both disks, it won't be copied, either. A message is displayed when a file is transferred.

When the copy is complete, CoCo asks if you want to transfer files from yet another disk. This keeps up as long as you answer "yes" and as long as the disk has room. If you run out of room while a copy is in progress, the program crashes with a DF Error — Disk Full. This is an acceptable concession when compared to having to type each COPY command manually.

When all the files are copied, answer "no" to the "search another disk" prompt. When you key in 'N', CoCo performs a cold start, just like on power up. If you just want to stop, change Line 1800 to CLOSE:END.

Of special note to single drive users: The program changes your selected single drive to the default drive for the system. As well, when disk changing prompts are displayed, a tone is generated to get your attention. Two tones are used: A low tone is emitted for required disk changes in the program proper; a higher tone is emitted when the BASIC system's COPY command is executed. If you don't change disks in the order requested, you get either an NE Error from the target disk not having the source program, or an AE Error from the source disk in the drive when CoCo is looking for the target disk.

If you don't get RAINBOW ON TAPE and have to type the program in manually, you can leave out all REMARK (\*) lines and lines 10 through 80. Suggestions and questions can be sent to me at the address at the start of the program listing. □



```

200 .....255
800 .....114
1390 .....130
2090 .....253
2600 .....15
END .....42
  
```

## The listing: FILESrch

```

Ø 1 * LINES Ø THROUGH 9Ø AND ALL
REMARKS LINES CAN BE DELETED WITH
OUT
1 1 * AFFECTING PROGRAM OPERATION
2 1 * SINCE I'M PROUD OF THE PROGRAM,
I'D RATHER YOU LEFT LINES
3 1 * 1Ø THROUGH 8Ø ALONE.
5 CLS
1Ø PRINT " *****
  
```

```

*****
2Ø PRINT " * FILESrch - DISK
FILE *"
3Ø PRINT " * COPY ROUTINE FOR
THE *"
4Ø PRINT " * COLOR COMPUTER W
/16K *"
5Ø PRINT " * BY PETE EICHSTAE
DT *"
6Ø PRINT " * APT D-3Ø8
*"
65 PRINT " * 2Ø45 PRENTISS DR
IVE *"
7Ø PRINT " * DOWNERS GROVE, I
L *"
75 PRINT " * 6Ø5
16 *"
8Ø PRINT " *****
*****
9Ø 1 * CLEAR AND ALLOCATE STRING
SPACE
95 GOTO 36ØØ
1ØØ CLEAR 1ØØØ: DIM PG$(72): DIM P
P$(72)
  
```

```

2ØØ INPUT "WHICH IS INPUT DRIVE"
; ID$: ID = VAL(ID$): IF ID < Ø OR I
D > 3 GOTO 25ØØ
3ØØ INPUT "WHICH IS OUTPUT DRIVE
"; OD$: OD = VAL(OD$): IF OD < Ø OR
OD > 3 GOTO 25ØØ
5ØØ PRINT "WHICH FILE TYPE SOULD
BE SOUGHT": PRINT " Ø = BASIC P
ROGRAM": PRINT " 1 = BASIC DATA
FILE": PRINT " 2 = MACHINE LANGU
AGE PROGRAM": PRINT " 3 = TEXT E
DITOR SOURCE FILE"
51Ø LINE INPUT " 4 = ALL "; FT$:
: IF FT$ < "Ø" OR FT$ > "4" GOTO 5Ø
Ø
52Ø FT = VAL(FT$)
55Ø IF ID <> OD THEN SOUND 5Ø, 3:
PRINT "PUT SOURCE DISK IN DRIVE"
; ID: INPUT "AND PRESS <ENTER>"; Z$
59Ø 1 * EACH DISK HAS 9 SECTORS F
OR RECORD ENTRIES
6ØØ FOR S = 3 TO 11
69Ø 1 * CLEAR THE PROGRAM RECORD
COUNTER
  
```

continued on Page 22

# Quick Restore

By John Galus

If you write BASIC programs that use a lot of data or long tables, such as in an Adventure game, you know how long it takes a program to search for a particular data item. Here is a short machine language routine called *Quick Restore* that allows you to restore to a specified line number.

As you may know, the RESTORE command permits repetitive use of the same data. It does this by resetting the data item pointer in \$33 to the beginning of your BASIC program. Whenever a READ command is performed, the interpreter looks through the entire BASIC program until it finds a DATA

statement, a somewhat slow process.

This machine language program gets the line number (in the variable LN) passed by theUSR function and stores it in \$2B. Then the line search routine is called at \$AD01. If this line is found, the address pointed to by Register X is bumped back by one and stored in \$33. If the line number is not found, a RESTORE is performed to the next higher numbered line in your program.

I have included a short program to illustrate the usefulness of this routine. It is written for a 32K Extended BASIC computer, but it is relocatable and will work on any size machine you have.

## The listing: RESTORE

```

1 'QUICK RESTORE
2 'JOHN GALUS
3 '55 WILKESBARRE AVENUE
4 'LACKAWANNA, NEW YORK 14218
10 CLEAR10, &H7FEF
20 CLS:X=&H7FF0:DEFUSR0=X
30 READ A:IF A=-99 THEN 50
40 POKE X,A:X=X+1:GOTO30
50 INPUT"ENTER ROOM NUMBER 1-4";
NU
60 LN=90+NU*10
70 Z=USR0(LN)
80 READ A$:PRINTA$:GOTO50
90 DATA 189,179,237,221,43,189,1
73,1,158,71,48,31,159,51,57,-99
100 DATA ROOM ONE
110 DATA ROOM TWO
120 DATA ROOM THREE
130 DATA ROOM FOUR

```

continued from Page 21

```

700 PG = 0:IF ID = OD THEN CLS:S
OUND 50,3:PRINT "PUT SOURCE DISK
IN DRIVE";ID:LINE INPUT "AND PR
ESS 'ENTER'";Z$
790 '* READ THE DIRECTORY SECTOR
S
800 DSKI$ ID, 17, S, DR$(1),DR$(
2)
890 '* IDENTIFY THE STRING TO MA
NIPULATE
900 FOR H = 1 TO 2
990 '* EACH RECORD HAS 32 BYTES
1000 FOR EN = 1 TO 128 STEP 32
1090 '* INCREMENT THE COUNTER
1100 PG = PG+1
1190 '* WE ONLY NEED THE FIRST 1
2 BYTES OF EACH RECORD
1195 '* BYTES 1-8 HAVE THE NAME,
9-11 THE EXTENSION, AND 12 HAS
THE FILE TYPE
1200 PG$(PG) = MID$(DR$(H),EN,12
)
1210 '* IF THE FIRST BYTE IS $0
THE RECORD WAS KILLED, GET THE N
EXT RECORD
1250 '* IF THE FIRST BYTE IS $FF
THERE ARE NO MORE ENTRIES (UNLE
SS YOU HAVE SOME WEIRD PROGRAM N
AMES)
1260 IF LEFT$(PG$(PG),1) = CHR$(
255) GOTO 1500
1290 '* CHECK THE FILE TYPE. IF
IT MATCHES, PROCEED, ELSE GET N
EXT RECORD
1300 IF RIGHT$(PG$(PG),1) = CHR$(
FT) THEN GOSUB 1900 ELSE IF FT=
4 GOSUB 1900
1390 '* GET NEXT ENTRY, CURRENT
STRING, SECTOR
1400 NEXT EN, H, S
1490 '* TRY AGAIN?
1500 CLS

```

```

1600 INPUT "SEARCH ANOTHER DISK"
;YN$
1700 IF LEFT$(YN$,1) = "Y" OR LE
FT$(YN$,1) = "y" THEN CLS:GOTO 2
00
1790 '* CLOSE OPENED FILES AND P
ERFORM A COLD START. REPLACE WI
TH "CLOSE:END" TO STOP COLD STAR
T
1800 CLOSE:POKE 113,0:EXEC &HA02
7
1890 '* PUT A "." BETWEEN THE NA
ME AND EXTENSION (SAME AS A "/"
)
1900 OP$ = LEFT$(PG$(PG),8)+"."+
MID$(PG$(PG),9,3)
1910 IF G<=1 THEN GOTO 2600 ELSE
GOTO 2630
1990 '* IF YOU'RE ONLY USING ON
E DRIVE, MAKE SURE IT'S THE DEFA
ULT DRIVE
2000 IF ID = OD THEN DRIVE ID
2030 PRINT:PRINT "COPYING ";OP$;
" TO DRIVE";OD
2050 IF LEFT$(PG$(PG),1) = CHR$(
0) GOTO 2300
2090 '* IF YOU'RE USING TWO DRIV
ES, COPY FROM THE INPUT DRIVE AN
D TO THE OUTPUT DRIVE
2100 IF ID <> OD THEN COPY OP$+
":"+RIGHT$(ID$,1) TO OP$+":"+RIGH
T$(OD$,1) ELSE COPY OP$
2190 '* IF YOU'RE USING ONLY ONE
DRIVE, PROMPT DISK SWITCH
2200 IF ID = OD AND PG <= 1 THEN
CLS:PRINT "INSERT SOURCE DISKET
TE AND PRESS 'ENTER'";:SOUN
D50,3:LINE INPUT NX$
2290 '* GET THE NEXT RECORD
2300 RETURN
2400 END
2490 '* IF YOU MESS UP, COCO TEL
LS YOU AND RESTARTS
2500 CLS 4:SOUND 100,1:SOUND 150
,1:SOUND 100,1:PRINT @ 232, "INV
ALID DRIVE!";:FOR X = 1 TO 1000:
NEXT:CLS:GOTO 200

```

```

2550 '* SHORTSTOP OVERFLOW INTO
THE NEXT ROUTINE
2560 '* IF IT GETS HERE IT DOESN
'T BELONG ... END!
2590 END
2595 '* CHECK FOR SINGLE DRIVE O
PERATION
2600 IF ID = OD THEN CLS:SOUND 5
0,1:PRINT "INSERT DESTINATION DI
SKETTE IN DRIVE";OD;:LINE INPUT
"AND PRESS 'ENTER'";Z$
2620 '* SEE IF FILE ALREADY EXIS
TS
2630 PRINT:PRINT "CHECKING DESTI
NATION DISKETTE":PRINT "FOR ";OP
$:PRINT "TO PREVENT <AE ERROR>"
2650 PP=0: FOR SS = 3 TO 11
2700 DSKI$ OD,17,SS,CK$(1),CK$(2
)
2800 FOR HH = 1 TO 2
2900 FOR EE = 1 TO 128 STEP 32
3000 PP = PP +1
3100 PP$(PP) = MID$(CK$(HH),EE,1
2)
3200 IF PP$(PP) = PG$(PG) THEN R
ETURN
3225 IF LEFT$(PP$(PP),1) = CHR$(
255) GOTO 3350
3250 PP$(PP) = ""
3300 NEXT EE,HH,SS
3330 '* IF YOU GET THIS FAR, THE
FILE MUST BE COPIED
3340 '* CHECK FOR SINGLE DISK OP
ERATION, THEN CALL THE COPY ROUT
INE
3350 IF ID = OD THEN CLS:SOUND 5
0,1:PRINT "INSERT SOURCE DISK IN
DRIVE";ID:LINE INPUT "AND PRESS
'ENTER'";Z$
3400 GOTO 2000
3450 '* SHORTSTOP RUNAWAY OPERAT
ION
3500 END
3550 '* CLEAR AS MUCH MEMORY AS
YOU CAN THROUGH BASIC
3600 PCLEAR 1:GOTO 100

```



# The Secret to Loading Those Double-Speed Tapes

By Craig Carmichael

**H**ave you got lots of long files to CLOAD and CSAVE? Or, have you accidentally taped your favorite game at double speed? Anyone who is familiar with the Color Computer, and whose POKE 65495,0 works, is probably also familiar with the double-speed CSAVE. These individuals also know that a simple POKE 65495,0, unfortunately, does not enable them to load the tapes back into the computer.

The problem is this: POKE 65495,0 takes the Color Computer from "slow" (normal) clock mode to "address-dependent" mode. In address-dependent mode, the computer runs at normal speed when the memory being accessed is RAM, from zero to 32767, and at double speed when accessing ROM, 32768 and up (BASIC, Extended BASIC). During a CSAVE, there aren't many calls to RAM, so the operation is nearly double speed. However, CLOAD makes considerable use of RAM, and thus does not operate at the same speed as the CSAVE. The exasperating thing is that double-speed tapes are perfectly good! All we need is a way to load them.

The first method I tried involved connecting my tape recorder's drive belt to a variable speed electric drill and running the tape recorder at my guess of half speed! As an emergency measure, it had its merits, working about one time in four with a steady hand, but this is not the method I shall detail here.

My next idea was to rewrite the cassette routines as machine language utility programs, which could be run as much as three times as fast as the regular ones, but luckily, other commitments and procrastination eliminated this idea.

Then I got a copy of *The Facts for the TRS-80 Color Computer* (a technical book by Spectral Associates, \$15.95)

and browsing through it, I noticed an overlooked detail. Memory Locations 8F, 90 and 91 Hex in RAM determine the width of pulses the computer accepts as a '0' or a '1' from the tape. So, here is the priceless secret: After you POKE 65495,0, POKE 143,15:POKE 144,20:POKE 145,7 and all those double-speed tapes will load perfectly!

Too many people buy disk drives because cassettes are slow to use, without realizing that the full speed of the cassette is not exploited. The cassette interface on my homemade computer runs at 4.5K Baud, three times the regular speed of the Color Computer's.

I usually program in machine language, and I have now incorporated the double-speed cassette functions as an automatic feature of my assembly editor and as an optional feature of my test editor for saving lengthy files. I have had no I/O Errors using double speed, except with a bad tape. There is, of course, less possibility of running over a bad section of tape since the programs save in a shorter time!

## Comments

1) To recap: When recording, simply use POKE 65495,0 "DOUBLE SPEED POKE", and when loading, use POKE 65495,0 with POKE 143,15:POKE 144,20:POKE145,7.

2) If your computer doesn't work in address-dependent mode, see Page 78 in the January 1983 RAINBOW for ideas on how to get it running.

3) To get back to regular speed loading without turning off the computer, POKE 65494,0:POKE 143,18:POKE 144,24:POKE 145,10.

4) If your tapes don't load flawlessly, you could try POKE 146,1 (or more) before you record to increase the length of leader tape sent at the start of each

block, since this is the most common trouble spot with any tape. Next, tape recorders that have a manual record level adjustment make better quality recordings than those with ALC record levels.

5) You could also try changing the POKE values at 143, 144 and 145. The values given are simply the first ones that gave me good results.

6) If all else fails, get a couple of adapters and hook up your stereo cassette deck. I confess to using an AKAI CS-34D at all times, which means I am only guessing about whether many people will have trouble with portable tape recorders at the higher speed.

## Rules

1) Use double speed only for your personal tapes. Even if others know how to load a tape at double speed, they won't be expecting to receive a tape in this format. The regular speed is standard.

2) Clearly indicate on the cassette label "FAST" when you have recorded a tape at double speed.

Due to speed differences between cassette recorders, the POKE values may have to be adjusted a bit if a program is CSAVEd on one unit and CLOADed with another model of cassette recorder.

There you have it! So simple, yet such a timesaver if you do a lot of CSAVEing and CLOADing, and a lifesaver if you've accidentally saved a tape at double speed.

(You may write to the author with any questions at 820 Dunsмур Road, Victoria, British Columbia, Canada, V9A 5B7. Please include an SASE.) ☺

Now you can use CoCo's assistance for  
compiling program documentation

# Cross-Reference Your Programs with XREF

By Douglas Van Dusen

**H**ave you ever had to go back to a program you wrote several months ago and couldn't find your way around the program? Well, you have fallen to the bane of all programmers: documentation! The worst part of writing a program is documenting it. *XREF* helps make it easier to do that documentation by making the CoCo do most of the work for you.

*XREF* will list your program and cross-reference it. It is easy to modify *XREF* since the line length and number of references per line have been placed in variables. This program also works for tape users as the device number has been placed in a variable, also.

Some of the rules for using this program are: 1) The program must be saved in ASCII; 2) In present form a PCLEAR 0 must be done for disk operation (see "Program Modifications" for more on this); 3) You must have a printer (any width will do); 4) Don't use the high speed POKes in the program with a software spooler program. It will mess up the printout; and 5) Be sure you have no machine language programs in memory — you need all the memory you can get.

Let's have a look at the program section by section:

Line	Description
1	Sets the printer Baud rate (9600 in my case).
2 - 3	Displays the status and statistics while the program operates (so you can tell it's doing something).
4 - 8	Sets up the parameters, finds out what options you have selected and enters the program line.
9 - 40	Breaks (parse) the input lines

## XREF Sample Run

"XREF" - February 26, 1986

PAGE 1

SYMBOL	REFERENCE LINE						
2	19						
4	99						
7	8	12	16	17	26	27	
12	15	16	18	21	26	29	41
	42						
22	14						
23	24						
30	12	16	17	29	25		
33	33						
35	33						
37	34						
38	31	34					
39	18	19					
41	14						
42	23						
43	7						
46	57						
59	55						
58	46						
62	44	47	54				
63	47	48					
64	8						
66	66						
67	67						
68	6	62	65				
69	68						
72	1						
76	78						
79	76						
82	86						
85	84						
87	83						
88	4	88					
BC	2	5	9	69			
BN	9	22	28	41			
BR	15						
C	22	23	31	32			

and finds the reserved words using the data in the RW\$ array. These come from the DATA statements.

- 41 - 58 Prints the cross-reference portion of the listing.
- 59 - 71 This is the page break portion of the program. It works no matter how wide the listing is to be created.
- 72 - 73 Clears the string space and dimensions the arrays that are necessary for program operation. This is where you can customize the program to suit your system (see "Program Modifications").
- 74 - 75 Displays the credits for the program (I request that these lines remain unchanged).
- 76 - 80 Reads the DATA statements and places the reserved words in the RW\$ array.
- 81 - 86 Takes the programs to be "XREFed" (10 MAX). The

program may or may not have the extension, however, if the extension is not BAS you must enter the extension (disk users don't use the drive number).

- 87 Used to input the date (it may also be used to put some comment of no more than 50 characters).
- 88 Enters the selection of what you want the program to do.
- 89 - 93 The loop that runs all of your programs you want to cross-reference.
- 94 - 96 The DATA statements that have the reserved words.

#### Program Modifications

Line 1: This may be changed to reflect the Baud rate used for your printer.

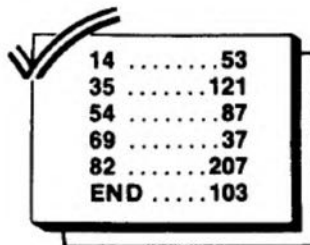
Line 72: The arrays RF and NX may be enlarged (for tape system or 40K

BASIC program use). These arrays must be the same size. The HI and LO variables are used for the high speed POKE; these may be deleted if your machine won't handle it. Be sure to remove all POKE HI and POKE LO references in the rest of the program.

Line 73: The DN variable is set to '1'. If it is set to -1 you will enter from tape. LW is the line width. Set it as you wish for your printer. ZR is the number of references per line. Use 6 for 80 cpl, 7 for 96 cpl and 11 for 132 cpl.

You can use the PCLEAR 0 POKE (POKE 25,6 for tape and POKE 25,14:POKE 3584,0:NEW for disk) to clear more memory for the program's use. The program needs a minimum of 500 bytes to operate.

(Any questions you have about XREF may be directed to the author at 2541-A Valencia Drive, Holloman AFB, NM 88330, phone 505-479-4035. Please enclose an SASE for a reply when writing.) □



14	.....	53
35	.....	121
54	.....	87
69	.....	37
82	.....	207
END	.....	103

The listing: XREF

```
1 POKE150,1:GOTO72
2 PRINT@0,STRING$(46,128);"xref"
;STRING$(46,128);:PRINT@129,"PRO
GRAM-ID: ";F$(F):PRINT@196,"LINE
NO: ";:PRINTUSING"#####";LN:PR
INT@257,"LINE COUNT: ";:PRINTUSI
NG"#####";LC-1:PRINT@321,"BYTE
COUNT: ";:PRINTUSING"#####";BC
3 PRINT@386,"REF COUNT: ";:PRINT
USING"#####";RC+1:PRINT@453,"ME
MORY: ";:PRINTUSING"#####";MEM:
RETURN
4 M=VAL(M$):IFM<0ORM>3THEN88
5 LC=0:BC=0:PZ=0:V$="":C$="":VC=
91:RC=-1:SZ=0
6 CLS:FORI=0TO91:VN(I)=-1:NEXT:G
OSUB68
7 POKELO,0:IFEOF(DN)THEN43
8 LINEINPUT#DN,L$:POKEHI,0:IFM>1
GOSUB64:IFM=2THEN7
9 LG=LEN(L$):BN=0:ER$="":LC=LC+1
:BC=BC+LG
10 LP=INSTR(L$," "):LN=VAL(LEFT$(
L$,LP)):GOSUB2
11 IFLN>32767THENLN=LN-65536
12 LP=LP+1:IFLP>LG GOSUB30:GOTO7
13 C$=MID$(L$,LP,1)
14 IFC$>="A"ANDC$<="Z"THEN22ELSE
IFC$>="0"ANDC$<="9"THEN41
```

```
15 IFC$=" "THEN12ELSEIFC$<>,"TH
ENBR=0
16 IFC$=CHR$(34)GOSUB30:LP=INSTR
(LP+1,L$,C$):IFLP>0THEN12ELSE7
17 IFC$=" "GOSUB30:GOTO7
18 IFC$="$"GOSUB39:GOTO12
19 IFC$="("GOSUB39
20 GOSUB30:IFC$<>,"THENER$="
21 GOTO12
22 C=ASC(C$):P=PT(C-65):BN=0
23 IFC<ASC(RW$(P))THEN42
24 IFINSTR(LP,L$,RW$(P))<>LP THE
NP=P+1:GOTO23
25 GOSUB30:RW$=RW$(P)
26 IFRW$="DATA"THENLP=INSTR(LP,L
$,"):IFLP>0THEN12ELSE7
27 IFRW$="REM"THEN7
28 IFRW$="GOTO"ORRW$="THEN"ORRW$
="ELSE"ORRW$="GOSUB"THENBN=1
29 LP=LP+LEN(RW$)-1:GOTO12
30 IFV$=" "THENRETURN
31 IFV$>="A"THENV$=V$+ER$:C=ASC(
V$)+1ELSEIFV$>="0"THENV$=RIGHT$(
" "+V$,5):C=VAL(LEFT$(V$,2))E
LSE38
32 IL=-1:I=C
33 IFV$>V$(I)THENIL=I:I=VN(I):IF
I>0THEN33ELSE35
34 IFV$=V$(I)THENJ=LS(I-91):IFRF
(J)=LN THEN38ELSERC=RC+1:NX(J)=R
C:GOTO37
35 VC=VC+1:IFIL>=0THENVN(IL)=VC
36 V$(VC)=V$:VN(VC)=I:RC=RC+1:FR
(VC-91)=RC:I=VC
37 RF(RC)=LN:NX(RC)=-1:LS(I-91)=
RC
38 V$="":RETURN
39 IFV$<>" "THENV$=V$+C$
```

```

40 RETURN
41 IFV$=""ANDBN=ØTHEN12
42 V$=V$+C$:GOTO12
43 IFM=2THENRETURN
44 PZ=Ø:GOSUB62
45 FORJ=ØTO91:V=J
46 V=VN(V):IFV<ØTHEN58
47 IFLZ>54GOSUB62ELSESZ=SZ+1:IFS
Z=3GOSUB63
48 IFLEFT$(V$(V),1)<>"ANDQQ=ØA
NDRZ<>3THENQQ=1:GOSUB63
49 RZ=Ø:I=FR(V-91):POKELO,Ø:PRIN
T#-2,V$(V):POKEHI,Ø
50 IFRZ=ØTHENPOKELO,Ø:PRINT#-2,T
AB(16):POKEHI,Ø
51 LN=RF(I):IFLN<ØTHENLN=LN+6553
6
52 POKELO,Ø:PRINT#-2,USING" #
###";LN,:POKEHI,Ø
53 RZ=RZ+1
54 IFRZ>ZR THENRZ=Ø:POKELO,Ø:PRI
NT#-2:POKEHI,Ø:LZ=LZ+1:IFLZ>74GO
SUB62
55 I=NX(I):IFI>ØTHEN5Ø
56 IFRZ>ØTHENPOKELO,Ø:PRINT#-2:P
OKEHI,Ø:LZ=LZ+1
57 GOTO46
58 NEXTJ
59 POKELO,Ø:PRINT#-2,STRING$(LW,
"=)
60 PRINT#-2,"LINE: ";LC-1;"
BYTE: ";BC;" SYMBOLS: ";VC-91;
" REFERENCES: ";RC+1
61 LZ=LZ+3:POKEHI,Ø:RETURN
62 GOSUB68:POKELO,Ø:PRINT#-2,"SY
MBOL";TAB(2Ø)"REFERENCE LINE":LZ
=LZ+1
63 POKELO,Ø:PRINT#-2,STRING$(LW,
"-"):LZ=LZ+1:SZ=Ø:POKEHI,Ø:RETUR
N
64 X=1
65 IFLZ>56ORRIGHT$(L$,3)=" ,PG"GO
SUB68
66 Y=INSTR(X,L$,CHR$(1Ø)):IFY>ØT
HENPOKELO,Ø:PRINT#-2,MID$(L$,X,Y
-X):LZ=LZ+1:POKEHI,Ø:X=Y+1:GOTO6
6
67 POKELO,Ø:PRINT#-2,MID$(L$,X,L
W):LZ=LZ+1:POKEHI,Ø:X=X+LW:IFX<L
EN(L$)THEN67ELSEReturn
68 POKELO,Ø:IFZZ=ØTHENZZ=1:GOTO6
9ELSEPRINT#-2,CHR$(12)
69 PZ=PZ+1:PRINT#-2:PRINT#-2,TAB
(LW-8)"PAGE "":PRINT#-2,USING"##
#";PZ
70 PRINT#-2,PR$:PRINT#-2
71 LZ=4:POKEHI,Ø:RETURN
72 CLEAR15ØØ:I=4ØØ:DIMVN(49Ø),V$
(49Ø),FR(4ØØ),LS(4ØØ),RF(13ØØ),N
X(13ØØ),RW$(12Ø),PT(25):HI=65495
:LO=65494
73 DN=1:LW=8Ø:ZR=6:CLS:PRINTSTRI
NG$(32,166);
74 PRINT" XREF COLOR BASIC VERS
ION 1.Ø":PRINTSTRING$(32,166):;P
RINT" (C)1984 WESTERN HORIZON"
:PRINTTAB(11)"SOFTWARE LTD.":PRI
NTSTRING$(32,166);
75 PRINT"LISTS ALL VARIABLES & RE
F LINE #":PRINTSTRING$(32,166):;
POKEHI,Ø:RW=Ø
76 READRW$:RW=RW+1:RW$(RW)=RW$:I
FRW$=""\THEN79
77 I=ASC(RW$)-ASC("A"):IFPT(I)=Ø
THENPT(I)=RW
78 GOTO76
79 FORI=ØTO25:IFPT(I)=ØTHENPT(I)
=RW
80 NEXT:POKELO,Ø
81 FX=Ø
82 PRINT"PROGRAM-ID"FX+1":":LIN
EINPUTL$
83 IFL$=""THENIFFX<1THENENDElse8
7
84 IF(DN=-1)THEN85ELSEIFINSTR(L$
,"/")=ØTHENL$=L$+"/BAS"
85 FX=FX+1:F$(FX)=L$
86 GOTO82
87 PRINT:POKE282,Ø:LINEINPUT"DAT
E = ";D$:POKE282,255:PRINT:PRINT
"1) XREF 2) LIST 3) BOTH "
88 M$=INKEY$:IFM$=""THEN88
89 FORF=1TOFX
90 POKELO,Ø:CLOSE:OPEN"I",#DN,F$
(F):PR$=CHR$(34)+" "+F$(F)+CHR$(3
4)+" - "+D$:POKEHI,Ø:GOSUB4
91 NEXTF
92 POKELO,Ø:PRINT#-2,CHR$(12)
93 END
94 DATAABS,AND,ASC,AS,ATN,AUDIO,
CIRCLE,CLS,CHR$,CLEAR,CLOSE,COLO
R,COS,CSAVE,CSAVEM,CLOAD,CLOADM,
CVN,DATA,DEF,FN,DLOAD,USR,DEL,DI
M,DSKI$,DSKO$,DRAW,ELSE,END,EOF,
EXP,EXEC,FREE,FIELD,FILES,FIX,FO
R,GET,GOSUB
95 DATAGOTO,HEX$,IF,INKEY$,INPUT
,INSTR,INT,JOYSTK,KILL,LEFT$,LEN
,LINE,LOAD,LOADM,LOC,LOF,LOG,LSE
T,MID$,MKN$,MEM,RENAME,NEW,NEXT,
NOT,ON,OPEN,OR,MOTOR,OFF,PEEK,PO
INT,PPOINT,POKE,POS,PUT,PRESET,P
SET,PAINT,PCLS,PCLEAR,PCOPY,PLAY
,PMODE,PRINT
96 DATAREAD,REM,RESET,RESTORE,RE
TURN,RIGHT$,RND,RSET,RUN,SAVE,SA
VEM,SGN,SIN,SQR,STEP,STOP,STR$,S
TRING$,SCREEN,SKIPF,SOUND,STOP,T
AB,TAN,THEN,TIMER,TO,UNLOAD,USIN
G,VAL,VARPTR,WRITE,VERIFY,"\"

```

*CLOAD and RUN all in one fell swoop!*

# AUTO-EXECUTING YOUR TAPE PROGRAMS

By Harold Nickel

**W**hile I have had my CoCo, I have come to appreciate its abilities. One I missed having, though, is the ability to load a program from tape and execute it all with one command. Without this, you can't "chain" programs (have one program execute another). Techniques have been written giving assembler programs the ability to auto-execute, but you are still stuck typing in CLOAD and RUN for your BASIC stock.

The following machine code provides this ability; it differs from the assembler techniques. With assembler auto-executes, the program loaded stores values into memory locations that cause it to begin executing. The machine language program runs itself. With *Crun*, the logic is external to the program. Like a BASIC command, executing the *Crun* code is done either manually or from a currently running program.

## Modifying CLOAD

Since much of the logic I wanted to use was already a part of the CLOAD command, I decided to use a variation of that logic for *Crun*. The first step was to determine how CLOAD worked.

I used the program *Memdump* (Listing 1) to print the machine code for CLOAD. *Memdump* prints selected areas of memory to either the screen or a printer. (I use a Microline 82A printer.) It prints memory in hexadecimal with 8 bytes per line for the screen display and 16 bytes per line for the printed output. Each line is preceded with the first byte's address. Printed output is double-spaced to allow room for nota-

tions. It also allows each dump to be titled for future reference.

After printing the CLOAD code, I interpreted it into assembler instructions using a 6809 assembly language book. I selected the portion of CLOAD that loads BASIC files and wrote it as a separate machine language routine.

To transform this routine into *Crun*, it had to run the newly loaded file. It does this by storing the values 'R', 'U' and 'N' in the keyboard buffer, then jumping to the command execution logic. This simulates the entry of the RUN command from the keyboard and causes the program to be executed.

The final version of *Crun* is presented in Listing 2. Not having an assembler, the code is shown in a three-column table rather than as an assembler program. The first column contains the actual machine code. The second column contains the assembler instruction associated with each line's function. The third column contains comments to help interpret the function being performed.

## Using Crun

I used the program in Listing 3 to install the *Crun* code. *Crun* takes 102 bytes of memory. The Variable A contains the address of the start of *Crun*. This value, therefore, must be less than or equal to the highest address in RAM minus 101. The CLEAR statement keeps the routine from being written over. Its address value should be less than or equal to the value used in Variable A.

Once installed, *Crun* can be called by using the EXEC command with the start address. Since the logic used is similar to that of CLOAD, it can also be used with a filename. Simply follow the EXEC

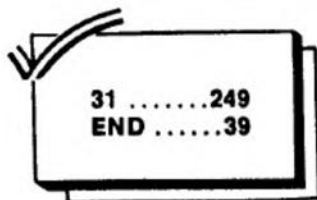
command and address with either the filename in quotes when entering it through the keyboard, or as a variable value if executed from a program.

One use I have found for *Crun* is to equip each of my program tapes with a directory program (Listing 4). I generally keep a few tapes as a program library containing a number of BASIC programs. The *TapeDir* program provides a list of all programs on a tape, and the ability to load and run them from a menu.

*TapeDir* first protects the highest 102 bytes of RAM (my CoCo has 32K) and pokes in *Crun*. It then displays the program names on the tape. You can select one from the menu by pressing its letter (or exit *TapeDir* with the SHIFT-CLEAR keys). The selected program will be loaded and run. I use *TapeDir* itself as one of the selections. Then, if the program I want is not on the first tape, I can insert a new tape and select *TapeDir* to display its menu.

Since I add programs to my tapes periodically, I wrote *TapeDir* so that adding new program names would not increase its length. The new name is added as one of the T\$ values. Names with less than eight characters are padded with blanks. The new directory can then be saved over the old one without writing into the next file on tape.

An additional technique I use is to place a tape header file on each tape. It consists of one comment line and is always the first file on a tape (before *TapeDir*). This lets me position the tape exactly at the beginning of *TapeDir*, using SKIPF, when I want to save a new menu. □



**Listing 1: MEMDUMP**

```
Ø 'FORMATTED HEX MEMORY DUMP
1 CLS
2 M$=""
1Ø INPUT"ENTER TITLE: ";T$
11 IF T$="Q" THEN END
12 INPUT"ENTER START (HEX): ";S$
13 INPUT"ENTER END ADDRESS: ";E$
14 PRINT
2Ø INPUT"(S)CREEN OR (P)RINT): "
;O$
21 IF O$="S" THEN O=Ø:PRINT:GOTO
25
22 IF O$="P" THEN O=-2:PRINT:GOT
O 26
23 PRINT:PRINT"ENTER S OR P":GOT
O 2Ø
25 CLS:S=8:GOTO 3Ø
26 PRINT:PRINT"READY PRINTER.":P
RINT"PUSH ENTER WHEN READY."
27 S=16:M$=""
28 INPUT O$
```

```
29 IF O$="Q" THEN GOTO 7Ø
3Ø PRINT#O,M$;T$:PRINT#O," ":L=2
31 FOR A=VAL("&H"+S$) TO VAL("&H
"+E$) STEP S
32 PRINT#O,M$;
33 IF LEN(HEX$(A))<4 THEN FOR P=
LEN(HEX$(A))+1 TO 4:PRINT#O,"Ø";
:NEXT P
35 PRINT#O,HEX$(A);" : ";
4Ø FOR SA=Ø TO S-1
41 PRINT#O," ";
42 IF LEN(HEX$(PEEK(A+SA)))<2 TH
EN PRINT#O,"Ø";
45 PRINT#O,HEX$(PEEK(A+SA));
5Ø NEXT SA
55 PRINT#O,""
56 IF O=-2 THEN PRINT#O," ":GOTO
65
6Ø L=L+1
61 IF L<15 THEN GOTO 65
62 T$=INKEY$:IF T$="" THEN GOTO
62
63 L=Ø
65 NEXT A
7Ø PRINT
71 INPUT"MORE ?";O$
72 IF O$="YES" THEN GOTO 1
73 IF O$="Y" THEN GOTO 1
8Ø CLS
81 END
```

## Communications Specialists

**AUTO ANSWER \$399.00**

### INFO CENTRE

THE FIRST BULLETIN BOARD SYSTEM  
for Tandy's computers  
(02) 344 9511

### SPECIAL!

Avtek Mini Modem + Cable + CoCo  
Tex Program - the total Viatel System -  
\$279.00

We also have the largest range of Software for  
OS-9 and Flex operating systems.

### PARIS RADIO ELECTRONICS

161 Bunnerong Rd., Kingsford, N.S.W. 2032.  
(02) 344 9111

## MK 1 SERIAL/PARALLEL PRINTER INTERFACE

CONNECT CO-CO I or II to a PARALLEL PRINTER  
Revised MK1 PRINTER INTERFACE

**Features:**

- \* EXTRA SERIAL PORT for MODEM, no more plugging/unplugging cables.
- \* Compatible with Standard Centronics Parallel Printers eg: EPSON, GEMINI, BMC, CP80, TANDY ETC.
- \* Plugs into CO-CO or CO-CO II Serial Port and includes all cables and connectors.
- \* SIX Switch selectable Baud rates 300, 600, 1200, 2400, 4800, 9600
- \* Power Pack is required for Printers not supplying power at pin 18 on the Parallel Connector eg: EPSON, BMC, CP80.
- \* Increases Printing Speed by up to 30% on TANDY DMP100/200 Printers

**NOW ONLY - \$89.95 (including postage)**  
Add \$9 for Power Pack if required

AVAILABLE FROM: G. & G. PIALA  
P.O. BOX 46  
THORNLEIGH, NSW. 2120  
phone: (02)-84-3172

Listing 2: CRUN

Machine Code	Assmblr Instr.	Description
0F 78	CLR	Flag CLOSE
32 62	LEAS	Clear Stack
BD A5 C5	JSR	Evaluate Filename
BD A6 48	JSR	Locate the File
7D 01 E4	TST	Check If Binary
26 05	BNE	Jump If Not Binary
B6 01 E2	LDA	Check If Basic
27 03	BEQ	Jump If Basic
7E A6 16	JMP	Jump to FM Error
BD AD 19	JSR	Do NEW
BD A7 7C	JSR	Read File Leader
9E 19	LDX	X=Program-Area Ptr
9F 7E	STX	Cassette-Buffer Ptr=X
DC 7E	LDD	D=Cassette-Buffer Ptr
4C	INCA	Bump Cassette-Buffer Ptr MSB
BD AC 37	JSR	Do Memory Check
BD A7 0B	JSR	Read a Block
26 34	BNE	Jump If I/O Error
96 7C	LDA	A=Block Type
27 30	BEQ	Jump If Header Block
2A ED	BPL	Loop If Data Block
9F 1B	STX	Save Buffer Ptr as End of Prog.
BD A7 E9	JSR	Turn Off Cassette
8E AB EC	LDX	X=O.K. Message
BD B9 9C	JSR	Display O.K. Message
BD AD 21	JSR	Reset Basic Memory Ptrs
BD AC EF	JSR	Reset Basic Line Ptrs
BD 01 82	JSR	Call Extended Basic Link
8E 02 DD	LDX	X=Start of Input Buffer
86 52	LDA	A='R'
A7 80	STA	Save 'R' in Buffer
86 55	LDA	A='U'
A7 80	STA	Save 'U' in Buffer
86 4E	LDA	A='N'
A7 80	STA	Save 'N' in Buffer
6F 84	CLR	Flag End of Input
C6 04	LDB	B=Length of Input
8E 02 DC	LDX	X=Start of Input minus 1
4F	CLRA	Signal No Break Key
7E AC 7F	JMP	Jump to Command Mode
BD AD 19	JSR	Do NEW
7E A6 19	JMP	Jump to Display I/O Error

Listing 3: INSTALL

```

0 'INSTALL MACHINE LANGUAGE CODE
10 CLS
20 CLEAR 200,30000
30 A=30000:'START ADDRESS
40 L=102:'NUMBER OF DATA VALUES
50 FOR X=A TO (A+L)-1
60 READ HS
70 POKE X,VAL("&H"+H$)
80 NEXT X
90 PRINT"CODE INSERTED AT";A
100 END
110 DATA 0F,78,32,62,BD,A5,C5,BD
,A6,48,7D,01,E4,26,05,B6,01,E2,2
7,03,7E,A6,16
120 DATA BD,AD,19,BD,A7,7C,9E,19
,9F,7E,DC,7E,4C,BD,AC,37,BD,A7,0
B,26,34,96,7C,27,30,2A,ED
130 DATA 9F,1B,BD,A7,E9,8E,AB,EC
,BD,B9,9C,BD,AD,21,BD,AC,EF
140 DATA BD,01,82,8E,02,DD,86,52
,A7,80,86,55,A7,80,86,4E,A7,80,6
F,84,C6,04,8E,02,DC,4F,7E,AC,7F
150 DATA BD,AD,19,7E,A6,19
    
```

29 .....228  
58 .....122  
END .....179

Listing 4: TAPEDIR

```

0 'TAPE DIRECTORY WITH CRUN
1 CLS
2 CLEAR 200,32666:A=32666
3 DIM T$(24)
10 'INSTALL CRUN
11 DATA 0F,78,32,62,BD,A5,C5,BD,
A6,48,7D,01,E4,26,05,B6,01,E2,27
,03,7E,A6,16,BD,AD,19,BD,A7,7C,9
E,19,9F,7E,DC,7E,4C,BD,AC,37,BD,
A7,0B,26,34,96,7C,27,30,2A,ED
12 DATA 9F,1B,BD,A7,E9,8E,AB,EC,
BD,B9,9C,BD,AD,21,BD,AC,EF,BD,01
,82,8E,02,DD,86,52,A7,80,86,55,A
7,80,86,4E,A7,80,6F,84,C6,04,8E,
02,DC,4F,7E,AC,7F
13 DATA BD,AD,19,7E,A6,19
14 FOR P=A TO A+101
15 READ D$:POKE P,VAL("&H"+D$)
16 NEXT P
20 'INITIALIZE DIRECTORY TABLE
21 T$(1)="DIR"
22 T$(2)="MEMDUMP"
23 T$(3)="INSTALL"
24 T$(4)="CRUN"
25 T$(5)=" "
26 T$(6)=" "
27 T$(7)=" "
28 T$(8)=" "
29 T$(9)=" "
30 T$(10)=" "
31 T$(11)=" "
32 T$(12)=" "
33 T$(13)=" "
34 T$(14)=" "
35 T$(15)=" "
36 T$(16)=" "
37 T$(17)=" "
38 T$(18)=" "
39 T$(19)=" "
40 T$(20)=" "
41 T$(21)=" "
42 T$(22)=" "
43 T$(23)=" "
44 T$(24)=" "
50 'DISPLAY TAPE DIRECTORY
51 PRINT" TAPE DIRECTORY"
":PRINT
52 PRINT" A - ";T$(1);" M -
";T$(13)
53 PRINT" B - ";T$(2);" N -
";T$(14)
54 PRINT" C - ";T$(3);" O -
";T$(15)
55 PRINT" D - ";T$(4);" P -
";T$(16)
56 PRINT" E - ";T$(5);" Q -
";T$(17)
57 PRINT" F - ";T$(6);" R -
";T$(18)
58 PRINT" G - ";T$(7);" S -
";T$(19)
59 PRINT" H - ";T$(8);" T -
";T$(20)
60 PRINT" I - ";T$(9);" U -
";T$(21)
61 PRINT" J - ";T$(10);" V -
";T$(22)
62 PRINT" K - ";T$(11);" W -
";T$(23)
63 PRINT" L - ";T$(12);" X -
";T$(24)
64 PRINT:PRINT" USE (shift)(cl
ear) TO EXIT";
65 SCREEN 0,1
70 'PROGRAM SELECTION
71 D$=INKEY$:IF D$="" THEN GOTO
71
72 IF ASC(D$)=92 THEN END
73 IF ASC(D$)<65 OR ASC(D$)>88 T
HEN GOTO 71
74 IF T$(ASC(D$)-64)=" "
THEN SOUND 1,3:GOTO 71
75 SCREEN 0,0:EXEC 32664 T$(ASC(
D$)-64)
    
```

# CoCo MERGE

by John Nicoletto

One of the most significant deficiencies of the Color Computer is the lack of a cassette MERGE command. I find this particularly frustrating because I prefer to design programs in a modular fashion. By this I mean that I divide the application program into sub-tasks. Each sub-task is performed by a program module. Each module is written, debugged, and tested separately. The modules are then assembled into the final application program. This approach leads to the development of a library of standard modules since modules tend to be useful in different places and in other programs. Much of the usefulness is lost, however, if modules must be typed each time they are to be used.

Thus, COCO MERGE was written to preserve my programming style and to overcome what I believe to be a major deficiency of the Color Computer. It will append one BASIC program to the end of another at the touch of the command keys. COCO MERGE even has its own prompts. However, a word of caution is needed. COCO MERGE is habit forming. Once you have used COCO MERGE you'll wonder how you ever did without it.

## THEORY OF OPERATION.

BASIC programs can reside almost anywhere in the Color Computer's memory. The PCLEAR command (available with Extended BASIC) provides the mechanism for relocating BASIC programs. It is used primarily to reserve memory for graphics. The Color Computer does a PCLEAR 4 when first turned on. This reserves 4 graphics pages (1536 Bytes per page) in lower RAM memory. BASIC programs will start loading at memory location Dec. 7681. A PCLEAR 1 will start loading BASIC at location Dec. 3073 while a PCLEAR 8 will start BASIC at Dec. 13825. With all this moving around the Color Computer must have some way to keep track of the BASIC program. Memory locations Dec. 25 - 28 provide this service. Locations 25 and 26 are the start of BASIC pointer while 27 and 28 are the end of BASIC pointer. As their names imply, these memory locations point to the beginning and end addresses of the BASIC program.

Each line of the BASIC program, when translated to machine language by the Color Computer, contains the address of the next line. The Color Computer will append a two Byte flag at the end of the BASIC program. Thus, the lines of the BASIC program are linked together from start to finish. All the Color Computer needs is the location of the first line. From that point on it daisy chains its way through the program executing each line along the way. When it gets to the end of BASIC flag the Color Computer will stop executing and return to the command input mode.

COCO MERGE makes use of this linking of BASIC

program lines to merge programs. Once the first program has been completed and entered into memory; the SHIFT and CLEAR keys are pressed. The first time this is done the start of BASIC pointer (Dec. 25 & 26) is stored in location STR. The end of BASIC address is then decremented by two and stored in locations 25 and 26. This will cause the second program to overwrite the end of BASIC flag and will link the first line of the second program to the last line of the first program. Now the second program may be entered. This program may be manipulated freely, without fear of affecting the first, since the Color Computer is no longer aware of the first program.

When the second program has been completed the command keys are once again pressed. This time COCO MERGE places the original start of BASIC address, which was stored in STR, back into locations 25 and 26. The Color Computer will now run both programs as though they were one.

## PROGRAM DESCRIPTION

Program listing 1. contains the machine language source code for COCO MERGE. Lines 110-140 will place the address of COCO MERGE into the Color Computer's Jump Table and initialize the mode switch. This alters the subroutine which outputs a character to the screen. Thus, COCO MERGE will be executed just before the computer PRINTs a character on the screen.

Lines 150-160 reserve memory to store the start of BASIC pointer and the state of the mode switch. Lines 170-200 store the two prompts.

COCO MERGE starts with line 210. Lines 210-220 check each character looking for the command key. If the command key was not pressed then the program returns to the Color Computer's character output routine. When the command key is recognized then lines 230-240 are executed. These lines check the setting of the mode switch. If the switch is set to merge (zero) then lines 250-340 are executed. If the switch is set to restore (256) then lines 350-410 are executed.

The merge sequence consists of getting the initial start of BASIC address and storing it in STR. Then, decrementing the end of BASIC address by two and storing it in the start of BASIC pointer locations. Line 300 selects the appropriate prompt (MERGE INITIATED) address for the screen print subroutine. Line 310 toggles the switch and line 320 calls the COCO MERGE screen print subroutine (lines 430-480). Lines 330 and 340 return to BASIC through the Color Computer's normal prompt subroutine.

The restore sequence retrieves the initial address, from STR, and returns it to the start of BASIC pointer locations. Line 370 select the MERGE COMPLETED prompt address and line 380 toggles the switch. Line 390 calls the screen print subroutine. Lines 400 and 410 return control to BASIC through the prompt subroutine.

The screen print subroutine (lines 430-480) will display the appropriate prompt. Line 430 establishes a counter for the two 16 character prompts (including the



carriage return). Line 450 is the Color Computer's normal screen print subroutine jump address.

#### OPERATING INSTRUCTIONS

Program listing 2. contains the BASIC language driver for COCO MERGE. The program is very friendly and will instruct you on its use. In addition, it will automatically load the machine language program in the highest location of available RAM. This is true independent of the size of memory.

The program contains a simple arithmetic check (lines 200 and 220) on the DATA statements. This check will assure that the program is typed and entered correctly. If there are no errors then the program will EXEC COCO MERGE and protect it from BASIC.

To use COCO MERGE simply type or CLOAD the first BASIC program. When you have finished note the last line number of this program. The SHIFT and CLEAR keys are the command keys. To execute COCO MERGE simply press the SHIFT and then the CLEAR key (simultaneously). Your Color Computer will respond with the prompt MERGE INITIATED. You will no longer be able to RUN or LIST the program. The Color Computer has essentially forgotten this program. But do not fear, COCO MERGE is in control.

You may now enter the next program. You may type or CLOAD the program into memory. This program should start with a program line which is greater than the last line of the first program. You may RUN, EDIT, RENUM, or do just about anything else to this second program without affecting the first. The only exception is the PCLEAR command. If you are working with graphics programs then one additional word of caution is in order. Since the first program is in lower memory it could be overwritten by RUNNING a graphics program.

Once you have completed the second program; press the SHIFT and CLEAR keys once again. This time your Color Computer will respond with the prompt MERGE COMPLETED. Now if you execute a RUN or LIST command the Color Computer will execute or display the merged programs.

While merging cassette programs is very easy with COCO MERGE the following precautions should be observed:

Make sure that your merged program does not contain duplicate line numbers. A program which contains two lines with the same number will not execute properly.

The second program should start with a line number greater than the last line number of the first program. If you didn't observe this precaution, and you have Enhanced BASIC, all is not lost. You may be able to save your merged program by executing the RENUM command. The Color Computer will automatically renumber the program lines. However, transfer statements such as IF THEN, GO TO, GO SUB, ON...GOTO, or ON...GOSUB must be checked.

Double check the application of variables common to both programs. One program may accidentally alter variables used by the other program.

Check the merged program's logic flow. Make sure that the new (merged) program performs the functions which you intended and in the order that you intended.

Check the order of DATA statements to assure that the data is being READ in the proper sequence. Remember that DATA statements are arranged in order of use. The RESTORE command can also cause problems in a merged program.

Finally, check all transfers between the two original programs. Make sure that you know where each transfer is and under what conditions it will be executed.

```

10 ' *****
20 ' *      CO CO MERGE      *
30 ' *          BY          *
40 ' *  JOHN L. NICOLETTOS  *
50 ' *    APRIL 1, 1983    *
60 ' * ALL RIGHTS RESERVED *
70 ' *****
80 CLS
90 PRINT"      CO CO MERGE":PRIN
TQ32,STRING$(32,131);
100 PRINT"  THIS PROGRAM WILL LOAD,
":PRINT"EXECUTE, AND PROTECT A MACHI
NE"
110 PRINT"LANGUAGE UTILITY WHICH WIL
L":PRINT"ALLOW YOU TO MERGE TWO OR M
ORE"
120 PRINT"CASSETTE PROGRAMS. TO USE
":PRINT"SIMPLY TYPE OR CLOAD YOUR FI
RST"
130 PRINT"PROGRAM. BEFORE ENTERING
THE":PRINT"NEXT PROGRAM PRESS THE [S
HIFT]"
140 PRINT"AND CLEAR KEYS. TYPE OR C
LOAD":PRINT"THE NEXT PROGRAM. WHEN
DONE"
150 PRINT"PRESS THE [SHIFT] AND CLEA
R KEYS";:PRINT"ONCE AGAIN TO COMPLET
E MERGE."
160 PRINTQ484,"PRESS ENTER TO CONTIN
UE";:LINEINPUT Z2#
170 ED=PEEK(39)*256+PEEK(40)
180 ST=ED- 118
190 FOR X=ST TO ED
200 READ D:POKE X,D:SUM=SUM+D
210 NEXT X
220 IF SUM (<) 12939THEN CLS:PRINTQ26
3,"!!!DATA ERROR!!!":END
230 EXEC ST
240 CLEAR 200,ST
250 DATA 48, 141, 0, 43, 191, 1, 104
, 111
260 DATA 141, 0, 3, 57, 0, 255, 16,
77
270 DATA 69, 82, 71, 69, 32, 73, 78,
73
280 DATA 84, 73, 65, 84, 69, 68, 13,
77
290 DATA 69, 82, 71, 69, 32, 67, 79,
77
300 DATA 80, 76, 69, 84, 69, 68, 13,
129
310 DATA 92, 38, 54, 109, 141, 255,
215, 38
320 DATA 27, 158, 25, 175, 141, 255,
205, 158
330 DATA 27, 48, 30, 159, 25, 49, 14
0, 199
340 DATA 99, 141, 255, 194, 141, 30,
142, 171
350 DATA 239, 126, 172, 121, 174, 14
1, 255, 180
360 DATA 159, 25, 49, 140, 194, 111,
141, 255
370 DATA 173, 141, 9, 142, 171, 239,
126, 172
380 DATA 121, 126, 130, 115, 198, 16
, 166, 160
390 DATA 189, 162, 130, 90, 38, 248,
57
400 CLS
410 PRINTQ128,"      CO"
420 PRINT:PRINT"      CO"
430 PRINT:PRINT"      MERGE"
440 PRINT:PRINT"      LOADE
D"
450 END

```

continued on Page 55

# Tandy ELECTRONICS

**EXCLUSIVE OFFER TO  
RAINBOW/COCO READERS!**

## Computer Cassette — Leaderless and Certified

**20% Off 3<sup>95</sup>**  
Reg. 4.95

- Certified high quality ensures there is no data loss.
- Leaderless for instant recording without loss of initial data.
- Comes with protective case.

26-301



**Color Print**

**Save \$7** Reg. 14.95  
**7<sup>95</sup>**



**Color Screen Print Utility.** Provides multi-color printouts of color graphics screens produced by your graphics program. For use with color ink-jet printer or any dot matrix printer with bit-image capabilities for black and white printouts. 26-3121

**Bedlam**

**Save \$10**

Reg. 24.95

**14<sup>95</sup>**



You're a patient in an insane asylum! The maze unfolds as you try to find an escape route. Can you trust Napoleon? Or the guy who calls himself X-Ray? The way out of the asylum changes every time. It will drive you mad! 26-3312



**Reactoids**  
**\$15 Off!**

Reg. 39.95

**24<sup>95</sup>**

The code is red! There is meltdown at the fusion reactor! Test your speed and co-ordination as you take control of the reactoid and race against time to contain the atoms. Can you keep it up long enough to stabilise the atoms. Requires joystick. 26-3092



**Shamus**  
**Save \$20**

Reg. 49.95

**29<sup>95</sup>**

Take advantage of great value and help Shamus conquer the evil Shadow in his maze of danger! Retrieve coloured keys from the chambers of Shadows Lair, but beware of his terrifying guardians. Has three levels of difficulty. 26-3289



**Color LOGO**  
**Save \$20**

Reg. 69.95

**49<sup>95</sup>**

**Color LOGO Programming Language.** Students grasp graphic relationships and develop problem solving skills through programming and controlling turtles on the screen. As a pattern is drawn, it is stored for future reference. 16K program pack. 26-2722

```
00100 START LDA #40F9 L3
00110 ASCII CHAR LDH #4500 0E
00120 VIDEO MEM STA J64 PU
00130 CHAR ON SCREEN
00140 CHPI #40FF 0E
00150 E IF END VIDEO MEM
00160 ONE SCREEN 0E
00170 ARCH IF NOT
00180 DONE SWI
00190 END
```

**Programming**  
**\$20 Off!**

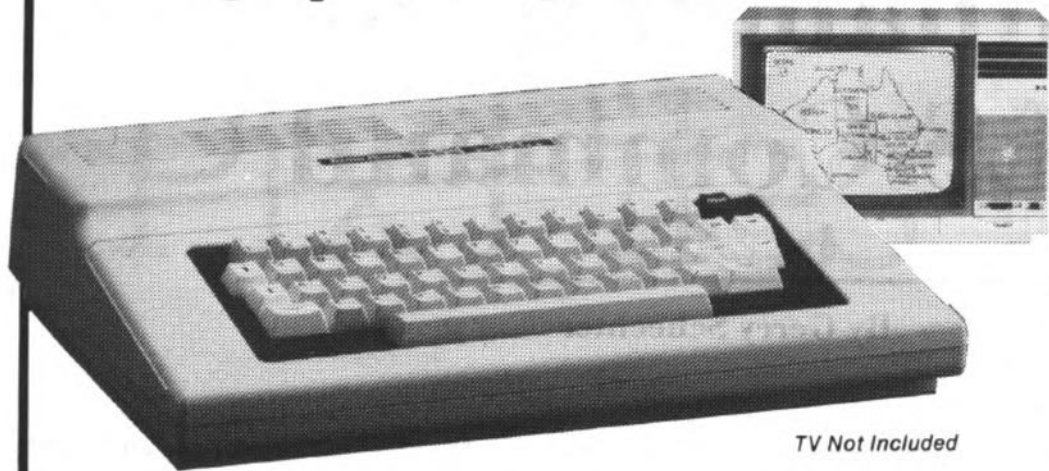
Reg. 79.95

**59<sup>95</sup>**

**Color Editor/Assembler.** Develop 6809 software programs or subroutines. Comes with trial assembly in memory so that you can run your program before final assembly to tape. Features an editor to change your program and Z-Bug for testing. 26-3250

**Sale Ends: March 31st, 1986**

# 23% OFF! A TRUE FAMILY COMPUTER



**16K Extended  
Basic Color  
Computer 2**

**Save \$70**

Reg 299.95

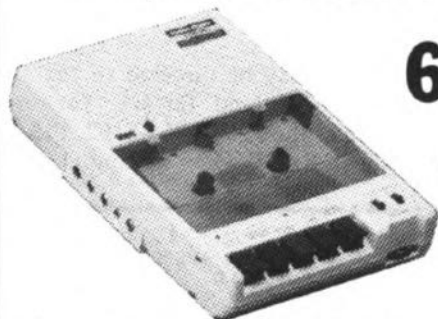
**229<sup>95</sup>**

*TV Not Included*

Tandy offers you an easily affordable introduction to the exciting world of computers with this very versatile home system! Plug it into your TV and begin by creating eight color graphics with sound using the built-in BASIC language. Simple one-line commands make it easy. Next write your own programs, or snap in an instant-loading Program Pak™ cartridge and enjoy hours of family fun playing games from Tandy's huge range of entertainment and educational software. The Color Computer 2 is ideal for people of all ages — young

children can use it to improve their maths, spelling, reading and writing skills. Students can store homework data in it with an optional cassette recorder, and tap into information databases using an optional modem. Parents can work out finances, plan budgets and type correspondence with an optional printer. The Color Computer 2 has 16K of memory, which is plenty for all these purposes, but it easily expands if you require more. Add an optional disk drive and you've got a complete personal computer system. 26-3136

## Computer Cassette Recorder

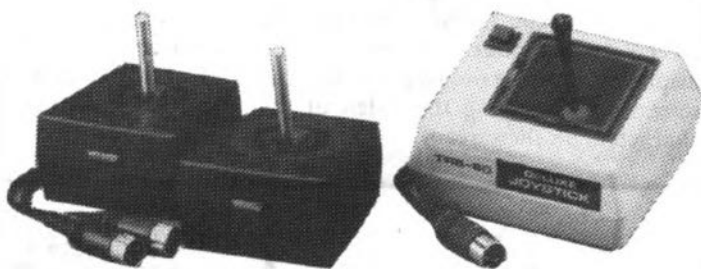


**69<sup>95</sup>**

- For Use With Tandy Computers
- Switchable Pause

Specially designed features for loading, saving and reviewing! Has on/off sound monitor, remote off, tape counter, LEDs for load/review/cue and battery, plus auto-level record. Jacks: earphone, remote, mike, aux. With case and connecting cable. 26-1209 Batteries not included.

## Color Computer Joysticks



**Joystick.** Precise control makes games and graphics come alive! 360° movement. 26-3008 ..... **29.95**  
**Deluxe Joystick.** Feel the difference! Patented for instant response! Dual axis trim controls for fine-tuning joystick to software. 26-3012 ..... **39.95**

# Tandy ELECTRONICS

A DIVISION OF TANDY  
AUSTRALIA LIMITED  
INC. IN N.S.W.

Nearly  
350 Stores  
Australia-  
Wide

WE SERVICE WHAT WE SELL!

Available from  
**350 Stores Australiawide**  
including Tandy Computer Centres

*\*Independent Tandy Dealers may not be participating  
in this ad or have every item advertised.  
Prices may also vary at individual Dealer Stores*

# Enhancing the CLS Command

By Gerry Schechter

With the possible exception of PRINT, the CLS command is probably the most often-used command when it comes to writing text-based BASIC programs. The CLS command has nine variations that correspond to the available colors in the text mode. I concluded these nine variations were not enough and decided to do something about it.

*SuperCLS* is a machine language subroutine that enhances the CLS command. Aside from the standard options accessible with the normal CLS command, several other options are also available. These include clearing the screen to any character that can be displayed, clearing only a portion of the screen and inverting the video of the characters on the screen.

The short demonstration program accompanying this article should serve to illustrate most of these features. However, some additional explanation is in order. The machine language subroutine is completely relocatable, so it can be placed anywhere in memory that won't be clobbered by BASIC. The subroutine uses BASIC's current cursor position in order to determine the starting point for the *SuperCLS* operation. This is controlled from your BASIC program by using the PRINT and PRINT@ statements. Therefore, the subroutine starts its operation from wherever BASIC normally prints its next character.

Control is passed to the subroutine via a USR call. The value in the parentheses is the value that is used for the

*SuperCLS* operation. This can be any value ranging from zero to 255. These are the same values you normally use in a PRINT CHR\$( ) statement. The only exception to this is the value of 32, which inverts the video on the screen instead of clearing it.

If it is still unclear as to how the subroutine works, take a few minutes to type in and run the demonstration program. As someone once said, "A picture is worth a thousand words." Have fun, and feel free to use this subroutine in your next program.

(Any questions you have regarding *SuperCLS* may be directed to the author at 75 Midland Terrace, Yonkers, NY 10704, phone 914-965-8102. Please include an SASE when writing.) □

310 .....204  
570 .....152  
END ....1000

## The listing: SUPERCLS

```

1 '=>SUPERCLS V1.0<=
2 ' GERRY SCHECHTER
3 '75 MIDLAND TERRACE
4 'YONKERS, NY 10704
5 ' FEBRUARY 1984
6 '*****
100 CLS
110 PRINT" ** SUPERCLS
120 S **"
120 GOTO 420
130 X = USR(169):GOSUB 510
140 X = USR(32):GOSUB 510
150 PRINT@64,"SUPERCLS"
160 X = USR(179):GOSUB 510
170 X = USR(32):GOSUB 510
180 PRINT@128," SUPERCLS"
190 X = USR(154):GOSUB 510
200 X = USR(32):GOSUB 510
210 PRINT@192," SUPERCLS"
220 X = USR(236):GOSUB 510
230 X = USR(32):GOSUB 510
240 PRINT@256," SUPERCLS"
250 X = USR(42):GOSUB 510
260 X = USR(32):GOSUB 510
270 PRINT@320,"";
280 FOR Z = 1 TO 255
290 X = USR(Z)
300 SOUND Z,1
310 NEXT Z
320 PRINT@0,"";
330 X = USR(32)
340 FOR Z = 32 TO 480 STEP 32
350 PRINT@Z,"";
360 X = USR(32)
370 PLAY"T255L255O1V31;1V<1V<1"
380 GOSUB 520
390 NEXT Z
400 PRINT@448,"";:END
410 'PROTECT MEMORY AND
420 'DEFINE USER CALL
420 IF PEEK(116) = 127
430 THEN CLEAR 200,32735
440 ELSE CLEAR 200,16351
450 IF PEEK(116) = 127
460 THEN ML = 32736
470 ELSE ML = 16352
480 DEF USR0 = ML
490 'POKE ML PROGRAM INTO MEMORY
500 FOR X = ML TO ML+29
510 READ X$
520 POKE X,VAL("&H"+X$)
530 NEXT X
540 GOTO 130
550 SOUND 255,2
560 FOR X = 1 TO 500
570 NEXT X
580 RETURN
590 'DATA FOR ML SUBROUTINE
600 DATA BD,B3,ED
610 DATA 9E,88
620 DATA C1,20
630 DATA 26,0D
640 DATA A6,84
650 DATA 88,40
660 DATA A7,80
670 DATA 8C,05,FF
680 DATA 23,F5
690 DATA 20,07
700 DATA E7,80
710 DATA 8C,05,FF
720 DATA 23,F9
730 DATA 39
740 'SOURCE FOR ML SUBROUTINE
750 ' ORG $7FE0
760 ' EQU $88
770 ' JSR $B3ED
780 ' LDX CURSOR
790 ' CMPB #32
800 ' BNE LOOP2
810 ' LDA ,X
820 ' EORA #$40
830 ' STA ,X+
840 ' CMPX #$5FF
850 ' BLS LOOP1
860 ' BRA RETURN
870 ' LOOP2 STB ,X+
880 ' CMPX #$5FF
890 ' BLS LOOP2
900 ' RTS
910 ' END START

```

Getting picture formats together

# Pix Files

By Joseph Kohn

Supposing we consider the standard format PMODE4 picture to be the infant of high resolution CoCo graphics, then *Graphicom* (by Cheshire Cat) is probably the teenager, and surely *CoCo Max* (by Colorware) is the young adult. As is typical of these "generation gaps," they have difficulty communicating with one another. Although all three use PMODE4, their picture file formats are sufficiently different, so moving pictures between them requires some thought or even special transfer routines.

The program listed here, *PixFiles*, provides a convenient means for inter-format picture file transfer. The picture formats include:

1) Standard PMODE4 Picture — This is the normal Extended BASIC format with the picture LOADMed and SAVEMed between RAM locations \$E00 and \$25FF.

2) *CoCo Max* Picture — This format is similar to Standard, except two screens, all eight graphics pages, are LOADMed and SAVEMed between RAM locations \$E00 and \$3DFF. *CoCo Max* files always have the extension MAX. It should be noted that the single screen file produced by *CoCo Max*, SCREEN/BIN, is a Standard format picture.

3) *Graphicom* Picture — The *Graphicom* file format is completely unique. It stores 24 pictures plus the *Graphicom* directory and working font on sequential sectors and tracks, skipping over the standard disk directory, Track 17. Pictures are loaded and saved from specific areas of the disk by selection

from the *Graphicom* illustrated directory.

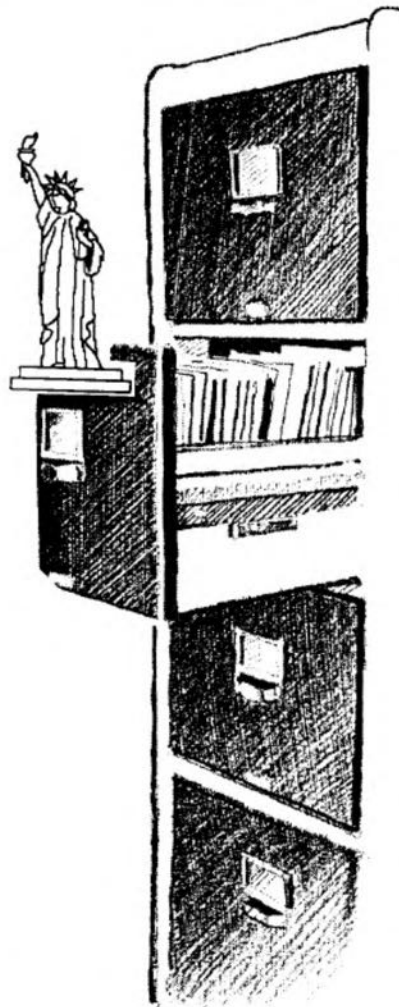
*PixFiles* is completely self-prompting and menu driven. Most operations are performed using the right joystick. Pictures can be loaded and saved to any disk drive. If you have more than two, change DX in Line 740 to the number of drives you intend to use.

Several error traps are built-in:

- *Graphicom* disks are checked for proper format.
- Before SAVEM for Standard and *CoCo Max* pictures, the disk is checked for sufficient storage space. This also prevents a SAVEM to a *Graphicom* disk.
- A check is made for Standard files to ensure that they are 6,144 bytes long.
- Filenames cannot exceed the maximum number of characters.
- A *Graphicom* LOAD/SAVE can be aborted by selecting a blank screen at the top of the illustrated directory.
- *CoCo Max* files must have the extension MAX.

The principle behind *PixFiles* is to first retrieve the picture you want to transfer from its source disk and place it in the Standard picture area of RAM. You can then examine it using "see working pix" on the main menu. This working screen picture is then saved to the destination disk in the selected format.

The only tricky part to keep track of is that *CoCo Max* pictures use two screens. After loading, if you intend to transfer the first (upper) screen, then



answer "no" to the "copy screen 2 to working pix?" prompt. To transfer the second (lower) screen, answer "yes."

To transfer pictures to *CoCo Max*, they are first saved in temporary files. Pictures can be temporarily saved as Screen 1 or Screen 2. You must have pictures temporarily saved to both screens before the final "save screens 1/2." The temporary files and the final save must be on the same drive and disk.

If you are typing in the listing, the comments may be deleted. After entering and saving the program, start debugging, but be sure to use backups of all the picture files you are working with!

A blank space has been left on the main menu. This is for you to add your own routine or call another program. This is a handy place for a screen print routine, for example.

(You may contact the author of this program with any questions at 4333 Larchwood Circle, NW, Canton, OH 44718, phone 216-492-7819. Please include an SASE when writing.) □

120	.....	126
290	.....	54
430	.....	52
550	.....	111
700	.....	17
880	.....	165
970	.....	206
END	.....	255

### The listing: PIXFILES

```

1# 'PIX FILES
2# 'JOSEPH KOHN
3# '4333 LARCHWOOD CR.,NW
4# 'CANTON, OH 44718
5# IFX=0THENPCLEAR8:X=1:GOTO5#
6# CLEAR3#0#0#,&H7FFF:DIMF$(68),GS
(4#),X$(22):FB=&HFFF#DK=&HC#
#4:PB=PEEK(DK+2)*256+PEEK(DK+3):
DR=PEEK(PB+1)
7# DATA GRAPHICOM PIX,graphicom
pix,COCO MAX PIX,coco max pix,ST
ANDARD PIX,standard pix,SEE WORK
ING PIX,see working pix,PIX DRIV
E,pix drive,,QUIT,quit,TEMPOR
ARY SAVE SCREEN 1,temporary save
screen 1,TEMPORARY SAVE SCREEN
2,temporary save screen 2
8# DATA SAVE SCREENS 1/2,save sc
reens 1/2,ABORT SAVE,abort save,
X
9# READX$(X):IFX$(X)<>"X"THENX=X
+1:GOTO9#
10# PMODE4,1:X$="PIX FILES":GOSU
B81#LN=#:FORX=#TO6:PRINT#66+64*
X,X$(2*X):NEXT:GOTO7#
11# JX=JOYSTK(#):JY=INT(JOYSTK(1
)/1#):JY=JY-(JY=5):IFJY<>LN THEN
PRINT#66+64*LN,X$(2*LN):SOUND1#0#
,1
12# PRINT#66+64*JY,X$(2*JY+1):I
FJY>2THEN1#ELSEIFJX<32THENPRINT
": load SAVE":LS=#ELSEPRINT": L
OAD save":LS=1
13# IFPEEK(FB)AND1THENLN=JY:PRIN
T#331,DR:GOTO1#ELSELN=JY+1 GOTO
15#39#56#71#74#11#77#
14# '-----graphicom load/save
15# X$=X$(#):GOSUB81#
16# GOSUB87#IFYN=#THEN1#0#
17# GOSUB9#IFN=#THEN1#2#
18# GOSUB81#GOSUB88#IFYN=#THEN
1#ELSEGOSUB25#IFSN<#THEN1#2#EL
SEIF LS THEN2#
19# '-----load graphicom
20# PCLS1:SCREEN1:W=2:AD=&HE#:#G
OSUB32#GOTO1#0#
21# '-----save graphicom
22# SCREEN1:AD=&HE#:#W=3:GOSUB32
#IFSN=#THEN1#0#
23# GET(216,162)-(255,191),GS,G:
PMODE4,5:SCREEN1:PUT(X1,Y1)-(X2-
2,Y2-2),GS,PSET:SN=#:AD=&H26#:#W
=3:GOSUB32#GOTO1#0#
24# '-----load graphicom directo
ry
25# PMODE4,5:PCLS1:SCREEN1
26# AD=&H26#:#SN=#:W=2:GOSUB32#
27# '-----select pix
28# X=INT(JOYSTK(#)/11):Y=INT(JO
YSTK(1)/13)
29# Y1=(Y-(Y>#))*32:X1=X*42+1:X2
=X1+41:Y2=Y1+31:FORC=#TO1:COLORC
:LINE(X1,Y1)-(X2,Y2),PSET,B:NEXT
30# IFPEEK(FB)AND1THEN28#ELSESN=
Y*6+X-4:PMODE4,1:IFSN<#THENCLS:P
RINT"abort from graphicom":RETURN
NELSERETURN
31# '-----graphicom i/o,w=2/3=re
ad/write,tr=track,s=sector,a=add
ress,dk=dskcon
32# S=SN*24:TR=INT(S/18):S=S-(TR
*18)+1

```

```

33# IFTR>17 OR (TR=17 AND S>1)TH
ENS=S+2:IFS>18THENS=S-18:TR=TR+1
34# FORI=#TO23:A=AD+256*I:POKEPB
,W:POKEPB+1,DR:POKEPB+2,TR:POKEP
B+3,S:POKEPB+4,INT(A/256):POKEPB
+5,A-256*INT(A/256):EXEC PEEK(DK
)*256+PEEK(DK+1)
35# S=S+1:IFS>18THENS=1:TR=TR+1
36# IFTR=17 AND S=2THENS=4
37# NEXT:RETURN
38# '-----coco max load/save
39# X$=X$(2):GOSUB81#
40# GOSUB87#IFYN=#THEN1#0#
41# GOSUB81#GOSUB88#IFYN=#THEN
1#ELSEIF LS THEN46#
42# '-----load coco max
43# GOSUB9#IFN=#THEN1#2#ELSEPC
LS1:SCREEN1:LOADMF$(VAL(K$))+DR$
44# GOSUB81#PRINT"COPY SCREEN 2
TO WORKING PIX":GOSUB85#IF YN
THENFORX=#TO8:PCOPY X TO X-4:NE
XT:GOTO1#ELSE1#
45# '-----save coco max
46# GOSUB81#LN=#:FORX=#TO3:PRIN
T#66+64*X,X$(2*X+14):NEXT
47# JX=JOYSTK(#):JY=INT(JOYSTK(1
)/2#):IFLN<>JY THENPRINT#66+LN*6
4,X$(2*LN+14):SOUND1#0#,1
48# PRINT#66+64*JY,X$(2*JY+15):I
FPEEK(FB)AND1THENLN=JY:GOTO47#
49# ON JY+1 GOTO5#0#,51#,53#,1#
50# IFFREE(DR)>2THEN1#(DR)=1:SCR
EEN1:SAVEM"TEMP/#0#"+DR$,&HE#,#,&
H25FF,&HA#27:GOTO47#ELSE1#3#
51# IFFREE(DR)>2THEN2#(DR)=1:SCR
EEN1:FORX=1TO4:PCOPY X TO X+4:NE
XT:SAVEM"TEMP/#0#"+DR$,&H26#,#,&H
3DFE,&HA#27:GOTO47#ELSE1#3#
52# IFJY=3THEN1#0#
53# GOSUB81#IFT1(DR)=# OR T2(DR
)=#THEN1#5#ELSELINEINPUT"FILE NA
ME?":#:#:IF F$="" OR LEN(F$)>8 T
HEN1#4#
54# T1(DR)=#:T2(DR)=#:PCLS1:SCRE
EN1:LOADM"TEMP/#0#"+DR$:#KILL"TEM
P/#0#"+DR$:#PMODE4,5:PCLS1:SCREEN
1:LOADM"TEMP/#0#"+DR$:#KILL"TEMP/
#0#"+DR$:#SAVEM F$+"/MAX"+DR$,&HE
#,#,&H3DFE,&HA#27:GOTO1#0#
55# '-----standard load/save
56# X$=X$(4):GOSUB81#
57# GOSUB87#GOSUB83#IFYN=#THEN
1#0#
58# GOSUB81#GOSUB88#IFYN=#THEN
1#ELSEIF LS THEN68#
59# '-----load standard
60# GOSUB9#IFN=#THEN1#2#
61# '-----check file length
62# NA$=F$(VAL(K$))+DR$:OPEN"D",
1,NA$,1:FIELD1,LAS C$:R=1
63# GET#1,R:IFASC(C$)=255THEN65#
64# GET#1,R+1:L=256+ASC(C$):GET#
1,R+2:L=L+ASC(C$):GET#1,R+3:A=25
6*ASC(C$):GET#1,R+4:A=A+ASC(C$):
SA=A:R=R+L+5:GOTO63#
65# GET#1,R+3:E=256*ASC(C$):GET#
1,R+4:E=E+ASC(C$):EA=A+L-1:CLOSE
#1
66# IF EA-SA<>&H17FF THENCLS:PRI
NT"not a picture file":GOTO1#2#E
LSEPCLS1:SCREEN1:LOADMNA$:GOTO1#
0#
67# '-----save standard
68# IFFREE(DR)<3THEN1#3#
69# GOSUB81#LINEINPUT"FILE NAME
/EXT?":F$:IFLEN(F$)>13 OR F$
=""THEN1#5#ELSESCREEN1:SAVEM F$+
DR$,&HE#,#,&H25FF,&HA#27:GOTO1#0#
70# '-----see working pix
71# SCREEN1:GOSUB83#
72# IFPEEK(FB)AND1THEN72#ELSE1#0#
73# '-----pix drive,dx=number of
drives
74# DX=2:DR=DR+1:IFDR=DX THENDR=
#
75# DR$="":MID$(STR$(DR),2,1):P
RINT#331,DR:SOUND1#0#,1:GOTO1#
76# '-----quit
77# X$=X$(12):GOSUB81#
78# PRINT"ARE YOU SURE?":GOSUB85
#

```

```

79# IF YN THENUNLOAD:CLS:END ELS
E1#0#
80# '-----title display
81# CLS:X=LEN(X$):Y=INT((32-X)/2
):PRINTSTRING$(Y,"*")X$STRING$(3
2-X-Y,"*")
82# '-----fire button debounce
83# FORX=#TO1#0#NEXT:IFPEEK(FB)A
ND1THENRETURNELSE3#
84# '-----prompts
85# IFJOYSTK(#)<32THENPRINT#135,
"yes NO":YN=1ELSEPRINT#135,"YES
no":YN=#
86# IFPEEK(FB)AND1THEN85#ELSE83#
87# PRINTX$(LN*2)" DISK IN DRIVE
"DR?":GOTO85#
88# PRINT"READY TO ";:IF LS THEN
PRINT"SAVE?":GOTO85#ELSEPRINT"LO
AD?":GOTO85#
89# '-----disk file search
90# CLS:PRINT"SEARCHING...":N=#:
FORZ=#TO11:DSKIS$ DR,17,Z,B$(#),B
$(1):FORQ=#TO1:FORW=#TO3:K$=MID$
(B$(Q),W*32+1,32):IF ASC(K$)=255
THENZ=99:W=Z:Q=Z:GOTO95#
91# IFASC(K$)=#THEN95#ELSEON LN+
1 GOTO92#93#94#
92# IFMID$(K$,12,1)=CHR$(1) AND
LEFT$(K$,11)="PICTURESGCM"THENN=
1:GOTO95#ELSE95#
93# IFMID$(K$,9,3)="MAX" AND MID
$(K$,12,1)=CHR$(2) THENN=N+1:F$(N
)=LEFT$(K$,8)+"/MAX":GOTO95#ELSE
95#
94# IFMID$(K$,12,1)=CHR$(2) AND
MID$(K$,9,3)<"MAX"THENN=N+1:F$(N
)=LEFT$(K$,8)+"/"+MID$(K$,9,3)
95# NEXTW,Q,Z:IFN=# AND LN=#THEN
CLS:PRINT"not a graphicom disk":
RETURN
96# IFLN=#THENRETURN
97# IFN=#THENCLS:PRINT"no pictur
es":RETURN
98# CLS:Q=1:FORZ=1TO3:FORW=1TO3#
:PRINT#(W-1)*16,"":PRINTUSING"#
#":Q:PRINT".":F$(Q):IFQ=N THENW=
99:Z=W
99# IFINT(Q/3#)=Q/3#THENPRINT#48
#,"CONTINUE...":LINEINPUTK$:NEX
TZ
1#0# Q=Q+1:NEXTW,Z:PRINT#48#,"":
INPUT"NUMBER OF PIX TO LOAD":K$:
IFK$=""ORVAL(K$)<1 OR VAL(K$)>N
THENN=#:GOTO97#ELSERETURN
1#2# SOUNDS5#,1#FORX=#TO1#0#NEX
T:GOTO1#0#
1#3# CLS:PRINT"no room on disk":
GOTO1#2#
1#4# CLS:PRINT"improper file nam
e":GOTO1#2#
1#5# CLS:PRINT"no temporary file
":IFT1(DR)=# AND T2(DR)=#THENPR
INT"s"ELSEIFT1(DR)=#THENPRINT" 1
"ELSEPRINT" 2"
1#6# GOTO1#2#

```



# UTILPACK

by Rod Hoskinson

**RIDDLE:** Extended Color Basic users have the EDIT command, MC-10ers have their little e, but how do CoCo users without Extended Basic edit a basic program line without retyping the whole of it all over again?

**ANSWER:** Using the LEDIT command, of course. LEDIT? No, it has nothing to do with those little red lights. LEDIT stands for LINE EDIT, and is available via UTILPACK.

Ready for another riddle? Okay, why is this new thing called LEDIT, rather than just edit?

**ANSWER:** for the very good reason that this preserves compatibility with ECB and DECB, because there is something in UTILPACK for all CoCo Users.

I bet there is not a reader out there who has not at some time cursed CoCo's destructive cursor. Everyone must know the feeling - you have just typed a long line from command mode or during an INPUT, then you realise you have made an error at the start of the line.

Well curse no more, and enjoy the luxury of on screen editing with UTILPACK!

Have you ever wondered what it would be like to have a few more BASIC commands at your disposal? Can't remember all those peeks and pokes? Then know that UTILPACK also features FAST, to put CoCo in the 1.78 MHz mode, SLOW to bring CoCo back to a more leisurely 0.89 MHz, and COLD to perform a cold start.

ECHO ON will copy all screen output to your printer if it is online, and ECHO OFF will stop this.

To get this package of utilities up and running, type the following program, then do a few CSAVEs before RUNNING. The program includes a checksum to help detect typing errors. If the program detects a checksum error, you will have to carefully re-examine the DATA lines.

The BASIC program pokes UTILPACK, which is 806 bytes of machine code, into high memory. The machine code is position

independent (it can go anywhere in memory), and it is automatically poked into the top of RAM for a 16k or 32k user.

Additionally the program automatically configures itself for non-ECB, ECB, or Disk ECB. (I have tried it on non-ECB and ECB and it works fine. I haven't tried it with a disk system but am almost certain it will run with no trouble).

Once you have RUN UTILPACK, you should see a sign on message and immediately notice a change - the cursor should be a steady block square. If you have got this far, chances are things are 100% O.K. If you have repeated trouble, I suggest you obtain this program on a CoCoOz cassette, if you have not already done so.

Now you should be ready to try out UTILPACK. Type away at the keyboard, then hit the left arrow (backspace) key. The cursor will move, but will not delete any characters. Try the right arrow key for forward cursor movement. The cursor will be the inverse of the current character. Hitting the shift left or right arrow will move the cursor to the start of end of the buffer.

To delete the character under the cursor, hold down the clear key (which has been reprogrammed as a control type key), and press the left arrow. To open up a space for insertion, hold clear down and press the right arrow key. To use the CLEAR key as normal, hold clear and press the commercial at (@) sign. This clears the screen and homes the cursor.

Because the shifted right arrow is now used for cursor movement, you obtain a closed square bracket by pressing the down arrow. Pressing any key other than those described will replace the current character.

You can use this screen editor in command mode, or from an INPUT or LINEINPUT command within a program.

To use the new LEDIT command, type LEDIT, followed by the line number of the line you wish to edit. If the line does not exist you will generate a UL ERROR.

The line you are editing will be printed with the cursor at the start. You have all the features of the screen editor at your disposal. When you have finished editing, press <ENTER>. Pressing BREAK at this stage will cancel any changes you have made. You can always enter or break no matter where in the line the cursor is located.

Apart from LEDIT, try the other new commands: FAST, SLOW, COLD, ECHO ON or

ECHO OFF.

If you like the features of UTILPACK, you should get into the habit of installing it at the start of any programming session. The new commands are tokenised and can be used in immediate mode within a program. Make sure UTILPACK is running before CLOADing a program that uses the new commands.

I hope you will find this program useful.

The listing:

```

1 REM UTILPACK=ENHANCEMENTS TO B
  ASIC BY ROD HOSKINSON V1.1 1986
2 GOTO 10
3 CSAVE"UTILPACK"
10 POKE32767,50
20 IFPEEK(32767)=50 THEN AD=3196
2 ELSE AD=15578
30 CLEAR200,AD-1
40 IFPEEK(32767)=50 THEN AD=3196
2 ELSE AD=15578
50 CHECKSUM=0
60 FOR I=AD TO AD+805
70 READ Q$:GOSUB270
80 POKEI,Q
90 CHECKSUM=CHECKSUM+Q
100 NEXT I
110 IF CHECKSUM<>84404 THEN PRIN
T"ERROR IN DATA-RETRY":END
120 IFPEEK(298)=0 THEN POKE AD+6
9,42:POKEAD+172,185:POKEAD+176,1
81:ELSE IFPEEK(308)<>0 THEN POKE
AD+69,62:POKE AD+172,229:POKE A
D+176,225
130 EXEC AD
140 CLS:PRINT"BASIC ENHANCEMENTS
  V1.1 1986 BY ROD HOSKINSON A
  RE NOW LOADED":PRINT"MEM="MEM+30
93
150 NEW
160 DATA 30,8D,0,49,31,8D,0,8C,1
0,AF,8D,0,41,31,8D,0,98,10,AF,8D
,0,3A,31,8D,0,AC,10,AF,8D,0,9D,3
1,8D,0,AA,10,AF,8D,0,96,31,8D,0,
A8,10,AF,8D,0,8F,31,8D,0,A4,10,A
F,8D,0,88,31,8D,0,CO,10,AF,8D,0,
81,CE,1,34,C6,B,BD,A5,9A
170 DATA 20,B,5,75,C4,75,D9,0,0,

```

```

0,0,0,0,86,7E,B7,1,82,30,8D,0,F3
,BF,1,83,F6,1,67,E7,8C,26,BE,1,6
8,AF,8C,21,B7,1,67,30,8D,0,4,BF,
1,68,39,34,4,D6,6F,C1,FE,27,B,E6
,8D,0,C,C1,FF,26,3,BD,A2,BF,35,4
,7E,82,73,0,46,41
180 DATA 53,D4,53,4C,4F,D7,43,4F
,4C,C4,45,43,48,CF,4C,45,44,49,D
4,34,1,81,D2,22,9,80,CE,48,33,8D
,0,6,6E,D6,6E,9F,1,46,75,F6,75,F
D,76,4,76,9,76,2E,7F,FF,D7,9D,9F
,35,81,7F,FF,D6,9D,9F,35,81,F,71
,7E,A0,27,9D,9F,81,20,27,FA,81,A
A
190 DATA 27,7,81,88,27,B,BD,A5,C
9,6F,8D,FF,A5,9D,9F,35,81,6F,8D,
FF,9D,6A,8D,FF,99,9D,9F,35,81,9D
,9F,BD,AF,67,BD,AD,1,24,3,7E,AE,
D2,BD,B7,C2,8E,2,DD,A6,80,81,0,2
6,FA,30,1F,AF,8D,1,A7,DC,2B,BD,B
D,CC,BD,B9,AC,8E,2,DD,A6,80,AD,9
F
200 DATA A0,2,81,0,26,F6,EC,8D,1
,8E,83,2,DD,34,6,DC,88,A3,E1,DD,
88,8E,2,DD,17,0,1A,25,8,8E,2,DD,
9F,A6,7E,AC,A8,7E,AC,76,8E,2,DD,
32,62,AF,8D,1,67,6F,84,6F,1,C6,1
,17,0,93,17,1,52,81,BF,10,27,0,A
6,BD,A1,C1,81,0,27,F0,81
210 DATA 8,27,4A,81,9,27,5D,81,1
5,10,27,1,6A,81,5D,10,27,1,7D,81
,A,10,27,1,91,12,12,12,12,81,3,1
A,1,10,27,1,2A,81,D,26,3,16,1,48
,81,C,27,BF,8C,3,DB,27,BA,A7,80,
AC,8D,1,12,25,6,6F,84,AF,8D,1,A,
5C,AD,9F,A0,2,17,0,37
220 DATA 20,A2,8C,2,DD,27,9D,17,
0,2D,30,1F,DC,88,83,0,1,DD,88,17
,0,21,5A,20,8B,8C,3,DB,27,86,A6,

```

```

84,81,0,27,80,30,1,17,0,E,DC,88,
C3,0,1,DD,88,17,0,4,5C,16,FF,6D,
DE,88,A6,C4,81,40,25,F,81,80,25,
6,86,20,A7,C4,20,EE,80,40
230 DATA A7,C4,39,8B,40,A7,C4,39
,BD,A1,C1,81,0,10,27,FF,48,81,8,
27,1C,81,9,27,44,81,40,26,11,BD,
A9,28,8E,2,DD,AF,8D,0,90,6F,84,6
F,1,16,FF,26,16,FF,28,DE,88,34,5
0,1F,12,31,21,A6,A0,A7,80,81,0,2
7,6,AD,9F,A0,2,20,F2,86,20,AD,9F
240 DATA A0,2,30,1F,AF,8D,0,64,3
5,50,DF,88,17,FF,92,16,FE,FC,DE,
88,34,50,AE,8D,0,52,8C,3,DA,25,5
,35,50,16,FE,EA,A6,84,A7,1,AC,E4
,27,4,30,1F,20,F4,86,20,AD,9F,A0
,2,A7,80,A6,80,AD,9F,A0,2,34,2,1
F,A8,81,80,26,7,EE,63,33,C8,E0
250 DATA EF,63,35,2,81,0,26,E5,3
0,1F,AF,8D,0,13,35,50,DF,88,17,F
F,41,16,FE,AB,86,FD,B7,FF,2,B6,F
F,0,39,2,DD,34,1,17,FF,2E,AC,8D,
FF,F5,27,E,DE,88,33,41,DF,88,30,
1,AC,8D,FF,E7,26,F4,35,1,34,1,BD
,B9,58,8E,2,DC,35,81,4F,20,DB
260 DATA 17,FF,8,8C,2,DD,27,B,5A
,30,1F,DE,88,33,5F,DF,88,20,F0,1
7,FE,F5,16,FE,5F,17,FE,EF,AC,8D,
FF,B6,27,B,5C,30,1,DE,88,33,41,D
F,88,20,EF,17,FE,DB,16,FE,45,86,
5D,16,FE,7D
270 Q$=RIGHT$("00"+Q$,2)
280 A$=LEFT$(Q$,1):B$=RIGHT$(Q$,
1)
290 IFA$)="A"THENH=ASC(A$)-55 EL
SE H=ASC(A$)-48
300 IFB$)="A"THENL=ASC(B$)-55 EL
SE L=ASC(B$)-48
310 Q=H*16+L:RETURN

```

### One-Liner Contest Winner. . .

This short program, called *Grader*, is mighty useful to students who have a number of test grades and want to compute their grade average. All grades must be entered in number form.

The listing:

```

1 CLS:T=0:G=0:A=0:INPUT"# OF TES
TS";N:FORX=1TON:INPUT"GRADE";G:T
=T+G:NEXTX:A=T/N:PRINT"YOUR AVER
AGE IS ";A:INPUT"AGAIN";A$:IFA$=
"Y"THENGOTO1ELSEEND

```

Judy Zoll Leo  
Skillman, NJ

### Hint . . .

## Repairing Deluxe Joysticks

There have been a number of reports of problems with Radio Shack's Deluxe Joystick, where the stick becomes loose and you no longer have control. I've found that a pin that holds one of the control levers to the joystick ball tends to work itself out of the ball.

After opening the joystick case, you'll see a pin on one of the levers that now faces the case instead of the ball. Hold the stick so that the hole in the ball lines up with this pin, and use needle-nose pliers to push the pin back into place.

- Ed Ellers



# Bubble Wars!

By Richard Ramella

**"Ping!"** said the kid. "Captain Nick Hazard is sailing through deep space in his Blue Death Suit. He sends another dread Rotundo to its maker!"

"That's not Captain Nick Hazard," I told the kid. "It's a hummingbird. And the orange things are just bubbles, not some kind of space creatures."

"They're the most-feared space creatures of all! They want to turn Captain Nick inside out!"

"Will you stop it?" I asked. "This is a non-violent arcade game for CoCo computers with at least 16K memory, Extended Color BASIC and one joystick."

"This is the most violent game I've ever played!" yelled the kid. "Ping! Pow! Bar-room! Yucko, look at 'em splatter!"

I withdrew. The kid playing the game *Bubble War* was and still is my son. He's

10. At age 3 he wanted a toy gun. When my wife and I refused, he chewed a graham cracker into the shape of a pistol and genially "powed" anyone who came near his high chair.

Not much has changed in the last eight years.

Despite its name, *Bubble War* is a gentle but tricky arcade game that involves nothing more violent than a blue hummingbird popping orange bubbles. It demonstrates how to have fun without destroying the universe.

If your computer won't accept the speed POKE 65495,0, delete Line 130.

At the start of the game, a white screen bordered by green appears. Within it is a wing-flapping blue hummingbird. Using the joystick, you can move the bird around the screen at a fair clip. It goes either northeast, southeast, southwest or northwest, depending on the quadrant in which the joystick is being held. Push the firebutton on the joystick and the hummingbird fires in the general direction it's traveling — left

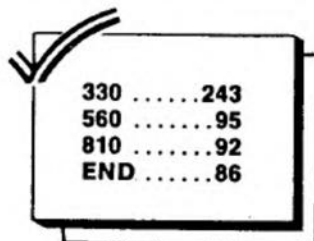
if moving westward, right if eastward.

Aim and fire to burst bubbles. Hits are scored according to the current size of orange bubbles appearing on the screen. Avoid taking the hummingbird too near any orange bubble or debris thereof; this will end the game.

From time to time, the action freezes and a colorful series of concentric circles spreads over the screen. The screen blanks and play resumes. This is done to wipe out bubble debris.

To see a current score during the game, press keyboard letter 'X', then press any other key except SHIFT or BREAK to continue play.

My son persists in imagining danger and triumph in his games. Sometimes I try to tell him that all computer arcade games are comprised in totality of but one fundamental idea. There are teams of light, whether two pixels or a thousand complex graphics shapes. One team chases, the other evades. As the player you may be on either side. The mind fills in the rest of the fantasy.



The listing: BUBBLWAR

```
100 REM * BUBBLE WAR * TRS-80 EX
TENDED COLOR BASIC 16K *
110 REM * BY RICHARD RAMELLA *
120 REM * REQUIRES ONE JOYSTICK
*
130 POKE 65495,0
140 CLEAR 256
150 PL$="L255;GFEDCBA"
```

```
160 DIM M1(1,6),M2(1,6)
170 CLS
180 HG=10
190 PG=10
200 PRINT @ 235,"BUBBLE WAR"
210 PMODE 3,1
220 PCLS 1
230 COLOR 3,5
240 P=128
250 Q=96
260 FOR G=1 TO 2
270 CIRCLE(10,10),5,3,1.5
280 CIRCLE(10,5),7,3,.3
290 LINE(4,11)-(17,11),PSET
300 IF G=1 THEN LINE -(10,15),PS
ET ELSE LINE -(10,5),PSET
310 LINE -(4,11),PSET
320 IF G=1 THEN GET(0,0)-(20,20)
,M1,G ELSE GET(0,0)-(20,20),M2,G
330 PCLS 1
340 NEXT G
350 SCREEN 1,1
360 CIRCLE(128,96),255,6
370 U=119
```

```
380 R=86
390 IF U<5 THEN U=5 ELSE IF U>22
9 THEN U=229
400 IF R<1 THEN R=1 ELSE IF R>17
0 THEN R=170
410 CT=CT+1: IF CT>150 THEN GOSU
B 1000: CT=0
420 PUT(U,R)-(U+20,R+20),M1,PSET
430 PSET(X,R-1,3)
440 IF PPOINT(U-1,R-1)=8 OR PPOI
NT(U+10,R-1)=8 OR PPOINT(U+21,R-
1)=8 OR PPOINT(U-1,R+10)=8 OR PP
OINT(U+21,R+10)=8 OR PPOINT(U-1,
R+21)=8 OR PPOINT(U+10,R+21)=8 O
R PPOINT(U+21,R+21)=8 THEN 870
450 S=RND(PG)
460 IF S=1 THEN GOSUB 620
470 K=JOYSTK(0)
480 L=JOYSTK(1)
490 PUT(U,R)-(U+20,R+20),M2,PSET
500 IF K>31 THEN U=U+4
510 IF K<31 THEN U=U-4
520 IF L>31 THEN R=R+4
530 IF L<31 THEN R=R-4
540 PK=PEEK(65280)
```

```

550 IF PK=126 OR PK=254 THEN GOS
UB 590
560 XG$=INKEY$
570 IF XG$="X" THEN GOSUB 980
580 GOTO 390
590 IF K>31 THEN GOSUB 710
600 IF K<31 THEN GOSUB 790
610 RETURN
620 G=RND(255)
630 P=RND(191)
640 IF G>U-21 AND G<U+42 AND P>R
-21 AND P<R+42 THEN 700
650 CIRCLE(G,P),HG,8
660 PAINT(G,P),8
670 NN=NN+1
680 IF NN=10 THEN HG=HG+5: NN=0:
JK=JK+1: IF JK=2 THEN PG=PG-1
690 IF PG<1 THEN PG=1
700 RETURN
710 M=U+22
720 N=R+9
730 IF M>253 THEN RETURN ELSE PS
ET(M,N,3)
740 S=RND(PG+20)
750 IF S=1 THEN GOSUB 620
760 V=PPOINT(M+2,N)
770 IF V<5 AND V>7 THEN 930 EL
SE PRESET(M,N): M=M+5: GOTO 730
780 RETURN
790 M=U-1
800 N=R+10
810 IF M<2 THEN RETURN ELSE PSET
(M,N,3)
820 S=RND(PG+20)
830 IF S=1 THEN GOSUB 620

```

```

840 V=PPOINT(M-2,N)
850 IF V<5 AND V>7 THEN 930 EL
SE PRESET(M,N): M=M-5: GOTO 810
860 RETURN
870 FOR T=1 TO 15
880 PUT(U,R)-(U+20,R+20),M1,NOT
890 PLAY PL$
900 NEXT T
910 XX=1
920 GOTO 980
930 IF V=6 THEN 950
940 IF V=8 THEN CIRCLE(M,N),25,5
: PAINT(M,N),5,5: PLAY PL$: SC=S
C+(HG*10)
950 IF V=6 THEN PRESET(M,N)
960 CIRCLE(128,96),255,6
970 RETURN
980 IF XX=1 THEN POKE 65494,0: P
RINT @ 331,"F I N A L";
990 PRINT @ 235,"BUBBLE WAR";
1000 PRINT @ 363,"S C O R E";
1010 PRINT @ 394,SC;
1020 IF XX=1 THEN YU=YU+1: PLAY
PL$
1030 IF XX=1 AND YU<40 THEN 1020
ELSE IF YU=40 THEN END
1040 PRINT @ 448,"TAP A KEY TO R
ETURN TO GAME";
1050 XX$=INKEY$
1060 IF XX$<>" " THEN CLS: SCREEN
1,1: RETURN ELSE 1050
1070 END
1080 FOR BN=0 TO 150 STEP 3
1090 CIRCLE(129,96),BN,1+RND(3),
1

```

```

1100 IF BN<20 THEN PLAY "T128:CD
EFGAB"
1110 NEXT BN
1120 PCLS1
1130 CIRCLE(128,96),255,6
1140 RETURN
1150 END
1160 REM * END OF LISTING

```



continued from Page 13

# Commandos



```

3052 I$=STR$(I):H$=STR$(H+5):V$=
STR$(V+5)
3054 DRAW"C"+I$+";BM"+H$+";"+V$+
BL$
3056 NEXT I,J
3060 LI=LI-1:IF LI<=0 THEN 3080
3070 ON PH GOTO 350,1000,2000
3080 CLS:PRINT"YOUR TEST IS OVER
"
3090 PRINT" **RATING**"
3100 PRINT"LAST LEVEL COMPLETED:
";LV-1
3110 PRINT" # OF LIVES SAVED:";
3120 IF PH>2 THEN PRINTLV ELSE P
RINT LV-1
3130 PRINT"RANK:";IF LV =1 THEN
PRINTR$(1) ELSE IF LV =6 AND I
I=1 THEN PRINTR$(6) ELSE PRINT R
$(LV-1)
3140 PRINT"SCORE:";SC
3150 IF LV<6 OR II=0 THEN 3155
ELSE PRINT"TIME TO COMPLETE COUR
SE:";TT/100:GOTO 3160
3155 PRINT"PRESS ANY KEY"
3156 A$=INKEY$:IF A$="" THEN 315
6 ELSE 6000
3160 PRINT"PRESS ANY KEY"
3170 A$=INKEY$:IF A$=""THEN 3170
3180 PMODE3,1:PCLS:SCREEN1,0
3190 CIRCLE(150,50),22,2:PAINT(1
50,50),2,2:CIRCLE(150,50),30,2
3200 PAINT(150,25),3,2
3210 DRAW"C2;BM160,80;ND70G10H10
ND70L10D80E10D20E10F10U20F10U80N
L10R10U20E10H10U20L20H10G10L20D2
0G10F10D20R10"
3220 DRAW"C3;BM140,40;R20BG20R20
BH20BG50D10R70U10BR5NR7D5NL7BR

```

```

5U5NR10E5F5D5"
3230 PAINT(135,100),4,2:PAINT(16
5,100),3,2
3240 COLOR2,1:LINE(40,60)-(100,6
0),PSET:LINE(200,60)-(254,60),PS
ET
3250 DRAW"BM110,20;NH20BR80NE20B
D80NF30BL80G30"
3260 A$=INKEY$:IF A$="" THEN 326
0
3270 GOTO 6000
4000 'WIN
4010 'LV1
4020 GET(5,130)-(20,130),C,G
4030 H=5:V=130
4040 PUT(H,V)-(H+15,V+10),D
4050 V=V-5:PUT(H,V)-(H+15,V+10),
C,PSET
4060 IF V<=10 THEN 4150 ELSE 404
0
4070 'LV3
4075 IF LV=6 THEN 4150
4080 PCLS(1)
4090 DRAW"C2;BM20,85;D10R10BR5NR
10U5NR5U5R10BR5D5F5E5U5BR5NR10D5
NR5D5R10BR5NR10U10"
4100 FORI=1TO LV+1:LINE(95+(I*5)
,85)-(95+(I*5),95),PSET:NEXT
4110 FORI=1TO1000:NEXT
4150 'WIN1,2,3
4155 II=1:GOSUB5000
4156 SC=SC+(LV*100)
4157 IF TI<200 THEN SC=SC+(200-T
I)
4158 TT=TT+TI
4160 PH=PH+1:IF PH>3 THEN PH=1:L
V=LV+1
4170 IF LV>6 THEN 3080
4180 ON PH GOTO 350,1000,2000
5000 'THEME
5010 Z$="O2BG":X$="AO3C"
5020 PLAY"T603L4DDXZ$;ACBG03DDXZ
$;XX$;O2L2AO3L4DD"

```

```

5030 PLAY"XZ$;XX$;O2BGAO3CXZ$;AO
3DO2L2G"
5040 RETURN
6000 'HIGH SCORE
6010 IF SC>SC(10) THEN 6020 ELSE
6140
6020 SC(11)=SC:N$(11)=N$:LV(11)=
LV
6030 SC=0
6040 IF SC=1 THEN 6140
6050 SC=1
6060 FORI=1TO10
6070 IF SC(I)<SC(I+1) THEN 6080
ELSE 6110
6080 SC=0:S=SC(I):SC(I)=SC(I+1):
SC(I+1)=S
6090 N$=N$(I):N$(I)=N$(I+1):N$(I
+1)=N$
6100 LV=LV(I):LV(I)=LV(I+1):LV(I
+1)=LV
6110 NEXT
6120 GOTO 6040
6140 CLS:PRINT@10,"*HIGH SCORE*"
6150 PRINT"-----"
6160 FORI=1TO10:IF SC(I)=0 THEN
6180
6170 PRINTN$(I) " "SC(I) " LE
VEL:"LV(I)
6180 NEXT
6190 PRINT:INPUT"WANT TO PLAY AG
AIN(Y/N)";A$
6200 IF A$="Y" THEN 150
6210 END
7000 '1
7010 N=LEN(N$)
7020 IF N>10 THEN 7030 ELSE IF N
<10 THEN 7040 ELSE RETURN
7030 N$=LEFT$(N$,10)
7035 RETURN
7040 FORI= 1 TO 10-N
7050 N$=N$+" "
7060 NEXT:RETURN

```

"It changes the DSKI\$, DSKO\$, DSKINI and BACKUP commands to operate at 36 tracks. It keeps a spare copy of the disk directory at Track 36. It can recover directories with logical errors and last, but most important, it can recover directories with physical errors."

# Crash-Proof It!

By Terry Wilson

After seeing a letter in "Downloads" from a reader who was having problems restoring a damaged Track 17 on a disk, I wrote this program, *Crashproof*. It is very short and only uses up one granule. I used to keep *Zapper* on every disk to store and retrieve directories, but at a cost of five granules.

*Crashproof* requires a 64K disk system. A copy should be kept on every disk for convenience. The program does four things: It changes the DSKI\$, DSKO\$, DSKINI and BACKUP commands to operate at 36 tracks instead of 35. It keeps a spare copy of the disk directory on Track 36. It can recover directories with logical errors and last, but most important, it can recover directories with physical errors.

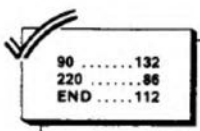
After the conclusion of an input/output session, I type RUN"CP" and press ENTER at the main menu. In a few seconds, Track 17 is safely stored at Track 36. If you maintain directories with more than 40 files it is advisable to change the '8' in lines 150 and 220 to 11. A logical crash can be repaired in about a minute and a physical crash in about five minutes.

It is important to note to beginners that all disks must have been formatted to 36 tracks before any files are stored on a disk. After choosing Option 1, any subsequent DSKINID formats 36 tracks. Therefore, a RUN"CP" only has to be done once to format a new box of disks. After all disks are formatted, LOAD"CP" again and RUN1010. Pressing any key puts a copy of *Crashproof* on your disk.

Option 2 performs the transfer of Track 36 to Track 17; Option 3 allows a backup of all 36 tracks to another disk. After this operation is complete, RUN"CP" again and choose Option 2. As with any new utility, practice on an insignificant disk first.

*Crashproof* has been written from information gathered from previous issues of RAINBOW, so no big feather in my cap. However, thanks to this information, *Crashproof* is a very useful, up-to-date utility.

(You may contact the author of this program with any questions you have at 3436 Casa Grande, Baton Rouge, LA 70814, phone 504-272-4652. Please enclose an SASE when writing.) □



Editor's Note: If you have the newer Radio Shack Disk BASIC I.I ROM, then you will need to change the POKE addresses in lines 60 and 200. Change the values in Line 60 from &HD446 and &HD180 to &HD534 and &HD29D respectively. Also, change the values in Line 200 from &HD572 and &HD595 to &HD65F and &HD682 respectively.

The listing: CRASHPRF

```

70 CLS:PRINT@39,"CRASH PROOF IT!
!!"
80 PRINT:PRINT" (1) FORMAT NEW
DISK TO 36 TRACKS":PRI
NT" (2) RECOVER CRASHED DISK":P
RINT" (COPY TRACK 36 TO 17)
"
90 PRINT" (3) STILL BAD?":PRIN
T" (OPTION #2 FAILED)"
100 PRINT:PRINT" ENTER YOUR
CHOICE OR"
110 PRINT:PRINT" PRESS <ENT
ER> TO":PRINT" COPY TRACK 17 TO
TRACK 36"
120 INPUT" ";A:IFA>3T
HEN70
130 CLS:ON A GOTO 170,210,230
140 PRINT" IF THIS DISK IS CR
ASHED":PRINT" CRASH PROOF W
ILL CATCH IT":DIR
150 FOR X=2 TO 8:DSKI$ 0,17,X,A$
,B$:DSKO$ 0,35,X,A$,B$:PRINT"TR.
17 SEC.":X"TO TR. 36 SEC.":X:PR
INTA$:B$:NEXT
160 PRINT"DIRECTORY STORED AT TR
ACK 36":FORX=1 TO 100:NEXT:GOTO
70
170 CLS:PRINT@73,">>>CAUTION<<<"
:PRINT:PRINT" YOU ARE ABOUT TO C
LEAN THAT DISK AND FORMAT IT
TO 36 TRACKS"
180 INPUT" SURE":X$:IFLEFT$(
X$,1)<>"Y"THEN70
190 PRINT" INSERT DISK TO BE FOR
MATED":PRINT" PRESS ANY KE
Y":EXEC44539
200 POKE&HD572,&H24:POKE&HD595,&
H24:DSKINI0
210 CLS:PRINT"INSERT BAD DISK IN
TO DRIVE 0":LINEINPUT" PRESS <E
NTER> WHEN READY":X$
220 FOR X=2 TO 8:PRINT"TRACK":X:D
SKI$ 0,35,X,A$,B$:DSKO$ 0,17,X,A
$,B$:PRINTA$:PRINTB$:NEXT:PRINT"
DIRECTORY RESTORED":FORX=1 TO 460:
NEXT:GOTO70
230 CLS:PRINT"THIS OPTION CREATE
S A BACKUP":PRINT"COPY OF THE CR
ASHED DISK. THE":PRINT"BACKUP0 C
OMMAND HAS BEEN CHANGED":PRINT"
TO BACKUP ALL 36 TRACKS SO WHEN"
:PRINT"YOU HAVE MADE A NEW COPY,
YOU":PRINT"MUST RUN CRASHPROOF
AND CHOOSE"
240 PRINT"OPTION #2 TO RESTORE T
HE DIREC- TORY ON YOUR NEW COPY.
"
250 PRINT:PRINT"TYPE 'YES' IF YO
U ARE READY TO BACKUP0"
260 INPUT" READY":X$:IF
X$="YES"THENBACKUP0
270 GOTO70
1000 ' USE A RUNI010 AFTER YOU
HAVE FORMATED ALL YOUR
NEW DISK
1010 CLS:PRINT@40,"CRASH PROOF I
T!!!":PRINT@99,"SAVE A COPY TO E
VERY DISK":PRINT@137,"PRESS ANY
KEY":EXEC44539:IFFREE(0)>0THENSA
VE"CP":PLAY"G":GOTO1010
1020 PRINT" THIS DISK IS F
ULL":EXEC44539:GOTO1010

```

# ASSEMBLY FILE

by Kevin

Appendix II of William Bardens' book COLOR COMPUTER ASSEMBLY LANGUAGE PROGRAMMING contains some very useful and very interesting information. Although it can take a bit of understanding.

Listed is the complete instruction set of the 6809 CPU along with the number of clock cycles required to execute the instruction, the number of bytes the instruction consumes and some good extra information.

Let's take a look at one of the load instructions, LDA for example. By now you should have a fair understanding of the various addressing modes of the 6809. From the table we can immediately see that the LDA instruction can take the forms of Immediate, Direct, Indexed or Extended addressing. When the assembler assembles the DIRECT form of the instruction it will convert the line LDA #34 to the numbers \$96 and \$34. These are the numbers you see as the object code for the line of source code you typed in using your assembler program. Obviously they consume two bytes of memory space and if you look to our table you will find that this is exactly the number of bytes that the instruction was supposed to consume. Also you may like to note that the instruction took 4 clock cycles, although that is generally of little use unless you really need to find ways to make your program run fast. Although I rather suspect that tightening up your programming techniques will gain you more speed in that case.

But what happens in the case of indexed addressing? You will see a separate table titled INDEXED ADDRESSING MODES.

Using Indexed addressing we could have the line LDA ,X+ which translates to \$A6 \$80. How did I get the \$80? Well if you look to the supplementary table you will see that with Auto Increment Indexed Addressing which is what we are doing you will find the binary code for the postbyte opcode. In our example this binary value is 1RR00000. So what do these RR bits

mean? Unfortunately it seems the publishers have left one very important footnote. Essentially the footnote says that for register X you substitute 00 for the RR within the binary number. Likewise Y=01, U=10 and S=11. Where this table shows the assembler form of the instruction merely substitute the name of the register you wish to index for the variable R.

Understand? No then play with and think about it for a while. Still looking at the same table you will see here the number of clock cycles and the number of bytes to be ADDED to the numbers given in the main table.

On to the description column. Again this information is most useful once you begin to understand the shorthand. Most of the shorthand is explained well enough in the Legend at the end of the table so let's just look at our own example. Quite simply the contents of memory are loaded into the A register. But note M does not have to be memory. It can also refer to the postbyte value when using Direct addressing, so don't get caught out. As always a bit of study and more practice will develop your understanding.

Now we come to the effect the instruction will have on the Condition Code Register. The only bits of the CC Register we worry about here are the Half-carry, Negative, Zero, Overflow and Carry bits. Again our LDA instruction will test both the Negative and Zero bits, set them if found to be true, otherwise cleared. The overflow bit will be cleared always upon execution of the LDA instruction.

Dig into this information, type up a few short listings comparing the assembled output with what the tables tell you the code should be. You should this way improve your understanding of Assembly Language no end and likewise have found the most useful ready-reference guide to the 6809 you could want.

# ODDS and ENDS #1

by Andrew Simpson

MACHINE LANGUAGE used to be this thing that I could load but not list; I could play but not change.

Then one day I got the book TRS-80 COLOR COMPUTER ASSEMBLY LANGUAGE PROGRAMMING.

Now Machine Language is this wonderful thing that lets me (when I learn) do almost anything with my CoCo.

When I was about 1/2 way through the book I discovered that the text screen is at &H400 to &H5FF. So if I POKE &H5FF,65 I put the letter A on the bottom right of the screen and the screen does not scroll up.

So can I reverse the process? I try a PEEK (&H5FF) <ENTER> & the computer replies with 65 - which is the decimal number for A, (refer to your CHR\$ codes in the back of GETTING STARTED WITH COLOR BASIC).

So now, every time I want to find out what is on the screen, I can! (I have not thought of a program that will use this, but one day I will.)

After a bit, I got to wondering why I couldn't do the same thing in the graphic screen. From all the things I had read, I understood the graphic screen was at &H600. So I POKEd into &H600.

I tried to put a few things there - &H12, &H76, &HFF etc., but nothing happened. So I thought - if I poke from &H400 on, while looking at the graphic screen, and wait for it to change color, I should be able to find where that happens. So I wrote this program:

```
10 PMODE4,1:SCREEN1,1:PCLS
20 FOR A= &H400 TO &HFFFF
30 POKE A,65
40 NEXTA
```

Then I ran the program and sat waiting, wondering if it would work. Then suddenly, a line went across the screen.

It was not a true line.

I pressed <break> and looked at A. The number I had was part of the way through the screen. So I thought "why can't I use the PPOINT so the computer will stop when it gets to the end? It could stop and tell me the number".

First I needed to make the line a true line, not a dot. I had learned from my book that 1 memory point = 8 screen points. For example, the letter A in binary = 01000010. So on the screen, I get the first 8 dots which are black white black black black black white black. So to make the screen white I must use a symbol that is 11111111, ie &HFF.

Then I wrote this program:

```
10 PMODE4,1:SCREEN1,1:PCLS
20 FOR A= &H2500 TO &H400 STEP-1
30 POKE A,&HFF
40 IF PPOINT(0,0)=5THEN PRINTA:S
TOP
50 NEXTA
```

Then I made the number &HE00 the start and I got the other points the same way.

All this proves that the if start of the screen is &HE00, then the end of the PMODE4,1 screen is &H25FF; the start of the PMODE 4,5 screen is &H2600 and the end is &H3DFF.

But when I took out my disk controller I found that the screens had moved. I tried my program again and the screen was at &H600. This means that without my (TANDY 1.0) disc controller the screen for PMODE4,1 is at &H600, the end at &H1DFF and PMODE 4,5 starts at &H1E00 and ends at &H35FF.

Now I have that information, I can save the screen to disk. I ran a good graphic screen program and typed SAVEM "NAME", &HE00, &H25FF, &HE00 <ENTER>. (For non disk users, change &HE00 to &H600 and &HE00 to &H1DFF.)

Then I wrote a program that will load the screen:

```
<non disc > 10 PMODE4,1:SCREEN1,1:PCLS
<use CLOAD> 20 LOADM"SCREEN"
30 RUN"NAME"
```

It loaded the screen so quick that if you blink you would miss it! (I am not kidding!) (Cassette was a bit slower.)

From my work with the book I had learned how to move memory locations around.

I thought, "why can't I move the screen up?"

Well, I can! And here is the program:

```

00010 * PMODE SCREEN UP
00020 * BY
00030 * ANDREW BRUCE SIMPSON
00040 * 20/12/1985
7FF0 00100 ORG $7FF0
7FF0 8E 0E00 00110 START LDX #$0E00
7FF3 A6 84 00120 LOOP LDA ,X
7FF5 A7 88 E0 00130 STA -#20,X
7FF8 A6 80 00135 LDA ,X+
7FFA 8C 25FF 00140 CMPX #$25FF
7FFD 25 F4 00150 BLO LOOP
7FFF 39 00160 END RTS
0000 00170 END

```

00000 TOTAL ERRORS

```

END 7FFF
LOOP 7FF3
START 7FF0

```

Type it in and save it FIRST.

Before you LOADM you must CLEAR 16, &H7FF0. (16K'ers CLEAR 16, &H3FF0 and change line 100 from 7FF0 to 3FF0.)

For those of you who do not have EDTASM+ then the BASIC listing is:

```
2 ' PMODE SCREEN UP
```

```

4 ' BY
6 ' ANDREW BRUCE SIMPSON
8 ' 20/12/1985
10 CLEAR19,&H7FED
20 FORA=&H7FED TO &H7FFF
30 READ B:POKE A,B
40 NEXTA
50 DATA &H41 ,&H42 ,&H53 ,&H8E ,&H0E
, &H00 ,&HA6 ,&H84 ,&HA7 ,&H88 ,&HE0
, &HA6 ,&H80 ,&H8C ,&H25 ,&HFF ,&H25 ,
&HF4 ,&H39

```

Those without disk drives should change the M.L. listing line 110 to 0600 instead of 0E00 and line 140 to 1DFF and not 25FF.


In the BASIC listing change the DATA statements from &H0E, &H00 to &H06, &H00 and &H25, &HFF to &H1D, &HFF, line 10 from &H7FED to &H3FED.

Now to use it, all you have to do is EXEC&H7FF0 and for 16K'ers EXEC&H3FF0. The PMODE ?,1 screen will scroll up one place.

You can save the normal screen to disk or tape by using SAVEM "NAME", &H400, &H5FF, &H400 then just use the same program that I used to load the graphic screen except take out line 10.

But can I make the screen scroll down? See you next time!

**COCOCONF**  
**'86**



**August**  
**30-31st**

venue. **SEAGULLS**  
**- GOLD COAST**

price **\$ 39.95**

sat nite meal included

**BE THERE!**

**STOCKTAKING SPECIALS!**

While they last - send a copy of this ad to get this special!!

**TEAC FD55F DSDD 80 Track Drives**  
with head load solenoid to reduce head ware.

Normally \$246.41

**While they last** **\$199.95**

Printers:

CPA 80 A .....	\$439.33
CPB 80 P .....	\$482.42
CPB 136 .....	\$707.70

**CoCo Disk Drive - includes Power Supply, Case, Cables, 1.4 DOS, 40 Track DSDD.**  
Drive expandable to 2 drives ..... **\$399.00**

Phone for further details  
All prices include Tax  
Please allow for P & P. Bankcard Welcome.  
Dealer enquires welcome.

**energy CONTROL**

ENERGY CONTROL INTERNATIONAL PTY. LTD.  
P.O. Box 6507 Goodna Qld 4300  
Brisbane AUSTRALIA  
Phone (07) 288 2455 Telex AA43778 ENICON  
P.O. Box 12153 Wellington North NEW ZEALAND  
Phone 4 726 462 TLX NZ 30135

Prices subject to change without notice.

# The Straight, Hard Facts about Assembly Language

By William Barden, Jr.

**“W**ant to speed up your programs 300 times? Want to learn skills that will make you rich? Try Color Computer assembly language! To see if you have the aptitude, code this problem in BASIC and send us the result:  $2 + 2 = ?$ . If you pass this simple test, we'd like to enroll you in 'Famous Programmers' School!"

I closed the coding pad cover on which this advertisement was printed and sat back in my Realistic DC-5 desk chair, reflecting. That's the trouble with assembly language, it's misunderstood — too many myths abound about it. Maybe I can dispel some of those myths in this column. I'll give you the straight, hard facts about assembly language. If you're satisfied, you might be interested in dropping in from month to month and following this column.

## Fact Number 1: Assembly Language is Fast

Assembly language *is* fast! As you know, every microcomputer (indeed, every computer) has a built-in set of machine language instructions. Every program, whether it's written in the OS-9 C language or Extended Color BASIC, must ultimately be translated into sequences of machine language instructions. If you can write *directly* in machine language or its fraternal twin brother, assembly language, you are operating at peak speed on the Color Computer.

As an example of this blinding speed, consider the program in Figure 1. It's a "bubble sort" in Extended BASIC that sorts a "worst case" set of characters on the screen. The Extended BASIC program takes an estimated 4,800 seconds to do the sort. Now look at Figure 2, an assembly language equivalent. (The assembly language here has been converted into machine language and embedded into Extended BASIC DATA statements, which are then moved and

executed.) The assembly language version takes six seconds!

Expect to see increases in speed in assembly language from dozens to hundreds of times over "interpretive" BASIC and from three times to dozens of times over compilers such as BASIC09 and C.

Assembly language is the standard by which every other language is based. When programmers want to develop impressive code, they invariably pick assembly language to do the development. Sure, C and PASCAL might be used for some programs where speed is not extremely critical, but assembly language is always the choice when the absolute fastest speed is required.

## Fact Number 2: Assembly Language is Tedious to Code

This is the most detrimental thing about assembly language, and I don't want to downplay it. There's no ques-

tion that assembly language is a very tedious language to code. It may take 10 times longer to code a large program in assembly language than in BASIC. Is there any solution to this? Not really. In spite of "macro" assemblers, interactive editors, debug packages and books that promise to teach you assembly language in days, it remains tough to use.

One approach in using assembly language is to use it sparingly. Use it in short assembly language subroutines to speed up BASIC or other languages in those areas where speed is important. The bulk of the code can be the higher level BASIC, C or PASCAL. That way, you can have the best of both worlds: the programming ease of the higher level language and the speed of assembly language.

## Fact Number 3: Assembly Language is Tedious to Learn

One of the reasons assembly language columns are so popular in magazines and why assembly language books sell so well is that computer hobbyists are continually looking for magic approaches to learning it. There really are none.

Surprisingly, it's relatively easy to learn how the machine language instructions for a microcomputer work. They are so rudimentary that they're easy to comprehend. One instruction transfers a byte from memory into a register. Another adds two bytes. Another compares two values. It's not too hard to sit down and memorize the actions of about 60 instructions, as found in the Color Computer's 6809 microprocessor.

There's much more to assembly language than memorizing the actions of instructions, though. It consists more of learning programming *algorithms* and approaches to doing things — constructing tables of data, sorting lists, using linked lists, building subroutines to print lines, and so forth. Learning

Figure 1: Bubble Sort in BASIC

```

100 REM EXT BASIC BUBBLE SORT
110 REM FILL SCREEN WITH CHARS
120 CLS
130 FOR I=&H400 TO &H5FF
140 POKE I,RND(127)
150 NEXT I
160 REM ACTUAL SORT
170 BEND=&H5FF
180 I=&H400
190 SWAP=0
200 IF PEEK(I)<=PEEK(I+1)THEN 260
210 FIRST=PEEK(I)
220 SECOND=PEEK(I+1)
230 POKE I,SECOND
240 POKE I+1,FIRST
250 SWAP=1
260 I=I+1
270 IF I<>BEND THEN 200
280 BEND=BEND-1
290 IF BEND=&H3FF THEN 310
300 IF SWAP<>0 THEN 180
310 GOTO 310

```

Figure 2: Bubble Sort in Assembly Language

			00100 * BUBBLE SORT			
4C5B	8E	0400	00110 BUB010	LDX	#\$400	POINT TO START OF TEXT SCREEN
4C5E	108E	0000	00120	LDY	#0	SET SWAP FLAG TO 0
4C62	A6	80	00130 BUB020	LDA	,X+	GET ITH ENTRY, INCREMENT
4C64	A1	84	00140	CMPA	,X	COMPARE TO ITH+1
4C66	23	0A	00150	BLS	BUB030	GO IF LESS THAN OR EQUAL
4C68	E6	84	00160	LDB	,X	SWAP HERE - GET ITH
4C6A	E7	1F	00170	STB	-1,X	STORE IN ITH
4C6C	A7	84	00180	STA	,X	STORE FIRST IN ITH+1
4C6E	108E	0001	00190	LDY	#1	SET SWAP FLAG
4C72	8C	05FF	00200 BUB030	CMPX	#\$5FF	AT END?
4C75	26	EB	00210	BNE	BUB020	GO IF NO
4C77	108C	0000	00220	CMPY	#0	YAY, ANY SWAPS?
4C7B	26	DE	00230	BNE	BUB010	IF YES, TRY AGAIN
4C7D	39		00240	RTS		RETURN
		0000	00250	END		

00000 TOTAL ERRORS

```

100 REM BASIC/AL BUBBLE SORT
110 CLEAR 200,16127
120 CLS
130 DATA &H8E,&H04,&H00,&H10
140 DATA &H8E,&H00,&H00
150 DATA &HA6,&H80,&HA1,&H84
160 DATA &H23,&H0A,&HE6,&H84
170 DATA &HE7,&H1F,&HA7,&H84
180 DATA &H10,&H8E,&H00,&H01
190 DATA &H8C,&H05,&HFF,&H26
200 DATA &HEB,&H10,&H8C,&H00
210 DATA &H00,&H26,&HDE,&H39
211 FOR I=16128 TO 16128+34
212 READ A
213 POKE I,A
214 NEXT I
220 FOR I=&H400 TO &H5FF
230 POKE I,RND(127)
240 NEXT I
241 DEFUSR0=&H3F00
242 SR=USR0(0)
243 GOTO 243
    
```

assembly language, then, is more a situation of learning its structures, approach and philosophy, a kind of excursion into microprocessor Zen. However, this leads us directly into . . .

**Fact Number 4: Once You've Learned One Assembly Language, You Know Them All**

Once you've studied the philosophy of one assembly language and mastered the techniques, it becomes extremely easy to learn the instruction set of the next microcomputer. Assembly language for the Apple Macintosh's 68000 is very similar to the Color Computer's 6809. Assembly language on the Tandy 1000's 8088 microprocessor is really not that different than on the Color Computer. Once you've learned one assembly language, regardless which it is,

you're in good stead for the next, since you've mastered the art of using assembly language instructions to accomplish useful things.

If you're looking for the ultimate microprocessor from which to learn assembly language, look no further — 90 percent of what you learn on the Color Computer is directly applicable to any other system. Incidentally, the 6809 on the Color Computer is regarded by many to be as good or better than the 8088 on IBM compatibles. The instruction set of the 6809 is built along classical programming lines, while the 8088 has more idiosyncrasies.

**Are You Still with Me?**

If so, you're a hard person to discourage. You must be a student, confirmed hacker or masochist, or possibly all

three. Sigh . . . If you must learn assembly language, then we'll give you some tips on how to go about it.

**Which Assembler is Best?**

As you probably know, an *assembler* is a program that takes the *source code* of your assembly language program and translates it into *object code* or machine language. In the process, it provides a listing of the program and the resulting machine language, as shown in Figure 3.

I have mixed feelings about recommending an assembler to use. In the case of the Color Computer, the Radio Shack versions are not bad and fairly inexpensive. In addition, they have become a standard. For that reason, we'll refer primarily to Radio Shack products here. Those with other products will find, for the most part, that the code still applies.

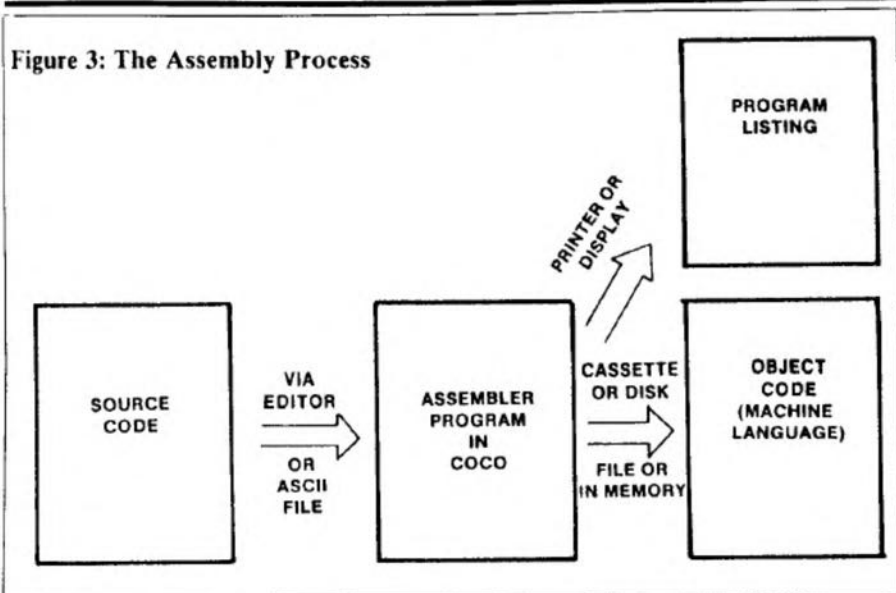
**Non-Disk System Users**

If you have a Color Computer without a disk, you'll probably want to get the Color Computer *EDTASM+* Assembler/Editor (Cat. No. 26-3250). This is a cassette-based system I like very much. The product contains three functions that would ordinarily be separate programs — the editor, assembler and debugger. Because the programs occupy memory at the same time, there's no loading from cassette between functions; you can simply switch from one to another with a single keystroke or two.

The editor does pretty much what Extended BASIC does in editing — characters on lines can be manipulated in different ways. The assembler, of course, translates the source code into object code and provides a listing. It also assembles into memory rather than creating an object file, although this can



Figure 3: The Assembly Process



also be done. Having the machine language code loaded directly into memory bypasses a cassette load of the object file and allows the debugger to be instantaneously called after assembly. The debugger (ZBUG) allows you to systematically debug the program by putting in stopping points (break-points), by stepping through instructions, by dumping selected areas of memory, and so forth.

#### Disk System Users without OS-9

If you're not an OS-9 user and run Disk BASIC, then Radio Shack provides a disk version of *EDTASM+* — *Color Disk EDTASM* (Cat. No. 26-3254). This version contains all of the commands of *EDTASM+* and a few more — it's essentially an upgrade of the cassette product. Using *Disk EDTASM*, you can save source and object files on diskette, a decided advantage with long programs.

#### Disk System Users with OS-9

I know Dale Puckett is going to kill me for this, but I have to say it: Learning assembly language with the OS-9 edi-

tor, assembler and debugger is much more of a task than using *Disk EDTASM*. The OS-9 program development tools are powerful, but more complex than the stand-alone *Disk EDTASM*, and you must pay more attention to the OS-9 environment in which you're operating.

However, if you are a confirmed OS-9 buff, it is certainly possible to learn OS-9 assembly language. The instructions and mnemonics for the 6809 are the same, as are many of the other commands provided for assembly. If you choose the OS-9 route, you'll learn more about how assembly language works in an operating system environment, if you can get past some of the frustration and complexities of OS-9 itself.

#### Other Assemblers

A non-Radio Shack product I like is the Micro Works Macro-80C Disk Assembler. I started using this product because there was no disk assembler from Radio Shack at the time. Although not as integrated as Radio Shack *EDTASM*, it is a nice, well-

thought out product.

#### Books and Tutorials

It will surprise some readers to find out I can't present an entire course on assembly language in the pages of *RAINBOW*. Think about it for a moment, though. In each column I have about 3,500 words to present my rambling discourses. That's 42,000 words a year, which might be enough for an introductory text on Color Computer assembly language. There's also the problem of new subscribers and just the overall length of time involved. It's difficult to take a semester course over a year or more. For that reason, I'll use the column to cover interesting points about assembly language programs, present some practical, short programs and, in general, act as a supplement to your own study.

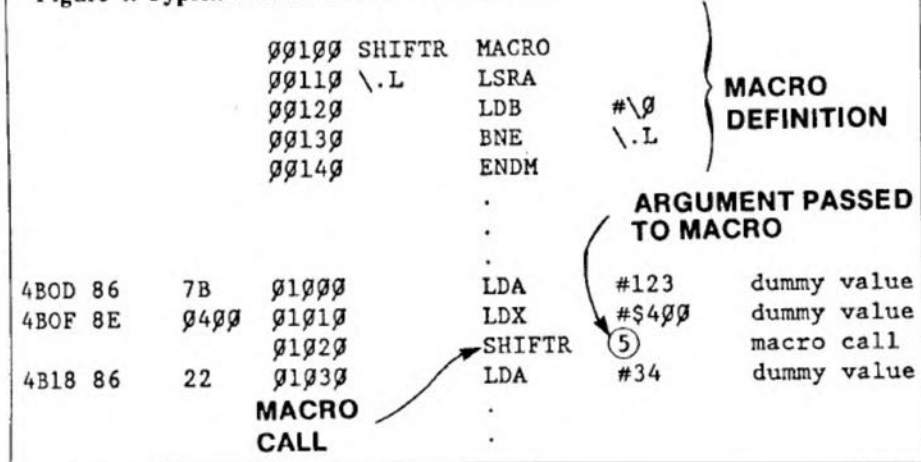
An obvious question you might have is, "What books can I use to learn assembly language?" It's embarrassing to recommend my own Radio Shack book, *Color Computer Assembly Language Programming* (Cat. No. 62-2077), but at \$6.95 you can't go too far wrong.

I honestly don't know of many books to recommend on the topic, other than the books on the 6809 microprocessor itself. Some of these do a good job of explaining the operation and use of 6809 instructions. A good one is Lance Leventhal's Osborne/McGraw Hill book *6809 Assembly Language Programming*. However, all of these generic books suffer from the same problem, though no fault of the author — they are not machine specific. They tell you about 6809 microprocessor instructions, but not how these instructions are used on any specific machine, such as the Color Computer. One must, if only for an absolute reference, is Motorola's *MC6809 Programming Manual*, available from Motorola Semiconductor Products, Inc., 3501 Ed Bluestein Blvd., Austin, TX 78721.

Another product is Dennis Kitz's Green Mountain Micro tutorial course called "Learning the 6809." It comes complete with audio tapes of 24 lessons and programs and a 224-page manual. This course is geared around *EDTASM+*, the cassette version of Radio Shack's editor/assembler/debugger, and is excellent. It covers not only the instruction set of the 6809, but also the structure and hardware of the Color Computer. About the only criticism that can be made of the course is its price of \$99, a little dear for many CoCoists.

Another product is the *Assembly*

Figure 4: Typical Macro Use in an Assembler



*Language Tutor* (26-3148, Page 46 of Computer Center Catalog RSC-15). The *Assembly Language Tutor* is a complete subset of an editor, assembler and debugger. It contains 30 lessons loadable from cassette, together with a large manual. The neat thing about the *Tutor* is that it contains an assembly language interpreter that runs your programs or lesson programs. As the interpreter oversees and controls things, it prevents you from making addressing errors, attempting to execute data rather than instructions and clobbering critical memory areas. It holds your hand, so to speak, to make assembly language learning a lot more palatable.

### 10 Most-Often Asked Questions about Assembly Language

In future columns I'll try to answer your questions about assembly language. If you have questions please drop me a line at RAINBOW, or leave a message on Delphi or the CoCo SIG of CompuServe. I'll try to answer the most common questions. For this first column, I'll answer 10 questions that keep popping up again and again.

#### Question 1: What are Macros and Why are They Used?

A "macro" is simply a sequence of instructions, ranging from one to hundreds. A typical macro might consist of four instructions, such as the ones shown in Figure 4. The instructions in this figure shift Register A right in a logical shift a specified number of positions. The macro is defined in a definition shown in the first part of the listing. Later, the macro can be "invoked" by writing down the macro name as shown in the SHIFTR mnemonic.

When the assembler sees the macro name in place of the usual instruction mnemonic, it searches a list of macro definitions, finds the instructions defined for the macro and automatically generates the instructions as if you had typed them in at that point. If the macro is invoked 10 times in a program, the same instructions are generated for each occurrence.

The advantage of the macro is that only one line of code can produce many lines automatically. Macros are a sort of "in-line" subroutine. In the example in Figure 4, the macro used one argument, but the line invoking the macro may also contain more arguments that are used within the macro body of code.

#### Question 2: What is Position-Independent Code and Why is it So Important?

Position-independent code is also

called relocatable code for some microprocessor instruction sets. Position-independent code is not at all important when instructions are assembled for a specific location in memory. Figure 5, for example, shows an LDA CONST1 instruction, which loads Register A in the 6809 with the contents of a memory location called CONST1, a constant. After assembly, the location of CONST1 is at location \$400A and the LDA address is \$4000. As long as the program is loaded in the \$4000 memory area, the LDA operates as it should, loading Register A with the contents of memory location \$400A.

Suppose the machine language code is moved to \$7000. The LDA should refer to a location (Hex A) 10 bytes away from the start of the program at \$700A. Instead, it refers to location \$400A! The LDA is not position-independent.

In the last part of the figure, the LDA CONST1 has been replaced with an LDA CONST1,PCR instruction. This instruction assembles without an absolute memory address — the address is computed from the current contents of the PC (Program Counter) register and an offset value in the instruction. The position-independent form of the LDA always loads the value 10 bytes away from the program start, and is position-independent.

It's important to have position-independent code in several cases. OS-9, for example, loads assembly language

code in different memory areas and much of the code must be position-independent. Even if you are not using OS-9, you cannot always guarantee that your program will be loaded in a specific memory area unless you take pains to do so. If your code has been converted to machine language DATA bytes and relocated to an array area in BASIC, for instance, you might not be able to know beforehand where that array area will be.

On the other hand, for simple programs outside of an OS-9 environment, you don't have to worry too much about position-independent code. Simply assemble your programs at a specific memory area and never move the machine language bytes anywhere else.

#### Question 3: What's the Proper Way to Write Assembly Language Programs?

Actually, there is no single way to solve a problem in assembly language code. There are usually many ways to write the assembly language code for a particular problem. Some ways might be more efficient than others, but assembly language is so fast that you can afford to be sloppy and still get the job done. At first, concentrate only on program design — using the right plan or algorithm to solve the problem. Later, as you become more experienced in assembly language, you can make your code more efficient and elegant.

Assembly language is a great deal less interactive than a higher level language

Figure 5: Position-Independent Code Example

#### Original Code: POINTS TO \$400A

4000		00100	ORG	\$4000	
4000 B6	400A	00110 START	LDA	CONST1	
4003 C6	10	00120	LDB	#\$10	constant
4005 8E	0400	00130	LDX	#\$400	screen start
4008 20	34	00140	BRA	NEXT	jump over CONST1
400A	FF	00150 CONST1	FCB	\$\$FF	constant

#### Relocated Code: STILL POINTS TO \$400A!

7000		00100	ORG	\$4000	
7000 B6	400A	00110 START	LDA	CONST1	
7003 C6	10	00120	LDB	#\$10	constant
7005 8E	0400	00130	LDX	#\$400	screen start
7008 20	34	00140	BRA	NEXT	jump over CONST1
700A	FF	00150 CONST1	FCB	\$\$FF	constant

#### Code with PCR: THIS DISPLACEMENT ADDED TO PC TO GET ADDRESS OF CONST1

4000		00100	ORG	\$4000	
4000 A6 8D 0007		00110 START	LDA	CONST1	

such as BASIC. You must carefully plan out the program design before even starting to code the problem. You should even consider "flow charting" the problem to get a clear idea of how to proceed. Breaking up a large problem into modules (subroutines) is also a good idea.

#### Question 4: What about I/O Operations in Assembly Language?

Input/output in Color Computer assembly language is best handled by using the BASIC "I/O drivers." The I/O drivers are assembly language code contained in BASIC ROM; they handle such I/O as text screen display, reading a character from the keyboard, reading and writing to tape and disk files, and other operations.

Some of the I/O calls are "documented," that is, defined in Radio Shack documentation. Other I/O calls are usable, but may change in subsequent versions of BASIC or new systems. Of course, you can write your own I/O drivers from scratch within your assembly language program, but it's less work to use standard drivers.

*Going Ahead with Extended Color BASIC* lists standard I/O drivers near the back of the book, and the Assembler manuals also reference I/O drivers. A typical driver is POLCAT (a little Texas humor there), the "Poll Keyboard for a Character" ROM subroutine, accessible by a call to location \$A000. If a key is being pressed, it will be returned in Register A with the 'Z' flag set.

#### Question 5: What are Condition Codes and What are They Used for?

The condition codes in the 6809 are a set of eight "flags." Although these flags are separate from each other, they are grouped together as the condition codes register to make them easier to handle. The main purpose of the flags is to record the results of arithmetic instructions.

In adding or subtracting two numbers, for example, it's handy to know whether the result of the operation is a negative number, zero or greater than zero. The condition codes record this information as part of the add or subtract instruction. The condition codes can be tested by Branch instructions. This sequence subtracts 12 from the contents of Register B and branches (a BASIC GOTO) to location NEXT1 if the result of the subtract is zero:

```
SUBB #12      subtract 12
BEQ  NEXT1    branch if result=0
```

Some instructions set the condition codes and some do not. All of the arithmetic instructions *do* set the condition codes so a Branch instruction can be used to alter the path of the program, if necessary.

#### Question 6: What are Interrupts?

Interrupts are used in computers to temporarily suspend execution of one program, called the "background" program, and to initiate a short new program called a "foreground" program. If the Color Computer is being used to monitor a nuclear reactor, for example, it might be beneficial to have the CoCo suspend printing paychecks and ring a bell when the coolant temperature reaches a critical point. An interrupt provides this ability.

There are two basic types of interrupts, maskable and non-maskable. A maskable interrupt can be enabled or

disabled under program control. A non-maskable interrupt is always active and cannot be disabled. The CoCo has both.

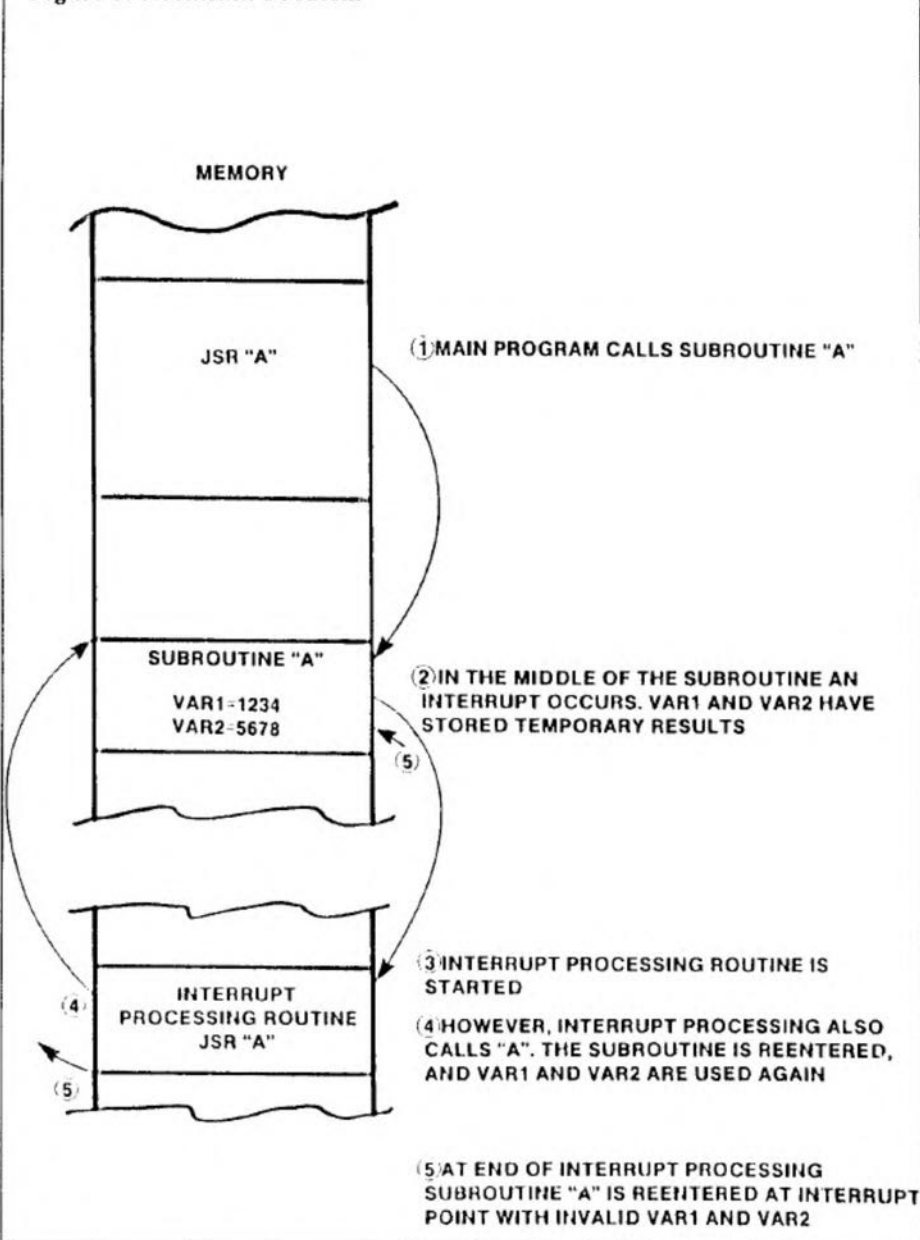
When an interrupt occurs and it is not catastrophic, the interrupt action is taken (such as ringing a bell for one second) and the interrupted program is then resumed. The interrupted program isn't aware that the interrupt occurred and goes blithely on its way, churning out the remainder of the paychecks. In a way, then, interrupts allow a type of "multitasking," where one task is a much higher priority than the other.

You may never use interrupts in your beginning assembly language programming and don't have to be aware of them in many short assembly language programs.

#### Question 7: What is Reentrant and Recursive Code?

Reentrant code relates to the inter-

Figure 6: Reentrant Problem



rupts just discussed. When a subroutine can be interrupted and is used by both the main program and an interrupt, it is said to be reentrant. Unless care is taken in the way variables are handled within the subroutine, it is possible to clobber the variables used by the main program when the interrupt code calls the subroutine again (see Figure 6). This problem is usually handled by not having a common subroutine for both the main program and interrupt procedure, or by using the stack to store temporary data.

Again, as in the case of Question 6, don't worry about reentrant code unless you are doing interrupt processing, and even then it often won't be a problem.

A recursive subroutine calls itself, possibly several times. Like reentrant code, special actions must be taken to save all levels of processing, usually in the stack.

#### Question 8: How Can I Use Assembly Language to Speed Up My Graphics?

Assembly language *can* be used to speed up graphics and with great success, but you may have more work than you bargained for. For one thing, to process graphics you'll need your own set of assembly language subroutines to handle graphics actions, such as drawing lines, creating shapes and implementing windows.

An alternative is to use some of the built-in graphics contained in ROM code. However, this latter course has

built-in dangers, as most of the ROM calls are not documented by Radio Shack. If they are used, be prepared to change addresses when the next version of BASIC or a new system appears.

Creating your own graphics subroutines may involve a great deal of work. The Extended BASIC CIRCLE command, for example, draws a circle by drawing a polygon of many sides. It takes some effort to implement such code in assembly language!

#### Question 9: How Can I Use Assembly Language for Sound?

Assembly language can be used for incredible sounds from the CoCo. Sound is produced in the CoCo by a digital-to-analog converter, which allows you to synthesize sounds by creating a wave form, as shown in Figure 7. The tones used for cassette tape, for example, are actually created from a sine wave table in ROM, a series of numeric values that are continually output to the D-to-A converter.

To use this feature of the CoCo, it's necessary to learn a little bit about the CoCo hardware interface. This is not too difficult, and we'll cover it in future columns. Assembly language, by the way, is fast enough to create sounds up to the frequency limits of the sound hardware used in the system, whereas BASIC is much too slow to produce custom sounds outside of the tones in the SOUND or PLAY commands.

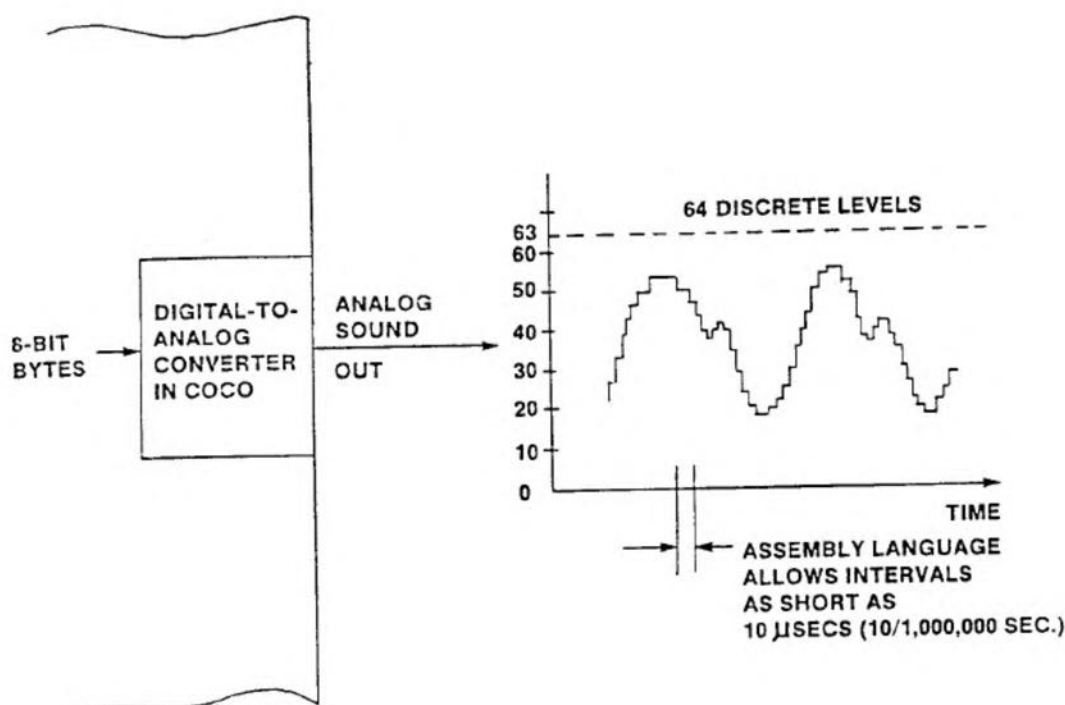
#### Question 10: How Can I Get a Listing of the BASIC Interpreter?

Microsoft, the author of CoCo BASIC, understandably doesn't pass out listings of the BASIC interpreter. However, several companies have produced listings of all versions of BASIC. These are not official listings, but have been compiled by "disassembling" BASIC to list the instructions used, figuring out what the instructions do, then adding their own comments. For the most part, these listings are very well-done and reveal such things as ROM subroutine calls and BASIC procedures. They're invaluable to anyone interested in the "internals" of BASIC and how functions and commands are implemented in assembly language.

Another alternative to discovering the secrets of BASIC is to disassemble it on your own. This can be done by using the ZBUG mnemonic mode in *EDTASM+* or *Disk EDTASM*. This mode displays the contents of memory as instruction mnemonics by converting the machine code numeric values into the appropriate instructions.

In future columns we'll talk in-depth about all of the topics mentioned here. Once again, if there are any topics you'd like discussed, write me at RAINBOW, contact me at Delphi or CompuServe, or simply write to P.O. Box 3568, Mission Viejo, CA 92692. See you next month. In the meantime, keep assembling! □

Figure 7: Digital-to-Analog Wave Forms



# What's the Diagnosis?

By Craig V. Bobbitt

**An** assembly language memory checker for the Color Computer, *Memory Diagnostic* has two modes:

1) Short — Every byte is tested to see that each of its bits can be cleared and set (compared to correct values in Register A), then the bytes adjacent to the target byte, which contain the complement of the target, are tested to make sure they don't follow the target (a common memory chip problem).

2) Long — All bit combinations are written into each memory location and checked against Register A.

The program is relocatable and jumps over itself during execution. It is heavily commented and should be fairly easy to follow. It has only been tested on a 64K machine, but it should work in any system configuration of the CoCo.

Lines 96-157 are the body of the short option.

Lines 159-177 are the body of the long option.

Lines 179-END display error messages.

The listing details how to force an error to see how that part of the program works. *Memory Diagnostic* is intended as a confidence check for the RAM-SAM portion of the Color Computer.

(If you have any questions regarding this program you may contact the author at P.O. Box 584, Greenville, TX 75401, phone 214-457-4476. Please include an SASE when writing.)

Lines 45-75 display the title and request memory size.

Lines 78-84 move BASIC to RAM if 64K is selected.

Lines 86-95 request long or short diagnostic.

(Craig Bobbitt lives in Greenville, Texas, and works on computer and peripheral hardware as a test engineer.)

## The listing: MEMDIAG

```

0001      NAM  MEMDIAG 2.2
0002      OPT  NOG
0003 *MEMORY DIAGNOSTIC FOR 64K TRS80 COLOR COMPUTER
0004 *20 APRIL 84
0005 *(C) BY CRAIG V. BOBBITT
0006 *P.O BOX 584
0007 *GREENVILLE, TX 75401
0008 *****
0009 * *
0010 * THIS PROGRAM RUNS A MEMORY TEST ON ALL *
0011 * LOCATIONS IN THE COLOR COMPUTER FROM *
0012 * 0000 TO END OF MEMORY. ANY DATA *
0013 * ERRORS ARE REPORTED TO *
0014 * THE SCREEN AND THE OPERATOR HAS THE OPTION *
0015 * OF ENDING THE TEST OR CONTINUING WITH *
0016 * THE NEXT LOCATION. *
0017 * THE SHORT TEST PUTS A PATTERN OF ALL ONES *
0018 * AND ALL ZEROS IN EACH LOCATION COMPARING *
0019 * IT VERIFIES BOTH OF THOSE VALUES. THE *
0020 * ADJACENT BYTES CONTAIN COMPLEMENTED DATA *
0021 * AND THEY ARE CHECKED TO INSURE THEY DON'T *
0022 * FOLLOW THE TARGET BYTE. *
0023 * THE LONG TEST WRITES ALL COMBINATIONS IN *
0024 * EACH BYTE OF MEMORY VERIFYING THAT THEY ARE *
0025 * READ BACK CORRECTLY. EXECUTION TIME FOR *
0026 * THE SHORT TEST IS ABOUT 12 SECONDS, THE *
0027 * LONG TEST TAKES ABOUT 6 MINUTES *
0028 * ALL LOCATIONS ARE RESTORED TO THEIR *
0029 * ORIGINAL VALUES *
0030 * THE PROGRAM JUMPS AROUND ITSELF. *
0031 * TO FORCE AN ERROR, LOAD PROGRAM AT $3000 *
0032 * (DEFAULT) AND EXEC &H3066 SELECT SHORT OPT *
0033 * *
0034 *****

```

```

0035
0036
0037
0038 *PROGRAM IS RELOCATABLE
3000 0039 ORG $3000
A1C1 0040 GETCHR EQU $A1C1 KEYBOARD INPUT
A002 0041 CHROUT EQU $A002 SCREEN OUTPUT
0042
3000 1A50 0043 START ORCC #$50 TURN OFF INTERRUPTS
0044
3002 BDA928 0045 .MSIZE JSR $A928 CLEAR SCREEN
3005 318D0203 0046 LEAY MSG1,PCR FIND THE MESSAGE
3009 170141 0047 LBSR DISPLA WRITE IT
300C BDA1C1 0048 GETSIZ JSR GETCHR LOOK FOR A CHARACTER
300F 27FB 0049 BEQ GETSIZ WAIT FOR KEYPRESS
3011 8131 0050 CMPA #'1 /16K/
3013 2710 0051 BEQ .16K
3015 8132 0052 CMPA #'2 /32K/
3017 271C 0053 BEQ .32K
3019 8133 0054 CMPA #'3 /64K/
301B 2728 0055 BEQ .64K
301D 8134 0056 CMPA #'4 /ABORT/
301F 102702B5 0057 LBEQ DONE
3023 20E7 0058 BRA GETSIZE ILLEGAL OPTION
0059
3025 8E3FFF 0060 .16K LDX #$3FFF TOP OF RAM FOR 16K
3028 AF8D02C5 0061 STX MEMEND,PCR SAVE IT
302C 8E3136 0062 LDX #"16
302F AF8D0136 0063 STX HEADER+23,PCR PUT 16 IN OUTPUT STRING
3033 2031 0064 BRA WRTHDR CONTINUE
0065
3035 8E7FFF 0066 .32K LDX #$7FFF TOP OF RAM FOR 32K
3038 AF8D02B5 0067 STX MEMEND,PCR SAVE IT
303C 8E3332 0068 LDX #"32
303F AF8D0126 0069 STX HEADER+23,PCR
3043 2021 0070 BRA WRTHDR
0071
3045 8EFEFF 0072 .64K LDX #$FEFF TOP OF RAM FOR 64K
3048 AF8D02A5 0073 STX MEMEND,PCR SAVE IT
304C 8E3634 0074 LDX #"64
304F AF8D0116 0075 STX HEADER+23,PCR
0076
0077 *GO TO 64K MODE
3053 8E8000 0078 LDX #$8000 START OF ROM
3056 B7FFDE 0079 MOVE STA $FFDE SWITCH PAGE
3059 A680 0080 LDA ,X+ GET BYTE FROM ROM
305B B7FFDF 0081 STA $FFDF SWITCH PAGE BACK
305E A71F 0082 STA -1,X STORE IN RAM
3060 AC8D028D 0083 CMPX MEMEND,PCR END OF ROM
3064 25F0 0084 BLO MOVE
0085
3066 BDA928 0086 WRTHDR JSR $A928 CLEAR SCREEN
3069 318D00E5 0087 LEAY HEADER,PCR GET ADX OF HEADER
306D 1700DD 0088 LBSR DISPLA WRITE IT
3070 BDA1C1 0089 INPUT JSR GETCHR GO GET A CHARACTER
3073 27FB 0090 BEQ INPUT WAIT FOR IT
3075 AD9FA002 0091 JSR [CHROUT] ECHO CHARACTER
3079 814C 0092 CMPA #'L LONG DIAGNOSTIC?
307B 1027009E 0093 LBEQ LONG YES
307F 8153 0094 CMPA #'S SHORT?
3081 26E3 0095 BNE WRTHDR BAD INPUT RETURN
3083 8EFFFF 0096 LDX #$FFFF FIRST LOCATION -1
3086 318DFF76 0097 LEAY START,PCR
308A 10AF8D0260 0098 STY TEMP,PCR SAVE START OF PROGRAM
308F 3001 0099 SLOOP LEAX 1,X
3091 E684 0100 LDB ,X SAVE THE BYTE
3093 E78D025D 0101 STB TARGET,PCR
3097 E61F 0102 LDB -1,X GET LOWER ADJACENT BYTE
3099 E78D0258 0103 STB LOWER,PCR STORE IT AWAY
309D E601 0104 LDB 1,X GET UPPER ADJACENT BYTE
309F E78D0250 0105 STB UPPER,PCR STORE IT

```

30A3 C6FF	0106	LDB	#\$FF	COMPLEMENT OF TARGET BYTE
30A5 E71F	0107	STB	-1,X	SET ADJ BYTES TO COMPLEMENT
30A7 E701	0108	STB	1,X	
30A9 6F84	0109	CLR	,X	CLEAR CURRENT LOCATION
30AB 4F	0110	CLRA		ZERO TO START
30AC A184	0111	CMPA	,X	ARE THEY EQUAL??
30AE 2703	0112	BEQ	SCONT	YES GO AHEAD
30B0 1701BB	0113	LBSR	ERROR	ERROR IF NOT EQUAL
	0114	* DON'T DO ADJACENT BYTES IF AT START		
	0115	* OR END OF MEMORY		
30B3 8C0000	0116	SCONT	CMPX #0	START OF MEMORY?
30B6 2714	0117	BEQ	SCONT2	YES DON'T DO ADJ BYTES
30B8 AC8D0235	0118	CMPX	MEMEND,PCR	END?
30BC 270E	0119	BEQ	SCONT2	YES DON'T CHECK ADJ BYTES
	0120	*HAVE ADJACENT BYTES CHANGED?		
30BE E101	0121	CMPB	1,X	UPPER ADJACENT
30C0 2703	0122	BEQ	SCONT1	
30C2 1701A9	0123	LBSR	ERROR	REPORT ERROR
30C5 E11F	0124	SCONT1	CMPB -1,X	LOWER ADJACENT
30C7 2703	0125	BEQ	SCONT2	
30C9 1701A2	0126	LBSR	ERROR	
30CC 43	0127	SCONT2	COMA	COMPLEMENT THE GOOD VALUE
30CD 53	0128	COMB		COMPLEMENT ADJ BYTES
30CE 6301	0129	COM	1,X	
30D0 631F	0130	COM	-1,X	
30D2 6384	0131	COM	,X	
30D4 A184	0132	CMPA	,X	ALL BITS SET??
30D6 2703	0133	BEQ	SCONT3	YES GO AHEAD
30D8 170193	0134	LBSR	ERROR	
30DB 8C0000	0135	SCONT3	CMPX #0	FIRST LOCATION?
30DE 2714	0136	BEQ	SCONT5	
30E0 AC8D020D	0137	CMPX	MEMEND,PCR	
30E4 270E	0138	BEQ	SCONT5	
30E6 E11F	0139	CMPB	-1,X	B SHOULD - ADJ BYTES
30E8 2703	0140	BEQ	SCONT4	
30EA 170181	0141	LBSR	ERROR	
30ED E101	0142	SCONT4	CMPB 1,X	
30EF 2703	0143	BEQ	SCONT5	
30F1 17017A	0144	LBSR	ERROR	
	0145	*RESTORE ALL BYTES		
30F4 E68D01FB	0146	SCONT5	LDB UPPER,PCR	
30F8 E701	0147	STB	1,X	
30FA E68D01F6	0148	LDB	TARGET,PCR	
30FE E784	0149	STB	,X	
3100 E68D01F1	0150	LDB	LOWER,PCR	
3104 E71F	0151	STB	-1,X	
3106 AC8D01E7	0152	CMPX	MEMEND,PCR	HAS ALL MEMORY BEEN TESTED
310A 102701CA	0153	LBEQ	DONE	IF ALL MEMORY TESTED
310E AC8D01DD	0154	CMPX	TEMP,PCR	ARE WE IN THE PROGRAM AREA
3112 1026FF79	0155	LBNE	SLOOP	NO KEEP GOING
3116 308D01DD	0156	LEAX	EXIT,PCR	FIND PROGRAM END
311A 16FF72	0157	LBRA	SLOOP	GO TO IT
	0158			
311D 8E0000	0159	LONG	LDX #0	FIRST LOCATION
	0160			
3120 E684	0161	LLOOP	LDB ,X	SAVE BYTE IN B
3122 6F84	0162	CLR	,X	CLEAR TARGET BYTE
3124 4F	0163	CLRA		CLEAR COMPARATOR
3125 A184	0164	LCONT	CMPA ,X	COMPARE
3127 2703	0165	BEQ	LCONT1	IF EQUAL NO ERROR
3129 170142	0166	LBSR	ERROR	REPORT THE ERROR
312C 6C84	0167	LCONT1	INC ,X	INCREMENT TARGET ADX
312E 4C	0168	INCA		INCREMENT COMPARATOR
312F 8100	0169	CMPA	#0	ALL COMBINATIONS TESTED?
3131 26F2	0170	BNE	LCONT	NO DO IT AGAIN
3133 E780	0171	STB	,X+	RESTORE BYTE AND POINT TO NEXT ONE
3135 AC8D01B8	0172	CMPX	MEMEND,PCR	ARE WE FINISHED
3139 1027019B	0173	LBEQ	DONE	YES
313D AC8D01AE	0174	CMPX	TEMP,PCR	PROGRAM AREA?
3141 26DD	0175	BNE	LLOOP	DO ANOTHER LOCATION
3143 308D01B0	0176	LEAX	EXIT,PCR	GO AROUND PROGRAM AREA

```

3147 20D7      0177      BRA   LOOP      GO DO IT AGAIN
                0178
3149 AD9FA002  0179 DISPL JSR   [CHROUT] SEND A CHAR TO SCREEN
314D A6A0      0180 DISPL LDA   ,Y+    GET A CHAR
314F 26F8      0181      BNE   DISPL    PRINT IT
3151 39        0182      RTS          RETURN
                0183
                0184 *SCREEN MESSAGES X'S WILL BE FILLED IN WITH
                0185 * CORRECT VALUES IN ERROR ROUTINE
                0186
3152 20        0187 HEADER FCS /      MEMORY DIAGNOSTIC 64K      <0D>SHORT OR LONG?/
3182 0D        0188 ERMSG FCS /<0D>DATA ERROR:<0D>ADDRESS=XXXX<0D>EXPECTED DATA=XX<0D>ACTUAL
DATA=XX/
31BC 0D        0189 CONTN FCS /<0D>WANT TO CONTINUE? (Y OR N)/
31D8 0D        0190 LAST FCS /<0D>DIAGNOSTIC COMPLETE<0D><0D>***PRESS ANY KEY TO CONTINUE**/
320C 43        0191 MSG1 FCS /COLOR COMPUTER MEMORY DIAGNOSTIC<0D> !ENTER MEMORY SIZE (1-4
)<0D><0D> 1) 16K<0D> 2) 32K<0D> 3) 64K<0D> 4)ABORT/
                0192
326E 3436      0193 ERROR PSHS D,X,Y    SAVE REGISTERS
3270 AF8D0078  0194      STX   LOCTN,PCR  SAVE LOCATION
3274 A78D0076  0195      STA   AREG,PCR  SAVE GOOD VALUE
3278 318DFF06  0196      LEAY  ERMSG,PCR  FIND ERROR MSG
327C 31A815    0197      LEAY  21,Y      POINT TO OUTPUT BUFFER
327F 308D0069  0198      LEAX  LOCTN,PCR  CONVERT THE LOCATION TO ASCII
3283 8D2E      0199      BSR   CONVRT
3285 3121      0200      LEAY  1,Y      NEXT OUTPUT LOC
3287 8D2A      0201      BSR   CONVRT   CONVERT IT
3289 31A810    0202      LEAY  16,Y     OUTPUT LOCATION DATA EXPECTED
328C 8D25      0203      BSR   CONVRT   CONVERT IT
328E AE62      0204      LDX   2,S      GET OLD X VALUE
3290 312E      0205      LEAY  14,Y     LOCATION FOR ACTUAL VALUE
3292 8D1F      0206      BSR   CONVRT   CONVERT
3294 318DFEEA  0207      LEAY  ERMSG,PCR  FIND ERROR MSG
3298 17FEB2    0208      LBSR  DISPLA   WRITE IT TO SCREEN
329B 318DFF1D  0209      LEAY  CONTN,PCR  FIND CONTINUE MESSAGE
329F 17FEAB    0210      LBSR  DISPLA   WRITE IT
32A2 176F1C    0211 ANSWER LBSR  GETCHR
32A5 27FB      0212      BEQ   ANSWER   KEEP LOOKING
32A7 AD9FA002  0213      JSR   [CHROUT] ECHO ANSWER
32AB 8159      0214      CMPA  #'Y      IS IT A YES?
32AD 2702      0215      BEQ   GOBACK   YES GO BACK WHERE YOU CAME FROM
32AF 2027      0216      BRA   DONE     RETURN
32B1 35B6      0217 GOBACK PULS D,X,Y,PC
                0218
32B3 A684      0219 CONVRT LDA   ,X      GET FIRST BYTE
32B5 84F0      0220      ANDA  #$F0     GET LEFT 4 BITS
0004          0221      RPT   4
                0222
                0223      ENDR
32B7 44        +      LSRA
32B8 44        +      LSRA
32B9 44        +      LSRA
32BA 44        +      LSRA
32BB 8109      0224      CMPA  #9      IS THIS A NUMBER
32BD 2E04      0225      BGT   LETR    NO ITS A LETTER
32BF 8B30      0226      ADDA  #$30
32C1 2002      0227      BRA   CONCNT
32C3 8B37      0228 LETR  ADDA  #$37    CHANGE IT TO ASCII
32C5 A7A0      0229 CONCNT STA   ,Y+    PUT IT IN OUTPUT STRING
32C7 A680      0230      LDA   ,X+    GET IT AGAIN
32C9 840F      0231      ANDA  #$0F    GET RIGHT BITS
32CB 8109      0232      CMPA  #9
32CD 2E04      0233      BGT   LETR1
32CF 8B30      0234      ADDA  #$30
32D1 2002      0235      BRA   CNCNT
32D3 8B37      0236 LETR1 ADDA  #$37
32D5 A7A4      0237 CNCNT STA   ,Y
32D7 39        0238      RTS
                0239
32D8 318DFEFC  0240 DONE  LEAY  LAST,PCR
32DC 17FE6E    0241      LBSR  DISPLA

```



```

32DF BDA1C1      0242 DONE1 JSR  GETCHR      IS THERE A KEY PRESSED
32E2 27FB       0243      BEQ  DONE1
32E4 1CA0       0244      ANDCC #A0      RESTORE INTERRUPTS
32E6 B7FFDE     0245      STA  $FFDE     TURN OFF 64K MODE
32E9 7EA027     0246      JMP  $A027     GO TO BASIC RESET ROUTINE
32EC           0247 LOCTN RMB  2
32EE           0248 AREG RMB  1
32EF           0249 TEMP RMB  2
32F1           0250 MEMEND RMB 2
32F3           0251 UPPER RMB 1
32F4           0252 TARGET RMB 1
32F5           0253 LOWER RMB 1
32F6           0254      RMB  1
32F7 12        0255 EXIT  NOP
3000          0256      END  START
NO ERROR(S) DETECTED

```

SYMBOL TABLE:

```

.16K 3025      .32K 3035      .64K 3045      .MSIZE 3002
ANSWER 32A2    AREG 32EE      CHROUT A002    CNCNT 32D5
CONCNT 32C5    CONTN 31BC     CONVRT 32B3    DISPL1 3149
DISPLA 314D    DONE 32D8     DONE1 32DF     ERMSG 3182
ERROR 326E     EXIT 32F7     GETCHR A1C1    GETSIZ 300C
GOBACK 32B1    HEADER 3152   INPUT 3070     LAST 31D8
LCONT 3125     LCONT1 312C   LETR 32C3     LETR1 32D3
LLOOP 3120     LOCTN 32EC    LONG 311D     LOWER 32F5
MEMEND 32F1    MOVE 3056     MSG1 320C     NARG 0000
SCONT 30B3     SCONT1 30C5   SCONT2 30CC    SCONT3 30DB
SCONT4 30ED    SCONT5 30F4   SLOOP 308F    START 3000
TARGET 32F4    TEMP 32EF     UPPER 32F3    WRTHDR 3066

```

CMD-MEMDIAG.TXT >/P

## CoCo MERGE

continued from Page 31

```

00100      ORG 32500
00110 START LEAX MERGE,PCR      PUT MERGE ADDR INTO
00120      STX $168              JMP TABLE
00130      CLR SW,PCR          SET SWITCH FOR MERGE
00140      RTS
00150 STR  RMB 2              START POINTER STORAGE
00160 SW   RMB 1              SWITCH STORAGE
00170 PMG  FCC /MERGE INITIATED/
00180      FCB $0D
00190 PRT  FCC /MERGE COMPLETED/
00200      FCB $0D
00210 MERGE CMPA #5C          LOOK FOR CONTROL CHR
00220      BNE EXIT            EXIT IF NOT CONTROL
00230      TST SW,PCR         CHECK SWITCH SETTING
00240      BNE RST
00250      LDX $19            GET START ADDR
00260      STX >STR,PCR       STORE IN STR
00270      LDX $1B            GET END ADDR
00280      LEAX -2,X          OVERWRITE END FLAG
00290      STX $19
00300      LEAY PMG,PCR
00310      COM SW,PCR        TOGGLE SWITCH
00320      BSR PRINT        GO TO SCRIN PRINT SUB
00330      LDX #ABEF
00340      JMP $AC79        RETURN TO BASIC
00350 RST  LDX >STR,PCR     GET START ADDR
00360      STX $19          PUT IN POINTER
00370      LEAY PRT,PCR    GET PROMPT ADDR
00380      CLR SW,PCR     TOGGLE SWITCH
00390      BSR PRINT        GO TO SCRIN PRINT SUB
00400      LDX #ABEF
00410      JMP $AC79        RETURN TO BASIC
00420 EXIT  JMP $8273      NORMAL RETURN
00430 PRINT LDB #10        # OF CHRS IN PROMPT
00440 PR1  LDA ,Y+        GET CHR
00450      JSR $A282        PRINT CHR ON SCREEN
00460      DECB
00470      BNE PR1
00480      RTS
00490      END

```

Alleviate your  
hand-assembling problems!

# Defeat de Bugs

By Mike Dean

**D**o you have a lot of assembly language listings, but do not have an editor/assembler? Are you hoping to purchase an editor/assembler in the future, but won't have the extra money to do so for some time? If you answered "yes" to either of these questions, then read on.

*Debug* is a utility that can display 30 bytes and their corresponding characters on the screen at any address. It can also make hard copy printouts in listing form, so if any of your computer pals want a listing, the program can easily generate one. *Debug* is essentially an address modifier. Anyone who has worked with the debug program on the Model III should be familiar with this version.

*Debug* requires a 16K Extended Color BASIC computer. No prior loading instructions are required unless the machine code you're going to type in requires them. To make sure the machine code starts after your BASIC program, type in the command PRINT PEEK(27)\*256+PEEK(28); this returns the end address for the program. If the address is where the code will be placed,

I suggest using the PCLEAR command to move the program back in memory.

The following keys are used in the program.

Key	Operation
N	Restarts program for a *N*ew address
J	*J*ump to specified address
:	Go to the next page of addresses
-	Go to the previous page of addresses
S	*S*aves machine code with the CSAVEM command
L	*L*oads machine code into memory with the CLOADM command
P	*P*rints a listing of machine code to the screen or printer
A-F 0-9	Hexadecimal numbers used when modifying addresses
Arrow keys	Move cursor in modification mode

When you have finished typing in *Debug*, save it since a possibility exists that the code you may JUMP to might not let you return to *Debug*. After saving *Debug*, type RUN and press ENTER. The credits appear and the program asks the address at which you would like to start. You must respond in hexadecimal.

After pressing ENTER, the address you requested will be in the upper left-hand corner of the screen. The contents of each address are displayed with each corresponding address. Press the semicolon (;) key and the next page of addresses will appear on the screen. Press the minus (-) key and the address

#### Sample Run

```
8000: 45 58 8E 80 DE CE
8006: 01 2A C6 0A BD A5
800C: 9A 8E B2 77 AF 43
8012: AF 48 8E 89 4C BF
8018: 01 0D 9E 8A BF 01
801E: 12 BD 82 9C CC 2C
8024: 05 DD E6 8E 01 3E
802A: 9F B0 CE B4 4A C6
8030: 0A EF 81 5A 26 FB
8036: 86 7E B7 01 9A 8E
```

typed in at the start of the program will be the address in the upper left corner.

Find an assembly listing in an issue of THE RAINBOW and type in the Hex numbers only. Once you reach the end of the page, a tone will sound. This indicates that you can either examine your typing accuracy or go to the next page. When you are finished typing in the code, press the 'S' key. Type the start address at the first prompt, press ENTER and type the ending address. Next, press ENTER and type the execution address. The program then asks for the filename. Type an appropriate name and press ENTER; the machine code will then be saved. The program goes back to the starting address you specified and

displays the memory contents. To test your program save, press the 'L' key. Type in the filename or press ENTER for the next file. The file will be loaded and you will go back to the starting address of the file just loaded.

For a better chance of finding typing errors, I have incorporated a print feature for those with printers. All you have to do is press the 'P' key and let the printer do the rest. The printout duplicates the format used on the screen. This makes it easy to type in from a Debug-generated listing.

To experiment with Debug, load it and type in Listing 2 — just type Hex numbers on the keyboard. The graphics that are produced should be easier to

test if you have a Hex to binary chart.

Debug has alleviated my hand-assembly problems. Once you get used to the format, listings can be entered in a matter of minutes. I have used Debug to type in the first two installments of Screen 51 by Chris Bone and R. Bartly Betts. (These installments can be found in the December 1984 and January 1985 issues of RAINBOW.) Use Debug as much as possible. You'll be amazed at the accuracy compared to typing DATA statements.

(Questions about Debug may be sent to Mike at R.R. 1, Box 117, Knoxville, IL 61448, phone 309-289-6987. Please include an SASE for a reply when writing.)

20 ..... 195  
38 ..... 199  
53 ..... 19  
END ..... 235

### The listing: DEBUG

```

1 CLS:PRINT@197,"MACHINE LANGUAGE
  DEBUG";
2 PRINT@233,"VERSION @2.@2";
3 PRINT@259,"BY MIKE DEAN JANUARY
  1985";
4 FORSW=@TO15@:NEXT
5 CLS:LINEINPUT"ADDRESS (HEX)";
  AS:LO=VAL("&H"+AS):CLS:IFLO>6544
6 THENLO=65446
7 ZV=LO:FORDP=@TO448STEP32:ZQS=HEX$
  (ZV)
8 IFLEN(ZQS)<4 THENZQS="@"+ZQS:GOTO
  7
9 PRINT@DP,ZQS";";
10 ZV=ZV+6:NEXT:ZV=LOC
11 FORDP=6TO454STEP32
12 ZQS=HEX$(PEEK(ZV))
13 IFLEN(ZQS)<2 THENZQS="@"+ZQS
14 PRINT@QP,ZQS";ZV=ZV+1:NEXTQP,
  DP:ZV=LOC
15 ZV=LOC:FORDP=1@5@TO1498STEP32
  :FORDP=@TO5
16 POKEQP+DP,PEEK(ZV):ZV=ZV+1:NE
  XTQP,DP
17 AP=@:BP=6:PO=LO:PX=1@5@
18 PRINT@ (AP+BP),STRING$(2,2@7);
19 AS=INKEY$
20 IFA$="N"THENRUN
  
```

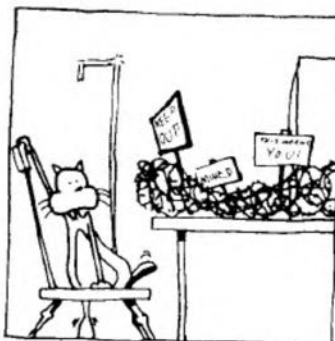
```

21 IFA$="";THENLO=LO+9@:IFLO>654
  46 THENLO=65446:GOTO6ELSE6
22 IFA$="-"THENLO=LO-9@:IFLO<@ETH
  ENLO=@:GOTO6ELSE6
23 IFA$<>CHR$(9)THEN26
24 GOSUB43:PRINT@ (AP+BP),ZQS;:BP
  =BP+3:IFBP>22 THENBP=6:AP=AP+32:P
  O=PO+1:PX=PX+27:GOTO44
25 PO=PO+1:PX=PX+1:GOTO18
26 IFA$<>CHR$(8)THEN29
27 GOSUB43:PRINT@ (AP+BP),ZQS;:BP
  =BP-3:IFBP<6 THENBP=21:AP=AP-32:P
  O=PO-1:PX=PX-27:GOTO45
28 PO=PO-1:PX=PX-1:GOTO18
29 IFA$<>CHR$(1@)THEN32
30 GOSUB43:PRINT@ (AP+BP),ZQS;:AP
  =AP+32:IFAP>448 THENAP=448:GOTO18
31 PO=PO+6:PX=PX+32:GOTO18
32 IFA$<>"^"THEN35
33 GOSUB43:PRINT@ (AP+BP),ZQS;:AP
  =AP-32:IFAP<@ THENAP=@:GOTO18
34 PO=PO-6:PX=PX-32:GOTO18
35 IFA$="L"ORAS="S"THEN46ELSEIFA
  $="J"THEN59ELSEIFA$="P"THEN6@
36 IFA$>"@"ANDAS<"9"ORAS=>"A"
  ANDAS<"F"THENPRINT@ (AP+BP),AS;:B
  P=BP+1:GOTO38
37 GOSUB43:PRINT@ (AP+BP),ZQS;:GO
  TO18
38 BS=INKEY$:IFBS$=""THEN38
39 IFBS$>"@"ANDBS$<"9"ORBS$=>"A"
  ANDBS$<"F"THENPRINT@ (AP+BP),BS;:E
  LSE38
40 POKEPO,VAL("&H"+(AS+BS)):POKE
  PX,VAL("&H"+(AS+BS)):GOSUB43:PRI
  NT@ (AP+BP-1),ZQS;:POKEPX,VAL("&H
  "+ZQS)
41 BP=BP+2:IFBP>22 THENBP=6:AP=AP
  +32:PX=PX+27:PO=PO+1:IFAP>448THE
  NSOUND128,3:GOTO44ELSE44
42 PO=PO+1:PX=PX+1:GOTO18
43 ZQS=HEX$(PEEK(PO)):IFLEN(ZQS)
  =1 THENZQS="@"+ZQS:RETURNELSERETU
  RN
44 IFAP>448 THENAP=448:BP=21:PO=L
  O+89:GOTO18ELSE18
45 IFAP<@ THENAP=@:BP=6:PO=LO:GOT
  O18ELSE18
  
```

```

46 CLS:IFPEEK(49152)=68 THENDV=1E
  LSE DV=-1
47 IFA$="L"THEN56
48 PRINT"RESPOND IN HEXADECI
  MAL"
49 LINEINPUT"START ADDRESS:";S@:
  S=VAL("&H"+S@)
50 LINEINPUT"ENDING ADDRESS:";E@
  :E=VAL("&H"+E@)
51 LINEINPUT"EXECUTION ADDRESS:"
  ;D@:D=VAL("&H"+D@)
52 LINEINPUT"FILENAME:";F@
53 IFDV=1 THEN55
54 PRINT"PRESS PLAY&RECORD.":PRI
  NT"PRESS ANY KEY WHEN READY":EXE
  C@HAL71:CSAVEMF$,S,E,D:LO=S:CLS:
  GOTO6
55 SAVEM F$,S,E,D:LO=S:CLS:GOTO6
56 IFDV=-1 THEN58
57 LINEINPUT"FILENAME:";F@:LOADM
  F@:LO=PEEK(487)*256+PEEK(488):C
  LS:GOTO6
58 PRINT"PRESS PLAY AND ENTER TH
  E","FILENAME:";:LINEINPUTF@:CLO
  ADM F@:LO=PEEK(487)*256+PEEK(488)
  :CLS:GOTO6
59 CLS:LINEINPUT"ADDRESS (HEX):"
  ;AS:EXEC VAL("&H"+AS):GOTO5
60 CLS:LINEINPUT"START ADDRESS:"
  ;S@:S=VAL("&H"+S@)
61 LINEINPUT"END ADDRESS:";E@:EN
  VAL("&H"+E@):LINEINPUT"SCREEN OR
  PRINTER?";DV$:IFDV$="P"THEN DV
  =-2 ELSE DV=@
62 IF(EN-ST)/6=INT((EN-ST)/6)THE
  N@63 ELSE EN=EN+1:GOTO62
63 FORDP=ST TO EN STEP6:ZQS=HEX$(
  DP)
64 IFLEN(ZQS)<4 THENZQS="@"+ZQS:G
  OTO64
65 PRINT#DV,ZQS";";
66 FORDD=DP TODP+5
67 ZQS=HEX$(PEEK(DD)):IFLEN(ZQS)
  <2 THENZQS="@"+ZQS
68 PRINT#DV," "+ZQS;
69 NEXTDD:PRINT#DV,"":NEXTDP
70 PRINT"PRESS A KEY TO CONTINUE
  ":EXEC@HAL71:LO=ST:CLS:GOTO6
  
```

### CoCo Cat



# FORTH FORUM

by John Poxon

This article is the first in a series of nine for 1986 dealing with the FORTH language. The emphasis throughout will be on simple explanations of aspects of FORTH and elementary routines which you may find useful for your FORTH programmes. More specifically, the topics for each month will (hopefully) be as follows:-

- March - A brief introduction to FORTH:  
A\*FORTH: Reverse Polish Notation.
- April - More about Reverse Polish Notation (RPN): using the STACK.
- May - Colon definitions: the A\*FORTH editor: CONSTANTS VARIABLES and BASEs.
- June - Comparisons and decisions. LOOPS
- July - More about LOOPS: Some interesting FORTH words.
- August - Precision: Single and Double length words.
- September- String handling.
- October - Data logging and control of external devices.
- November - To be decided.

FORTH is unlike any other language that you've probably met so far. Doubtless you've been used to a fixed syntax, and either interpretation or compilation of your code to make a program "RUN".

FORTH syntax may be (to a large extent) defined by you, and either interpreted or compiled, depending on your whim and the circumstances. More about these mysteries later!

FORTH is typically much faster than BASIC and a little slower than Assembly Language. It therefore is used primarily in applications which require speed, while avoiding the large programming time demanded for Assembly Language. Such applications may be games, operating systems, process control applications, word processing and spreadsheets, etc..

Its inventor, Charles Moore, created it to allow him to satisfactorily control

an astronomical (optical) telescope at the Kitt Peak Observatory in Arizona, USA.

FORTH possesses the additional merits of being compact, easily debugged and transportable between differing CPUs.

A\*FORTH is an implementation of FORTH created by John Redmond. It is available from Rainbow, Tandy or John Redmond (see ads).

John intended A\*FORTH to be almost entirely compatible with the syntax of the book "Starting FORTH", by Leo Brodie, of FORTH inc..

These articles assume that you have both A\*FORTH and Starting FORTH, though I hope that casual readers will also benefit from them.

Doubtless by now you're aching to run the A\*FORTH you bought recently. Well, load and EXEC it and we'll get on with using it. (I'll refrain from such murderously inane comments as "we'll set FORTH"). Forgive my sick humour!

To stimulate your fancy, let's muck about a bit.

Try

```
PRINT(2 + 2) <ENTER>
```

It didn't like that, did it? Basic-like statements cannot function within FORTH unless specially defined. Cast out thoughts of "doing" such definitions for the moment.

Try

```
2 2 + . <ENTER> (Put in all spaces as shown)
```

Much to your delight, I suspect, (and possibly, amazement) a 4 was printed. If not, you did something wrong. Try again, (no-one is looking).

Notice that the numbers which are to be operated on precede the operator (the +). This method of proceeding with calculations is called Postfix or Reverse Polish Notation (RPN).

Notice also that each number or other character in this example was isolated by spaces. In FORTH each character or group of characters isolated by spaces is a FORTH word. FORTH allows any

combination of characters to be defined as a FORTH word and added to the dictionary currently in use. If you think about this feature of FORTH for a moment you will probably see that this is a wonderfully powerful capability! Obviously, if you miss a space, FORTH will see the incorrect string of characters as an unknown FORTH word, and suitably admonish you!

The stop after the + is the FORTH version of PRINT (unless you make it be something else, of course). More of that later: don't worry about it for now. I tend to leap ahead of the main thread of my explanation from time to time.

Try  
10 10 \* . <ENTER>

You obtained, of course, the value 100. If you would like to make the same calculation in hexadecimal, try

HEX A A \* . <ENTER>

Did you get 64, the HEXIDECIMAL equivalent of DECIMAL 100?

To again obtain answers written to base 10, type DECIMAL before your next calculation. This will set all future calculations to decimal until you do something to alter the current base. Such a something might, for instance be a return to hexadecimal by using HEX. There are ways of using other bases, e.g. base 2 (binary), but I'll save that for the May issue.

Why not put the RAINBOW aside for a few minutes and play around with a few calculations in RPN. Check the results using your other computer. . . You know - the one between your ears! To further aid your imminent attempts to compute FORTH style, the standard operators are the same as in BASIC. Those of you who feel really adventurous might like to try chained calculations, with the . (PRINT) inserted at interesting points in the process, versus a . at the end only.

Good luck! If you would like to talk about FORTH, please feel free to call me on (07) 2087820.

continued from Page 5

```

143 IFF=0THEN149
144 FORI=1TOLEN(G$)
145 IFMID$(G$,I,1)=" "THEN147
146 NEXTI
147 T$=LEFT$(G$,I):Y$=MID$(G$,I+
1)
148 G$=T$+"{"+"Y$+"}"
149 E$="":FORI=ST TO S-1
150 T$=HEX$(PEEK(1)):GOSUB158
151 E$=E$+T$:NEXTI
152 J=INSTR(G$," "):IFJ=0 THEN 1
53 ELSEQ$=MID$(G$,J+1):G$=LEFT$(
G$,J)+STRING$(6-J," ")+Q$
153 POKE65494,0:PRINT#DN,RIGHT$(
"0000"+HEX$(PO),4);";E$;STRING
$(11-LEN(E$),32);G$:POKE65495,0
154 IFINKEY$=" "THEN10
155 PO=PO+S-PO
156 IFS<=E THEN25ELSE10
157 T$=RIGHT$("0000"+T$,4):RETUR
N
158 T$=RIGHT$("00"+T$,2):RETURN
159 RETURN
160 PRINT"ASCII DUMP"
161 FORS=S TOE
162 T$=RIGHT$("0000"+HEX$(S),4)+
" "+RIGHT$("00"+HEX$(PEEK(S)),2)
+" FCC "
163 Q=PEEK(S):IFQ>31 AND Q<128TH
ENT$=T$+CHR$(Q):GOTO166
164 IFQ>159THENT$=T$+"$80+"+CHR
$(Q-128):GOTO166
165 T$=T$+"??"
166 POKE65494,0:PRINT#DN,T$:POKE
65495,0:IFINKEY$=" "THEN10ELSENE
XT:GOTO10
167 PRINT"WORD DUMP"
168 FORS=S TOE STEP2:T$=RIGHT$("
0000"+HEX$(S),4)+" "+RIGHT$("000
0"+HEX$(PEEK(S)*256+PEEK(S+1)),4
)+" FDB $" +RIGHT$("0000"
+HEX$(PEEK(S)*256+PEEK(S+1)),4)
169 POKE65494,0:PRINT#DN,T$:POKE

```

```

65495,0:IFINKEY$=" "THEN10ELSENE
XT:GOTO10
170 PRINT"CLOADM PROGRAM":INPUT"
PROGRAM NAME";N$:INPUT"LOADING O
FFSET";LO:POKE65494,0:CLOADM$,L
O:POKE65495,0
171 S=PEEK(487)*256+PEEK(488):E=
PEEK(126)*256+PEEK(127)-1:PRINT"
START ADDRESS=$";RIGHT$("0000"+H
EX$(S),4):PRINT"END ADDRESS=$";R
IGHT$("0000"+HEX$(E),4):PRINT"EX
ECUTE ADDRESS=$";RIGHT$("0000"+H
EX$(PEEK(157)*256+PEEK(158)),4):
GOTO10
172 PRINT"BYTE DUMP"
173 FORS=S TOE
174 T$=RIGHT$("0000"+HEX$(S),4)+
" "+RIGHT$("00"+HEX$(PEEK(S)),2)
+" FCB $" +RIGHT$("00"+
HEX$(PEEK(S)),2)
175 POKE65494,0:PRINT#DN,T$:POKE
65495,0
176 IFINKEY$=" "THEN10
177 NEXT
178 GOTO10
179 DN=-2:PRINT:GOTO10
180 DN=0:INPUT"COLOR";C$:IFC$="O
"THENSCREEN0,1ELSESCREEN0
181 GOTO10
182 IF(S+T)<0THENT=65536+T:RETUR
N
183 IF(S+T)>65535THENT=T-65536:R
ETURNELSERETURN
184 Q=VAL("&H"+RIGHT$(G$,2)):V$=
""
185 IF(Q AND128)=128THENV$="E"
186 IF(Q AND64)=64THENV$=V$+"F"
187 IF(Q AND32)=32THENV$=V$+"H"
188 IF(Q AND16)=16THENV$=V$+"I"
189 IF(Q AND8)=8THENV$=V$+"N"
190 IF(Q AND4)=4THENV$=V$+"Z"
191 IF(Q AND2)=2THENV$=V$+"V"
192 IF(Q AND1)=1THENV$=V$+"C":G$
=G$+" "+V$:RETURNELSEG$=G$+" "+V
$:RETURN

```

```

193 S$="NEGD *****COMD LSRD
*****RORD ASRD ASLD ROLD DECD *
****INCD TSTD JMPD CLRD ":RETURN
194 S$="*****NOPH SYNCH****
*****LBRARLBSRR*****DAAH ORCCM*
****----SEXH EXGM TFRM ":RETURN
195 S$="BRAR BRNR BHIR BLSR BHSR
BLOR BNER BEQR BVCR BVSr BPLR B
MIR BGER BLTR BGTR BLER ":RETURN
196 S$="LEAXILEAYILEASILEAUIPSHS
MPULSMPSHUMPULUM*****RTSH ABXH R
TIH CVAIHMULH *****SWIH ":RETURN
197 S$="NEGAH*****COMAHLsRA
H*****RORAHASRAHASLAHROLAHDCAH*
****INCAHTSTAH*****CLRAH":RETURN
198 S$="NEGBH*****COMBHLsRB
H*****RORBHASRBHASLBHROLBHDCEBH*
****INCBHTSTBH*****CLRBH":RETURN
199 S$="NEGI *****COMI LSRI
*****RORI ASRI ASLI ROLI DECI *
****INCI TSTI JMPI CLRI ":RETURN
200 S$="NEGE *****COME LSRE
*****RORE ASRE ASLE ROLE DECE *
****INCE TSTE JMPE CLRE ":RETURN
201 S$="SUBAICMPA1SBCA1SUBD2ANDA
1BITA1LDA1 *****EORA1ADCA1ORA1 A
DDAICMPX2BSRR LDX2 *****:RETURN
202 S$="SUBA CMPA SBCA SUBD ANDA
BITA LDA STA EORA ADCA ORA A
DDA CMPX JSR LDX STX ":RETURN
203 S$="SUBB CMPB SBCB ADDD ANDB
BITB LDB STB EORB ADCB ORB A
DDB LDD STD LDU STU ":RETURN
204 S$="SUBBICMPB1SBCB1ADDD2ANDB
1BITB1LDB1 *****EORB1ADCB1ORB1 A
DDB1LDD2 *****LDU2 *****:RETURN
205 S$="3FHSW12832CMPD8C2CMPY8E2
LDY 93DCMPD9DCMPY9EDLDY 9FDSTY
A3ICMPDACICMPYAEILDY AFISTY B3EC
MPDBCECMPYBEILDY BFESTY CE2LDS D
EDLDS DFDSTS BEILDS EFISTS FEELD
S FFESTS ":RETURN
206 S$="3FISW13832CMPU8C2CMPS93D
CMPU9CDCMPSA3ICMPUACICMPSB3ECMPU
BCECMPS":RETURN

```

## FRICKER'S FOLLIES

Before we get too far I had better clear up a couple of misprints that occurred in some of my previous columns, the first being the listings to fix the baud rate problems (Sept Rainbow). Unfortunately both listings were labelled version 1.1 when in fact they were 1.1 and 1.0 in order of printing.

The next problem that occurred is one you should have already picked up. The next article (Oct Rainbow) which had a Basic09 program called Hello also had misprints. The article talked about using delimiters on a single line when using multiple statements using the backslash. The problem is that when the listing was printed the word processor completely ignored the backslashes when time came to print it.

But enough of this, let's get on with what we are here for, this month's mess up! (Watch it! G.)

This month I am going to present a couple of programs of no great worth and try to explain how they work. These programs duplicate two of the utilities that are supplied with OS-9. They are here PURELY as examples and not to replace the Machine Language programs.

The first will read a disk ASCII file and list it on the screen. It can easily be modified to redirect the output to the printer or any other applicable device. The explanations will be after the listings.

## PROCEDURE BASIC09-LIST

```

10 dim filename:string[32]
   rem make filemane no more than 32
   characters long.
   dim inpath:byte
   rem make inpath a 1 byte integer
   dim char$:string[1000]
   rem make char$ long enough to handle
1000 bytes.
   print chr$(12)

20 input "Name of ASCII file to be
listed",in$

```

```

30 open #inpath,filename:read
40 while not eof(#inpath) do
   input (#inpath),char$
   print char$

50 endwhile
   end

```

Now we are going to look at this one line at a time.

We have already covered the Procedure statement in previous articles so we will look at the second line (10) where we introduce another form of the DIM statement. We have a different way of using DIM than you may be used to with the OS-9 format being DIM filename:string[32].

Just what is this :string[32] about and why did we leave the \$ out? Firstly as you may remember the colon (:) is NOT a delimiter as it is in Colour Basic. The colon is used in this case to pass the value in the square brackets to the variable and to let Basic09 know what type of variable it is. The value may be changed up or down as memory permits.

The second line has another form of the DIM statement, where inpath is defined as a BYTE integer variable. An integer variable is required where it is used in the open and input statements.

The next line is much the same as the first except that this time we have used the more conventional naming of a string by using the dollar sign (\$).

The print CHR\$(12) statement will clear the screen and home the cursor or it will send a form feed to your printer if your printer recognises one.

The INPUT statement is just the same as the one you are used to in Basic.

The OPEN statement comes next and if you look at it you will see that it does make sense. The inpath variable is the one that we earlier defined as an integer. Basic09 requires that it be an integer and it assigns its own value when it runs the program.

You must let Basic09 assign its own values instead of doing it yourself which is the way Disk Basic does it. OS9 can have other files and buffers being opened and closed by other users on other terminals and may assign them the values that you may want to use.

The next lines (40+) introduce you to the Control structure WHILE/DO ENDWHILE. This group of statements may be new to you. What the while/endwhile loop does is check for a condition much the same way as the if/then loops that you are used to.

What while means is that while a condition (test) remains true then it will do whatever is in the loop and when the test fails it will exit the loop much the same as the IF/THEN does. OK, so why did I use WHILE instead of IF. In this case I did it to illustrate the use of "while"!

I could have used LOOP/ENDLOOP or REPEAT/UNTIL control loops to do almost the same thing, but I will leave them until another time.

The next little offering takes the program we just looked at and makes it do something useful. In this case we are duplicating the MERGE command in OS9. This program will ask for two input file names and one output filename.

```
BASIC09_merge
```

```
dim first:string[32]
dim second:string[32]
```

```
dim output:string[32]
dim inpath:byte
dim outpath:byte
dim char$:string[1000]
print chr$(12)

input "first filename",first
input "second filename", second
input "output filename", output

open #inpath,first:read
open #outpath,output:write

while not (eof(#inpath)) do
  input #inpath,char$
  write #outpath,char$
  print char$
endwhile

close #inpath
open #inpath,second:read

while not (eof(#inpath)) do
  input #inpath,char$
  write #outpath,char$
  print char$
endwhile

close #inpath
close #outpath

end
```

As you will have noticed, this program does not use line numbers at all. As explained in earlier articles the use of line numbers is optional.

## Hardware Review

### XPNDR2 and Super Guide — an Ideal Expansion Card Set

The diversity and number of expansion boards available for the CoCo never ceases to amaze me. Robotic Microsystems plays a major role in this area with their XPNDR2 plug-in expansion card. This card measures 7 by 7/4 inches and provides traces connected to the CoCo 6809 microprocessor via the game or "expansion port," as I prefer to call it. The board is very well-made and features gold-plated edge connectors and plated through holes.

A nice addition is a red LED mounted at one corner of the board to remind you that CoCo is turned on. It's also a reminder not to unplug the board with the power turned on.

Attached to about the center of the card is a sturdy 40-pin edge connector socket suitable to plug in your disk controller, voice pack or any other applicable cartridge. The controller or cartridge sits vertically, leaving space on either side of the connector available for experimental circuits.

The 40-pin socket is mounted with wire-wrap pins so easy connection can be made for experimenting. In fact, the board is designed to accommodate wire-wrap sockets for ease of experimenting. A full 24 square inches of component layout space is also available.

Documentation is complete and detailed. A well-illustrated booklet titled "Application Notes" is included. This eight-page booklet is geared toward the experimenter and especially the beginner with its light touch of technical subject matter associated with microcomputer interfacing techniques.

Another item included in the package is called Super Guide. This little plastic gadget mounts inside the expansion port and contains a slot that serves to add support to the XPNDR2 card. This is a very helpful accessory to any expansion card and is recommended since it prevents stress and strain on the expansion connector. It also holds the flap door open and its thin slot (1/8 inch) prevents accidental contact with the connector. It does a great job in accomplishing all these tasks and it's not expensive!

If you're into interfacing, you need to look at what Robotic Microsystems has to offer.

— Jerry Semones

# DIR SORT

by Ross McKay

Ross supplied the following utility written in Basic09 which alphabetically sorts a directory before listing in columns.

Ross is the new meet contact for the Carlingford Colour Computer Club and can be contacted on 02-624-3353, or on Tandy Access BBS.

The Listing:

```

PROCEDURE adir
(* Adir v1.1 - courtesy : --- Rosko ! --- *)

(* This is a simple program that opens the specified *)
(* directory file, reads all valid names, and sorts them *)
(* alphabetically before printing them. *)
(* To run the source [unpacked] program from within BASIC09*)
(* enter 'run adir("<directoryname>")' *)
(* Packing the program will remove remarks and long *)
(* variable names. *)

(* Fixed since v1.0 : *)
(* does not truncate filenames to 16 chars *)
(* faster SEEK of third record *)
(* will not signal error for new (empty) directory *)

PARAM direct:STRING
TYPE xbyte=byt1,byt2,byt3:BYTE
BASE 1

DIM file(60):STRING[29]
DIM lsn:xbyte
DIM counter:BYTE
DIM path:BYTE
DIM x,xx:INTEGER
DIM m:BYTE
DIM swap:STRING[29]

OPEN #path,direct:READ+DIR

SEEK #path,64

counter:=0
WHILE NOT(EOF(#path)) DO
counter:=counter+1
GET #path,file(counter)
GET #path,lsn
IF ASC(file(counter))=0 THEN
counter:=counter-1
ENDIF
ENDWHILE
CLOSE #path

FOR x:=1 TO counter
FOR xx:=x+1 TO counter
IF file(xx)<file(x) THEN
swap:=file(xx)
file(xx):=file(x)
file(x):=swap
ENDIF
NEXT xx
NEXT x

FOR x:=1 TO counter
GOSUB 100
PRINT file(x),
NEXT x
PRINT
END

100 FOR xx:=1 TO 29
m:=ASC(MID$(file(x),xx,1))
EXITIF m>127 THEN
file(x):=LEFT$(file(x),xx-1)+CHR$(m-128)
ENDEXIT
NEXT xx
RETURN

```





**GOLDSOFT**  
Hardware & Software for your TANDY computer.

**HARDWARE**

**The CoCoConnection:**

Connect your CoCo to the real world and control robots, models, experiments, burglar alarms, water reticulation systems — most electrical things.  
Features two MC 6821 PIAs; provides four programmable ports; each port provides eight lines, which can be programmed as an input or output; comes complete with tutorial documentation and software; supplied with LED demonstration unit. Switchable memory addressing allows use with disk controller or other modules via a multipack interface; plugs into Cartridge Slot or Multipack, uses gold plate connectors; a MUST for the hardware designer and debugger!

\$206.00

**Video-Amp:**

Connects simply to your CoCo to drive a Colour or Mono monitor.

With instructions

\$25.00

With instructions and sound

\$35.00

**SOFTWARE**

**CoCoOz:**

The programs you see listed in Australian CoCo Magazine are available on CoCoOz!  
No laborious typing — just (C)LOAD and Go!

Each Tape

\$8.00

Subscription, 6 months

\$42.00

12 months

\$75.00

**NEW for 1986 ONLY** Each DISK

\$10.95

Subscription on disk, 12 months

\$102.50

**Rainbow on Tape:**

Australian. The programs you see listed in Australian Rainbow Magazine are available on tape. A boon if you don't understand the language!  
American. We also supply the programs found in American Rainbow on tape. Please specify either Australian or American.

Each Tape

\$12.00

Subscription, 12 months

\$144.00

**NEW for 1986 ONLY** Each DISK

\$15.00

Subscription on disk, 12 months

\$172.00

**MiCoOz:**

The programs in the MiCo section of Australian CoCo Magazine. (For MC 10 computers only)  
Back issues of CoCoOz and MiCoOz are always available on tape only.

Each Tape

\$8.00

Subscription, 12 months

\$75.00

**To be released soon:**

**GOLDDISK 1000** — programs from 'softgold' for your Tandy 1000 on disk, and,  
**GOLDDISK 4** — programs from 'softgold' for your Tandy Model 4!

\$10.95

**The Best of CoCoOz:**

#1 ... EDUCATION programs. Fourteen programs for the teacher or parent.  
#2 ... Part 1 ... 16K GAMES. (Mainly ECB). Sixteen programs to keep you on your toes!  
Part 2 ... 32K GAMES. Adventures, simulations and arcade games for everyone!  
#3 ... UTILITIES. Programs to make your use of the computer easier.  
Spoolers, Reverse Video, Disk utilities and more.  
#4 ... BUSINESS programs. Invoicing, Accounts, Creditors and more.

Per Tape

\$10.00

Per Disk

\$21.95

#3 on Disk

\$16.00

Any two tapes

\$17.95

**CoCoLink:**

CoCoLink is our Bulletin Board which you can access with any computer if you have a 300 baud modem and a suitable terminal program. There is a free visitor's facility, alternatively membership entitles you to greater access of the many files available. We can also be contacted through Minerva (OTC) and Viatel (Telecom).

Subscription to CoCoLink, 12 months

\$29.00

**Magazines:**

Australian Rainbow Magazine — THE magazine for advanced CoCo users!  
Australian CoCo Magazine — THE magazine for the new user of a Tandy computer.  
Also suits owners of CoCos, MC 10s, Tandy 1000s, 100s, 200s & 2000s.

**Back Issues:**

Australian Rainbow Magazine. (Dec '81 to now.)  
Australian CoCo Magazine. (Aug '84 to now.)  
CoCoBug Magazine. For CoCo — usually 8 programs in each magazine. (Sep '84 to Oct '85)  
Australian MiCo Magazine. For Tandy MC 10 computers. (Dec '83 to Jul '84)  
Australian GoCo Magazine. For Tandy Model 100 users. (Jul '83 to Jul '84)  
See Subscription Page for further details.

**Books:**

HELP: A quick reference guide for CoCo users.  
FACTS: THE reference for CoCo machine language programmers.  
MiCo HELP: A quick reference for owners of MC 10 computers.

\$9.95

\$11.95

\$9.95

**Othello:** by Darryl Berry

The board game for your CoCo.

Tape 16K ECB

\$15.95

**Say the Wordz:** by Oz Wiz & Pixel Software

Two curriculum based speller programs for your Tandy Speech/Sound Pack.

Tape 32K ECB

\$39.95

**Bric a Brac:**

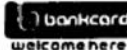
Blank tapes ... 12 for \$18.00 or \$1.70 each.  
Cassette cases ... 15 for \$5.00.  
Disks ... (they work!) \$3.50 each or \$28.95 per box of 10.

**HOW TO ORDER**

Option 1: Use the subscription form in this magazine.  
Option 2: Phone and have ready your Bankcard, Mastercard or Visa number.  
Option 3: Leave an order on Viatel, Minerva or CoCoLink, but be sure to include your Name, Address, Phone Number, Credit Card Number and a clear indication of what you require, plus the amount of money you are authorising us to bill you.



# COMPUTERWARE FOR MICROS.



Peter Collison  
11 Grantley Avenue,  
Rostrevor S.A. 5073  
Phone: (08) 336 6588

## DRIVE 0

Panasonic 40-track single-sided disk drive, slimline case and power supply. Plus CFM controller complete with B-DOS.

**Only \$399<sup>00</sup>**  
(Plus \$8.00 Shipping)

## DRIVE 1

Panasonic 40-track single-sided disk drive, slimline case and power supply.

**Only \$170<sup>00</sup>**  
(Plus \$8.00 Shipping)

## \*\*B-DOS\*\* ©

USER FRIENDLY DISK OPERATING SYSTEM NOW AVAILABLE ON DISK FOR YOUR CO-CO

AUTO (line number) : 35-40 TRACKS  
ERROR TRAPPING : BAUD (value)  
COLD (cold start) : PDIR (print dir)  
PCLEAR (16 pages) : SWAP (var1, var2)  
OS9/DOS (included) : UNNEW.....

A COMPREHENSIVE MANUAL + MUCH MORE  
**\*\*\* ONLY \$44.95 \*\*\***

## UPGRADE-KITS

Up-grade your short case 16K Coco II to 64K with our up-grade kit. Complete with instructions.

**Only \$85<sup>00</sup>**

# NOW DRIVE ZERO! \$399.

**\* LOWER CASE KIT \***  
TRUE LOWER CASE PLUS REVERSE VIDEO  
Now with dual 5:7 and 7:9 characters. Use your COLOR BURNER to put in your own special character sets. (optional)

For visual comfort and Pro programming send for LOWERKIT II-C... **\$89.95**

### \* COLOR BURNER \*

An EPROM programmer is the perfect tool for creating your own program packs. The COLOR BURNER programs the most popular erasable, programmable eproms: 2716(2K), 2732(4K), 2764(8K), 27128(16K) and 68764/66(8K).

EASY TO USE - STEP BY STEP INSTRUCTIONS.

**\*\* ONLY \$99.95 \*\***

## \*\* COLOR QUAVER \*\*

THE ULTIMATE ALL-SOFTWARE MUSIC SYNTHESIZER!  
COLOR QUAVER, an amazing Music experience from your Color Computer.

### FEATURES:

- Real electronic music synthesis that is more than bleeps.
- Full four part harmony, all in precise tempered tuning.
- Five usable octaves, variable tempos, rhythms from 32nd note to whole note.
- Fast compiler provides finished music in five seconds... up to 250 notes per voice line, 1000 notes in all.
- Over 40 pages of instructions, explanations, hints, listings and samples.

**\*\* INCREDIBLE VALUE AT ONLY... \$39.95 \*\* (tape only)**

## SUPER BACK UP UTILITY ©

... WITH S.B.U. FROM COMPUTIZE YOU'LL NEVER NEED ANOTHER BACK-UP UTILITY FOR YOUR CO-CO!!!

1. Tape to Tape
  2. Tape to Disk
  3. Auto Relocate
  4. Disk to Tape
  5. Disk to Disk
- Menu Driven!
  - Requires 32K Extended Co-Co
  - Requires 1 or 2 Drives
  - All Machine Language!!!
- \*\*\* ONLY \$49.95 \*\*\***  
(SUPPLIED ON DISK)

# user group CONTACTS

(Stop between numbers = b.h. else a.b.; but, hyphen between = both.)

ALBERT	JOHN HAIRNS 08 278 3560	MALDEN	LEW MALONEY 079511333&782	LIVERPOOL	LEONIE DUGAN 02-607-3791
ADELAIDE	ROB DUNCAN 060 43 1031	MARYBOROUGH	MAX HUCKERBY 051 45 4315	MACQUARIEFLDS	KEITH ROACH 02 618 2958
ALBURY	DANIEL CLARKSON 067 72 8031	MELBOURNE:	LYN DAVSON 049 49 8144	ROSEVILLE	KEN UZZELL 02 467 1619
AMIDALE	COLIN LEHMAN 051 57 1545	DANDENONG	NORM VERN 071 21 6638	SUTHERLAND	IAN ANNABEL 02 528 3391
BAIRNSDALE	MAR BEVELANDER 053 32 6733	DOMCASTER	DAVID HORSROCKS 03 793 5157	SYDNEY EAST	JACKY COCKINOS 02 344 9111
BALLARAT	ANNIE KEIJER 079.82.6931	FRANKSTON	JUSTIN LIPTON 03 887 5149	TAMWORTH	ROBERT WEBB 067 65 7256
BLAXLAND	BRUCE SULLIVAN 047 39 3903	MARIE WARRER	BOB HAYTER 03.783.9748	TAMWORTH	GARY SYLVESTER 046 81 9318
BOVEN	TERRY COITON C/O 077 86 2220	MTI ELSTERN	LEIGH EVANS 03 704 6680	TARA	STEVEN YOUNGBERRY
BRASSALL	BOB UNSWORTH 07 201 8659	MELTON	MARIO GERADA 03 743 1323	TONGALLA	TONY HILLIS 056 59 2251
BRIGHTON	GLENN DAVIES 08 296 7477	RINGWOOD	IVOR DAVIES 03 758 4496	TOOWOOMBA	GRAHAM BURGESS 076 30 4254
BUNDABERG	ROB THOMPSON 07 848 5410	MILDURA	JACK SMIT 03.744.1355	TOYNSVILLE	JOHN O'CALLAGHAN 077 73 2064
CAIRNS	SOUTH WEST GRAHAM BUTCHER 07 376 3400	MORPETHVILLE	SCOTT HOWISON 050 23 6016	TRARALGON	MORRIS GRADY 051 66 1931
CAMBERA NTH	PIKE RIVERS BARRY CLARKE 07 204 2806	KORBE	KEN RICHARDS 06 384 4503	UPPER HUNTER	TERRY GRAYOLIN 065 45 1698
CAMBERA STH	BROKEN HILL DEAN PARADICE 080 6701	KORWELL	ALF BATE 067 52 2465	URALLA	FRANK MUDFORD 067 78 4391
CARLINGFORD	RON SIMPSON C/O TANDY	MT ISA PAUL	GEORGE FRANCIS 051 34 5175	VAGGA VAGGA	CSS JENKINSON 069 28 2263
CHURCHILL	GLENN HODGES 070 54 6583	BUCKLEY-SIMONS	BRIAN STORE 063-72-1958	WRITENOCK	GLENN HODGES 070 54 6583
COFFS HARBOUR	JOHN BURGESS 062 58 3924	BRIAW STORE 063-72-1958	PETER ANGEL 071 68 1828	YARRAVONGA	PAT KERMODE 056 74 4583
COOMA	ROSIE MCNAY 02 624 3353	ORANGE	LYN DAVSON 049 49 8144	SPECIAL INTEREST GROUPS	ANDREW WILLIE 004 35 1839
COORANBORO	ROSS PRATT 0648 23 065	PARRES	STEVE LOVETT 063.62.4025	BRIZBIZ BRIAN HERE-STREETER 07 349 4696	OS9 GROUPS
DALBY	ANDREW B. SIMPSON 074.62.3228	PERKITH	JIM JAMES 063 62 8625	TDP - TELEWRITER, DYNAMIC PROCOLOR	GEORF TOLPOTT 07 44 6084
DEWILLOUGH	VAYNE PATTERSON 058 81 3014	PORT LINCOLN	DAVID SMALL 068 62 2682	BRISBANE	CARL STEEN 02 646 3619
DOBBO	GRAEME CLARKE 068 89 2095	PORT MACQUARIE	ALEX SCHOFIELD 047 31 5303	BRISBANE	JACK FRICKER 07 262 8869
EMERALD	LEIGH EVANS 059 68 3392	PORT PHILIP	IAN MACLEOD 09 448 2136	CARLINGFORD	ROSKO MCKAY 02 624 3353
FORBES	JOHANNA VAGG 068 52 2943	ROCKHAMPTON	ROB DALZELL 08 386 1647	COOMA	FRED BISSELLING 0648 23263
GIPPSLAND STH	GARY BAILEY 065 54 5029	SALE	KEVIN GOVAN 086 32 1368	KALGOORLIE	TERRY BURBETT 090.21.5212
GLADSTONE	PAT KERMODE 056 74 4583	SCARBOROUGH	BRYAN MCHUGH 051 44 4792	PERKITH	BOB THOMPSON 047 30 2468
GOLD COAST	CAROL CATHCART 079 78 3594	SEACOMBE HTS	PETER MAY 07 203 6723	STONEY EAST	JACKY COCKINOS 02.344.9111
GOSFORD	GRAHAM MORPHETT 075 51 0015	SHEPPARTON	GLENN DAVIS 08 296 7477	STONEY NTH	MARK ROTHWELL 02 817 4627
GOULDERN VALLEY	PETER SEIFFERT 043 32 7874	SPRINGWOOD	ROSS FARFAR 058 25 1007	LITHGOW	DAVID BERGER 063 52 2282
GRAFTON	GOULBURN VALLEY TONY HILLIS 058 59 2251	STURT	TONY PATTERSON 053 42 8815	PORT LINCOLN	BILL BOARDMAN 086 82 2385
GREENACRES	PETER LINDSAY 066 42 2503	SVAN HILL	DAVID SEAMONS 047 51 2107	ROCKHAMPTON	TIM SHANK 079 28 1846
GUYRA	BETTY LITTLE 08 261 4063	SYDNEY:	MARY DAVIS 08 296 7477	STONEY	RAJA VIJAY 02 519 4106
HASTINGS	MICHAEL J. HARTMAN 067 79 7547	BANKSTOWN	BARRIE GERARD 050.32.2838	WARRAKABOOL	GARY FURR 055 62 7440
HEBERT BAY	LESLIE HORWOOD 071 22 4989	BLACKTOWN	CARL STEEN 02 646 3619	BRISBANE	BRIAN DOUGAN 07 30 2072
HOBART	BOB DELBOURGO 002 25 3896	BLACKTOWN KEITH GALLAGHER 02-627-4627	ART PITTARD 02 72 2861	KELBOURNE	TONY LOYD 03 500 0876
IPSWICH	MILTON ROWE 07 281 4059	CAMPBELLTOWN	LEO GIBLEY 02 605 4572	STONEY	ROGER RUTHER 047.39.3903
JUNEE	PAUL MALONEY 069 24 1860	CHATSWOOD	BILL O'DONNELL 02 817 4627	BRISBANE	JOHN POKON 07 288 7820
KALGOORLIE	TERRY BURBETT 090.21.5212	CLAYTON HERMAN FREDRICKSON 02 6236379	MARK ROTHWELL 02 817 4627	BRISBANE	JOHN BOARDMAN 086 82 2385
KINGSTON	VIM DE PUIG 002 29 4950	DACEVILLE	DEWIS CORROY 02 671 4065	GOLD COAST	JOHN REDMOND 02 85 3751
LEETON	CHRIS MAGEL 069 53 2969	HILLS DIST	ATHALIE SMART 02 848 8830	TAMWORTH	TONY EVANS 077 86 2220
LEICHHARDT	STEVEN CHICOS 02 560 6207	HORNSEY		ROBERT WEBB 067 65 7256	ROBERT WEBB 067 65 7256
LITHGOW	DAVID BERGER 063 52 2282			CSS JENKINSON 069 25 2263	CSS JENKINSON 069 25 2263

AUSTRALIAN RAINBOW MAGAZINE  
 REGISTERED BY AUSTRALIAN POST —  
 PUBLICATION NO. QBG 4009  
 AUSTRALIAN COCO/sotfgold  
 REGISTERED BY AUSTRALIAN POST —  
 PUBLICATION NO. QBG 4007.  
 PO BOX 1742,  
 SOUTHPORT. QLD. 4215.

AUSTRALIAN CoCo this month.

SOFTWARE REVIEWS !!!!  
 QUICK ON THE DRAW WINNERS !!!  
 PLUS:  
 BRIDGE - THE CARD GAME,  
 EDUCATION,  
 PAT KERMODE RETURNS,  
 AND HEAPS MORE !!!

DONT FORGET - COCOCONF '86 IS ON IN AUGUST!  
 BE THERE !!

POSTAGE  
 PAID  
 AUSTRALIA