Ten Complete Programs Included

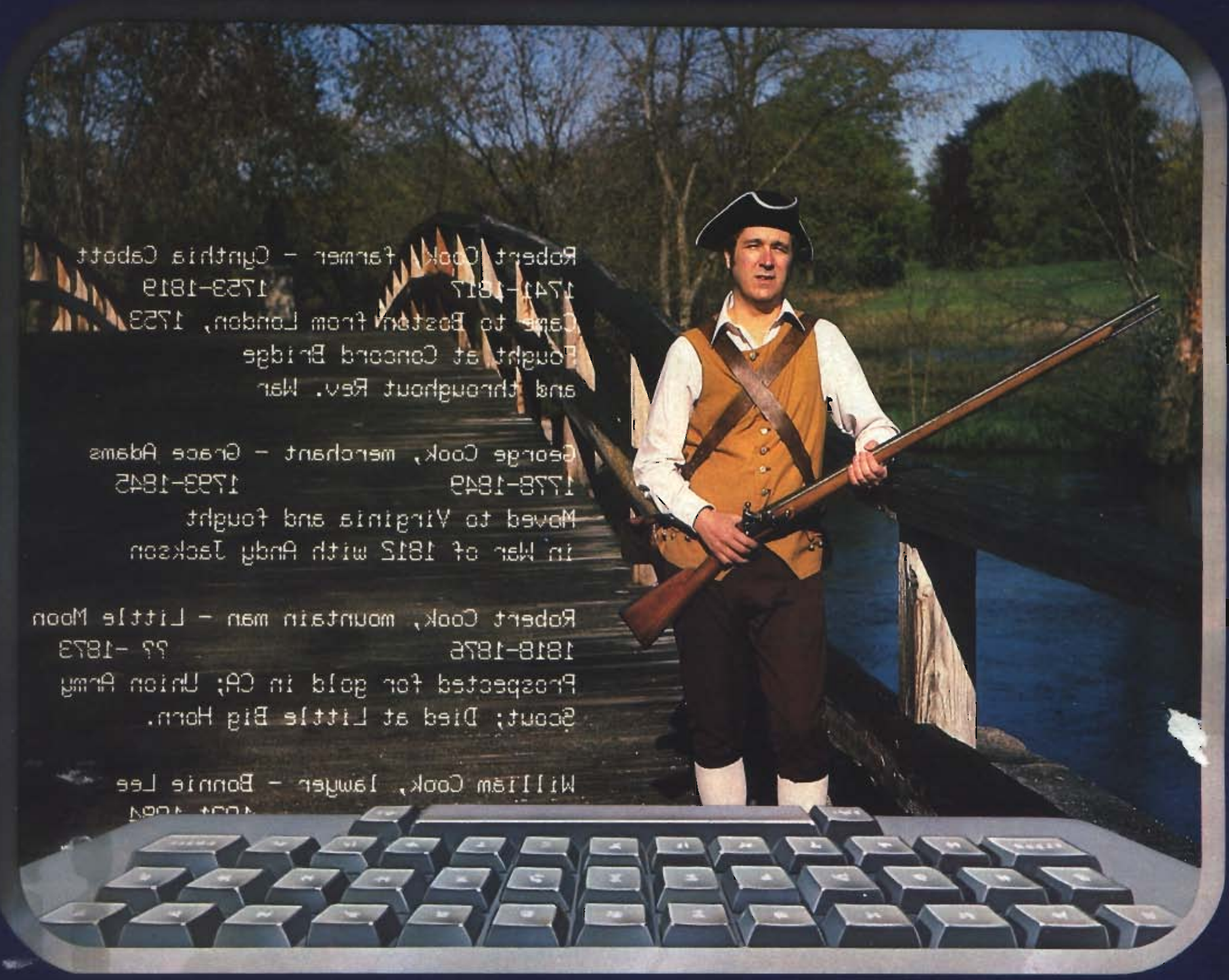
NO. 73

U.S. Edition: $2.50
International Edition: $3.00

JULY 1984

MICRO™

for the Serious Computerist

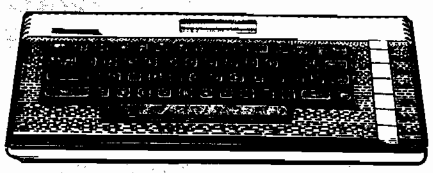

Robert Cook, farmer - Cynthia Cabott
1741-1817 1753-1819
Came to Boston from London, 1753
Fought at Concord Bridge
and throughout Rev. War
George Cook, merchant - Grace Adams
1778-1849 1793-1845
Moved to Virginia and fought
in War of 1812 with Andy Jackson
Robert Cook, mountain man - Little Moon
1818-1876 ?? - 1873
Prospected for gold in CA; Union Army
Scout; Died at Little Big Horn.
William Cook, lawyer - Bonnie Lee

Basic DVORAK Keyboard
Applesoft Compression
Better BASIC Hex Loader
HiRes Graphic Printouts
6809/68000 Comparison
Flight Simulator II

It all adds up...

HOME COMPUTERS

ATARI®



600XL....................$149
800XL....................$229
WHILE SUPPLY LASTS

1200XL.................CALL
1450XL.................CALL

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

DISK DRIVES FOR ATARI

PERCOM		RANA	
AT 88-S1	$279.00	1000	$299.00
AT 88-S1 PD	$299.00	**TRAK**	
INDUS		AT-D2	$389.00
GT Drive	$379.00	AT-D4	$589.00

MEMORY BOARDS

ATARI		APPLE/FRANKLIN	
Axlon 32K	$59.99	Axlon 128K	$299.00
Axlon 48K	$99.99	Axlon 320K	$849.00
Axlon 128K	$299.00		

ALIEN VOICE BOX

Atari	$119.00	Apple	$149.00

DISKETTES

MAXELL		ELEPHANT	
5¼" MD-1	$24.99	5¼" SS/SD	$18.49
5¼" MD-2	$34.99	5¼" SS/DD	$21.99
8" FD-1	$39.99	5¼" DS/DD	$26.99
8" FD-2	$49.99	**DISK HOLDERS**	
VERBATIM		**INNOVATIVE CONCEPTS**	
5¼" SSDD....Value Life	$22.99	Flip-n-File 10	$3.99
5¼" DSDD....Value Life	$29.99	Flip-n-File 50	$17.99
BIB		Flip-n-File 50 w/lock	$24.99
5¼" Disk Head Cleaner	$14.99	Flip-n-File (400/800 ROM)	$17.99

CONTROLLERS & JOYSTICKS

WICO		KRAFT	
Joystick	$21.99	Joystick	$41.99
3-way Joystick	$22.99	Atari Single Fire	$12.99
Power Grip	$21.99	Atari Switch Hitter	$15.99
BOSS Joystick	$17.99	Apple Paddles	$34.99
ATARI/VIC Trak Ball	$34.99	IBM Paddles	$34.99
Apple Trak Ball	$54.99	IBM Joystick	$46.99
Apple Analog	$37.99		

Koala

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

PRINTERS

AXIOM	
AT-100 Atari Interface Printer	$239.00
GP-100 Parallel Interface	$199.00
GP-550 Atari Bidirectional	$319.00
GP-700 Atari Color Printer	$489.00
GP-550 Parallel Printer	$269.00
BMC	
401 Letter Quality	$589.00
BX-80 Dot Matrix	$269.00
C.ITOH	
Gorilla Banana	$209.00
Prowriter 8510P	$379.00
Prowriter 1550P	$599.00
A10 (18 cps)	$569.00
Hot Dot Matrix	CALL
F10-40	$999.00
F10-55	$1349.00
COMREX	
ComWriter II Letter Quality	$499.00
DIABLO	
620 Letter Quality	$949.00
630 Letter Quality	$1749.00
DAISYWRITER	
2000	$999.00
Tractor Feed	$109.00
EPSON	
RX-80, RX-80FT, RX-100	CALL
FX-80, FX-100	CALL
LQ 1500	CALL
IDS	
Prism 80...For Configuration	CALL
Prism 32...For Configuration	CALL
JUKI	
6100	$499.00

MANNESMAN TALLY	
160L	$589.00
180L	$799.00
Spirit 80	$309.00
NEC	
8023 Dot Matrix	$389.00
8025 Dot Matrix	$669.00
2010/15/30	$749.00
3510/15/30	$1369.00
7710/15/30	$1799.00
OKIDATA	
82, 83, 84, 92, 93, 2350, 2410	CALL
OLYMPIA	
Compact 2	$479.00
Compact RO	$509.00
ESW 3000	$1449.00
SMITH CORONA	
TP-1000	$449.00
Tractor Feed	$119.00
SILVER REED	
500 Letter Quality	$449.00
550 Letter Quality	$549.00
770 Letter Quality	$899.00
STAR	
Gemini 10X	$299.00
Gemini 15X	$399.00
Serial Board	$75.00
Radix 10	$599.00
Radix 15	$699.00
TOSHIBA	
1340	$869.00
1351	$1699.00
TRANSTAR	
120P	$469.00
130P	$649.00
315 Color	$459.00

MODEMS

ANCHOR	
Volksmodem	$59.99
Mark IL Serial	$79.99
Mark VII (Auto Ans/Auto Dial)	$99.99
Mark XII (1200 Baud)	$299.99
Mark TRS-80	$99.99
9 Volt Power Supply	$9.99
HAYES	
Smartmodem 300	$209.00
Smartmodem 1200	$499.00
Smartmodem 1200B	$449.00
Micromodem IIe	$269.00
Micromodem 100	$299.00
Smart Com II	$89.99
Chronograph	$199.00

NOVATION	
J-Cat	$99.99
Cat	$139.00
Smart Cat 103	$179.00
Smart Cat 103/212	$399.00
AutoCat	$219.00
212 AutoCat	$549.00
Apple Cat II	$249.00
212 Apple Cat	$449.00
Apple Cat 212 Upgrade	$259.00
PC Cat Access 1 2 3	$399.00
ZENITH	
ZT-1	$339.00
ZT-10	$309.00
ZT-11	$369.00

MONITORS

AMDEK	
300 Green	$149.00
300 Amber	$159.00
310 Amber	$169.00
Color 1	$279.00
Color 1 Plus	$299.00
Color 2 Plus	$419.00
Color 3	$349.00
Color 4T IBM	$699.00
BMC	
1201 (12" Green)	$88.99
1201 Plus (12" Green Hi-Res)	$98.99
9191 Plus	$249.00
GORILLA	
12" Green	$88.99
12" Amber	$95.99
NEC	
JB 1260 Green	$109.00
JB 1201 Green	$149.99
JB 1205 Amber	$159.99
JB 1215 Color	$259.00
JC 1216 RGB	$429.00
JC 1460 Color	$359.00
PRINCETON GRAPHICS	
MAX-12 Amber	$199.00
HX-12 RGB	$539.00
SRT-2 RGB	$649.00

SAKATA	
SC-100 Color	$269.00
SG-1000 Green	$129.00
SA-1000 Amber	$139.00
TAXAN	
210 Color RGB	$299.00
400 Med-Res RGB	$319.00
415 Hi-Res RGB	$439.00
420 Hi-Res RGB (IBM)	$489.00
100 12" Green	$125.00
105 12" Amber	$135.00
USI	
Pi 1, 9" Green	$99.99
Pi 2, 12" Green	$119.99
Pi 3, 12" Amber	$129.99
Pi 4, 9" Amber	$119.99
1400 Color	$269.99
QUADRAM	
Quadchrome 8400 Color	$519.00
ZENITH	
ZVM122 Amber	$99.99
ZVM123 Green	$89.99
ZVM124 IBM-Amber	$149.00
ZVM131 Color	$309.00
ZVM133 RGB	$429.00
ZVM135 RGB/Composite	$469.00

COMPUTER MAIL ORDER

west
800-648-3311
In NV call (702)588-5654, Dept. 125
Order Status Number: 588-5654
P.O.Box 6689
Stateline, NV 89449

canada
Ontario/Quebec 800-268-3974
Other Provinces800-268-4559
In Toronto call
(416)828-0866, Dept. 125
Order Status Number: 828-0866
2505 Dunwin Drive, Unit 3B
Mississauga, Ontario, Canada L5L1T1

east
800-233-8950
In PA call (717)327-9575, Dept. 125
Order Status Number: 327-9576
Customer Service Number: 327-1450
477 E. 3rd St., Williamsport, PA 17701

VISA MasterCa

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

the best prices

APPLE/FRANKLIN DISK DRIVES

MICRO-SCI

A2	$219.00
A40	$299.00
A70	$319.00
C2 Controller	$79.99
C47 Controller	$89.99

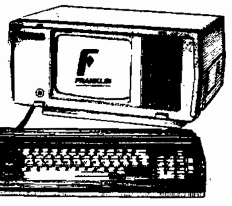
RANA

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

APPLE IIe STARTER PACK

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

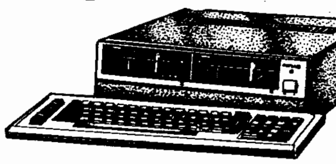
FRANKLIN



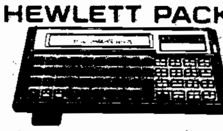
ACE 1000 Color Computer	CALL
ACE PRO PLUS System	CALL
ACE 1200 Office Mgmt. System	CALL
ACE PORTABLES	CALL

SANYO

MBC 550	**CALL**
MBC 555	**CALL**
MBC 555-2	**CALL**
MBC 1100	$1499.00
MBC 1150	$1899.00
MBC 1200	$1849.00
MBC 1250	$2099.00
PR 5500 Printer	$599.00



HEWLETT PACKARD



HP 71B	**$419.99**
41CV	**$189.99**
41CX	**$249.99**
HP 11C	$62.99
HP 12C	$92.99
HP 15C	$92.99
HP 16C	$92.99
HP 75D	$879.99
HPIL Module	$98.99
HPIL Cassette or Printer	$359.99
Card Reader	$143.99
Extended Function Module	$63.99
Time Module	$63.99

SHARP

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

NEC

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

commodore

CBM 8032	**$599**
CBM 8096	$869.00
CBM 9000	$999.00
B128-80	$769.00
8032 to 9000 Upgrade	$499.00
2031 LP Disk Drive	$299.00
8050 Disk Drive	$949.00
8250 Disk Drive	$1199.00
4023 Printer	$379.00
8023 Printer	$569.00
6400 Printer	$1399.00
Z-RAM	$499.00
Silicon Office	$699.00
The Manager	$199.00
SoftROM	$125.00
VisiCalc	$159.00

PROFESSIONAL SOFTWARE

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

SX-64 Portable	**$859**
CBM 64	**$199**
C1541 Disk Drive	$249.00
C1530 Datasette	$69.99
C1520 Color Printer/Plotter	$129.00
M-801 Dot Matrix Printer	$219.00
C1526 Dot Matrix/Serial	$299.00
C1702 Color Monitor	$259.00
C1311 Joystick	$4.99
C1312 Paddles	$11.99
C1600 VIC Modem	$59.99
C1650 Auto Mdoem	$89.99
Logo 64	$49.99
Pilot 64	$39.99
Word Pro 64 Plus	$59.99
Calc Result 64	$65.99
Calc Result Easy	$39.99
Codewriter 64	$75.99
MCS 801 Color Printer	$499.00
DPS 1101 Daisy Printer	$459.00
Magic Voice Speech Module	$54.99
Desk Organizer Lock	$49.99
Vidtex Telecommunications	$34.95

MSD

SD1 Disk Drive	$349.00
SD2 Disk Drive	$599.00

NEC PRINTERS

NEC 2050	$899.00
NEC 3550	$1669.00

PERCOM/TANDON DISK DRIVES

5¼" 320K Floppy	$219.00
5 Meg Hard w/Controller	$1049.00
10 Meg Hard w/Controller	$1349.00
20 Meg Hard w/Controller	$1899.00

VISICORP

VisiCalc IV	$159.00
VisiWord +	$249.00
Visi-on Application Manager	$79.99
Visi-on Calc	$269.00
Visi-on Graph	$179.00
Visi-on Word	$249.99
Optical Mouse	$189.99

AST RESEARCH

Six Pak Plus	from...$279.00
Combo Plus II	from...$279.00
Mega Plus	from...$309.00
I/O Plus	from...$139.00

QUADRAM

Quadlink	$479.00
Quadboard	as low as...$289.00
Quad 512 Plus	as low as...$249.00
Quadcolor	as low as...$219.00
Chronograph	$89.99
Parallel Interface Board	$89.99
64K RAM Chips Kit	$59.99

IBM

MICROPRO

WordStar Professional Pack	$389.00
InfoStar	$299.00
SpellStar	$159.00
CalcStar	$99.99

MICROSTUF

Crosstalk	$105.00

MICROSOFT

MultiPlan	$159.00

ASHTON-TATE

dBASE II	$339.00
dBASE III	$449.00
Friday!	$185.00

IUS

EasyWriter II	$249.00
EasySpeller	$119.00
EasyFiler	$229.00

CONTINENTAL SOFTWARE

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

LOTUS

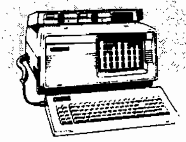
Symphony	$549.00
1-2-3	$339.00

PROFESSIONAL SOFTWARE

PC Plus/The Boss	$329.00

SYNAPSE

File Manager	$89.99



ZENITH data systems

PC COMPATIBLE 16 BIT SYSTEMS

Z-150 PC Z-160 PC

Call for price and configurations

SOFTWARE

	C-64	Atari	IBM	Apple
Electronic Arts				
One on One	$29.99	$29.99	$29.99	$29.99
Music Construction	$29.99	$29.99	$29.99	$29.99
Pinball Construction	$29.99	$29.99	$29.99	$29.99
Cut & Paste	$39.99	$39.99	$39.99	$39.99
Hard Hat Mack	$27 99	$27.99	$27.99	$27 99
InfoCom				
Witness	$29.99	$29.99	$29 99	$29 99
Infidel	$29.99	$29 99	$29.99	$29 99
Deadline	$29.99	$29 99	$29.99	$29.99
Planetfall	$29.99	$29.99	$29.99	$29.99
Enchanter	$29.99	$29.99	$29.99	$29.99
Zork1,2,3 ea	$27.99	$27.99	$27.99	$27.99
Suspended	$29.99	$29.99	$29.99	$29.99
Sorcerer	$29.99	$39.99	$29.99	$29.99
AtariSoft				
Joust	$35.99	N/A	$28.99	$28.99
Moon Patrol	$35 99	N/A	$28.99	$28 99
Ms. PacMan	$35.99	N/A	$28 99	$28 99
PacMan	$35 99	N/A	$28.99	$28.99
Donkey Kong	$35.99	N/A	$28.99	$28.99
Pole Position	$35.99	N/A	$28.99	$28.99
Spinnaker				
Aerobics	$28.99	$28.99	$28.99	$28.99
Trans	$24.99	$24.99	$24.99	$24.99
Adventure Creature	$24.99	$24.99	$24.99	$24.99
Aegean Voyage	$24.99	$24.99	$24.99	$24.99
Snooper Troops 1,2 ea	$28.99	$28.99	$28.99	$28.99
Traction Fever	$22.99	$22.99	$22 99	$22.99
Alphabet Zoo	$22 99	$22.99	$22.99	$22.99
In Search of..	$24.99	$24.99	$24.99	$24.99
Facemaker	$22.99	$22.99	$22.99	$22.99
Kinder Comp	$17 99	$17.99	$17 99	$17 99
Dynatech				
Code Writer	$79.99	$79.99	$175.99	$155.99
VisiCorp				
VisiCalc	$159.99	$149.99	$159.99	$159.99
VisiCalc Advanced	N/A	N/A	N/A	$269.99
pfs:				
Write	N/A	N/A	$89.99	$89.99
Graph	N/A	N/A	$89 99	$79.99
Report	N/A	N/A	$79.99	$79.99
File	N/A	N/A	$89.99	$79.99
Solutions:* as low as	N/A	N/A	$16 99	$16 99

*Call on Titles

COMPUTER MAIL ORDER

west
800-648-3311
In NV call (702)588-5654, Dept. 125
Order Status Number: 588-5654
P.O.Box 6689
Stateline, NV 89449

canada
Ontario/Quebec 800-268-3974
Other Provinces 800-268-4559
In Toronto call
(416)828-0866, Dept. 125
Order Status Number: 828-0866
2505 Dunwin Drive, Unit 3B
Mississauga, Ontario, Canada L5L1T1

east
800-233-8950
In PA call (717)327-9575, Dept. 125
Order Status Number: 327-9576
Customer Service Number: 327-1450
477 E. 3rd St., Williamsport, PA 17701

VISA MasterCard

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

This Month in Micro

This month we have 10 complete, useful, exciting programs for you on a diverse group of topics. The longer ones are available on MicroDisk as well to save you time and effort.

Featured This Month

DVORAK Keyboard — Try out a new keyboard arrangement that can increase your typing speed dramatically. The keyboard now commonly used on computers was deliberately designed to avoid jamming slow typewriter keyboards. Technology eliminated the problem, but the awkward solution is still with us. However, a different layout is becoming more widely accepted, which results in productivity and typing speed skyrocketing. This demo program will allow you to convert your keyboard temporarily and see if you like the arrangement.

6809 vs. 68000 — While the 68000 based computer is far more expensive than the 6809, it can be 100 times more powerful, but, what are the real differences. A checkbook offers a good way to compare their abilities. This program contains the main subroutines to create a machine language program which runs on either kind of machine to allow comparison.

Flight Simulator II — Studying an accepted masterpiece of program design is one way to learn really fine programming skills. Flight Simulator II is just such an exciting state-of-the-art package. Looking into its details and the way it was created will give even experienced programmers more than a few pointers.

C-64 Graphics Dump — This "perfect" dump for the impressive C64 graphics works in either HiRes or multi-color mode, allows large size printouts, works with many printers and graphics packages, can vary color and intensity, and is very fast. This program is available on a MicroDisk.

Communication Between Computers — What do you do when you have several different computers and only one printer? Interface and merge it all into one efficient system.

HILISTER — Highlighting lines of text and programs can be very useful for emphasis or clarity when discussing material on the screen in business meetings, classrooms, seminars. This program also allows easy movement within a program or text.

Simple Numeric Sorting — This simple method lets long lists be arranged in order, without user supplied programs. It takes advantage of a built-in BASIC feature.

Applesoft Compression Program — With other programs, extra long listings often do not work, overflowing the Called Line Number Table. This program has several unusual features which surpass other Compression routines.

Useful Math Functions — Save time and mathematical aggrevation with a compilation of defined functions.

Commodore to Apple — Sort of a poor man's modem. Commodore cassette files can be sent to Apple disks for storage or interfacing with peripherals which don't work with Commodore. This works with data files, BASIC programs and memory ranges.

Circles for the C64 — In a HiRes environment, creating circles can be a problem. The code for this mathematical way of defining and plotting circles in a game or business type analysis is most helpful. The theory will generally work on any 6502 based computer with HiRes capabilities.

BASIC Hex Loader — This handy BASIC Utility will load Machine Language code in Hex, and a special version for the C64 will even generate the DATA statements.

MORE THAN JUST THE TIP
subscribe to!
The Transactor
The Tech/News Journal For Commodore
AND GET MORE FROM YOU
COMMODORE 64 AND VIC 20
"The Transactor is a … delight and a constant resource." - JIM BUTTERFIELD
"The Material is fabulous … of the Transactor. Very impressive. … the friend - LIZ
CHECK THESE FEATURES:
• Up-to-date: new … new products and services.
• Insights and stra… talk from respected expert… like Jim Butterfield.
• Interesting bits o… info shared among Commod… users.
SIMPLY THE BEST
Subscribe now a… you'll wonder how yo… managed without it.
THE TRANSACTOR SUBSC… S DEPT
500 Steeles Avenue, Milton, O… 9T 3P7
Telephone: 876-4741
Year's subscription (6 issues) $15.00
(Separate issues $2.95 each)
Back issues available.

MICRO is published monthly by:
MICRO, Chelmsford, MA 01824.
Second Class postage paid at:
Chelmsford, MA 01824 and additional mailing offices.
USPS Publication Number: 483470.
ISSN: 0271-9002.
Send subscriptions, change of address, USPS Form 3579, requests for back issues and all othe fulfillment questions to:
MICRO
P.O.Box 6502
Chelmsford, MA 01824
or call 617/256-3649.
Subscription Rates: (per year):
U.S. $24.00 or $42.00 for two years
Foreign surface mail: $27.00
Air mail: Europe $42.00
Mexico, Central America, Middle East, North Africa, Central Africa $48.00
South America, South Africa, Far East, Australia, New Zealand $72.00

MICRO™

for the **Serious Computerist**

JULY 1984

13 **A Basic DVORAK Keyboard for the VIC-20 and C-64**
Alfred J. Bruey
A "new" key arrangement is gaining acceptance, increasing typing speed and productivity enormously.

15 **A Comparison of 6809 and 68000**
Mike Rosing
The checkbook offers a simple, effective way to compare these two microprocessors.

19 **Flight Simulator II: Microcomputer Simulation At Its Best**
Chris Williams
By analyzing this design masterpiece, programmers may discover the elements needed to make their own software great.

22 **Graphic Print for C-64**
Michael J. Keryan
Create a full-page graphic printout from a Commodore 64 high resolution display.

30 **Interface Clinic: Communication Between Different Computers**
Ralph Tenny
Merge several computers into one efficient system, sharing a single printer.

34 **HILISTER — A Study and Teaching Aid**
J. Morris Prosser
Move easily within your programs and highlight parts of text or listings for emphasis, drama, clarity.

38 **Super Simple Numeric Sort**
Robert L. Martin
Arrange a list in numerical order without the need for a user supplied sorting program.

42 **CMPRSS: Improved Applesoft Compression Program**
Ian R. Humphreys
Compress large programs and retain comments without overflowing Called Line Number Table.

52 **Useful Functions — Part II**
Paul Garrison
Save time and mathematical aggrevation with a compilation of defined functions.

55 **Commodore-To-Apple Cassette File Loader**
Art Matheny
Transfer cassette files written on VIC-20 or C64 to an Apple disk for interfacing, etc.

65 **BASIC Hex Loader**
Robert M. Tripp
Handy BASIC utility to load Machine Language code in Hex.

66 **Circles for the Commodore 64**
Lester Cain
An interesting mathematical way to plot circles on the C64.

Product Reviews

11 Paint Magic — Easy to use graphics with joystick and keyboard.

11 Promenade Model C1 EPROM Programmer — Add-on programmer which handles 12 models of EPROM and at least 8 of EEPROM.

11 TimeTrax — Time management system for personal or business life, including printed schedules.

12 Spell Perfect — Machine language spelling checker for Letter Perfect or any standard text files.

12 The Complete Graphics System — 2 and 3 dimensional graphics including 108 colors.

Departments

2 Highlights
6 Editorial
7 Feedback
8 Spotlight: Sage
10 Lyte Bytes
11 Reviews
72 CoCo Bits
73 Microbes
74 Catalog
78 Books
79 Listing Conventions
80 Advertiser Index
80 Next Month in Micro

editorial

Dear Readers,

As we approach the midpoint of 1984, I find myself looking towards the future. In the field of computers so much happens so quickly that it is hard to imagine what will transpire in the remainder of this year, let alone five years hence. One way to approach the future is by examining the present, noting the trends and then projecting. At this time the world of the microcomputer continues to dish up new surprises. It seems every time you turn around a new computer is being launched. Although the appearance may differ from machine to machine they are all based on a few standard chips. At its inception, MICRO chose to focus on the 6502 chip. This chip has proven itself to be a well designed and dependable innovation. Although the heyday of the 6502 has passed, it is not dead. This is clearly evidenced by Apple releasing yet another 6502-based computer - the Apple IIc. Apple seems to also be aware of the need to move onward and did so with the introduction of the Macintosh. The 68000 brings the general populace in touch with 16-bit machines. (I will not go into the advantages of a 16-bit over an 8-bit because, if there weren't any, the 68000 would never have surfaced.) Presently the big name in chips seems to be Intel, not Motorola. The 8088, 8086 and other chips developed by Intel have become the backbones of micros made by IBM, Hewlett-Packard, and Digital, to name a few. These are not names to scoff at. As popular as 6502 based machines (Apple, Atari, Commodore, etc.) are, the bulk of sales is starting to shift to machines based on other chips. Unfortunately or fortunately, depending on your viewpoint, there are rumors that Intel is only going to be able to fill 25 percent of its orders. If this proves to be true then someone will have to pick up the slack. The question is who. Perhaps Motorola will seize the opportunity and cover the deficit, using their chips.

But, even if Intel completely dominates the market, the 6502 will carry on. People don't throw away computers because they become outdated. The fact that there are still many IBM mainframes using cards is a testimony to this. Why do people continue to use outdated computers? Certainly the monetary aspect can't be overlooked. Even with drastic reductions in the price of memory (the new HP Nomad has as many words of memory as the old IBM 360 series), and the lowering of the price of computers in general, they are still not cheap. For many it is a matter of loyalty. Others are content with the familiar and prefer the comfort of an old friend to the fear of the unknown. And there are those people who prefer to live in the past, not be bothered and are perfectly content, thank you very much. For these and other reasons there will be a need for 6502 machines, journals, software and support for many years to come.

But what about the future? Certainly one cannot ignore the 68000 or Intel's 80186. To pretend they aren't improvements on previous chips is folly. Rather than seek to delude ourselves I suggest we embrace new technology with open arms and open eyes. To blindly accept something simply because it has been billed as new and improved is foolish. I think the best approach is one of open skepticism. A willingness to explore new territory and seek new frontiers. After all, isn't that what the world of computers has always been about? Let's examine the innovations and carefully separate the wheat from the chaff. Bearing in mind past mistakes, we will always find room to improve and go forward. We have built better mousetraps; we have even built better "mouses"; why not now create men? Because, of mice and men, there is no end.

Mark S. Morano

Mark S. Morano
Technical Editor

On The Cover

```
Robert Cook, farmer - Cynthia Cabott
1741-1817            1753-1819
Came to Boston from London, 1753
Fought at Concord Bridge
and throughout Rev. War

George Cook, merchant - Grace Adams
1778-1849              1793-1845
Moved to Virginia and fought
in War of 1812 with Andy Jackson

Robert Cook, mountain man - Little Moon
1818-1876                    ?? -1873
Prospected for gold in CA; Union Army
Scout; Died at Little Big Horn.

William Cook, lawyer - Bonnie Lee
1823-1863             1831-1884
Confederate Major; Died at Gettysberg
```

On the bridge at Concord, Massachusetts, a colonial minuteman dreams of past and future glories of family and country. Data Bases, long thought of as tools for business and government, have many useful applications in personal life as well. Keeping family trees, health information, employment records are just a few uses which can make you paper-independent. Happy Independence Day!

feedback

Dear Ian,

(RE: Micro 67, Dec. 1983)

I have a question about your program 'C-64 Alarm Clock'. For some unknown reason, when I use 'GOSUB 9140' to reset the alarm, the computer displays 'SYNTAX ERROR IN 48'. It does not affect the operation of the clock, but I would like to know why this statement appears, since there is no statement 48 in this program. I have tried to list statement 48, however, nothing lists. Please reply as soon as possible. Thank you.

Kenneth K. Choy
San Francisco, CA

Dear Kenneth,

The situation you describe, getting a 'syntax error' after 'gosub 9140', seems to occur only occasionally. The simplest explanation is that the GOSUB command is intended to be used from within a program. If you type it into the keyboard directly, then BASIC will execute the subroutine ok. When it is finished, however, it will try to resume executing the program at the next statement after the GOSUB. Since there is no program running, it gets confused and gives an error message.

The error seems to be quite harmless, and does not affect anything. If you use the 'gosub 9140' statement within a program, you should not incur an error.

There is no line 48, of course, and that number is meaningless.

I hope you enjoy the alarm clock program, Kenneth, and that this odd error doesn't cause any problems.

Ian Adam
Vancouver, BC, Canada

To the editor,

Ref. Micro No.51 August 1982, page 97.

First things first. I truly enjoy your magazine. Similarly for Mr. Bongers articles.

In Mr. C. Bongers program on an improved method of garbage collection, MICRO No. 51 page 90, the program works as advertised. However, I found a slight problem when I attempted to use it with string arrays. The second paragraph on page 97 appears to be too brief. I tried using the string version of:

&CLEAR A:DIM A(20,20)

to initialize a string array to zero. This version:

&CLEAR A$:DIM A$(20,20)

didn't do anything until it was modified to force a cleanup as follows:

&CLEAR A$:FRE (1,K) : DIM A$(20,20)

From then on I was smiling.

James Fulton
Corona Del Mar, CA

MICRO

One Month Added to All Subscriptions

Because of our combined April/May issue, we've gotten some questions from readers wanting to know if we were going to be bimonthly, if they were going to loose an issue, if we were taking a vacation early, etc.

The answer is much simpler. When we redesigned MICRO to make it more readable,we needed some extra time between issues to gear up our production department (artistic temperament and all that). So we gained the needed time by combining two issues.

It was a one-time thing. We are not going to be bimonthly. More importantly, you will not lose an issue. If you subscribed for 12 issues, you will receive just that — and the combined issue counts as only one. All subscriptions will be extended one month.

While we're on the subject of subscriptions, please check your mailing labels to be sure all information is correct; tell us about problems right away.

At last!... A dual 6522 versatile interface adapter (VIA) board for the Commodore-64.

The 6522 VIA, long the preferred input/output chip for 6502 microcomputers, is now available for the C-64. 6522 programming techniques, covered in many available books, can now be applied to the C-64 for even the most sophisticated real-time control applications. Board allows full use of the IRQ interrupt. When combined with the C-64's memory capacity, it provides an extremely powerful yet cost-effective development system and controller in one package. Includes extensive application notes and programming examples.

Up to four boards can be connected together, providing sixteen 8-bit ports. Order Model 64IF22, $169 for one, postpaid USA. Each additional $149.

Complete reconstructed Assembly Language source code for the C-64's BASIC and KERNAL ROMs, all 16 K!

Extensively commented and cross-referenced. Far more than a mere "memory map" of useful locations, this book really does tell all. An incredible time-saver in effective C-64 programming and understanding. Order C-64 Source $29.95, postpaid USA.

SCHNEDLER SYSTEMS
1501 N. Ivanhoe, Dept. M7
Arlington, VA 22205
Telephone orders/information: (703) 237-4796
VISA MASTERCARD

Sage Microcomputer System

Distributor

Sage Computer
4905 Energy Way
Reno, NV 89502

Introduction

The SAGE II is a fast 32-bit computer using the p-System Operating System with a 68000 Interpreter to emulate the 'p-machine.' SAGE chose this operating system for a number of reasons. To develop their own Operating System would have been time consuming and costly, and once it was finished they would be incompatible with everyone else. Instead they opted for a highly portable system which would allow programs to be transferred from one machine to another with very little difficulty. Portability being the key, many programmers purchased SAGEs to use as developmental tools. The SAGE also had the added attraction of being very fast. With these points in mind, the majority of the SAGEs sold during the first year were bought by programmers and developers. Since that time the market and support of the SAGE has greatly expanded.

The Processor

The SAGE II uses an 8mhz, interrupt driven 68000 microprocessor. It has a 16-bit data bus and a 24-bit address bus, directly addressing 16 million bytes. There are more than 1000 executable instructions, the set containing 56 instruction types with 14 different addressing modes. With 17 general purpose registers, each 32 bits long, a 24-bit program counter and a 16-bit status register, the SAGE is a powerful machine. Using an 8 Mhz clock the MC68000 (without wait states) runs at 2 million instructions per second. There is a light on the processor which indicates when the bus is active, inactive or the processor is in process.

Memory

RAM memory for the SAGE II is configurable from 128K to 1024K bytes in 128K increments. On the Main processor board (CPU board) up to 512K bytes may be stored, with an additional 512K on the Winchester board. A self-test, DEBUGGER, and bootstraps are in the EPROM firmware.

Keyboard and Physical Description

Basically a standard Qwerty keyboard, the entire unit is connected with a telephone-like cord allowing the user to move the keyboard to his lap or any convenient position. The basic alphanumeric keys are laid out in the usual manner with a numeric pad to the right. Above this pad are four programmable function keys (their function changing from program to program). The SAGE II is contained in an aluminum case measuring 3.5'' x 12.5'' x 17''. Weighing in at 15lb. 8 oz., it is easily moved.

Interfaces

SAGE decided to simplify I/O implementation by using I/O memory-mapped assignment. The connections provided are: Terminal - RS232-C, Modem - RS232-C, Printer - parallel, Group-A and B - dipswitch, and IEEE-488 -GPIB bus. A second RS232-C port is available. With the Winchester board 4 serial ports can be supported.

Documentation

The documentation we received included a Getting Started/Word Processing volume, a Technical Manual, and a p-System Operating System Manual. Each manual

was contained in a 3-ring hard-cover binder which fit into another hard-covered box. The documentation was clearly written, with indexes and table of contents that were very helpful. Most of the information was easily accessed and references were provided where appropriate.

Software

There are some fine software packages available for the SAGE II. These include some excellent business,spread-sheet and database products. As the SAGE II uses the p-System Operating System, it lends itself to easy transferral of software developed on other p-System machines. Given this portability of programs, I would expect a steady influx of software for this microcomputer.

Peripherals

The SAGE II supports single and dual disk drives, Winchester disk, dot matrix and daisy-wheel printers, monochrome and color monitors. The system came with a QUME monitor which is ergonomically designed (i.e., takes people into consideration). This was a very nice addition, being able to rotate and swivel the screen to avoid glare, and position the monitor to suit the user's preferences and body (tall, short, etc.).

Price

The SAGE II with one 640K floppy drive is listed at $3,200, with two 640K floppy drives it is listed at $3,900. If you choose to expand to 512K bytes of parity RAM (which is necessary for either the Sage Multi-User system or the Idris Operating System), it is an additional $500. The Qume CRT comes in a variety of flavors, prices ranging from $690 for the green QVT-102 to $1,310 for the amber QVT-211GX which has full graphics capabilities.

Conclusion

The SAGE II is a well designed and competent computer. SAGE is the only low-cost multi-user (2 users) and multi-tasking micro on the market. Allowing foreground and background activites to run concurrently, you can compile while using the word processor. Although this not the micro for everyone it is definitely one of the best 68000 micros currently available. For those who are interested in a more serious micro, particularly for developmental or business purposes this is definitely a machine worth considering.

MICRO

OS9

APPLICATION SOFTWARE

ACCOUNTS PAYABLE **$299**	GENERAL LEDGER with CASH JOURNAL **$399**	PAYROLL **$499**
ACCOUNTS RECEIVABLE **$299**		SMALL BUSINESS INVENTORY **$299**

COMPLETE DOCUMENTATION $19.95

OS9 & BASIC 09 ARE TRADEMARK OF MICROWARE, INC. & MOTOROLA CORP.

SPECIALTY ELECTRONICS

(405) 233-5564
2110 W. WILLOW — ENID, OK 73701

ISOBAR... most complete computer protection!

More features to prevent errors, false printout, disc skips! *Only ISOBAR has 3-way spike protection, noise suppression for RFI PLUS isolated filter banks!* Individual filter banks isolate each load from other loads minimizing data errors of *any* kind. MOV surge suppressors arrest both common mode and differential mode surges. L/C filter network rejects radio frequency noise at any amplitude. Torroidal coils for greatest efficiency! All-metal housing.

Order toll free 1-800-662-5021 IN ILLINOIS, CALL 1-312-642-6871

Indus-Tool, 325 W. Huron, Dept. M
Chicago, IL 60610 MasterCard VISA
Send model #________
Enclosed is $________ or charge on
☐ MasterCard or ☐ Visa *Expires*______
*Card no.*________
Name________
Signature________
Address________
City________ State____ Zip____

Model IBAR 4-6 (4 outlets, 6 ft. cord) **Only $79.95**

Model IBAR 2-6 (2 outlets, 6 ft. cord) **Only $54.95**

Model IBAR 8-15 (8 outlets, 15 ft. cord) **Only $97.95**

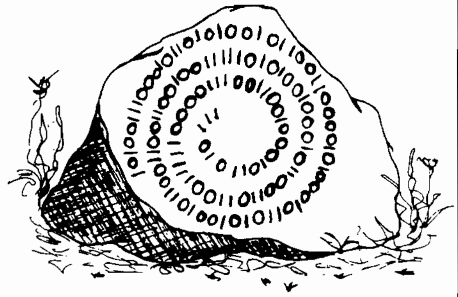
Why Winchester failed his computer course?

_ _ _ _ _ _ _ / _ _ / _ _ _ _ / _ _ _ / _ _ _ _ _ .

NCEMMUOIN	☐☐◯☐☐☐☐◯☐
LXEPI	☐◯☐☐☐
VICRSEURE	☐☐☐☐☐◯☐◯☐
RPHEOP	◯☐☐☐◯☐
EDPCOUESOD	☐◯☐◯☐☐☐☐☐☐
GRMIOTLAH	☐☐☐◯☐☐☐☐☐
CIKDOLRG	☐◯☐◯☐☐☐☐
CAUPDETIL	☐☐☐◯☐☐◯☐☐
ITUNENOMLITOEMCAC	☐☐☐◯☐☐☐☐☐☐☐◯☐◯☐☐☐
RIBYAN	◯☐☐☐☐☐
ILECDHAMXIE	◯☐☐☐☐◯☐☐☐☐☐
CBYISOML	◯☐☐☐☐☐☐☐

Last month we printed a puzzle, (see copy). The secret is now revealed -read the slashes and circles as ones and zeros, divide them into groups of seven, translate each group into its ASCII equivalent, then read the letters in reverse order; you get the following message -"welcome to lyte bytes."

This month to text your computer literacy we have a word scramble. To find the answer — first decipher each word and write it in the adjacent box; extract the letters that fall within the circles; take these letters and unscramble them to arrive at the final answer using the blank lines under the cartoon. We will of course provide the answer in next month's Lyte Bytes.



Product Name:	**Paint Magic**
Equip. Req'd:	Commodore 64 with disk drive, joystick and color monitor
Price:	$50
Manufacturer:	Datamost, Inc. 8943 Fullbright Avenue Chatsworth, CA 91311

Description: A graphics program that creates pictures with the help of a joystick and the keyboard. You advance from circles and boxes with one color fills, to sketches with self-designed color patterns which can be transposed, exchanged and saved for later recall. Portions of the screen can be magnified for detailed work. Sample pictures are provided to show you what Paint Magic is capable of.

Pluses: Any screens you design can be saved and included in your own BASIC programs. Because of the numerous color and pattern choices you have amazing flexibility to experiment with.

Minuses: Only five colors can be used at a time. A joystick with eight positions is essential and being able to select diagonal lockout is a very useful feature.

Documentation: An attractive and simple tutorial provides the needed information

Skill level: Beginner and up

Reviewer: Mike Cherry

Product Name:	**Time-Trax**
Equip. Req'd:	Apple II, II or IIe, monitor (preferably Black and White), disk drive, blank diskette, 2 AA alakaline batteries
Price:	$99.95
Manufacturer:	Creative Peripherals Unlimited, Inc. 1606 S. Clementine Anaheim, CA 92802

Description: An easy to use time management system, designed to help you keep track of events, scheduled meetings, etc., in your personal or business environment. One package can manage an infinite number of users. The program keeps a calendar of scheduled events for one year, and enables the user to print out a daily, weekly, or monthly schedule. It has a search of entries option, using keyword(s) and wildcards.

Pluses: Very simple to use, clean, clear and helpful menus. Hitting an escape (at most three times) will return you from anywhere in program to the main menu. Will not allow you to make an entry into the past. Has two kinds of cursors: blinking — displayed when you are to type information in; and non-blinking — displayed when you are to select an option. Retains data for the present month, and eleven months past and in the future, deleting any month that becomes 12 months old. Maximum of 311 entries per month or 9079 characters of text. Maximum of 99 entries per day. Good error messages. A clock is included (hardware and assembly instructions). This maintains the correct time and date, using two AA batteries as a backup. The clock itself makes this package worth the price. The clock can also be used in Applesoft BASIC or 6502 assembly language programs, a machine language program is included on the disk. Clear readable graphic display of calendar (month at a time).

Minuses: Time-Trax has a feature which reminds you of upcoming appointments and tells you when you have missed a scheduled event. A great idea, but one that is limited by the necessities of 1) your computer must be on, 2) it must be running Time-Trax, and 3) a menu or calendar must be displayed. If you haven't met these requirements your reminder becomes a missed event. Not very practical in practice, since most people will not choose to keep their computer always running and tie up their system with one program, i.e., Time-Trax. Rather, I suggest they should have made this a background instead of a foreground task.

Documentation: Thorough, easy to understand. Unlike much documentation, an index has been provided.

Skill level: Beginner and up.

Reviewer: Mark S. Morano

Product Name:	**Promenade model C1 EPROM Programmer**
Equip. Req'd:	Commodore 64 or VIC-20 Computer, Disk or Tape
Price:	$99.50 plus $3 postage/handling
Manufacturer:	JASON-RANHEIM 580 Parrott Street San Jose, CA 95112

Description: The Promenade is a highly capable EPROM programmer which operates from the User Port of the VIC or C-64 computers. It can program at least 12 models of 5-volt only EPROM (Erasable Programmable Read Only Memory) ranging in size from 1K x 8 to 32K x 8 and 8 models of EEPROM (Electrically Erasable PROM). In addition to programming EPROMs and EEPROMs (and erasing EEPROMs) the unit will save assembly language object code (as will any programmer) and also will put BASIC object code into ROM. An auto-start loader is furnished which can make any ROM auto-start when plugged into the computer's expansion port. Promenade's own software will put several BASIC programs on an EPROM, along with a directory of those programs. Thus, working programs can be "cast in silicon" on EPROM and simply plugged in to change job assignments for a computer. This feature is being widely used in industry where the low cost of a VIC-20 makes it attractive to dedicate a computer. The ease of BASIC programming and subsequent installation of the program in EPROM, allows major cost savings for computerized projects. Rapid turnaround of modified programs is possible with EEPROMs: the time for erasure and reprogramming an EEPROM can be as short as 2 minutes or less!

Pluses: This package outperforms most other add-on programmers, yet the cost is lower than any I've heard of. If you have the computer, all you need is mass storage, a Promenade and EPROMs to start generating programs which don't go away if the power fails. It is rugged, attractive, highly engineered and well made. Their immediate concern is to get the customer's problems solved as promptly as possible, even if this requires express mail delivery of a replacement unit.

Minuses: The major lack of this equipment is in documentation for programming EPROMs with assembly object code, and on how to manipulate assembly files with a debug monitor co-resident with the Promenade software. Everything works well together - it is just hard to learn how from the documentation. It is my personal prejudice that electrical schematics should be furnished with all electronic products, but the low cost of Promenade overcomes this feeling somewhat.

Documentation: A 16 page manual (but no schematic) is furnished. It covers saving BASIC programs to EPROM in meticulous detail. The manual is not well organized, but it is small enough that everything can be found rather easily. Documentation regarding use of Promenade for "normal" assembly-language programming is very sparse.

Skill level: In general, using EPROM programmers requires considerable knowledge about preparing assembly code for use in a read-only environment. However, this combination of equipment and documentation should allow inexperienced persons to save BASIC programs readily.

Reviewer: Ralph Tenny

Product Name:	**Spell Perfect**
Equip. Req'd:	Apple II w/48K and drive
Price:	$89.95
Manufacturer:	LJK Enterprises, Inc. 7852 Big Bend Blvd. St. Louis, MO 63119

Description: A machine-language spelling checker program operating on Letter Perfect or any standard text files. It is compatible with most 80 column cards and has a file buffer of over 40,000 characters. Words are easily added to the dictionary from corrected documents and up to 255 dictionary disks are allowed - the program prompts for disk insertions.

Pluses: The well written manual is not needed for the most part being menu driven and having easily understood prompts. The program is fast (a 100 sector file took less than 2 minutes) and offers words to be corrected in context with the surrounding text. A "help" command is available to prompt you with similar sounding words from the dictionary or you can edit the word in place.

Minuses: The program doesn't recognize " ' " or " - " leading to problems with hyphenated or contracted words. A prompt to add word to dictionary instead of rerunning the program on the corrected file would be nice.

Documentation: The 72 page manual nicely complements the on-line prompting and answers all questions with specific examples.

Skill level: No particular computer knowledge necessary.

Reviewer: Phil Daley

Product Name:	**The Complete Graphics System**
Equip. Req'd:	Apple II, II, IIe, Color Monitor, disk drive, extra diskettes for backup copies and work disks
Price:	*
Manufacturer:	Penguin Software 830 4th Avenue P.O. Box 311 Geneva, IL 60134

Description: As the title says, this is a complete graphics system. Easy enough for those who aren't programmers and sophisticated enough for those who are. You can create two and three dimensional graphics, use 108 blended colors, outline areas, fill them in, draw with lines, brushes (96 choices), use freehand drawing, employ preprogrammed boxes, arcs, circles, triangles, and ellipses. There is a program in which you can create your own shapes, store them in a table, and then draw on them whenever you choose. A variety of input devices are compatible: ordinary keyboard, joystick, trackball, touch tablet, paddles, Apple graphics tablet, a mouse, and Houston Instruments HiPad. (What's left?) An object can be magnified 2, 4, or 8 times its original size, rotated, shrunk, varied in intensity, and easily transferred to any drawing. Text can be added to graphics using another special program. As originally stated — this is a complete graphics system.

Pluses: The pluses are many. The fact that it can do all of the above is a plus; that it does them well merits special applause.

Minuses: Overall, there is no such thing as a perfect graphics package. There will always be flaws. As far as minuses go with this product they are truly insignificant, bordering on non-existent.

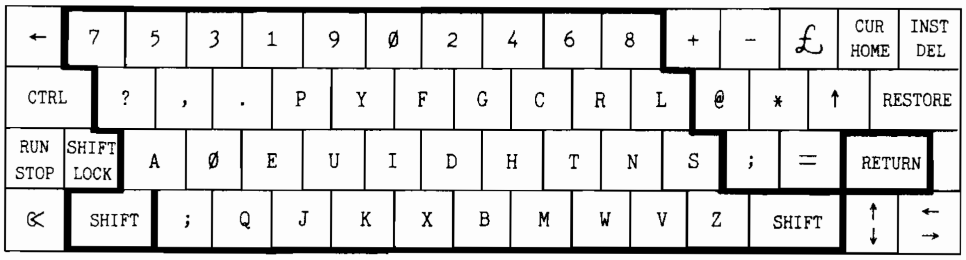
Documentation: The documentation is generally clearly written. There are some sections that could be more lucid, but with some rereading most everything can be figured out.

Skill level: Intermediate to advanced.

Reviewer: Mark S. Morano

MICRO

A Basic DVORAK Keyboard for the VIC-20 and Commodore 64



by Alfred J. Bruey

The current keyboard was designed to slow typists down. A new arrangement can increase productivity enormously

At the 1876 Centennial Exposition one exhibitor presented a strange gadget which is now known as the "typewriter." It did not receive as much attention as it should have because this new, practical discovery was overshadowed by the "telephone," another strange new invention.

One of the first typewriter designers, Christopher Sholes, found that if the keys were arranged in a reasonable order, they would jam because of their slow action. So he rearranged them so the keys that were often hit together would not get tangled with each other. His arrangement, which assigns the letters QWERTYUIOP to the top row of alphabetic keys, is still used today. I will refer to this arrangement as the QWERTY keyboard, for obvious reasons. If there is a QWERTY keyboard, there must, of course, be a non-QWERTY keyboard. Otherwise, what would I be writing about?

Actually, there are, or have been, many non-QWERTY keyboards. The one that I'll be discussing here, the Dvorak keyboard, was designed by August Dvorak in the 1930's. Dvorak wasn't the first to develop a non-QWERTY keyboard; in the last quarter of the nineteenth and first quarter of the twentieth century, there were a great variety of typewriter keyboard arrangements from which to choose. When I was collecting old typewriters a few years ago, before a lack of storage space put an end to that hobby, I found that probably the easiest-to-find non-QWERTY keyboard was found on the old Oliver typewriter whose model numbers went all the way to Number 9 before they were discontinued.

The DVORAK Keyboard

Figure 1 shows a drawing of the VIC-20 and C-64 keyboard with the commonly used keys changed to represent a simplified version of the Dvorak keyboard. Notice that no attempt was made to incorporate all the special characters. The arrangement in this figure follows that shown in an article (Dvorak Keyboard for Your Computer) by John Raines in the August, 1983 issue of MICRO Magazine. This article presented a 6502 machine language program for the Apple Computer, which allows the Dvorak arrangement to be used to input data to Apple programs.

The VIC DVORAK Program

The Dvorak keyboard program shown in Listing 1 is a demonstration program that you can run to see whether or not you like this "new" arrangement. All it does is put whatever you type on the screen.

The program logic is straightforward. A GET instruction is used to get characters, one at a time, from the keyboard buffer. Then the ASCII value of the character is obtained. A conversion table, entered with a DATA and READ statement, is used to convert the QWERTY characters to the equivalent Dvorak keyboard positions. Then the character is printed on the screen (in

Listing 1

```
10 DIM MT(47)
50 DATA 87,ØØ,86,9Ø,56,55,53,51,49,57,48,5Ø,52,54,83,ØØ
55 DATA ØØ,ØØ,ØØ,ØØ,ØØ,65,88,74,69,46,85,73,68,67,72,84
6Ø DATA 78,77,66,82,76,63,8Ø,79,89,71,75,44,81,7Ø,59
65 FOR I=1 TO 47:READ MT(I):NEXT I
1ØØ GET K$:IF K$=""THEN 1ØØ
1Ø1 K=ASC(K$)
1Ø2 IF K=6Ø THEN K=44:UC=1:GOTO 115
1Ø3 IF K=62 THEN K=46:UC=1:GOTO 115
1Ø4 IF K=63 THEN K=47:UC=1:GOTO 115
1Ø5 IF K=91 THEN K=58:UC=1:GOTO 115
1Ø9 UC=Ø:IF K< >1ØØ THEN UC=1:K=K-128
11Ø IF K$=CHR$(13) THEN PRINT CHR$(13);:GOTO 1ØØ
111 IF K$=CHR$(32) THEN PRINT CHR$(32);:GOTO 1ØØ
115 PRINT CHR$(MT(K-43)+128*UC);
12Ø GOTO 1ØØ
```

lines 110, 111, or 115). Then execution is returned to line 100 to GET the next character.

Using the Program

First press the SHIFT and COMMODORE keys to put the VIC into text mode. Next load the program (QWERTY LOAD translates to Dvorak NRAE) and the RUN it (RUN becomes PGB). Then you begin typing as though you had a Dvorak keyboard. When you are done using the program, press the RUN/STOP key to get out of the program and revert to the QWERTY keyboard.

Notice that only the characters outlined in the heavy black lines in Figure 1 are defined. You can use other characters, but you will probably get the message

```
?ILLEGAL QUANTITY ERROR IN 115
```

if you do.

Changing Your Keyboard

There are various ways to change your keyboard:

1. The easiest way is to put squares of masking tape on the keytops and write on the proper leters with a felt-tip pen. You might write the QWERTY symbols in one corner of the tape and the DVORAK in another.

2. You can change keycaps. This is not a trivial task and you should consider it only if you are making a permanent change.

3. Another temporary solution is to put the Dvorak character on tape on the front of the keycap, the way APL characters are often imprinted on keys. These characters can also be painted on the keyfronts for a permanent change.

Getting New Keyboard Arrangements Adopted

The major problem in trying to get a new keyboard arrangement adopted is that there are millions of people trained on the QWERTY keyboard. Another problem is that there are millions of QWERTY Keyboards in use. Tests performed since the 1940's have shown convincingly that it does not take long for the increased productivity possible with the Dvorak keyboard to recover the investment in re-training QWERTY typists on Dvorak keyboards. But many companies don't have the money to hire replacement help to keep up with the day-to-day work as their typists are being retrained. They also do not have the money to replace all their QWERTY hardware.

A simple solution to the hardware problem is in sight. The availability of computers with programmable keyboards makes it possible for users trained on two different keyboards to use the same computer (at different times, of course) by plugging in differently defined keyboards. By using this method, companies can gradually switch their employees to the Dvorak layout. A Dvorak keyboard is already available as an option for the IBM PC.

Program Extensions

As this program now stands, it is only useful as a demonstration of the Dvorak keyboard. You can't use this program to input data into a different program without some programming effort.

1. You can change this program to an input subroutine which you can attach to a more useful program. Then you can use the subroutine to enter data for the main program.

2. If you are going to use the Dvorak keyboard for your permanent keyboard arrangement, you will probably want to re-write this technique in machine language and use this program as a replacement for your computer's input routine. You can get help doing this from the MICRO article referenced earlier.

3. You might want to extend this system to handle the characters that I didn't include in my program.

4. You can add coding to print the characters on the printer as well as the screen, so you can have a record of your typing progress if you are using this program to learn the new keyboard.

MICRO™

Sure it's insured?

SAFEWARE™ Insurance provides full replacement of hardware, media and purchased software. As little as $35/yr covers:
• Fire • Theft • Power Surges
• Earthquake • Water Damage • Auto Accident

For information or immediate coverage call:

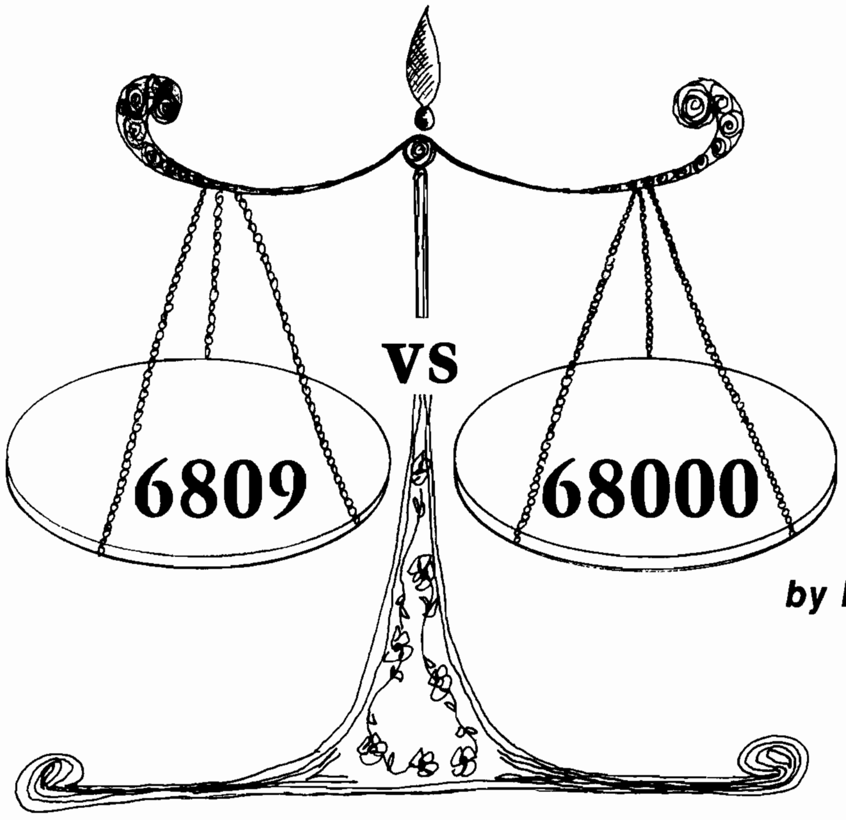
1-800-848-3469

In Ohio call (614) 262-0559

SAFEWARE™

SAFEWARE, THE INSURANCE AGENCY INC.

A Comparison:



by Mike Rosing

The checkbook offers a simple but effective way to compare these two microprocessors

The 6809 microprocessor is found in several computers, including the Radio Shack color computer which is available just about anywhere. The 68000 microprocessor is also found in several computers. Some of these are APPLE's LISA and MACINTOSH computers and the SAGE II. While the 68000 based machines can cost 10 times the price of the 6809 based machines, they are easily 100 times more powerful.

To compare these two machines at the machine level requires a specific project; the check book is simple, but illustrative. This requires addition, subtraction, movement of values, the conversion of ASCII to binary. What follows is not a complete program. It does contain the main subroutines required to create a simple check book program in machine language on either the 6809 or 68000.

To avoid rounding problems the choice of integer arithmetic is preferred. The smallest unit of money is the penny, so all calculations are done in pennies.

Next we have to decide the maximum value with which we are going to deal. This value should be a power of two and so large that we will never reach it. Since 16 bits leaves us with $327.67 as a maximum value we take 32 bits as the size. This gives us $21,474,863.54 as a maximum value. Very few check books exceed this value (positive or negative).

Good machine code writing involves subroutines. Because the comparisons here are so simple, the subroutines may look silly. Remember that the purpose is comparison and not necessarily good code.

An implicit assumption in these subroutines is that some operating system is involved. Thus the user stack on the 6809 is presumed to be initialized. The 68000 is presumed to be in user mode and the stack pointer is initialized.

Movement

The first subroutine (MOVEATOB) is to move a quantity from point A to point B in memory. The 6809 code requires two load and two store instructions. These destroy the A and B registers so they are pushed on the stack before and recovered at the end of the subroutine. The 68000 code can move 32 bits from memory to memory in one instruction without disturbing any other registers.

Addition

Next we need a subroutine to add numbers into an accumulator (see SUM). For the 6809 adding the least significant 16 bits is no problem. Since the carry can not be added to the D register, we have to go to byte addressing to sum the most significant bytes. Another way to do this is to create a loop count with the B register and use it as an offset. This runs slower than straight inline code.

The 68000 code can add 32 bit quantities in a single crack, so there is no need to worry about the carry bit. The ADD instruction is not as powerful as the MOVE instruction. It can only add with a data register. So we bring the 32 bit value into a data register and then sum this into the accumulator. Note that the MOVEM (move multiple registers) can be used with a single register as well as many registers.

ASCII to Binary

The simple example so far has assumed that the numbers are already in memory. Since most computers have keyboards which work in ASCII, we need a routine (GETNUM) to convert an ASCII string to a binary number which our subroutines can then add. Every operating system has its own method of getting characters from the keyboard. Here we assume that a subroutine can be written called GETBYTE which will return a byte from the keyboard into a register.

Once the string is pulled into memory and all the digits are in the range ASCII '0' to ASCII '9', the process of conversion can begin. Multiplying the result by 10 and adding in each byte of the string converts from human base 10 to computer base 2. A simple way to multiply 32 bits by 10 is to first multiply by 2 and save this in a temporary location. Then multiply by 4 (giving a final multiplication by 8) and add in the temporary value. Multiplication by 2 consists of a shift left.

For the 6809, the subroutine ROTL rotates the result area left one bit. Calling this 3 times with a MOVEATOB and the SUM subroutines completes the multiplication. Finally, a digit from the input string is masked off and added to the result. The addition requires propagating the carry

```
* MOVING 32 BIT VALUES CODE COMPARISON
*
*   6809 CODE
*  SUBROUTINE MOVEATOB MOVES A 32 BIT VALUE POINTED
*  TO BY X TO THE PLACE POINTED TO BY Y.
*
MOVEATOB: PSHU     D      SAVE D REGISTER
          LDD      ,X     GET 16 BITS
          STD      ,Y     SAVE 16 BITS
          LDD      2,X    NEXT 16 BITS
          STD      2,Y    SAVED
          PULU     D      RECOVER D REGISTER
          RTS             AND LEAVE
*
*  68000 CODE
*  SUBROUTINE MOVEATOB MOVES A 32 BIT VALUE POINTED
*  TO BY A0 TO THE PLACE POINTED TO BY A1.
*
MOVEATOB: MOVE.L  (A0),(A1)  MOVE 32 BITS
          RTS                AND LEAVE
*
-----------------------------------------------------
*
* SUMMING 32 BIT VALUES CODE COMPARISON
*
*  6809 CODE
*  SUM ADDS A 32 BIT NUMBER POINTED TO BY X TO AN
*  ACCUMULATOR POINTED TO BY Y.
*
SUM:      PSHU     D      SAVE REGISTER
          LDD      2,X    GET LEAST SIGN. BITS
          ADDD     2,Y    ADD TO ACCUMULATOR
          STD      2,Y    SAVE RESULT
          LDA      1,X    ONE BYTE UP
          ADCA     1,Y    ADD IN CARRY TO NEXT BYTE
          STA      1,Y    SAVE BYTE
          LDA      ,X     MOST SIGN. BYTE
          ADCA     ,Y     ADD TO ACCUMULATOR AND CARRY
          STA      ,Y     SAVE RESULT
          PULU     D      RESTORE REGISTER
          RTS             AND LEAVE
*
*  68000 CODE
*  SUM ADDS A 32 BIT NUMBER POINTED TO BY A0 TO AN
*  ACCUMULATOR POINTED TO BY A2
*
SUM:      MOVEM.L D0,-(SP)  SAVE A REGISTER
          MOVE.L  (A0),D0   GET NUMBER
          ADD.L   D0,(A2)   SUM INTO ACCUMULATOR
          MOVEM.L (SP)+,D0  RECOVER REGISTER
          RTS               AND LEAVE
*
-----------------------------------------------------
*
*  CONVERTING ASCII TO BINARY CODE
*
*  6809 CODE
*  GETNUM BRINGS AN ASCII STRING INTO MEMORY AND CONVERTS
*  IT TO A BINARY NUMBER. ALL ENTRIES ARE IN PENNIES.
*  ENTER WITH X POINTING TO PLACE FOR NUMBER TO GO.
```

```
*
GETNUM:   PSHU    D,X,Y     SAVE REGISTERS
          CLR     3,X       ZERO RESULT AREA
          CLR     2,X
          CLR     1,X
          CLR     ,X
          LEAY    INSTRING  POINT TO INPUT AREA
GNLOOP:   BSR     GETBYTE   GET BYTE FROM KEYBOARD
          CMPA    #13       WAS IT A CARRIAGE RETURN ?
          BEQ     KRUNCH    THEN PROCESS STRING
          CMPA    #'Ø'      WAS IT TOO SMALL ?
          BLT     GNLOOP    THE IGNORE IT
          CMPA    #'9'      WAS IT TOO BIG ?
          BGT     GNLOOP    THEN IGNORE IT
          STA     ,Y+       SAVE BYTE INTO STRING
          BRA     GNLOOP    AND GET NEXT CHARACTER
*
*  HAVE STRING IN MEMORY, NOW PROCESS IT.
*
KRUNCH:   CLR     ,Y        MARK END OF STRING
          CLR     COUNT     BYTE COUNT INTO STRING
*
*  MULTIPLY RESULT BY TEN.
*
CNVRT:    LEAY    TEMP      POINT TO TEMP AREA
          BSR     ROTL      RESULT TIMES 2
          BSR     MOVEATOB  PUT INTO TEMP
          BSR     ROTL      RESULT TIMES 4
          BSR     ROTL      RESULT TIMES 8
          EXG     X,Y       ADD RESULT
          BSR     SUM       TO TEMP AND SAVE
          EXG     X,Y       INTO RESULT
          BCS     TOOBIG    ERROR: NUMBER TOO BIG
*
*  ADD IN BYTE FROM STRING
*
          LEAY    INSTRING  GET NEXT
          LDA     COUNT     BYTE
          LDB     A,Y       FROM STRING
          BEQ     DONE      NO MORE TO DO
          CLRA              HIGH BYTE OF D CLEARED
          ANDB    #15       KEEP LOW NIBBLE ONLY
          ADDD    2,X       ADD IN RESULT
          STD     2,X       SAVE RESULT
          BCC     BMPCNT    NO CARRY TO PROPOGATE
          LDA     1,X       ADD IN
          ADCA    #Ø        CARRY BIT
          STA     1,X       TO EACH
          BCC     BMPCNT    BYTE IF
          LDA     ,X        NECESSARY
          ADCA    #Ø
          STA     ,X

*  BUMP TO NEXT BYTE IN STRING AND CHECK FOR DONE
*
BMPCNT:   INC     COUNT     BUMP STRING COUNTER
          LDA     COUNT     DONE WITH STRING ?
          TST     A,Y
          BNE     CNVRT     NOT YET
DONE:     PULU    D,X,Y     RECOVER REGISTERS
          RTS               AND LEAVE
*
*  ERROR HANDLER WILL BE MACHINE DEPENDENT
*
TOOBIG:   (SEND ERROR MESSAGE TO SCREEN)
*
*  DATA AREA
*
INSTRING: 2Ø BYTES
TEMP:      4 BYTES
COUNT:     1 BYTE
```

through all 32 bits of the result. The loop is repeated until all string digits have been converted or an error occurs.

Comparing the 68000 version of GETNUM to the 6809 version, we see that one instruction of the 68000 does the same as two calls to a 10 line subroutine of 6809 code. To shift 32 bits left once, takes ROTL for the 6809. To shift 32 bits left twice, takes only one line of code for the 68000. The number of registers on the 68000, reduces a lot of memory requirements. While the 6809 must continually swap pointers from register to memory, the 68000 keeps all values in registers, for this simple example at any rate.

Conclusion

These simple comparisons are intended to be educational. Experience with the 68000 sometimes makes writing code on the 6809 frustrating. The ability to address 16 megabytes of RAM on the 68000 versus 64 kilobytes on the 6809 makes one wonder if the term "micro" really applies anymore.

The reduced coding required for the 68000, increases programmer productivity and decreases the time for producing a final result. Obviously, there are many ways to solve each problem. The flexibility of the 68000 and the number of registers, makes this microprocessor the most powerful chip to date. While the 6809 makes a great home based computer, the power of the 68000 makes it far more useful in the business or scientific environment.

Bibliography

"MC6809 Preliminary Programming Manual", Motorola Inc., 1979

"Color Computer Assembly Language Programming", William Barden, Radio Shack

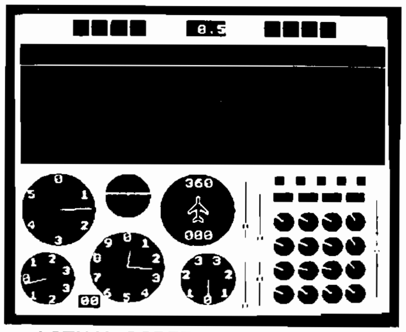
"16-Bit Microprocessor Users Manual", Motorola, Prentice-Hall, 1982

"Motorola Microprocessors Data Manual", Motorola, 1981, pgs. 4-298 to 4-329 and pgs. 4-661 to 4-710

Mr. Rosing received a B.S. Engineering Physics from Univ. of Colorado in 1976, and a Ph.D. in Nuclear Engineering from Univ. of Wisconsin in 1982. He is presently Chief Engineer for Network Telecommunications in Denver.

ATARI 48K * TRS C/C 32K COMMODORE 64

747 FLIGHT SIMULATOR



ACTUAL SCREEN PHOTOGRAPH

Superbly realistic instrumentation and pilot's view in lifelike simulation which includes emergencies such as engine fires and systems failures. This program uses high resolution graphics to the full to produce the most realistic flight-deck display yet seen on a home computer. There are 21 real dials and 25 other indicators. Your controls operate throttle, ailerons, elevators, flaps, slats, spoilers, landing gear, reverse thrust, brakes, etc. You see the runway in true perspective. Uses joysticks and includes options to start with take-off or random landing approach. A real simulation, not just another game! **Cassette only, $27.95 (add 6% in Calif.). Sole U.S. distributor for D.A.C.C. Ltd., England.**

F. Ashton
P.O. Box 7037
Chula Vista, CA 92012

Come See Us at Commodore Convention

Sat. & Sun, July 28-29
Hershey, Pennsylvania

The show is being run by the Mid-Atlantic Regional Commodore Association (MARCA) and there will be speakers, workshops, and exhibitors. For info. call Mindy at 717/486-3274.

It's an easy trip from all areas of the east, from CT to VA, and a lovely place with lots of family fun.

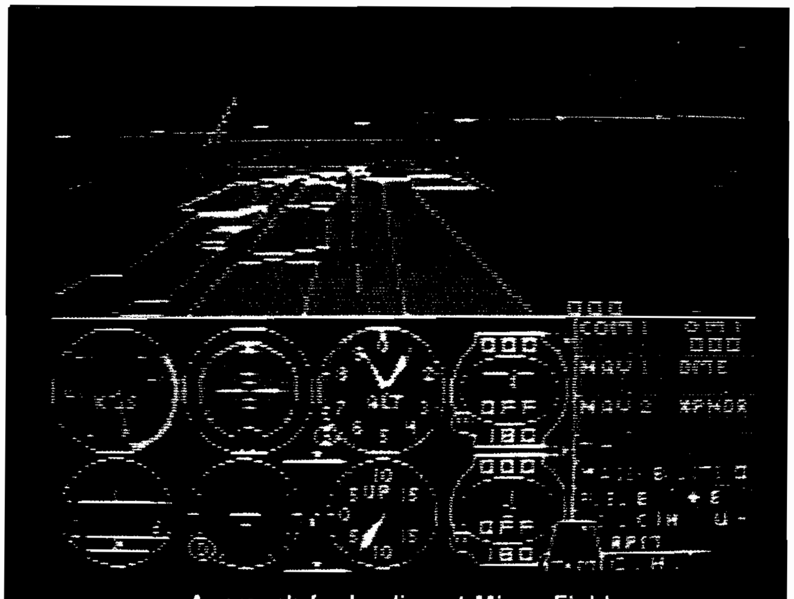
Come visit us at
Booth Number 101
try some of our latest software packages, and say "Hi" to the MICRO Staff.

```
*
*  SUBROUTINE TO ROTATE 4 BYTES LEFT ONCE
*  ENTER WITH X POINTING TO BYTES TO ROTATE
*
ROTL:       PSHU     D          SAVE REGISTER
            ANDCC    #Ø         CLEAR CARRY BIT
            LDB      #3         SET COUNTER
ROTLOOP:    LDA      B,X        GET BYTE
            ROLA                TIMES 2
            STA      B,X        SAVE BYTE
            DECB                DO 4 TIMES
            BPL      ROTLOOP
            PULU     D          RECOVER REGISTER
            RTS                 AND LEAVE
*
*  68ØØØ CODE
*  GETNUM BRINGS AND ASCII STRING INTO MEMORY AND
*  CONVERTS IT TO A BINARY NUMBER. ENTRIES ARE
*  ASSUMED TO BE IN PENNIES. ENTER WITH A3
*  POINTING TO THE PLACE FOR THE RESULT.
*
GETNUM:     MOVEM.L DØ-D2/AØ,-(SP)  SAVE REGISTERS
            LEA      INSTRING,AØ    POINT TO INPUT AREA
GNLOOP:     BSR      GETBYTE        GET KEYBOARD INPUT
            CMP.B    #13,DØ     WAS IT A CARRIAGE RETURN ?
            BEQ      KRUNCH     THEN PROCESS STRING
            CMP.B    #'Ø',DØ    WAS IT TOO SMALL ?
            BLT      GNLOOP     THEN IGNORE IT
            CMP.B    #'9',DØ    WAS IT TOO BIG ?
            BGT      GNLOOP     THEN IGNORE IT
            MOVE.B   DØ,(AØ)+   SAVE BYTE INTO STRING
            BRA      GNLOOP     AND GET NEXT BYTE
*
*  HAVE STRING IN MEMORY. NOW PROCESS INTO BINARY
*
KRUNCH:     CLR.B    (AØ)       MARK END OF STRING
            CLR.L    D1         CLEAR RESULT
            LEA      INSTRING,AØ    POINT TO TOP OF STRING
*
*  MULTIPLY RESULT BY TEN
*
CNVRT:      LSL.L    #1,D1      RESULT TIMES 2
            MOVE.L   D1,D2      SAVE THIS RESULT
            LSL.L    #2,D1      RESULT TIMES 4 MORE FOR 8
            ADD.L    D2,D1      ADD IN 2 FOR 1Ø TIMES
            BVS      TOOBIG     NUMBER TOO BIG
*
*  NOW ADD IN BYTE FROM STRING
*
            MOVE.B   (AØ)+,DØ   GET BYTE FROM STRING
            AND.L    #15,DØ     MASK OFF ALL BUT LOW NIBBLE
            ADD.L    DØ,D1      ADD TO RESULT
            BVS      TOOBIG     TOO MANY DIGITS
            TST.B    (AØ)       DONE YET ?
            BNE      CNVRT      NOPE, KEEP ADDING BYTES
            MOVE.L   D1(A3)     SAVE RESULT MEMORY
            MOVEM.L (SP)+,DØ-D2/AØ  RECOVER REGISTERS
            RTS                 AND LEAVE
*
*  SUBROUTINE TO SEND ERROR MESSAGE TO SCREEN
*
TOOBIG:     SEND ERROR MESSAGE TO SCREEN
*
*  DATA AREA
*
INSTRING: 2Ø BYTES
*
*
```

Flight Simulator II
Microcomputer Simulation At Its Best

by Chris Williams

By analyzing this design masterpiece, programmers may discover the elements needed to make their own software great


Approach for landing at Miegs Field

Until now, simulations designed for microcomputers have been unexciting, crude approximations of whatever real-life phenomenon they were trying to model. They were slow. They lacked detail. And all too often, the modeling equations employed were out-and-out wrong. But no longer. A company called SubLogic Corporation has seen fit to single-handedly advance the state-of-the-art in microcomputer simulation technology beyond its childhood stage into exciting, energetic adolescence.

SubLogic was the manufacturer of Flight Simulator, the first popular microcomputer flight simulation. It was designed to run on a 16K Apple II, and it did so -- more or less. Amid relatively little fanfare, they've now released a sequel designed for the newer crop of Apples that sport 64K. There are also versions out for other machines. They call it Flight Simulator II, but there all similarity between sequel and original ends.

Flight Simulator was revolutionary in its day. No one had done a flight simulation on a microcomputer before Bruce Artwick, co-founder of SubLogic, worked his magic. The final product ran reasonably well, but it was slow and the graphics lacked pizazz.

Not so with Flight Simulator II. The screen updates are faster and detailed scenery for four different parts of the U.S. are included with the package. Additionally, the company advertises the availablility of scenery disks for other areas of the U.S. It all makes for a degree of realism never before approached on a microcomputer.

Flight

The airplane modeled in Flight Simulator II is a Piper PA-28-181 Archer II; a single engine, 148 mph., non-retractable gear general aviation aircraft. In real-life, the Archer II performs very well while remaining easy to fly. It is, consequently, an excellent choice for the product.

The simulation flight controls are on the keyboard. SubLogic includes helpful cue-cards with the package that specify which keys do what. As a pilot, I found flying with keys instead of a control yoke and rudder pedals disconcerting at first, but I soon adjusted. At my request, other pilots tried it and agreed the adjustment came easy. A non-pilot would probably never notice.

The layout of the keyboard is fascinating and all computerists writing user-interactive routines could learn from it. The T,F,H,B diamond is used as the control yoke of the aircraft. It's perfect for one hand operation and easily learned.

But it's in the use of the G key that something innovative has been added. Whatever the value of the aileron control variables (set by F and H), they are nulled to neutral with a single press of G. Without this, several key presses of either F or H would be necessary to return a given setting to zero. They gave this problem a lot of thought and came up with an excellent answer.

Some of the most interesting features of the product are in the navigation and communications radios. Here the simulation uses cntl-C and cntl-N followed by greater-than or less-than signs to simulate changing a frequency. This is a good choice as cntl keys are generally a bit awkward. Why is that good? Because nothing in flying is as awkward as changing radio frequencies in turbulence. Making it difficult on the simulation is entirely appropriate.

The Editor

The product includes a particularly valuable feature called "The Editor". At any time during flight, a touch of the ESC key sends you to The Editor, and from there you can change the current flight situation to be anything you wish.

The procedure is interesting and, again, programmers should take note. When you press the ESC key, a menu entitled "Simulation Control" is displayed. The menu is two pages long. Moving off the bottom of one page automatically sends you to the other. These two pages contain a list of simulation variables and their current values. By positioning the cursor at the proper variable line and entering a new value, the user can quickly change his situation without having to fly into it.

There are two valuable applications for this feature. First is the ability to set North and East coordinates which allows the user to instantly change from, say, the Chicago scenery area to the Boston-N.Y. scenery area without a time consuming crossing of the intervening distance between.

The second valuable application has to do with Critical Attitude Recovery. CAR is required by the FAA (Federal Aviation Administration) as an integral part of the instrument flight training curriculum for pilots attempting to add an instrument rating to their license. CAR is taught in an actual airplane, generally as follows. The student, wearing a hood to restrict his vision to the instrument panel, is told to close his eyes or cover them while the instructor takes control of the aircraft. The instructor then places the aircraft in an "unusual" or "critical" attitude. This is typically an extreme nose high or low configuration with a very steep bank included.

After a few seconds delay (to let the gee-forces confuse the student's equilibrium), the instructor tells the student to open his eyes and, using no outside visual references (i.e., instruments only), recover the aircraft to normal, straight-and-level flight.

The Editor allows a user to practice this procedure. Extreme values for the pitch, roll, and yaw variables can be entered at the Simulation Control menu and then, when the user exits Edit mode, he is faced with a critical attitude. Recovery technique is the same on the simulator as in real life so the exercise is excellent practice.

The Weather

Any pilot will tell you that the single most important factor in flying is the weather. Winds aloft, turbulence, and clouds often determine more about a flight than the pilot's wishes. Therefore, a simulation predicated on its accuracy in modeling real-life operation must have user variable weather. Naturally, Flight Simulator II does.

This is another area where the computerist can learn from what SubLogic has done. They've devoted attention to detail and implemented features to promote realism even where it makes the programming complex. Having this sort of professional attitude is probably more important than sheer technical skill in producing excellence in a program.

SubLogic handled the weather by allowing the user to define two layers of clouds and four of wind. Wind adjusts the airplane's ground speed for given airspeeds and clouds simply clear the screen to white when the airplane is at a blanketed altitude. With cloud bases set at about 500 feet, the airplane "breaks out" on an ILS (Instrument Landing System) instrument approach lined up nicely with the runway, making final descent and landing both easy and immensely satisfying.

Incidently, when the #1 Nav. radio is tuned to the ILS frequency, the glideslope needle on the indicator becomes active. The Localizer needle acts as it does for all the VOR navigational beacons. The pilots reading this will appreciate the level of detail SubLogic is covering there.

Turbulence is also permitted as a user-defined feature. Its effect is random motion of the instruments which makes the airplane harder to fly.

Lastly, the user can specify a given season. The effect of this is to change the time of day when night falls. Oh yes, there's a night mode, and it *is* hairy. Would you have expected anything less?

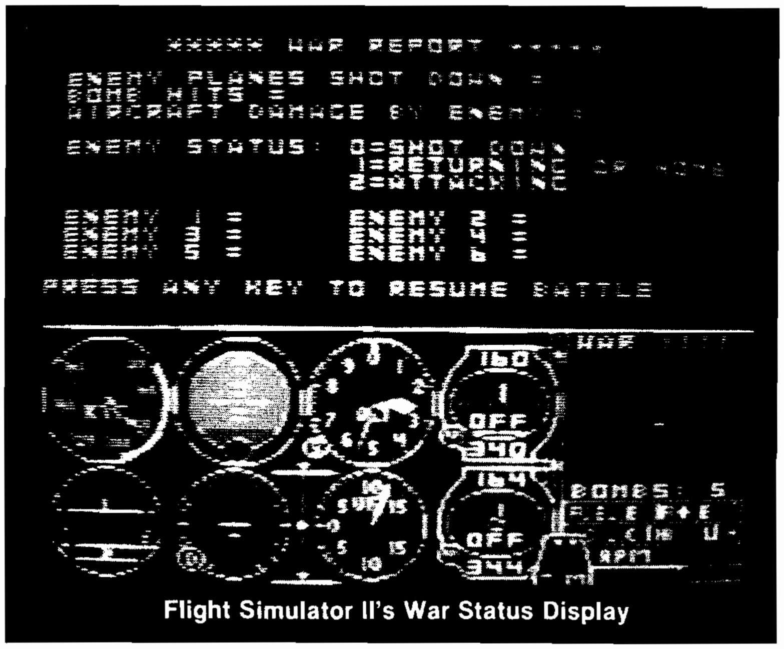
Seeing the World

The reason most pilots love to fly is nowhere near as esoteric and romantic as they'd have you believe. It's really very simple. The higher you are, the more pleasant things you can see. Flight Simulator II was clearly designed with that in mind. The original Flight Simulator was a forward-looking simulation that had nothing of consequence to see in its database. This product allows the user to look in all directions by using a special key sequence. Such is the attention to detail that when you look out the rear window of the cabin, the rudder is superimposed on the screen as a thick vertical line. And, of course, when you look out the side, the wingtip is prominent at the bottom of the screen.

There's another viewing mode included that is not realistic. It's called Radar Mode. In this mode, the user can get a top view of the world and an impression of where the airplane is with respect to landmarks. This is unavailable on a real airplane and therefore somewhat bizzare, but for users to whom flying is unfamiliar it probably is a valuable, perhaps even vital, feature.

Emergency Procedures

What do you do if the engine quits? That is the first question people new to single-engine flying ask. The answer (which I've found is always responded to with a chuckle) is to execute the emergency procedures all pilots are trained to perform. But there are also other emergencies in flying that a pilot can encounter. Flight Simulator II has a feature that will throw them all at a pilot randomly to see how he reacts. It's called the Reliability Factor. This is a number the user selects from the Editor's Simulation Control menu. Anything less than 100 percent here and things start to go wrong. The lower the number, the more they go wrong.



```
***** WAR REPORT *****

ENEMY PLANES SHOT DOWN =
BOMB HITS =
AIRCRAFT DAMAGE BY ENEMY =

ENEMY STATUS: 0=SHOT DOWN
              1=RETURNING TO HOME
              2=ATTACKING

ENEMY 1 =          ENEMY 2 =
ENEMY 3 =          ENEMY 4 =
ENEMY 5 =          ENEMY 6 =

PRESS ANY KEY TO RESUME BATTLE
```

Flight Simulator II's War Status Display

This is an excellent feature. The malfunctions modeled are often subtle and a pilot's inattention to his instruments can result in a simple problem becoming fatal. It's a good training aid in that it really brings home to the user the importance of staying sharp and alert.

The Dogfight Game

They call it World War I Ace, and since today's general aviation airplanes are similar in performance to World War I fighters, I suppose it was inevitable. As an option of the Simulation Control menu, the user may select the dogfight game and fly against enemy fighter aircraft.

Actually, it's not bad. It's not simply a shoot 'em up. The user still has to fly his airplane properly and manuever into position in order to bomb ground targets or shoot down enemy fighters. If he fails to fly properly, the airplane will stall and crash, just as it would in the pure simulation mode.

Rules of the game are standard; you get points for shooting fighters down or bombing fuel depots, and you lose points for getting shot. Additionally, your plane degrades in performance each time it gets hit.

One rather interesting feature of the game is worth special mention because of its educational value to computerists. Unlike any actual World War I fighter, the one in this game has air-to-air radar. What this does is provide the user with information concerning targets where no information would otherwise have been available.

That is important because it demonstrates a flexibility on the part of SubLogic. They concentrated hard on realism throughout the product, but they didn't lose their ability to perceive the need for a feature that wasn't real. That's rare. I often see programmers who, once they learn to juggle assembly language routines, refuse to take advantage of those features of BASIC that simply cannot run any faster. That sort of locked-in attitude costs hours of programming time. One should guard against it.

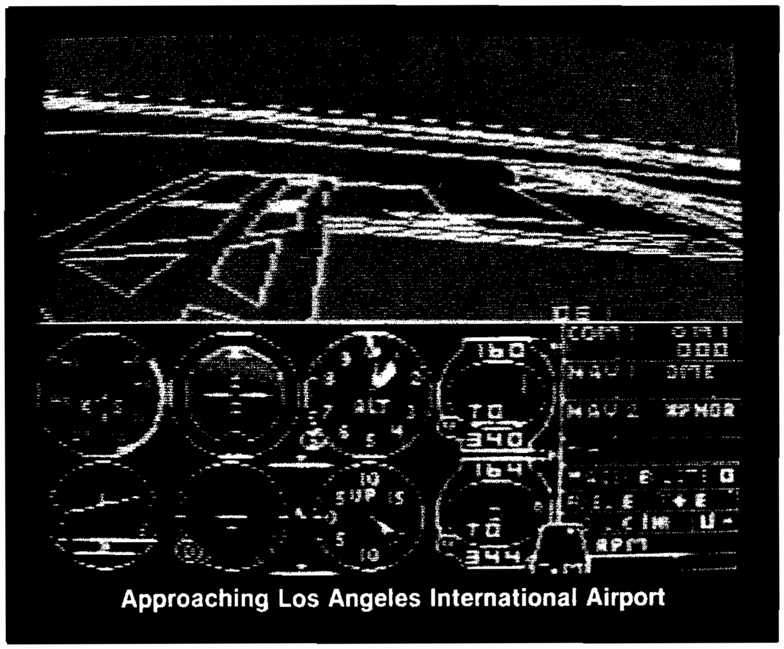
Conclusions

This product is one of those that can be perceived as something special even before the marketplace has passed its judgement. As such, one feels compelled to examine it and determine what core characteristic makes it what it is and, further, what does it have in common with other software programs already acknowleged as masterpieces of design.

Through this sort of analysis, programmers can remove a bit of the uncertainty in software design. They can find certain prerequisite things their programs must have to excel. They can make the process more of a science and less of an art. So what is it about Flight Simulator II? What is it that makes it superb? Is it something that can be emulated?

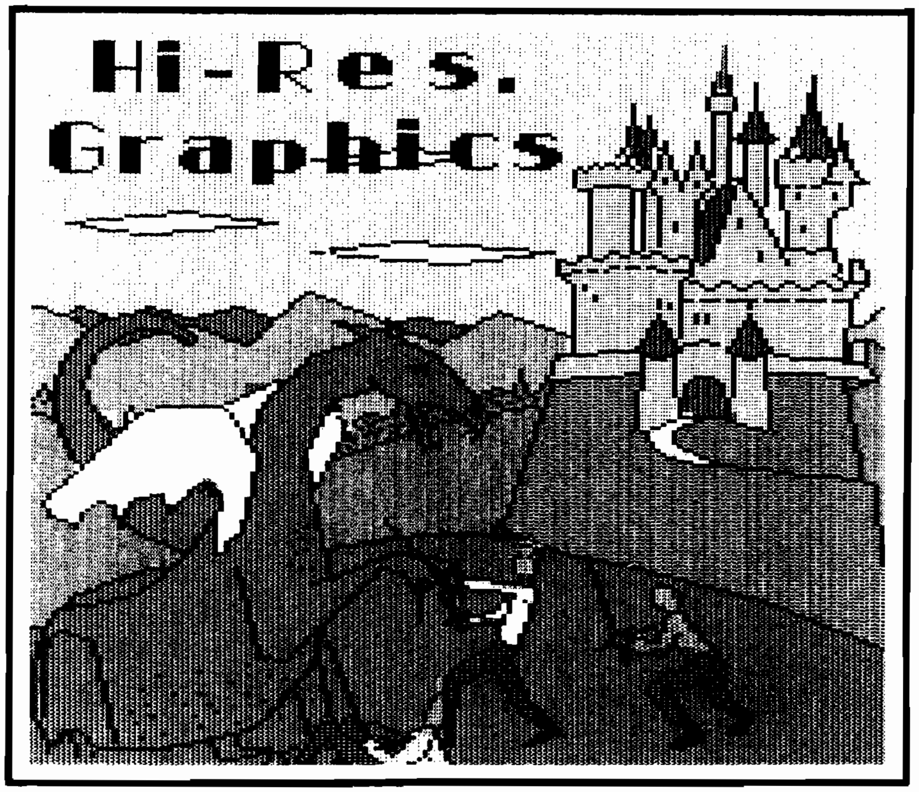
My opinion is that the program was planned intricately, written intricately, and, most important, debugged intricately. That all comes down to one phrase - attention to detail. They covered *everything*. Frankly, most programs don't cover half of what they could — and therefore should. Programmers need to make a rule for themselves. This rule would say that on the day the "Finished!!" tag is hung on a program, an X is placed on the calendar for two weeks in the future. The programmer must continue testing and working on the program until that day. Just think of how many bugs would never find their way to market.

MICRO


Approaching Los Angeles International Airport

Graphic Print for Commodore 64
(Part 1)

by Michael J. Keryan



Editor's Note: This is part 1 of a three part series. Parts 2 and 3 will appear in subsequent issues.

Create a full-page printout from a Commdore 64 high resolution display

The Commodore 64 is capable of displaying some pretty impressive graphics. Take a look at a few of the games recently introduced, like Neutral Zone, Blue Max, or Pogo Joe. Most sophisticated games use a high-resolution bit-mapped display rather than the alphanumeric/graphic-symbol display that most of you use for your programs.

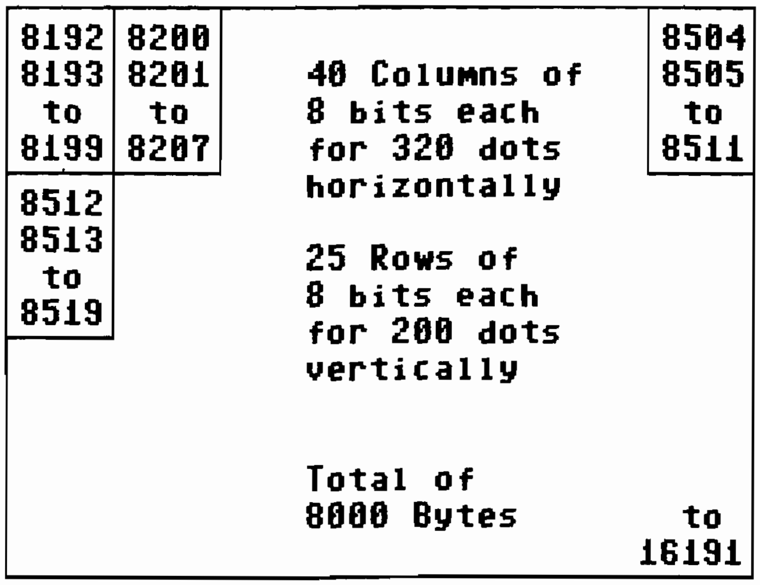
High-resolution bit-mapped graphics (and the multi-color variation) are described in the Commodore 64 Programmer's Reference Guide. The manual even shows you how to create a display using PEEKs and POKEs. However, since several thousand memory locations are involved, BASIC is extremely slow. Any practical use of high resolution graphics must use machine language routines. Since most people are not familiar with assembly or machine language programming, quite a few graphic aid and drawing programs for the Commodore 64 have been developed.

I was quite disappointed when I learned that pictures that were created on my Koala Pad could not be dumped to my printer. I also found that even though other graphic packages contained graphic dump routines, the resulting printouts were much less than perfect. Many routines give rather small drawings, one dot on the screen to one printed dot--this results in a picture a little smaller than 3 inches by 4 inches. Many graphic dump routines use the Commodore 1525 graphic mode which can be emulated by a number of interfaces with non-Commodore printers, but this technique is very slow. The most serious fault of all of the routines I've seen is their inability to recognize a color on the screen and translate it to a pattern that is approximately the same darkness of the color. Most graphic dumps print, at most, 3 or 4 varying shades of black dots, even though one of the colors represented is white.

Since a perfect graphic dump program wasn't available, I decided to write one. These were the objectives that I set for this program:

1. It will work in either standard HiRes or multi-color mode.
2. Printouts should be large, about the same size as the display on my Commodore 1701 color monitor (approx. 7" x 9"). This will fit nicely on a normal sheet of paper with one inch borders on all sides.

Figure 1. Graphics Bit-map Mode



8192
8193
to
8199
8200
8201
to
8207
8504
8505
to
8511
8512
8513
to
8519
40 Columns of 8 bits each for 320 dots horizontally
25 Rows of 8 bits each for 200 dots vertically
Total of 8000 Bytes
to 16191

3. The dump routine should work on my printer as well as those of my friends. These include NEC 8023, Prowriter (C. Itoh), Epson MX-80 and FX-80, and Gemini (Star) printers. Sorry 1525 owners, you're on your own.

4. Fast--to get the needed speed to print a full page of graphics, the print commands should directly access the printhead (transparent interface operation).

5. A unique dot pattern should be used for each of the 16 colors, so that any two adjacent colors can be distinguished. Each pattern should vary in intensity roughly in proportion to the darkness of the color on the CRT. Needless to say, the program should be able to determine the color of each dot on the screen.

6. Printouts of any part of the screen or the whole screen should be possible.

7. Most important, the program should be able to access graphic displays made from a number of graphic aid and drawing programs.

All of these objectives have been met and the resulting Assembly language program, GDUMP, is shown in Listing 1. The program is not especially compact; in fact, it uses quite a bit of memory for lookup tables. However, it works as per the above objectives and is the best graphic screen dump program that I have seen for the Commodore 64.

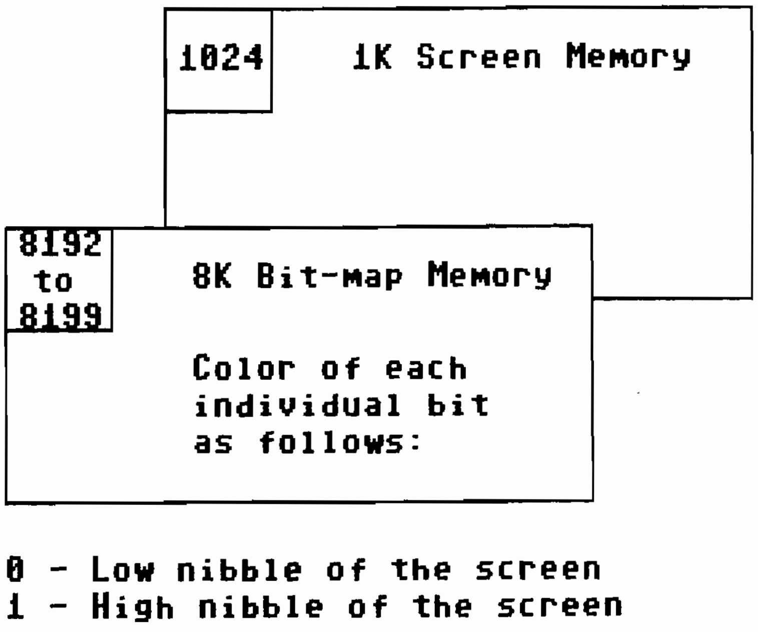
High Resolution Bit Map

Before describing how the program works, a short review of Commodore 64 bit map graphics is helpful. The standard high resolution bit map screen of the 64 is divided into 320 dots horizontally and 200 dots vertically. Each dot corresponds to a bit in memory. Therefore, 320 x 200 = 64000 bits, or exactly 8000 bytes of memory is required to hold this bit map pattern of ones (bit is on) and zeros (bit is off). Let's assume our bit map memory starts at $2000 hexadecimal (or 8192 decimal). The order of the bytes in memory do not correspond to the manner in which the lines are scanned on the CRT--they are arranged in 8 byte blocks as shown in Figure 1.

Despite the fact that the bytes are arranged in memory a little strangely, you can see that the screen is made up of 320 bits across and 200 bits down. You can think of this as: when a bit is off (0) the corresponding dot will be off (black), and when a bit is on (1) the dot will be on (white). Many two-color screens are set up like this, but the HiRes screen (HIRES) is a little more complicated than this, as shown in Figure 2. For every 8 byte block of bit map memory (or every 8x8 dot square) there exists a corresponding one byte of screen memory.

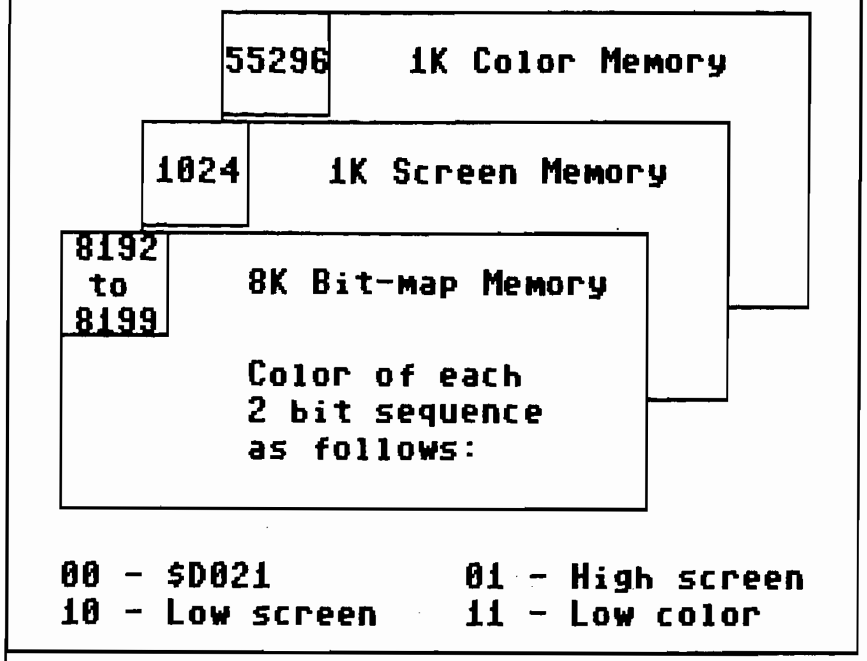
Let's assume this 1K block of memory starts at $0400 (1024 decimal). The colors of the foreground and background are picked up in the screen byte. The way one byte can hold two colors is by breaking the 8 bit byte into two 4 bit nibbles. With 4 bits, each nibble can hold a number from 0 to 15, for one of the 16 colors. Therefore, for every 8x8 square of dots, the color displayed for any of these 64 dots can be found in the high nibble of the corresponding screen memory if the bit is on (1) and in the low nibble if the bit is off (0). Note that only two unique colors can be displayed in any 8x8 block of dots, but an adjacent block can have any two other (or the same) colors.

Figure 2. Memory - High-res Mode



1024
1K Screen Memory
8192
to
8199
8K Bit-map Memory
Color of each individual bit as follows:
0 - Low nibble of the screen
1 - High nibble of the screen

Figure 3. Memory - Multicolor Mode



Multi-Color Bit Map Mode

If you thought the last section was difficult, you may as well skip this section right now. With the HIRES mode, there are two separate blocks of memory to worry about. In multi-color mode (MULTI) there are three blocks of memory, as shown in Figure 3. An additional 1K block of memory (usually starting at $D800 or 55296 decimal) is also used to store color information. In MULTI-color mode, the horizontal resolution is reduced to 160 dots, half of that as HIRES mode. Actually, there are still 320 dots on the screen, but the color can only change for every two dots. In every two-dot sequence of the bit-map memory, we can get four possible patterns of bits: 00, 01, 10, or 11. The pattern determines where the color for these two dots can be found. So in any 8x8 square of dots, a total of 4 colors are possible. Three of these colors can be different for every 8x8 square, but one color is common to all squares--the sequence of two zeros calls for the color in the background color register $D021.

To get an accurate graphic screen dump, we must first determine the location of each bit in an 8K bit-map block, and determine the corresponding colors from either the upper or lower nibble of screen memory, the lower nibble of color memory, or from the background color register. Each color must be translated to a unique pattern for a dot-matrix printer, and these patterns must be sent to the printer. A method is also required to duplicate dot patterns for grids larger than the original 320x200 dot grid.

GDUMP

The assembler (Listing 1) is commented, so you should be able to follow along, if you are familiar with machine language. The program is assembled to begin at $5000. There were very few memory areas left to put this code, when you want it to be compatible with the files containing graphic data from various third party routines. I decided to stick it right in the middle of your BASIC workspace. All the important constants were brought near the beginning to allow easy changes. The minimum and maximum horizontal and vertical byte numbers are located at $5003-$5006. The upper left of the screen is 0,0; the lower right is 39,199. You can change these if you want only part of the screen printed (but you will also have to change N1-N4 and EN1-EN2 in GSETUP and ESETUP).

There are four modes of operation:

0. Mode 0 is for two-color HIRES printouts. Every bit equal to 1 prints a 2x2 black square.

1. Mode 1 inverts the dots of mode 0. Bits that are equal to 1 print a 2x2 white area; bits equal to 0 print black dots.

2. This is MULTIcolor mode in which colors are determined from one of four possibilities as in Figure 3.

3. This is HIRES color mode in which colors are determined from either high or low nibbles of the screen memory as in Figure 2.

The starting page number for the bit-map memory, screen memory, and color memory are stored in $5008-$500A. These can be changed from the defaults ($2000, $0400 and $D800) for non-standard screen configurations.

The program begins by jumping to a printer setup routine. For TYMAC CONNECTION interfaces, an extra sequence is required before any other sequences. This is equivalent to CHR$(27) "W"CHR$(00). It disables the width command in the interface and is necessary to disable printing a carriage return after 80 graphic bytes. The printer channel is opened with a secondary address which puts the interface into transparent mode (5 for CARDCO, 6 for CONNECTION). Next the correct codes are sent to change the printer spacing to 1/9 inch vertically, to eliminate blank spaces between lines. These sequences are different for NEC/C.ITOH and EPSON/GEMINI printers. Then a carriage return is sent to start the printer at a known state.

Three loops can be found in the code: LOOPH, LOOPV and LOOPN. LOOPH cycles through the 40 horizontal screen bytes. LOOPV cycles through the 200 vertical bytes. LOOPN cycles through the repeat counter REPT several times for each of the 200 lines. REPT is set up to 3 for NEC/C.ITOH and 2 for EPSON/GEMINI. This gives a total of 600 or 400 dots, respectively for the top to bottom CRT scan (left to right on the printer). For both types of printers, this gives a line length of about 7 inches. Actually LOOPH is cycled through twice, since two dots are printed for every horizontal dot on the screen. If you follow through the logic in the area of LOOPN, you will see that every byte sent to the printer (for the 8 dots on the printhead) is made up of two 4 bit nibbles, each derived from a two-bit horizontal dot sequence on the screen.

Subroutine CHKREV simply reverses the 8-bit pattern for EPSON type printers since the printhead is set up the opposite of NEC type printheads. This routine also replaces every $0D bit pattern with $0B. For an

Listing 1

```
                ; GRAPHIC SCREEN DUMP  V1.2
                ;   M. J. KERYAN     3-27-84
                ;
                ; TO BE USED WITH 'TYMAC CONNECTION'
                ; OR SIMILAR TYPE OF INTERFACE
                ; AND PRINTERS—
                ;   NEC 8Ø23, PROWRITER, C.ITOH 851Ø
                ;   OR EPSON WITH GRAFTRAX OR
                ;   EPSON COMPATIBLE PRINTER.
                ;
5ØØØ                     ORG $5ØØØ
                ;
5ØØØ 4C 39 5Ø   GDUMP    JMP GSTART
                ;
5ØØ3 FF         MINH     BYT $FF     ; HORIZ. MIN.-1
5ØØ4 27         MAXH     BYT 39      ; HORIZ. MAX.
5ØØ5 ØØ         MINV     BYT Ø       ; VERT. MIN.
5ØØ6 C8         MAXV     BYT 2ØØ     ; VERT. MAX.+1
5ØØ7 Ø3         REPT     BYT 3       ; REPEAT BYTES
5ØØ8 2Ø         BMPG     BYT $2Ø     ; BIT MAP PAGE #
5ØØ9 Ø4         SCPG     BYT $Ø4     ; SCREEN PAGE #
5ØØA D8         CLPG     BYT $D8     ; COLOR PAGE #
5ØØB ØØ         PTYPE    BYT $ØØ     ; PRINTER
                ; Ø = NEC/C.ITOH     ; TYPE —
                ; 1 = EPSON TYPE
5ØØC Ø6         SECADR   BYT $Ø6     ; SECONDARY
                ; (TRANSPARENT)      ; ADDR
5ØØD ØØ         INTERF   BYT $ØØ     ; INTERFACE
                ; Ø = CONNECTION     ; TYPE —
                ; 1 = OTHER
5ØØE Ø2         MODE     BYT $Ø2     ; MODE TYPE
                ;
                ; MODE Ø = NORMAL HIRES B/W
                ;      1 = INVERTED HIRES B/W
                ;      2 = MULTI-COLOR
                ;      3 = HIRES COLOR
                ;
5ØØF ØD         GSETUP   BYT $ØD     ;SET UP CARR RET
5Ø1Ø 2Ø         SP1      BYT $2Ø     ;AND 4 SPACES
5Ø11 2Ø         SP2      BYT $2Ø     ;FOLLOWED BY
5Ø12 2Ø         SP3      BYT $2Ø     ;THE NEC/C.ITOH
5Ø13 2Ø         SP4      BYT $2Ø     ;REQUIRED
5Ø14 1B         ESC      BYT $1B     ;GRAPHIC CONTROL
5Ø15 53         ES       BYT $53     ;SEQUENCE—
5Ø16 3Ø         N1       BYT $3Ø     ; ESC, S, N1, N2,
5Ø17 36         N2       BYT $36     ; N3, N4 WHERE
5Ø18 3Ø         N3       BYT $3Ø     ; N'S ARE 4 DIG.
5Ø19 3Ø         N4       BYT $3Ø     ; BYTE COUNT
                ;
5Ø1A ØD         ESETUP   BYT $ØD     ;SET UP CARR RET
5Ø1B 2Ø         ESP1     BYT $2Ø     ;AND 4 SPACES
5Ø1C 2Ø         ESP2     BYT $2Ø     ;FOLLOWED BY
5Ø1D 2Ø         ESP3     BYT $2Ø     ;THE EPSON
5Ø1E 2Ø         ESP4     BYT $2Ø     ;REQUIRED
5Ø1F 1B         EESC     BYT $1B     ;GRAPHIC CONTROL
5Ø2Ø 4B         EK       BYT $4B     ;SEQUENCE—
5Ø21 9Ø         EN1      BYT $9Ø     ; ESC, K, N1, N2
5Ø22 Ø1         EN2      BYT $Ø1     ;
                ;
5Ø23 1B         SPC      BYT $1B     ;LINE SPACING
5Ø24 54         TEE      BYT $54     ;OF 16/144 INCH
5Ø25 31         NN1      BYT $31     ;FOR C.ITOH/NEC
5Ø26 36         NN2      BYT $36
5Ø27 ØD         RET2     BYT $ØD
                ;
5Ø28 1B         ESPC     BYT $1B     ;LINE SPACING
5Ø29 41         EA       BYT $41     ;OF 8/72 INCH
5Ø2A Ø8         ENN1     BYT $Ø8     ;FOR EPSON TYPE
5Ø2B ØD         ERET2    BYT $ØD
                ;
ØØFD            PL       EQU $FD     ;MEMORY USED FOR
ØØFE            PH       EQU $FE     ;INDIRECT
                ;                     POINTERS
5Ø2C ØØ         DATA     BYT Ø       ;MEMORY REGISTERS
5Ø2D ØØ         VBYT     BYT Ø       ;USED IN THIS
5Ø2E ØØ         HBYT     BYT Ø       ;PROGRAM
5Ø2F ØØ         NBYT     BYT Ø
5Ø3Ø ØØ         TBYT     BYT Ø
5Ø31 ØØ         NIBL     BYT Ø
5Ø32 ØØ         DATAXX   BYT Ø
5Ø33 ØØ         DATAYY   BYT Ø
5Ø34 ØØ         DATATM   BYT Ø
5Ø35 ØØ         COLORB   BYT Ø
5Ø36 ØØ         SCREEN   BYT Ø
5Ø37 ØØ         ETEMP1   BYT Ø
5Ø38 ØØ         ETEMP2   BYT Ø
                ;
ØØ71            GFILE    EQU $71     ;PRINTER FILE #
                ;
FFCC            CLRCHN   EQU $FFCC   ;KERNAL ROUTINES
FFC3            CLOSE    EQU $FFC3
FFBA            SETLFS   EQU $FFBA
FFBD            SETNAM   EQU $FFBD
FFCØ            OPEN     EQU $FFCØ
FFC9            CHKOUT   EQU $FFC9
                ;
5Ø39 2Ø 21 52   GSTART   JSR SETUP   ;OPEN PORT, ETC.
5Ø3C AD Ø4 5Ø            LDA MAXH
5Ø3F 8D 2E 5Ø            STA HBYT    ;INIT. WIDTH
5Ø42 A9 ØØ               LDA #$ØØ
5Ø44 8D 31 5Ø            STA NIBL    ;FIRST NIBBLE
5Ø47 AD Ø5 5Ø   LOOPH    LDA MINV
5Ø4A 8D 2D 5Ø            STA VBYT    ;INIT. HEIGHT
5Ø4D AØ ØØ               LDY #$ØØ
5Ø4F AD ØB 5Ø   OUTNUM   LDA PTYPE      ;PRINTER TYPE
5Ø52 DØ ØD               BNE OUTN2
5Ø54 B9 ØF 5Ø   OUTN1    LDA GSETUP,Y   ;OUTPUT
5Ø57 2Ø CA F1            JSR $F1CA      ;GRAPHIC
5Ø5A C8                  INY            ;CONTROL CODES
5Ø5B CØ ØB               CPY #$ØB       ;FOR 1 LINE
5Ø5D DØ F5               BNE OUTN1      ;11 BITS
5Ø5F FØ ØB               BEQ LOOPV
5Ø61 B9 1A 5Ø   OUTN2    LDA ESETUP,Y   ;OUTPUT
5Ø64 2Ø CA F1            JSR $F1CA      ;GRAPHIC
5Ø67 C8                  INY            ;CONTROL CODES
5Ø68 CØ Ø9               CPY #$Ø9       ;FOR 1 LINE
5Ø6A DØ F5               BNE OUTN2      ;9 BYTES
5Ø6C AD Ø7 5Ø   LOOPV    LDA REPT
5Ø6F 8D 2F 5Ø            STA NBYT    ;INIT. COUNTER
5Ø72 A9 ØØ               LDA #$ØØ
5Ø74 8D 3Ø 5Ø            STA TBYT    ;RIGHT BYTE
5Ø77 2Ø B4 51            JSR DATACL
5Ø7A 8D 2C 5Ø            STA DATA
5Ø7D AD 2C 5Ø   LOOPN    LDA DATA
5Ø8Ø 29 Ø3               AND #$Ø3    ;ØØØØØØ11
5Ø82 2Ø ØB 51            JSR DATACO ;CONVERT TO
5Ø85 29 ØF               AND #$ØF    ;4 BITS
5Ø87 8D 34 5Ø            STA DATATM ;HOLD IT
5Ø8A AD 2C 5Ø            LDA DATA
```

```
508D 29 ØC              AND #$ØC   ;ØØØØ11ØØ
508F 4A                 LSR A
5Ø9Ø 4A                 LSR A
5Ø91 2Ø ØB 51           JSR DATACO ;4 MORE BITS
5Ø94 ØA                 ASL A
5Ø95 ØA                 ASL A
5Ø96 ØA                 ASL A
5Ø97 ØA                 ASL A
5Ø98 ØD 34 5Ø           ORA DATATM ;COMBINE 8 BITS
5Ø9B 2Ø DB 5Ø           JSR CHKREV ;CHECK IF REVERSE
5Ø9E 2Ø CA F1           JSR $F1CA  ;OUTPUT BYTE
5ØA1 CE 2F 5Ø           DEC NBYT   ;END OF REPEAT?
5ØA4 FØ ØB              BEQ NEND
5ØA6 AD 3Ø 5Ø           LDA TBYT
5ØA9 49 Ø1              EOR #$Ø1   ;TOGGLE BYTE #
5ØAB 8D 3Ø 5Ø           STA TBYT
5ØAE 18                 CLC
5ØAF 9Ø CC              BCC LOOPN  ;CONTINUE REPEAT
5ØB1 EE 2D 5Ø  NEND     INC VBYT
5ØB4 AD 2D 5Ø           LDA VBYT
5ØB7 CD Ø6 5Ø           CMP MAXV   ;END OF VERT.?
5ØBA DØ BØ              BNE LOOPV  ;CONTINUE VERT.
5ØBC AD 31 5Ø           LDA NIBL
5ØBF 49 Ø1              EOR #$Ø1   ;TOGGLE NIBBLE
5ØC1 8D 31 5Ø           STA NIBL
5ØC4 AD 31 5Ø           LDA NIBL
5ØC7 DØ ØF              BNE TOLPH
5ØC9 CE 2E 5Ø           DEC HBYT
5ØCC AD 2E 5Ø           LDA HBYT
5ØCF CD Ø3 5Ø           CMP MINH
5ØD2 DØ Ø4              BNE TOLPH
5ØD4 2Ø 98 52           JSR SETDWN ;UNDO SETUP
5ØD7 6Ø                 RTS
5ØD8 4C 47 5Ø  TOLPH    JMP LOOPH  ;BRANCH TOO LONG
              ;
5ØDB 8D 37 5Ø  CHKREV   STA ETEMP1
5ØDE 8D 38 5Ø           STA ETEMP2
5ØE1 AD ØB 5Ø           LDA PTYPE  ;IF PRINTER IS
5ØE4 FØ 1B              BEQ PCR    ;EPSON, THEN
5ØE6 A9 ØØ              LDA #$ØØ   ;REVERSE DOT
5ØE8 8D 38 5Ø           STA ETEMP2 ;ORDER
5ØEB AØ Ø8              LDY #$Ø8
5ØED B9 F2 52  EP1      LDA TABBIT-1,Y
5ØFØ 2D 37 5Ø           AND ETEMP1
5ØF3 FØ Ø9              BEQ EP2
5ØF5 B9 FA 52           LDA TABTIB-1,Y
5ØF8 ØD 38 5Ø           ORA ETEMP2
5ØFB 8D 38 5Ø           STA ETEMP2
5ØFE 88        EP2      DEY
5ØFF DØ EC              BNE EP1
51Ø1 AD 38 5Ø  PCR      LDA ETEMP2 ;IF BIT CODE
51Ø4 C9 ØD              CMP #$ØD   ;IS SAME AS
51Ø6 DØ Ø2              BNE PRET   ;CARR RETURN,
51Ø8 A9 ØB              LDA #$ØB   ;CHANGE IT
51ØA 6Ø        PRET     RTS
          ;
51ØB AA        DATACO   TAX        ;X = 2 BITS
51ØC AD ØE 5Ø           LDA MODE
51ØF C9 Ø2              CMP #$Ø2   ;< 2?
5111 BØ ØB              BCS DØ     ;NO, GO ON
5113 BD DE 52  ZERONE   LDA HICOD,X   ;YES, Ø OR 1
5116 AE ØE 5Ø           LDX MODE
5119 FØ Ø2              BEQ D1
511B 49 ØF              EOR #$ØF   ;INVERT GRAPHICS
511D 6Ø        D1       RTS
511E C9 Ø3     DØ       CMP #$Ø3   ;MODE 3?
512Ø FØ 33              BEQ HIRØ   ;YES, HIRES COLOR
5122 EØ ØØ     MULTI    CPX #$ØØ   ;TWO BITS = ØØ?
5124 DØ Ø6              BNE ONE
5126 AD 21 DØ           LDA $DØ21  ;COLOR IN $DØ21
```

```
5129 18                 CLC
512A 9Ø ØF              BCC ONETWO
512C EØ Ø3     ONE      CPX #$Ø3   ;TWO BITS = 11?
512E FØ 1F              BEQ THREE
513Ø AD 36 5Ø           LDA SCREEN ;TWO BITS = 1Ø
5133 EØ Ø2              CPX #$Ø2
5135 FØ Ø4              BEQ ONETWO
5137 4A        HINIB    LSR A
5138 4A                 LSR A
5139 4A                 LSR A      ;HIGH NIBBLE
513A 4A                 LSR A      ;CONTAINS COLOR
513B 29 ØF     ONETWO   AND #$ØF
513D AA                 TAX
513E BD BE 52           LDA TABCOL,X  ;GET SHADE #
5141 AA        GETCOD   TAX
5142 BD CE 52           LDA TABCOD,X  ;GET CODE
5145 AE 3Ø 5Ø           LDX TBYT
5148 FØ Ø4              BEQ DATAE  ;ALTERNATE LOW
514A 4A                 LSR A      ;AND HIGH
514B 4A                 LSR A      ;NIBBLES OF
514C 4A                 LSR A      ;CODE
514D 4A                 LSR A
514E 6Ø        DATAE    RTS
514F AD 35 5Ø  THREE    LDA COLORB ;COLOR IN COLOR
5152 18                 CLC        ;MEMORY
5153 9Ø E6              BCC ONETWO
5155 EØ ØØ     HIRØ     CPX #$ØØ   ;BITS ØØ
5157 DØ Ø6              BNE HIR3
5159 AD 36 5Ø           LDA SCREEN ;USE LOWER
515C 4C 3B 51           JMP ONETWO ;NIBBLE
515F EØ Ø3     HIR3     CPX #$Ø3   ;BITS 11
5161 DØ Ø6              BNE HIR2
5163 AD 36 5Ø           LDA SCREEN ;USE UPPER
5166 4C 37 51           JMP HINIB  ;NIBBLE
5169 EØ Ø2     HIR2     CPX #$Ø2   ;BITS 1Ø
516B DØ 1B              BNE HIR1
516D AD 36 5Ø           LDA SCREEN ;GET UPPER
517Ø 2Ø 37 51           JSR HINIB
5173 2Ø A3 51           JSR HIRC
5176 ØA                 ASL A      ;DATA IN BITS
5177 ØA                 ASL A      ;———**——
5178 8D 32 5Ø           STA DATAXX
517B AD 36 5Ø           LDA SCREEN
517E 2Ø 3B 51           JSR ONETWO ;GET LOWER
5181 2Ø A3 51           JSR HIRC   ;————————**
5184 ØD 32 5Ø           ORA DATAXX ;COMBINE
5187 6Ø                 RTS
5188 AD 36 5Ø  HIR1     LDA SCREEN ;BITS Ø1
518B 2Ø 3B 51           JSR ONETWO ;GET UPPER
518E 2Ø A3 51           JSR HIRC
5191 ØA                 ASL A      ;DATA BITS
5192 ØA                 ASL A      ;——————**——
5193 8D 32 5Ø           STA DATAXX
5196 AD 36 5Ø           LDA SCREEN
5199 2Ø 37 51           JSR HINIB  ;GET LOWER
519C 2Ø A3 51           JSR HIRC   ;————————**
519F ØD 32 5Ø           ORA DATAXX ;COMBINE
51A2 6Ø                 RTS
              ;
51A3 48        HIRC     PHA        ;THIS ROUTINE
51A4 29 Ø3              AND #$Ø3   ;AVERAGES THE
51A6 8D 33 5Ø           STA DATAYY ;THE BITS
              ;
51A3 48        HIRC     PHA        ;THIS ROUTINE
51A4 29 Ø3              AND #$Ø3   ;AVERAGES THE
51A6 8D 33 5Ø           STA DATAYY ;THE BITS
51A9 68                 PLA        ;————————**
51AA 4A                 LSR A      ;AND
51AB 4A                 LSR A      ;—————**——
51AC 29 Ø3              AND #$Ø3
```

```
51AE 18                CLC
51AF 6D 33 5Ø          ADC DATAYY
51B2 4A                LSR A      ;DIVIDE BY 2
51B3 6Ø                RTS
               ;
51B4 AD 2D 5Ø   DATACL LDA VBYT   ;GET MEMORY
51B7 4A                LSR A
51B8 4A                LSR A
51B9 4A                LSR A
51BA AA                TAX
51BB BD FC 54          LDA HCTAB,X
51BE 85 FE             STA PH
51CØ BD E3 54          LDA LCTAB,X
51C3 18                CLC
51C4 6D 2E 5Ø          ADC HBYT
51C7 85 FD             STA PL
51C9 9Ø Ø2             BCC CL3
51CB E6 FE             INC PH
51CD A5 FE      CL3    LDA PH
51CF 48                PHA
51DØ 18                CLC
51D1 6D Ø9 5Ø          ADC SCPG
51D4 85 FE             STA PH
51D6 AØ ØØ             LDY #$ØØ
51D8 B1 FD             LDA (PL),Y
51DA 8D 36 5Ø          STA SCREEN ;SCREEN MEMORY
               ;
51DD 68                PLA
51DE 18                CLC
51DF 6D ØA 5Ø          ADC CLPG
51E2 85 FE             STA PH
51E4 B1 FD             LDA (PL),Y
51E6 8D 35 5Ø          STA COLORB ;COLOR MEMORY
               ;
51E9 AC 2E 5Ø          LDY HBYT
51EC AE 2D 5Ø          LDX VBYT
51EF BD Ø3 53          LDA LTAB,X
51F2 85 FD             STA PL
51F4 BD CB 53          LDA HTAB,X
51F7 85 FE             STA PH
51F9 B9 93 54          LDA LTABA,Y
51FC 18                CLC
51FD 65 FD             ADC PL
51FF 85 FD             STA PL
52Ø1 9Ø Ø2             BCC CL1
52Ø3 E6 FE             INC PH
52Ø5 B9 BB 54   CL1    LDA HTABA,Y
52Ø8 18                CLC
52Ø9 65 FE             ADC PH
52ØB 85 FE             STA PH
52ØD 18                CLC
52ØE 6D Ø8 5Ø          ADC BMPG
5211 85 FE             STA PH
5213 AØ ØØ             LDY #$ØØ
5215 B1 FD             LDA (PL),Y
5217 AE 31 5Ø          LDX NIBL
521A FØ Ø4             BEQ CL2
521C 4A                LSR A
521D 4A                LSR A
521E 4A                LSR A
521F 4A                LSR A      ;ACCUM = BIT MAP
522Ø 6Ø         CL2    RTS        ;BYTE
               ;
5221 A9 71      SETUP  LDA #GFILE
5223 2Ø C3 FF          JSR CLOSE
5226 AD ØD 5Ø          LDA INTERF
5229 DØ 31             BNE SET2
522B A9 71             LDA #GFILE ;FOR CONNECTION,
522D AØ ØØ             LDY #$ØØ   ;WIDTH MUST BE
522F A2 Ø4             LDX #$Ø4   ;SET TO ZERO TO
```

```
5231 2Ø BA FF          JSR SETLFS ;TO AVOID EXTRA
5234 A9 ØØ             LDA #$ØØ   ;CARR RETURNS
5236 2Ø BD FF          JSR SETNAM
5239 2Ø CØ FF          JSR OPEN
523C BØ 56             BCS GCLOSE
523E A2 71             LDX #GFILE
524Ø 2Ø C9 FF          JSR CHKOUT
5243 A9 1B             LDA #$1B
5245 2Ø CA F1          JSR $F1CA
5248 A9 57             LDA #$57
524A 2Ø CA F1          JSR $F1CA
524D A9 ØØ             LDA #$ØØ
524F 2Ø CA F1          JSR $F1CA
5252 A9 ØD             LDA #$ØD
5254 2Ø CA F1          JSR $F1CA
5257 A9 71             LDA #GFILE
5259 2Ø C3 FF          JSR CLOSE
525C A9 71      SET2   LDA #GFILE
525E AC ØC 5Ø          LDY SECADR
5261 A2 Ø4             LDX #$Ø4
5263 2Ø BA FF          JSR SETLFS
5266 A9 ØØ             LDA #$ØØ
5268 2Ø BD FF          JSR SETNAM
526B 2Ø CØ FF          JSR OPEN
526E BØ 24             BCS GCLOSE
527Ø A2 71             LDX #GFILE
5272 2Ø C9 FF          JSR CHKOUT
5275 AØ ØØ             LDY #$ØØ
5277 AD ØB 5Ø          LDA PTYPE
527A DØ ØC             BNE OUTSP2
527C B9 23 5Ø   OUTSP  LDA SPC,Y
527F 2Ø CA F1          JSR $F1CA
5282 C8                INY
5283 CØ Ø5             CPY #$Ø5
5285 DØ F5             BNE OUTSP
5287 6Ø                RTS
5288 B9 28 5Ø   OUTSP2 LDA ESPC,Y
528B 2Ø CA F1          JSR $F1CA
528E C8                INY
528F CØ Ø4             CPY #$Ø4
5291 DØ F5             BNE OUTSP2
5293 6Ø                RTS
5294 2Ø 98 52   GCLOSE JSR SETDWN
5297 6Ø                RTS
               ;
5298 A9 ØD      SETDWN LDA #$ØD   ;CARR RETURN
529A 2Ø CA F1          JSR $F1CA
529D A9 ØC             LDA #$ØC   ;FORM FEED
529F 2Ø CA F1          JSR $F1CA
52A2 A9 1B             LDA #$1B   ;LINE SPACING
52A4 2Ø CA F1          JSR $F1CA  ;BACK TO 1/6 IN.
52A7 AD ØB 5Ø          LDA PTYPE
52AA DØ Ø4             BNE EPCL
52AC A9 41             LDA #$41   ;ESC A FOR NEC/
52AE DØ Ø2             BNE LSPC   ; OR C. ITOH
52BØ A9 32      EPCL   LDA #$32   ;ESC 2 FOR
52B2 2Ø CA F1   LSPC   JSR $F1CA  ; EPSON
52B5 2Ø CC FF          JSR CLRCHN
52B8 A9 71             LDA #GFILE
52BA 2Ø C3 FF          JSR CLOSE
52BD 6Ø                RTS
               ;
52BE ØF    TABCOL  BYT 15,Ø,11,3,1Ø,7,12,1
52C6 Ø8            BYT 8,14,5,13,9,2,6,4
52CE ØØ    TABCOD  BYT $ØØ,$2Ø,$Ø4,$28
52D2 ØA            BYT $ØA,$25,$4A,$A5
52D6 69            BYT $69,$87,$2D,$A7
52DA 6D            BYT $6D,$DB,$9F,$FF
52DE ØØ    HICOD   BYT $ØØ,$Ø3,$ØC,$ØF
52E2 28    AUTHOR  BYT '(C) M.KERYAN 1984'
```

unexplainable reason, my printer-interface would print two $0D patterns for every one sent, messing up the 600 byte counter. Instead of tracking down the reason for this, I eliminated any chance for this glitch to occur.

At the beginning of every line a carriage return is sent, followed by 4 spaces (to center the drawing), then a code is sent to set up the printer to accept the correct number of graphic characters (600 or 400 as explained above). These are the labeled GSETUP and ESETUP.

Subroutine DATACL returns the contents of three memory cells, based on the current horizontal and vertical coordinates: the SCREEN memory, the COLOR memory and the bit-map memory in the accumulator. To avoid confusing calculations and to speed things up a bit, lookup tables are used extensively in this routine.

Subroutine DATACO is entered with the lower two bits of the accumulator equal to two bits from the bit-map memory. When finished, this routine returns with a four bit matrix pattern that eventually gets sent as half of a byte to the printhead. This routine works differently for the four modes of operation. In modes 0 and 1, simple 4 bit patterns duplicate (or invert) the original 2 bit sequence. In modes 2 and 3, the correct colors are determined. Then unique patterns are found through lookup tables TABCOL and TABCOD. Note that each of the 16 colors are associated with two different 4 bit patterns--the high and low nibbles of TABCOD. These two different codes are alternately used when the same byte is repeated to avoid vertical lines on the printed.

After the picture is printed, SETDWN sends a carriage return and a form feed to the printer and then changes the line spacing back to 1/6 inch for normal printer operation.

GDUMP can be run by your BASIC programs by POKEing the required set-up parameters into the area in the beginning of the program, then SYS 20480. Next month we'll continue this series by adding another small machine language program and a BASIC program that will allow GDUMP to print pictures made from SIMONS' BASIC, ULTRABASIC-64, DOODLE, KOALA-PAINTER and TPUG's SLIDESHOW. For those of you who don't have an Assembler to enter GDUMP, MICRO will provide these programs on 1541 disks for $15 (US). Order MicroDisk No. MD-4.

```
52F3 8Ø 4Ø 2Ø   TABBIT  BYT $8Ø,$4Ø,$2Ø,$1Ø,$Ø8,$Ø4,$Ø2,$Ø1
52FB Ø1 Ø2 Ø4   TABTIB  BYT $Ø1,$Ø2,$Ø4,$Ø8,$1Ø,$2Ø,$4Ø,$8Ø
53Ø3 ØØ Ø1 Ø2   LTAB    BYT $ØØ,$Ø1,$Ø2,$Ø3,$Ø4,$Ø5,$Ø6,$Ø7
53ØB 4Ø 41 42           BYT $4Ø,$41,$42,$43,$44,$45,$46,$47
5313 8Ø 81 82           BYT $8Ø,$81,$82,$83,$84,$85,$86,$87
531B CØ C1 C2           BYT $CØ,$C1,$C2,$C3,$C4,$C5,$C6,$C7
5323 ØØ Ø1 Ø2           BYT $ØØ,$Ø1,$Ø2,$Ø3,$Ø4,$Ø5,$Ø6,$Ø7
532B 4Ø 41 42           BYT $4Ø,$41,$42,$43,$44,$45,$46,$47
5333 8Ø 81 82           BYT $8Ø,$81,$82,$83,$84,$85,$86,$87
533B CØ C1 C2           BYT $CØ,$C1,$C2,$C3,$C4,$C5,$C6,$C7
5343 ØØ Ø1 Ø2           BYT $ØØ,$Ø1,$Ø2,$Ø3,$Ø4,$Ø5,$Ø6,$Ø7
534B 4Ø 41 42           BYT $4Ø,$41,$42,$43,$44,$45,$46,$47
5353 8Ø 81 82           BYT $8Ø,$81,$82,$83,$84,$85,$86,$87
535B CØ C1 C2           BYT $CØ,$C1,$C2,$C3,$C4,$C5,$C6,$C7
5363 ØØ Ø1 Ø2           BYT $ØØ,$Ø1,$Ø2,$Ø3,$Ø4,$Ø5,$Ø6,$Ø7
536B 4Ø 41 42           BYT $4Ø,$41,$42,$43,$44,$45,$46,$47
5373 8Ø 81 82           BYT $8Ø,$81,$82,$83,$84,$85,$86,$87
537B CØ C1 C2           BYT $CØ,$C1,$C2,$C3,$C4,$C5,$C6,$C7
5383 ØØ Ø1 Ø2           BYT $ØØ,$Ø1,$Ø2,$Ø3,$Ø4,$Ø5,$Ø6,$Ø7
538B 4Ø 41 42           BYT $4Ø,$41,$42,$43,$44,$45,$46,$47
5393 8Ø 81 82           BYT $8Ø,$81,$82,$83,$84,$85,$86,$87
539B CØ C1 C2           BYT $CØ,$C1,$C2,$C3,$C4,$C5,$C6,$C7
53A3 ØØ Ø1 Ø2           BYT $ØØ,$Ø1,$Ø2,$Ø3,$Ø4,$Ø5,$Ø6,$Ø7
53AB 4Ø 41 42           BYT $4Ø,$41,$42,$43,$44,$45,$46,$47
53B3 8Ø 81 82           BYT $8Ø,$81,$82,$83,$84,$85,$86,$87
53BB CØ C1 C2           BYT $CØ,$C1,$C2,$C3,$C4,$C5,$C6,$C7
53C3 ØØ Ø1 Ø2           BYT $ØØ,$Ø1,$Ø2,$Ø3,$Ø4,$Ø5,$Ø6,$Ø7
53CB ØØ ØØ ØØ   HTAB    BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
53D3 Ø1 Ø1 Ø1           BYT $Ø1,$Ø1,$Ø1,$Ø1,$Ø1,$Ø1,$Ø1,$Ø1
53DB Ø2 Ø2 Ø2           BYT $Ø2,$Ø2,$Ø2,$Ø2,$Ø2,$Ø2,$Ø2,$Ø2
53E3 Ø3 Ø3 Ø3           BYT $Ø3,$Ø3,$Ø3,$Ø3,$Ø3,$Ø3,$Ø3,$Ø3
53EB Ø5 Ø5 Ø5           BYT $Ø5,$Ø5,$Ø5,$Ø5,$Ø5,$Ø5,$Ø5,$Ø5
53F3 Ø6 Ø6 Ø6           BYT $Ø6,$Ø6,$Ø6,$Ø6,$Ø6,$Ø6,$Ø6,$Ø6
53FB Ø7 Ø7 Ø7           BYT $Ø7,$Ø7,$Ø7,$Ø7,$Ø7,$Ø7,$Ø7,$Ø7
54Ø3 Ø8 Ø8 Ø8           BYT $Ø8,$Ø8,$Ø8,$Ø8,$Ø8,$Ø8,$Ø8,$Ø8
54ØB ØA ØA ØA           BYT $ØA,$ØA,$ØA,$ØA,$ØA,$ØA,$ØA,$ØA
5413 ØB ØB ØB           BYT $ØB,$ØB,$ØB,$ØB,$ØB,$ØB,$ØB,$ØB
541B ØC ØC ØC           BYT $ØC,$ØC,$ØC,$ØC,$ØC,$ØC,$ØC,$ØC
5423 ØD ØD ØD           BYT $ØD,$ØD,$ØD,$ØD,$ØD,$ØD,$ØD,$ØD
542B ØF ØF ØF           BYT $ØF,$ØF,$ØF,$ØF,$ØF,$ØF,$ØF,$ØF
5433 1Ø 1Ø 1Ø           BYT $1Ø,$1Ø,$1Ø,$1Ø,$1Ø,$1Ø,$1Ø,$1Ø
543B 11 11 11           BYT $11,$11,$11,$11,$11,$11,$11,$11
5443 12 12 12           BYT $12,$12,$12,$12,$12,$12,$12,$12
544B 14 14 14           BYT $14,$14,$14,$14,$14,$14,$14,$14
5453 15 15 15           BYT $15,$15,$15,$15,$15,$15,$15,$15
545B 16 16 16           BYT $16,$16,$16,$16,$16,$16,$16,$16
5463 17 17 17           BYT $17,$17,$17,$17,$17,$17,$17,$17
546B 19 19 19           BYT $19,$19,$19,$19,$19,$19,$19,$19
5473 1A 1A 1A           BYT $1A,$1A,$1A,$1A,$1A,$1A,$1A,$1A
547B 1B 1B 1B           BYT $1B,$1B,$1B,$1B,$1B,$1B,$1B,$1B
5483 1C 1C 1C           BYT $1C,$1C,$1C,$1C,$1C,$1C,$1C,$1C
548B 1E 1E 1E           BYT $1E,$1E,$1E,$1E,$1E,$1E,$1E,$1E
5493 ØØ Ø8 1Ø   LTABA   BYT $ØØ,$Ø8,$1Ø,$18,$2Ø,$28,$3Ø,$38
549B 4Ø 48 5Ø           BYT $4Ø,$48,$5Ø,$58,$6Ø,$68,$7Ø,$78
54A3 8Ø 88 9Ø           BYT $8Ø,$88,$9Ø,$98,$AØ,$A8,$BØ,$B8
54AB CØ C8 DØ           BYT $CØ,$C8,$DØ,$D8,$EØ,$E8,$FØ,$F8
54B3 ØØ Ø8 1Ø           BYT $ØØ,$Ø8,$1Ø,$18,$2Ø,$28,$3Ø,$38
54BB ØØ ØØ ØØ   HTABA   BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
54C3 ØØ ØØ ØØ           BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
54CB ØØ ØØ ØØ           BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
54D3 ØØ ØØ ØØ           BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
54DB Ø1 Ø1 Ø1           BYT $Ø1,$Ø1,$Ø1,$Ø1,$Ø1,$Ø1,$Ø1,$Ø1
54E3 ØØ 28 5Ø   LCTAB   BYT $ØØ,$28,$5Ø,$78,$AØ,$C8,$FØ,$18
54EB 4Ø 68 9Ø           BYT $4Ø,$68,$9Ø,$B8,$EØ,$Ø8,$3Ø,$58
54F3 8Ø A8 DØ           BYT $8Ø,$A8,$DØ,$F8,$2Ø,$48,$7Ø,$98
54FB CØ                 BYT $CØ
54FC ØØ ØØ ØØ   HCTAB   BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$Ø1
55Ø4 Ø1 Ø1 Ø1           BYT $Ø1,$Ø1,$Ø1,$Ø1,$Ø1,$Ø2,$Ø2,$Ø2
55ØC Ø2 Ø2 Ø2           BYT $Ø2,$Ø2,$Ø2,$Ø2,$Ø3,$Ø3,$Ø3,$Ø3
5514 Ø3                 BYT $Ø3
5515                    END
```

INTERFACE CLINIC: *Communication Between Different Computers*

How to merge several computers into one efficient system

by Ralph Tenny

A few columns ago I answered a letter query about communication between different computers. Here's another example: I have two Radio Shack Color Computers and one Commodore 64, but only one printer (EPSON MX-80). The 64K Color Computer is in use constantly, mostly as a word processor; the 32K (home brew) Color Computer is usually idle. Both computer systems (computer, disk, cassette and display) are plugged into separate power strips. Thus, each system is individually controllable. In order to drive the printer from the Color Computer using standard software, the EPSON switch SW2 needs to be set to 0000. For the Commodore, using a "The Connection" serial interface, the settings must be 0010. Thus, whenever I print from the other computer, I must move the printer power cord to the other power strip, open the printer case and move one switch, and connect the other drive cable. The C-64 printer interface has a 2K buffer, but the Color Computer interface has no buffer. All my writing is done using ELITE*WORD, and I often must wait for one file to print out before working on another.

Obviously, things would go better if I had a large printer buffer to capture several pages of data and print it while I work on another file. Figure 1 shows how to merge my existing computers into a single, more efficient system. The printer and the 32K CoCo will be powered from a third power strip which turns on when either or both the other systems are active. A special interface board for the CoCo will have a serial input from the 64K CoCo printer port and a parallel input from the C-64 system. A separate parallel output will drive the printer. Either computer will be able to direct output to the printer. If the printer is busy, the requesting computer will have to wait as usual. I expect that 28K of memory would be available in the 32K CoCo after allowing for display memory, stack and controller program workspace. 28K of buffer is enough for more than 15 pages of double-spaced text, which exceeds any need I have had so far.

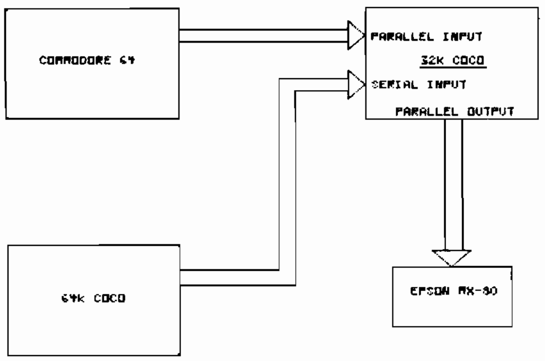


Figure 1. A special network connection will allow two computers to feed a third computer which will serve as a printer buffer.

Let me share some of my philosophy used in designing this system. Three primary considerations were involved: first, the new system should be compatible with commercial software running on both the 64K CoCo and the C-64. Primarily, that means no special printer drivers will be written for any commercial software. Second, the expansion will be modular. As I complete some part of the task, an improvement in system efficiency will result. Finally, no internal modifications will be made to either the 64K CoCo or the C-64. All these considerations are met by the (apparently) clumsy plan to configure the 32K CoCo interface to respond to either of the other computers as if it were a printer. That is, the input interfaces will handshake with the driver computers exactly as does the existing printer interface. Software options for straight-through printing or formatting by the 32K CoCo will be written.

At some future time, I may consider eliminating the "Connection" interface; most commercial software uses the Commodore serial port. To eliminate this interface would require hours of experimentation and study, designing an interface to convert from Commodore serial to RS-232 format, and there isn't time or need for that. The C-64 claims to have an RS-232 serial port available, but this requires a special output interface. Also, much commercial software for the C-64 does not support this port which is implemented by simulating a 6850 ACIA in software. Finally, the data transfer rate of the serial port is faster than the RS-232 transfer rate.

I am beginning to implement this printer buffer system as outlined above.

Due to various time pressures, the conversion will need to be made in several phases. Each phase will be reported in the column as the work is performed. A separate problem had to be solved first. The 32K CoCo must be capable of booting (starting up) unaided, so it must have an autostart ROM in the expansion (cartridge) port. I have an EPROM programmer for the C-64, along with 6502 development software which will handle the Commodore programming required. My 6809 development software has no way to send 6809 code to the C-64 programmer. The temporary link between the CoCo and the C-64 is presented this month; probably, the CoCo expansion interface will follow next month.

The simplest way to transfer data between dissimilar computers is to use a standard data rate and interface at the transmitting computer. If the software and hardware at the receiving computer is fast enough to capture the data as it comes, no handshake is needed. For this one-way data flow, the CoCo/C-64 interface can be a one-transistor level translator and inverter (Figure 2). R1 and D1 limit base drive to Q1, while Q1 and R2 drive PB7 of the Commodore User Port. The CoCo printer port incorporates a BUSY* signal, so a third wire is needed to feed back a high level ("not busy") to the serial in-line.

The program listing is a rudimentary data input program which services the interface of Figure 2. Figure 3 shows the flowchart for the program, which assembles incoming serial data into bytes and saves the data in sequential locations beginning at $2000. Since the C-64 has a timer available, complicated bit timing is not needed. Using a timer means that less experimentation is needed to get the timing correct. Instead of counting down a software loop, the CPU polls the CIA Interrupt Status bit to learn when the timer has finished.

For those who need the review, Figure 4 shows how the 8-bit serial asychronous data is formatted. A Start bit (TTL low level) is sent first, followed by eight data bits which may be either low or high. At least one Stop Bit (high level) is sent to complete the transmission of a single byte. Note that Radio Shack 1.0 BASIC sends only seven bits with one Stop bit; later

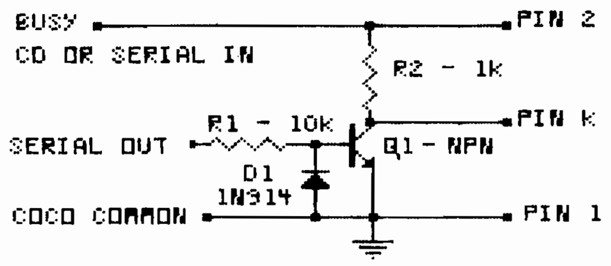
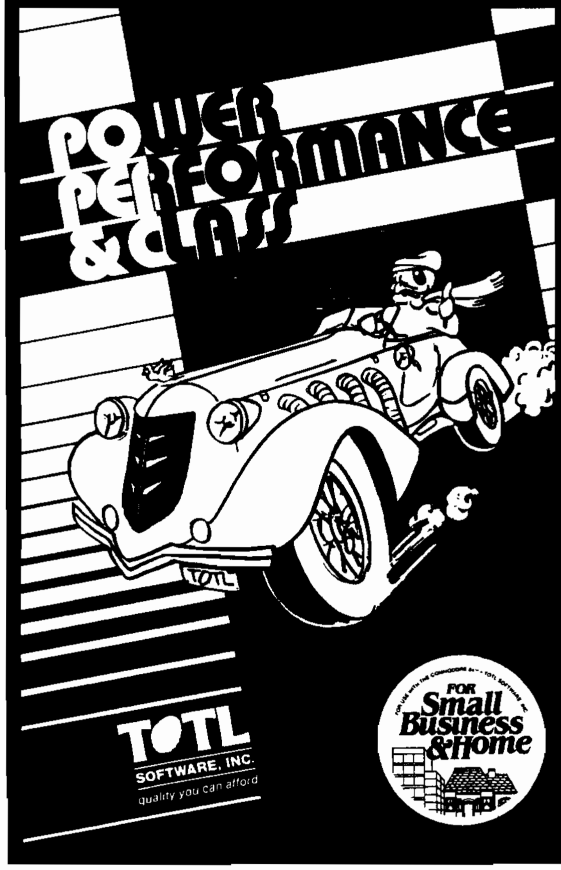


Figure 2. Two resistors, a diode and one transistor make up a data transmitter to send data from the Color Computer to the Commodore 64 (see text).

Announcing . . . TOTL. MONEYMINDER

FOR THE COMMODORE 64™

The home accounting package that will make your budget, not break it! ONLY $39.95



Money-Saving Bonus Paks of 64 Software

(BP-1)—(disk)
totl.text/
totl.speller/totl.label
reg. price $103 NOW **$79**

(BP-2)—(disk)
totl.business/
totl time manager/
totl.infomaster/totl.text
reg. price $228 NOW **$159**

(BP-3)—(disk)
totl.infomaster/
totl.text/totl.speller
reg. price $129 NOW **$99**

(BP-4)—(disk)
totl.text/
totl.speller/
research assistant
reg. price $118 NOW **$89**

(BP-5)—(tape)
totl.text/totl.label
reg. price $60 NOW **$49**

Commodore 64 and VIC 20 are trademarks of Commodore Business Machines Inc.

INFORMATION AND ORDER COUPON

	TAPE	DISK
TOTL.TEXT 2.0 (VIC + 8K)	☐ 24.95	☐ 28.95
TOTL.TEXT 2.5 (VIC + 16K)	☐ 34.95	☐ 38.95
TOTL.LABEL 2.1 (VIC + 16K)	☐ 19.95	☐ 23.95
TOTL TIME MGR. 2.1 (VIC + 8K)	☐ 29.95	☐ 33.95
RESEARCH ASST. 2.0 (VIC + 8K)	☐ 29.95	☐ 33.95
TOTL.BUSINESS 3.0 (VIC + 24K)		☐ 84.95
TOTL.TEXT 2.6 (C-64)	☐ 39.95	☐ 43.95
TOTL.SPELLER 3.6 (-C64)		☐ 34.95
TOTL.LABEL 2.6 (C-64)	☐ 19.95	☐ 23.95
TOTL TIME MGR. 2.6 (C-64)	☐ 34.95	☐ 38.95
RESEARCH ASST. 2.0 (C-64)	☐ 34.95	☐ 38.95
TOTL.INFOMASTER 3.6 (C-64)		☐ 49.95
TOTL.BUSINESS 3.6 (C-64)		☐ 94.95
TOTL.MONEYMINDER 3.6 (C-64)		☐ 39.95
BONUS PAK #________		________

Check, Money Order or C.O.D.* also accepted. *C.O.D. orders $2 00 additional (CA residents add 6½% sales tax)

Total ________
C.O D. Charges/Sales Tax ________
Shipping & Handling $3.00
Amount Enclosed ________

FOR ORDERING ONLY—CALL OUR TOLL FREE NUMBERS
Continental U.S. 1-800-351-1555, California 1-800-351-1551
Hawaii and Alaska 415-943-7877

☐ SEND MORE INFORMATION (no charge for catalog)

Name ________________
Street ________________
City ________ State ____ Zip ____
Phone ( ) ________ ☐ MC ☐ VISA
Card # ________ Exp. Date ____

TOTL SOFTWARE, INC.

quality you can afford
1555 Third Avenue
Walnut Creek, CA 94596
415/943-7877

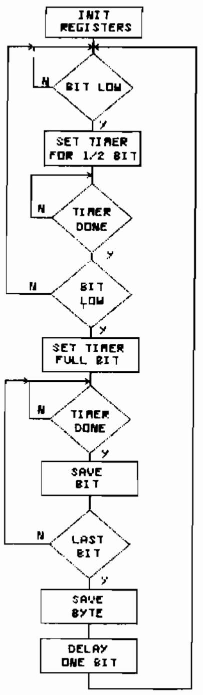


Figure 3. This flow chart describes the program listing for the Commodore to receive data from the Color Computer.

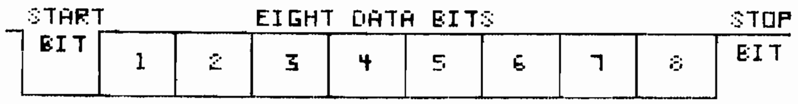


Figure 4. This diagram illustrates a typical binary data byte as transferred by the circuit of Figure 2.

versions send eight data bits and one Stop bit.

Refer to Figure 3 and Listing 2 for the following discussion. **NEW** initializes various locations and the program waits for Bit 7 to go low. **GET** performs a 6502 BIT test which sets the N Flag equal to Bit 7. Until the Start bit takes PB7 low, the BMI test forces the testing to continue. When the Start bit arrives, the half-bit delay is called to be sure the input is still low. This test provides noise rejection only. If the line is still low (valid start bit), INBIT calls a one-bit delay. This allows time for the first data bit to arrive and settle. Next, the incoming data is captured and a test for eight bits received is made. The loop is executed eight times, until **SAVX** becomes zero.

Incoming data appears on PB7. The **LDA BPORT** reads all eight bits, but Bit 7 is stripped off by shifting the bit into the Carry Bit. Then the Carry Bit is shifted into the location named WORD. After eight bits have been received, **DUMP** saves the assembled data byte into the next sequential buffer location and increments the pointer. When the Y Index "rolls over" from $FF to $00, the location **PAGE**, which is the high order byte of the buffer address, is incremented. This way, the transferred data can be as large as necessary up to $8000 (32768) bytes. *Special Note: The listing was assembled at $3000 to avoid destroying the Editor package at $C000. In normal use, this program is intended to reside at $C000, thus providing $8000 bytes of data buffer. Otherwise, only $1000 (4096) bytes of buffer is available with Listing 2.*

SKIP makes sure the CIA Interrupt Status bit is clear before a full-bit delay is called. After the delay, control returns to **FIX** to test for the next Start bit. The delay routine is called once (enter at HLFBIT) for a one-half bit delay, or twice (enter at FULBIT) for a one bit delay. The timer is started and the Timer A Interrupt Status Bit (Bit 0 in the CIA Interrupt Register) is repeatedly polled. When this bit goes high, the timer has timed out, so the RTS causes normal program operation to resume. There is a special caution regarding use of the CIA timers. Timer A can be operated in the free-running mode to allow generation of arbitrary waveforms for special purposes. The one-shot mode, as demonstrated here, should always be used for normal timing. This mode selection is shown controlled by the assignments for **TMRNIT** and **TMROFF**.

This program is intended to be loaded and operated under control of HESMON 64 or another debug monitor; the RESTORE key forces a stop. CoCo can send data using a simple BASIC program. Data integrity can be verified by using another BASIC program to checksum the data in CoCo

Listing 1

```
  5 REM DATA TRANSFER IN BINARY
 1Ø FOR X=49152 TO 49168
 2Ø HØ=PEEK(X)
 3Ø PRINT#-2,CHR$(HØ);
 4Ø NEXT
 42 REM CHECKSUM COMPUTATION
 45 AT=1:LM=65536
 5Ø FOR X=49152 TO 49168
 6Ø A=PEEK(X)
 7Ø AT=AT+A
 8Ø IF AT> LM THEN AT=AT-LM
 9Ø NEXT X
 1ØØ PRINT''CHECKSUM = '',AT
```

Listing 2

```
                ; THIS PROGRAM IMPLEMENTS A TTL
                ; LEVEL SERIAL INPUT PORT ON THE
                ; COMMODORE 64 COMPUTER WHICH WILL
                ; INPUT AND STORE BINARY DATA
                ;
                ; EQUATES
DDØØ             APORT    EQU $DDØØ    ; CIA REGISTERS
DDØ1             BPORT    EQU APORT+1
DDØ2             ADDR     EQU APORT+2
DDØ3             BDDR     EQU APORT+3
DDØ4             TMRALO   EQU APORT+4
DDØ5             TMRAHI   EQU APORT+5
DDØ6             TMRBLO   EQU APORT+6
```

and the same program to checksum the data in C-64 memory. A more "automatic" data transfer would require far more programming, so this simpler approach is a good compromise.

The BASIC program, Listing 1, will transfer binary data between a CoCo and a C-64 and checksum the data at both ends. Lines 10-40 send the data across to the C-64 which receives the data with the program in Listing 2. Compute the CoCo memory checksum before or after sending data by typing "GOTO 45". Lines 45 100 of the same program, entered into the C-64, will compute the checksum after the transmission. Note that line 10 specifies addresses 49152-49168 ($C000-$C010), which happens to be the first 16 bytes of the expansion area (Disk BASIC for CoCo if you have a disk). Obviously, this could have been any set of locations, as long as the C-64 buffer area is long enough. Note also that line 50 must specify the same addresses as line 10. The C-64 version must use the target addresses set up by the C-64 receive program.

I recommend the following sequence for data transfers using these programs:

1. Connect and test the interface.
2. If data is to be transferred for programming in an EPROM, use HESMON 64 to prepare the buffer area:

   F2000 2FFF FF

This command fills 4096 locations (a full 2732 EPROM) with $FF. Thus, if the code transferred is smaller than 4096 bytes, unused EPROM locations will remain undisturbed.

3. Set up the CoCo by entering the BASIC program. Compute the checksum now or later.
4. Start the receiving program in the C-64 (it will wait on data if the interface is connected) using:

   G3000

5. Type RUN on CoCo.
6. When CoCo prints "BREAK IN 40", hit RESTORE on the C-64.
7. Save the data using this HESMON command: (disk assumed)

   S"filename" 08 2000 2FFF

8. Return to BASIC (C-64) with the HESMON command XC; enter the checksum program and compute the checksum. In case other than HESMON is used, it may be necessary to load the data from disk with an offset to avoid conflicts with BASIC. If the checksum is OK, you are free to program the EPROM.

```
DDØ7               TMRBHI    EQU APORT+7
DDØD               CIA2IR    EQU APORT+$D
DDØE               TMRACR    EQU APORT+$E
DDØF               TMRBCR    EQU APORT+$F
                   ;
                   ; CONSTANTS
ØØØ9               TMRNIT    EQU $Ø9        ; TIMER ON/ONE SHOT
ØØØ8               TMROFF    EQU $Ø8        ; TIMER OFF
ØØ2C               BAUDLO    EQU $2C        ; TIMER VALUE FOR
ØØØ3               BAUDHI    EQU $Ø3        ; 6ØØ BAUD
                   ;
                   ; BUFFERS
ØØ7C               SAVA      EQU $7C
ØØ7E               SAVX      EQU $7E
ØØ7F               SAVY      EQU $7F
ØØ8Ø               POINTR    EQU $8Ø        ; DATA BUFFER POINTER
ØØ81               PAGE      EQU POINTR+1   ; BUFFER HI BYTE
ØØ82               WORD      EQU POINTR+2   ; INPUT SCRATCH BYTE
                   ;
3ØØØ                         ORG $3ØØØ
                   ; MAIN PROGRAM
3ØØØ A9 Ø8         NEW       LDA #TMROFF    ; INSURE TIMER OFF
3ØØ2 8D ØE DD                STA TMRACR
3ØØ5 A9 ØØ                   LDA #$ØØ       ; INIT DATA POINTER
3ØØ7 85 8Ø                   STA POINTR     ; LOW BYTE
3ØØ9 A8                      TAY            ; AND INDEX POINTER
3ØØA A9 2C                   LDA #BAUDLO    ; SET TIMER FOR
3ØØC 8D Ø4 DD                STA TMRALO     ; HALF-BIT TIME
3ØØF A9 Ø3                   LDA #BAUDHI
3Ø11 8D Ø5 DD                STA TMRAHI
3Ø14 A9 2Ø                   LDA #$2Ø       ; INIT DATA POINTER
3Ø16 85 81                   STA PAGE       ; HI BYTE
3Ø18 A9 Ø8         FIX       LDA #Ø8        ; INIT BIT COUNTER
3Ø1A 85 7E                   STA SAVX
3Ø1C 78                      SEI            ; KILL C64 INTERRUPTS
3Ø1D A9 ØØ                   LDA #ØØ        ; INIT INPUT
3Ø1F 85 82                   STA WORD       ; SCRATCH PAD
                   ;
                   ; INPUT LOOP
3Ø21 2C Ø1 DD      GET       BIT BPORT      ; TEST FOR START BIT
3Ø24 3Ø FB                   BMI GET        ; WAIT FOR IT
3Ø26 2Ø 53 3Ø                JSR HLFBIT     ; FOUND IT
3Ø29 2C Ø1 DD                BIT BPORT      ; WAIT ONE-HALF BIT
3Ø2C DØ F3                   BNE GET        ; FALSE START BIT?
3Ø2E 2Ø 5Ø 3Ø      INBIT     JSR FULBIT     ; SAMPLE NEXT BIT
3Ø31 AD Ø1 DD                LDA BPORT      ; READ PORT
3Ø34 ØA                      ASL A          ; GET INPUT DATA BIT
3Ø35 66 82                   ROR WORD       ; ROTATE INTO BUFFER
3Ø37 C6 7E                   DEC SAVX       ; COUNT BIT AND
3Ø39 FØ Ø3                   BEQ DUMP       ; TEST FOR LAS
3Ø3B 4C 2E 3Ø                JMP INBIT      ; GET MORE
3Ø3E A5 82         DUMP      LDA WORD       ; SAVE ASSEMBLED
3Ø4Ø 91 8Ø                   STA (POINTR),Y      ; DATA
3Ø42 C8                      INY            ; BUMP POINTER
3Ø43 DØ Ø2                   BNE SKIP       ; PAGE BOUNDARY?
3Ø45 E6 81                   INC PAGE       ; INCREMENT PAGE BIT
3Ø47 AD ØD DD      SKIP      LDA CIA2IR     ; CLEAR STATUS BIT
3Ø4A 2Ø 5Ø 3Ø                JSR FULBIT     ; WAIT FOR STOP BIT
3Ø4D 4C 18 3Ø                JMP FIX        ; AND CONTINUE
                   ;
                   ; POLLED TIMER DELAY
3Ø5Ø 2Ø 53 3Ø      FULBIT    JSR HLFBIT     ; TWICE FOR FULL BIT
3Ø53 A9 Ø9         HLFBIT    LDA #TMRNIT    ; START TIMER
3Ø55 8D ØE DD                STA TMRACR
3Ø58 AD ØD DD      TEST      LDA CIA2IR     ; WAIT FOR
3Ø5B 29 Ø1                   AND #$Ø1       ; STATUS BIT
3Ø5D FØ F9                   BEQ TEST
3Ø5F 6Ø                      RTS            ; RETURN
3Ø6Ø                         END
```

HILISTER - A Study and Teaching Aid (Part 1)

Move easily within your programs and highlight parts of text or listings to add emphasis, drama or clarity

by J. Morris Prosser

HiLister is a machine language program which may be called from either Applesoft or the monitor to invert one line at a time on the screen display, thus "highlighting" that line. In addition, an Applesoft program, a block of disassembled memory locations, a disk catalog (either drive), a memory dump (in both hex and ASCII), or almost anything else may be listed to the screen, after which one can jump to the beginning or end of the listing, move forward or backward by screen "pages", scroll either up or down, or step up or down one line at a time. Lines may be highlighted in this mode also.

HiLister began as a simple line inverter, to highlight lines on the screen while teaching a beginner's programming class. The instructor sat at the keyboard and used a separate monitor to show the class what was happening. In order to point out a particular line for discussion, he had to get up and point to it on the monitor. HiLister made it possible for him to remain seated, pointing out the line by causing it to be printed in inverse characters.

At that point, it was possible to highlight only those lines already on the screen display, so I added a list function to allow an entire Applesoft program to be examined with the highlighter. When the list function is in effect, if the highlight is moved to the bottom of the screen and an attempt is made to move it further, the screen scrolls up one line, and the bottom line is again highlighted. A similar action occurs at the top of the screen. The additional functions of jumping to beginning or end, paging, scrolling, and stepping are icing on the cake.

Once the Applesoft list function was in operation, I found that the program was very helpful for studying program listings at any time, rather than being useful only in a teaching situation. It was at this point that I decided to add a list function for machine language disassembly listings.

It also appeared that some other functions might be useful, so I added a command to dump a block of memory to the screen in hex and ASCII and another to allow the listing of long catalogs from either drive. The final step was to add a method of listing other things I had perhaps overlooked.

HiLister is initialized by "BRUN HILISTER" or by "BLOAD HILISTER" and "CALL 3276 8". The initialization consists of setting the ampersand (&) and ctrl-Y vectors. The program is then accessed by entering ctrl-Y from the monitor (for the highlighter function only), or "&" from Applesoft (for all functions). "&LIST" causes the Applesoft program in memory to be listed in its entirety to both the screen and to a buffer area used by HiLister for the list function. Commas or hyphens and beginning and ending addresses may be used as in the standard Applesoft LIST command to obtain a partial listing.

To get a listing of a machine language program or other disassembled machine code, the command is an ampersand followed by a dollar sign and the start address (in hex) of the memory to be disassembled. Thus, "&$8000" would print 256 lines of disassembled code starting at $8000 (a partial listing of HiLister, for example). "&$8000L" would produce the same result. Addition of a plus sign after the address (for example, &$8000) causes 512 lines of disassembled code to be listed. Note that "&$8000L" would produce only 256 lines of code, since the program looks for only one character following the address.

To obtain a memory dump, the command is "&$" followed by the range of memory to be dumped. For example, "&$8000.84FF" would dump the range $8000 to $84FF, just as in the normal monitor command.

Disk catalogs are listed by using the command "$C" for the default drive, or "&C1" or "&C2" to specify the drive.

To list anything else to the program buffer, use "&B" to initialize the output detour and the buffer, then list or print whatever is desired, then enter the HiLister program with "&E".

While the program is listing to the screen and buffer, ctrl-S and ctrl-C may be used to pause and end the listing, respectively, just as with the normal Applesoft LIST command. Note, however, that ctrl-C is not effective in a catalog listing.

If a program is too long to be completely listed to the buffer, the bell sounds and a message is displayed offering the options of using the part of the program already listed or leaving the HiLister program and re-entering it with only an elected part of the program to be listed. The buffer normally starts at $4000, so an Applesoft program of more than 57 sectors would overwrite it. The Applesoft program length is checked by HiLister, however, and if necessary the start of the buffer is moved up in memory. In this event, of course, the buffer size is decreased and it will not hold as long a listing.

Applesoft programs of this length or longer may be too long for complete listing. For very long programs it is better to load the program, delete those lines not required for study, and then invoke the list function of HiLister. This will provide for a larger buffer and make the maximum number of lines available for study. Note that an Applesoft program longer than 120 sectors will overwrite the HiLister program itself. In this case it is possible to load the Applesoft program, delete part of it, then BRUN HILISTER.

The assembly listing for HiLister is quite long and is liberally commented, so only a brief description of how the program works will be provided here (Listing 1).

Upon first running the program, the ampersand and ctrl-Y vectors are set up and control is returned to BASIC. Upon entry to the main program, the program determines whether the highlighter alone is requested, or one of the other options is desired. If a listing is required, the program sets the output vector (subroutine OUTSET) to cause all output to pass through the program, so that it may be listed to the buffer as well as to the screen. It also fills the buffer with carriage returns so there will be no extraneous material at the end of the listing. If an Applesoft listing, the program goes to a portion of code which replaces the standard Applesoft "LIST" routine. The standard routine could not be used, since it does not normally return to the caller and, in addition, some special formatting was required.

If a disassembly listing is requested, the program determines the start address for the listing, then checks to see whether 256 or 512 lines should be listed. This is done in subroutine "MEMLST," which also checks to see whether "DEF" is part of the address entered. The reason this is needed is that Applesoft would interpret this as the beginning of a "DEF FN" command, and so would replace it with the token for "DEF" ($B8). If this happens, the "DEF" address must be restored so the listing will start at the correct address. While this situation will seldom arise, I thought it should be covered.

MEMLST also checks to determine if a memory dump is desired rather than a disassembly listing. It does this by looking for a period between addresses.

When all is well, if a disassembly listing is requested, the program goes to "MONLIST," which replaces the monitor "LIST2" subroutine. It is called twice if 512 lines are to be listed.

If a memory dump is required, the program jumps to "DUMP," which performs a function similar to the "XAM" function in the monitor, with the added feature that the hex code is converted to ASCII and shown at the same time. Control (non-printing) characters are shown as blanks.

If a catalog listing has been requested, the program jumps to "CTLG," which first removes the pause from the DOS CATALOG routine, then calls it. When the catalog

Listing 1

```
0800            *  HILISTER1     (REV  04/16/84)
0800            *
0800            *     Written by
0800            *
0800            *  J. Morris Prosser
0800            *
0006            LINE     EQU $06        ;LINE NUMBER FOR HIGHLIGHTER
0007            TEMPY    EQU $07        ;TEMPORARY STORAGE FOR Y REGISTER
0009            TEMPX    EQU $09        ;TEMPORARY STORAGE FOR X REGISTER
0019            FLAG     EQU $19        ; FLAG FOR USE BY HIGHLIGHTER
001A            LSTFLG   EQU $1A        ; A/S LIST FLAG
001B            COUNT    EQU $1B        ; COUNTER
001C            PLUSFLG  EQU $1C        ;FLAG FOR EXTENDED MONITOR LIST
001D            CATFLG   EQU $1D        ; FLAG FOR CATALOG LISTING
001E            DIRFLG   EQU $1E        ;FLAG FOR STEP DIRECTION
0024            CH       EQU $24        ;CURSOR HORIZONTAL POSITION
0025            CV       EQU $25        ;CURSOR VERTICAL POSITION
0031            MODE     EQU $31        ;MODE OF MONITOR COMMAND
0036            CSWL     EQU $36        ;CHARACTER OUTPUT VECTOR
003A            PCL      EQU $3A        ;PROGRAM COUNTER
003C            A1L      EQU $3C        ; GENERAL PURPOSE COUNTER
003E            A2L      EQU $3E        ;GENERAL PURPOSE COUNTER
0040            A3L      EQU $40        ; GENERAL PURPOSE COUNTER
0042            A4L      EQU $42        ;GENERAL PURPOSE COUNTER
0050            LINNUM   EQU $50        ;GENERAL PURPOSE 16-BIT REGISTER
0085            FORPNT   EQU $85        ;GENERAL POINTER
009B            LOWTR    EQU $9B        ; GENERAL PURPOSE REGISTER
009D            DSCTMP   EQU $9D        ; TEMP STRING DESCRIPTOR
00B1            CHRGET   EQU $B1        ;GET CHAR.,INCREMENT POINTER
00B7            CHRGOT   EQU $B7        ;GET CHAR., NO INCREMENT
00F9            MEMFLG   EQU $F9        ;MONITOR LIST FLAG
00FA            BUFST    EQU $FA        ;  BEGINNING OF LIST BUFFER
```

Listing 1 *(continued)*

```
ØØFC            SCRST    EQU $FC       ; BEGINNING OF SCREEN BUFFER
ØØFE            LSTEND   EQU $FE       ;END OF LISTING
Ø2ØØ            IN       =   $2ØØ      ;Input buffer
Ø3DØ            BASIC    =   $3DØ      ;Soft entry to BASIC
Ø3EA            TELLDOS  =   $3EA      ;DOS routine to get change in
Ø3F5            AMP      =   $3F5      ;Ampersand vector
Ø3F8            CTRLY    =   $3F8      ;Control-Y vector
4ØØØ            BUFLE    =   $4ØØØ     ;Buffer low end
CØØØ            KBD      =   $CØØØ     ;Keyboard input address
CØ1Ø            KBDSTRB  =   $CØ1Ø     ;Keyboard strobe
D61A            FNDLIN   =   $D61A     ;Find mem. loc. of line in LINNUM
DAØC            LINGET   =   $DAØC     ;Get line no. from input buffer
DAFB            CRDO     =   $DAFB     ;Print carriage return
DB5C            OUTDO    =   $DB5C     ;Print character in accumulator
DEC9            SYNERR   =   $DEC9     ;Syntax error routine
ED24            LINPRT   =   $ED24     ;Print line number
F8DØ            INSTDSP  =   $F8DØ     ;Print disassembled instruction
F94Ø            PRNTYX   =   $F94Ø     ;Print Y and X registers
F953            PCADJ    =   $F953     ;Adjust program counter
FBC1            BASCALC  =   $FBC1     ;Calc. start addr. of screen line
FC22            VTAB     =   $FC22     ;Set cursor vertical position
FC58            HOME     =   $FC58     ;Clear screen - home cursor
FC9C            CLREOL   =   $FC9C     ;Clear to end of line
FCBA            NXTA1    =   $FCBA     ;Increment pointer A1L,A1H
FDDA            PRBYTE   =   $FDDA     ;Print accumulator as hex
                                        byte
FDED         63 COUT     =   $FDED     ;Print to output device
FDFØ            COUT1    =   $FDFØ     ;Print to screen
FE2C            MOVE     =   $FE2C     ;Move memory block
FF3A            BELL     =   $FF3A     ;Sound bell
FFA7            GETNUM   =   $FFA7     ;Get hex bytes from input buffer
FFC7            ZMODE    =   $FFC7     ;Set MODE for GETNUM
8ØØØ                     ORG $8ØØØ
8ØØØ                     NOG
8ØØØ            *
8ØØØ            * Set ampersand and ctrl-Y vectors
8ØØØ            *
8ØØØ A9 4C      START    LDA #$4C
8ØØ2 8D F5 Ø3            STA AMP
8ØØ5 8D F8 Ø3            STA CTRLY
8ØØ8 A9 8Ø               LDA /BEGIN
8ØØA 8D F6 Ø3            STA AMP+1
8ØØD 8D F9 Ø3            STA CTRLY+1
8Ø1Ø A9 1B               LDA #BEGIN
8Ø12 8D F7 Ø3            STA AMP+2
8Ø15 8D FA Ø3            STA CTRLY+2
8Ø18 4C DØ Ø3            JMP BASIC
8Ø1B A2 ØØ      BEGIN    LDX #Ø        ;Clear flags
8Ø1D 86 1D               STX CATFLG
8Ø1F 86 1A               STX LSTFLG
8Ø21 86 F9               STX MEMFLG
8Ø23 86 1C               STX PLUSFLG
8Ø25 86 1E               STX DIRFLG
8Ø27 C9 ØØ      HILITER  CMP #Ø        ;Other command
8Ø29 FØ Ø3               BFL HILITER1  ;No - HILITER
8Ø2B 4C CF 8Ø            JMP LISTER
8Ø2E            *
8Ø2E A2 ØØ      HILITER1 LDX #Ø        ;Set FLAG and LINE to zero
8Ø3Ø 86 19               STX FLAG
8Ø32 86 Ø6               STX LINE
8Ø34 FØ 5B               BFL NXTLN     ;Branch always
8Ø36 2C ØØ CØ   KEYCHK   BIT KBD       ;Check keyboard
8Ø39 1Ø FB               BPL KEYCHK    ;Key not pressed
8Ø3B AD ØØ CØ            LDA KBD       ;Key pressed - get it
8Ø3E 2C 1Ø CØ            BIT KBDSTRB   ;Reset keyboard strobe
8Ø41 C9 9B               CMP #$9B      ;Is it 'ESC'
8Ø43 DØ Ø5               BTR NOTESC    ;No - branch
8Ø45 85 19               STA FLAG      ;Yes - set FLAG
8Ø47 4C 91 8Ø            JMP NXTLN     ;Remove highlight and exit
8Ø4A C9 88      NOTESC   CMP #$88      ;Is it left arrow
```

listing is complete, the program restores the pause to DOS.

When listing is completed, the program pages back one screenful and sets the address at that point as the start of the screen buffer and as the address of the end of the listing. It then reprints this screen, sounds the bell, and prints a "LISTING COMPLETED" message.

The operation of the jumps to beginning and end of the listing is fairly obvious - simply a matter of setting the start of the screen buffer to the start of the listing buffer or the end address of the listing, as mentioned above.

The paging and scrolling are based on checking the buffer for the next previous or next following carriage return. For paging, 23 returns are counted before the next screen is printed, while for scrolling the screen is reprinted after each return is found, and then the next one is searched for.

Stepping one line at a time is accomplished by use of the space bar. The program checks to see whether the last movement called for was forward or backward (by looking at DIRFLG), then calls UPDO or DOWNDO, as appropriate. Default is UPDO, to scroll forward one line.

Commands available for manipulating the listing are:

B - jump to the beginning of the listing

E - jump to the end of the listing

+ or ; - page forward (previous bottom line becomes top line)

- or = - page back (previous top line becomes bottom line)

Right arrow - scroll up (stops on any keypress)

Left arrow - scroll down (stops on any keypress)

Space bar - step forward or backward one line.

& - calls highlighter

ESC - returns to BASIC

If the highlighter was requested, the top line of the screen is changed to the inverse of what it was; that is, normal characters become inverse, inverse characters become normal, and flashing characters are unchanged. The program then looks for keyboard input. If a right arrow is pressed, the top line is restored and the next line is inverted. Further presses of the right arrow key cause the highlighting line to move on down the screen in this manner. The left arrow works the same way, except that it moves the "highlight" up the screen.

If the highlighter was called from any list routine, then when the highlighted line is at the bottom of the screen, further right arrows make the screen scroll up one line. Left arrows work in an analogous fashion when the highlighted line is at the top of the screen. The "ESC" key causes the currently highlighted line to be restored and the program returns to the caller.

One problem occurs with the highlighter if your listing includes lower case letters, in that the Apple II cannot show lower case letters in inverse. I thought the best thing to do in this event was to convert the lower case to upper case before highlighting. Naturally, when the highlighting is removed the material remains in all upper case. If the list function is in effect, the lower case will be restored as soon as the screen is reprinted for any reason, such as scrolling, paging, or stepping. Another way of handling this situation would be to show all characters except lower case in inverse, leaving the lower case characters normal. If you would like to try this option, get into the monitor with CALL-151, then type "809C:B0 16 EA EA" and press RETURN - after having BLOADed HILISTER, of course.

While the highlighter is in operation, all keys except "ESC" and the right and left arrows are ignored.

The assembly listing for the highlighter portion of the program is included here as Listing 1. This is a stand-alone program as shown, so it can be put to use immediately after keying it in. It should be saved as HiLister1. If you are entering the code without using an assembler, the command is:

BSAVE HILISTER1, A$8000, L$D0.

Part 2 of this article will present a listing of the remainder of the program, and will include instructions for adding it on. Some of the code in the first part of the listing appears redundant, but it is necessary for interfacing to the other parts of the program.

Listing 1 *(continued)*

```
804C DØ 1F            BTR NOTLFT     ;No - branch
804E A6 Ø6            LDX LINE       ;Yes - get LINE
805Ø CA               DEX            ;and decrement it
8051 1Ø 14            BPL LFT1       ;Not top of screen
8Ø53 E8               INX            ;Top of screen
8Ø54 A5 1A            LDA LSTFLG     ;List in effect
8Ø56 Ø5 F9            ORA MEMFLG
8Ø58 Ø5 1D            ORA CATFLG
8Ø5A FØ ØB            BFL LFT1       ;No - branch
8Ø5C 85 19            STA FLAG       ;Yes
8Ø5E 2Ø 91 8Ø         JSR NXTLN      ;Restore top line
8Ø61 2Ø 83 83         JSR DOWNDO     ;Scroll down one line
8Ø64 4C 91 8Ø         JMP NXTLN      ;Invert it
8Ø67 86 Ø9     LFT1   STX TEMPX
8Ø69 A2 ØØ            LDX #Ø
8Ø6B FØ 23            BFL INVERT     ;Put in highlight
8Ø6D C9 95     NOTLFT CMP #$95       ;Is it right arrow
8Ø6F DØ C5            BTR KEYCHK     ;No - get next keypress
8Ø71 A6 Ø6            LDX LINE       ;Get line number
8Ø73 E8               INX            ;and increment it
8Ø74 EØ 18            CPX #24        ;Bottom line
8Ø76 DØ 14            BTR RT1        ;No - branch
8Ø78 CA               DEX            ;Yes
8Ø79 A5 1A            LDA LSTFLG     ;List in effect
8Ø7B Ø5 F9            ORA MEMFLG
8Ø7D Ø5 1D            ORA CATFLG
8Ø7F FØ ØB            BFL RT1        ;No - branch
8Ø81 85 19            STA FLAG       ;Yes
8Ø83 2Ø 91 8Ø         JSR NXTLN      ;Restore line
8Ø86 2Ø 65 83         JSR UPDO       ;Scroll up one line
8Ø89 4C 91 8Ø         JMP NXTLN      ;Invert it
8Ø8C 86 Ø9     RT1    STX TEMPX      ;Save line number
8Ø8E A2 ØØ            LDX #Ø
8Ø9Ø CA        INVERT DEX
8Ø91 A5 Ø6     NXTLN  LDA LINE       ;Get line number
8Ø93 2Ø C1 FB         JSR BASCALC    ;Find address of left end
8Ø96 AØ 27            LDY #39        ;Start at end of line
8Ø98 B1 28     GETCH  LDA ($28),Y    ;Get character
8Ø9A C9 EØ            CMP #$EØ       ;Is it lower case
8Ø9C 9Ø Ø2            BLT NOTLC      ;No - check further
8Ø9E 29 DF            AND #$DF       ;Yes - make it upper case
8ØAØ C9 AØ     NOTLC  CMP #$AØ       ;Is it normal
8ØA2 9Ø Ø4            BLT INV        ;No - check further
8ØA4 29 3F            AND #$3F       ;Yes - invert it
8ØA6 BØ ØC            BGE DISP       ;and display it
8ØA8 C9 4Ø     INV    CMP #$4Ø       ;Is it flashing
8ØAA BØ ØA            BGE NXTCH      ;Yes - don't change it
8ØAC 69 8Ø            ADC #$8Ø       ;Must be inverse - make it normal
8ØAE C9 AØ            CMP #$AØ       ;Normal now
8ØBØ BØ Ø2            BGE DISP       ;Yes - display it
8ØB2 69 4Ø            ADC #$4Ø       ;No - make it so
8ØB4 91 28     DISP   STA ($28),Y    ;And print it
8ØB6 88        NXTCH  DEY            ;Get next character
8ØB7 1Ø DF            BPL GETCH      ;Not done yet
8ØB9 A5 19            LDA FLAG       ;Is FLAG set
8ØBB FØ Ø5            BFL CONT       ;No - check X
8ØBD A2 ØØ            LDX #Ø         ;Yes - clear it
8ØBF 86 19            STX FLAG
8ØC1 6Ø               RTS            ;Done
8ØC2 8A        CONT   TXA            ;X=Ø
8ØC3 DØ Ø3            BTR CONT1      ;No - branch
8ØC5 4C 36 8Ø         JMP KEYCHK     ;Yes - get next command
8ØC8 A5 Ø9     CONT1  LDA TEMPX      ;Invert next line
8ØCA 85 Ø6            STA LINE
8ØCC E8               INX
8ØCD FØ C2            BFL NXTLN      ;Branch always
8ØCF           *
8ØCF D8        LISTER RTS
8ØDØ                  END
```

Super Simple Numeric Sort

by Robert L. Martin WB2KTG

Arrange a list in numerical order without a user supplied sorting program

Everyone, at some time, has had to take a list of numbers and arrange them in numerical order. The effort involved in accomplishing this task can, of course, be minimized by the use of a computer and a sorting program. Explained in this article is a sorting technique which doesn't require a user supplied program, but instead uses a built-in BASIC feature-automatic program statement sequencing.

All BASIC interpreters will allow non-sequential program statement entry. That is, the line numbers of statements need not be entered in any specific order. The BASIC interpreter will automatically LIST them in ascending order.

To arrange a list of numbers in ascending order, input each number followed by a period, asterisk, or some other non-numeric character. For non-integer values the decimal point will serve as the non-numeric character.

The Basic interpreter assumes that any digits input preceding a non-numeric character are line numbers. All alphanumeric characters entered following the first non-numeric character are assumed to be BASIC program statements. As long as no attempt is made to RUN the program, no error message will be given.

The example shown is the actual printed output from my Sharp PC-1500 pocket computer and CE-150 printer/plotter.

The use of this technique was discovered at work when I was given a list of 140 repair orders to sequence. Each repair order number was four digits long. Fortunately, I had my PC-1500 with me, along with a bit of imagination. I hope this example of using a computer's "hidden" talents will result in other non-standard techniques being developed to save the time and patience of the human interface. MICRO

Sample Printout From Sharp PC-1500/CE-150

```
29              29.
36.5            36.5
414             414.
13.2            13.2
5               5.
1019            1019.
7.25987         7.25987
```

a)List of Numbers

b)Numbers as Input to the Computer (note Decimal Points).

```
   5:.
   7:.25987
  13:.2
  29:.
  36:.5
 414:.
1019:.
```

c)Output of Computer in Response to a "LLIST" command.

FLOPPY DISKS SALE *$1.19 ea.

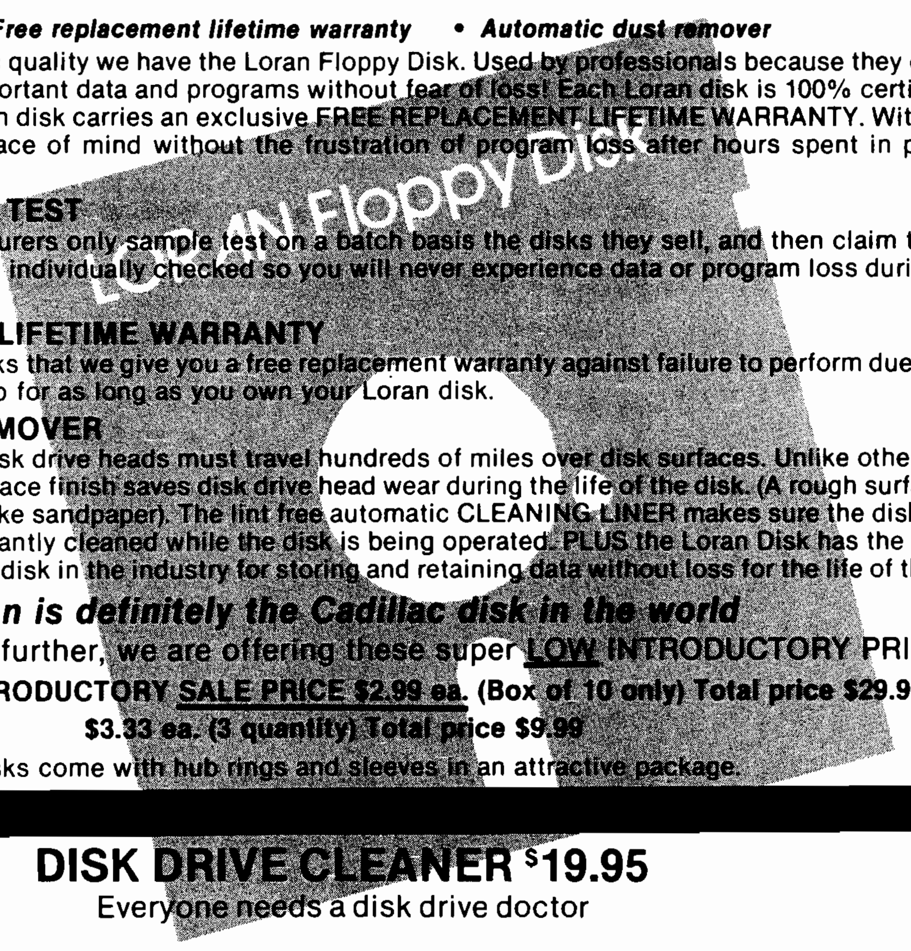
Economy Model or Cadillac Quality

LORAN CERTIFIED PERSONAL COMPUTER DISK ***We have the lowest prices!*** LORAN CERTIFIED PERSONAL COMPUTER DISK

***ECONOMY DISKS**

Good quality 5¼" single sided **double** density with hub rings.

Bulk Pac	100 Qty.	$1.19 ea.	Total Price	$119.00
	10 Qty.	1.39 ea.	Total Price	13.90



CADILLAC QUALITY

• ***Each disk certified*** • ***Free replacement lifetime warranty*** • ***Automatic dust remover***

For those who want cadillac quality we have the Loran Floppy Disk. Used by professionals because they can rely on Loran Disks to store important data and programs without fear of loss! Each Loran disk is 100% certified (an exclusive process) plus each disk carries an exclusive FREE REPLACEMENT LIFETIME WARRANTY. With Loran disks you can have the peace of mind without the frustration of program loss after hours spent in program development.

100% CERTIFICATION TEST

Some floppy disk manufacturers only sample test on a batch basis the disks they sell, and then claim they are certified. Each Loran disk is individually checked so you will never experience data or program loss during your lifetime!

FREE REPLACEMENT LIFETIME WARRANTY

We are so sure of Loran Disks that we give you a free replacement warranty against failure to perform due to faulty materials or workmanship for as long as you own your Loran disk.

AUTOMATIC DUST REMOVER

Just like a record needle, disk drive heads must travel hundreds of miles over disk surfaces. Unlike other floppy disks the Loran smooth surface finish saves disk drive head wear during the life of the disk. (A rough surface will grind your disk drive head like sandpaper). The lint free automatic CLEANING LINER makes sure the disk-killers (dust & dirt) are being constantly cleaned while the disk is being operated. PLUS the Loran Disk has the highest probability rate of any other disk in the industry for storing and retaining data without loss for the life of the disk.

Loran is definitely the Cadillac disk in the world

Just to prove it even further, we are offering these super LOW INTRODUCTORY PRICES

List $4.99 ea. INTRODUCTORY SALE PRICE $2.99 ea. (Box of 10 only) Total price $29.90

$3.33 ea. (3 quantity) Total price $9.99

All disks come with hub rings and sleeves in an attractive package.

DISK DRIVE CLEANER $19.95

Everyone needs a disk drive doctor

FACTS

- 60% of all drive downtime is directly related to poorly maintained drives.
- Drives should be cleaned each week regardless of use.
- Drives are sensitive to smoke, dust and all micro particles.
- Systematic operator performed maintenance is the best way of ensuring error free use of your computer system.

The Cheetah disk drive cleaner can be used with single or double sided 5¼" disk drives. The Cheetah is an easy to use fast method of maintaining efficient floppy diskette drive operation.

The Cheetah cleaner comes with 2 disks and is packed in a protective plastic folder to prevent contamination.

List $29.95 / Sale $19.95

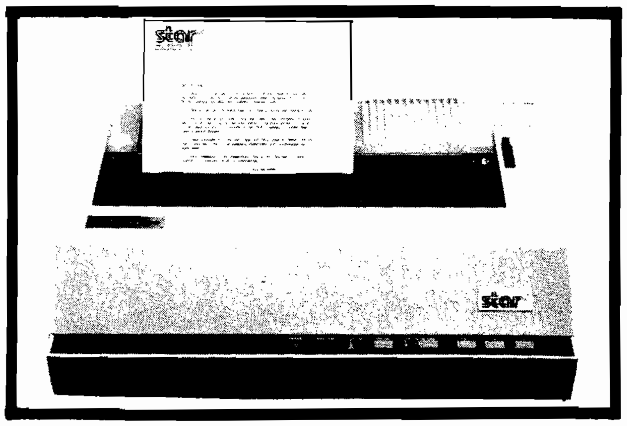
Add $10.00 for shipping, handling and insurance. Illinois residents please add 6% tax. Add $20.00 for CANADA, PUERTO RICO, HAWAII orders. WE DO NOT EXPORT TO OTHER COUNTRIES.

Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery, 2 to 7 days for phone orders, 1 day express mail! Canada orders must be in U.S. dollars. Visa - MasterCard - C.O.D.

PROTECTO
ENTERPRIZES (WE LOVE OUR CUSTOMERS)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

EXECUTIVE LETTER QUALITY
DAISY WHEEL PRINTER SALE 379^{00}

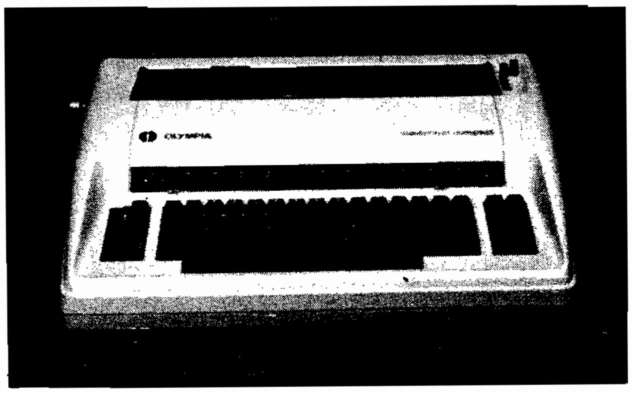
COMSTAR 13"



computer printer

COMSTAR 13" "DAISY WHEEL" POWER TYPE PRINTER is typewriter friendly. It uses a simple drop in cassette ribbon. Just turn on the COMSTAR 13" for Crip executive quality correspondence at 18 CPS with a daisy wheel that prints 96 power type flawless characters, bi-directional. Designed for personal and business applications. COMSTAR 13" carriage accepts paper from letter to legal size, continuous computer paper or single sheets, you can set right and left margins, vertical and horizontal tabs. (Serial and parallel interface). LIST PRICE 599^{00}
SALE PRICE 379^{00}

OLYMPIA "DAISY WHEEL" COMBINATION
PRINTER/TYPEWRITER SALE 489^{00}



Olympia (WORLD'S FINEST)

THE **Olympia** COMPUTER PRINTER **ELECTRONIC TYPEWRITER** is the ultimate for Home, Office, and Word Processing. You get **the best Electronic Typewriter made** and used by the world's largest corporations (better than IBM Selectric) plus a **Superb Executive Correspondence Computer Printer!!** (Two machines in one!) Just flick the switch for the option you want to use. The extra large carriage allows 14⅛" printer paper width. It has cassette ribbon lift off correction. Baud rates, Jumper selectable 75 through 19,200 (parallel interface)
LIST 799^{00} **SALE 489^{00}**

• 15 DAY FREE TRIAL — 90 DAY FREE REPLACEMENT GUARANTEE

Add $17.50 for shipping and handling!!

Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery, 2 to 7 days for phone orders, 1 day express mail! Canada orders must be in U.S. dollars. VISA — MASTER CARD ACCEPTED. We ship C.O.D.

PROTECTO
ENTERPRIZES (WE LOVE OUR CUSTOMERS)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

SANYO MONITOR SALE!!

9" Data Monitor

- 80 Columns × 24 lines
- Green text display
- Easy to read - no eye strain
- Up front brightness control
- High resolution graphics
- Quick start - no preheating
- Regulated power supply
- Attractive metal cabinet
- UL and FCC approved

- *15 Day Free Trial - 90 Day Immediate Replacement Warranty*

9" Screen - Green Text Display	**$ 69.00**
12" Screen - Green Text Display (anti-reflective screen)	**$ 99.00**
12" Screen - Amber Text Display (anti-reflective screen)	**$ 99.00**
12" Screen-Super 1000 Line Amber Text Display	**$129.00**
14" Screen - Color Monitor (national brand)	**$249.00**

Display Monitors From Sanyo

With the need for computing power growing every day, Sanyo has stepped in to meet the demand with a whole new line of low cost, high quality data monitors. Designed for commercial and personal computer use. All models come with an array of features, including up-front brightness and contrast controls. The capacity 5 × 7 dot characters as the input is 24 lines of characters with up to 80 characters per line.

Equally important, all are built with Sanyo's commitment to technological excellence. In the world of Audio/Video, Sanyo is synonymous with reliability and performance. And Sanyo quality is reflected in our reputation. Unlike some suppliers, Sanyo designs, manufactures and tests virtually all the parts that go into our products, from cameras to stereos. That's an assurance not everybody can give you!

SANYO

Official Video Products of the Los Angeles 1984 Olympics

• LOWEST PRICES • 15 DAY FREE TRIAL • 90 DAY FREE REPLACEMENT WARRANTY
• BEST SERVICE IN U.S.A. • ONE DAY EXPRESS MAIL • OVER 500 PROGRAMS • FREE CATALOGS

Add $10.00 for shipping, handling and insurance. Illinois residents please add 6% tax. Add $20.00 for CANADA, PUERTO RICO, HAWAII orders. WE DO NOT EXPORT TO OTHER COUNTRIES.

Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery, 2 to 7 days for phone orders, 1 day express mail! Canada orders must be in U.S. dollars. Visa - MasterCard - C.O.D.

PROTECTO
ENTERPRIZES (WE LOVE OUR CUSTOMERS)
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

feature

CMPRSS — *Improved Applesoft Compression Program*

Compress large programs easily and retain comments without overflowing Called Line Number Table

by Ian R. Humphreys

```
                 ;**APPLESOFT SUBROUTINES**
                 ;
D61A              FNDLIN   EQU $D61A     ;Find start of
Ø8ØØ                                     ;givn Applesft ln
D697              STXTPT   EQU $D697     ;Init TXTPTR for
Ø8ØØ                                     ;pass of program
DAØC              LINGET   EQU $DAØC     ;Convrt dec to hex
DAFB              CRDO     EQU $DAFB     ;Output carriage
Ø8ØØ                                     ;return to screen
DB3A              STROUT   EQU $DB3A     ;Output a text
Ø8ØØ                                     ;string to screen
ED24              LINPRT   EQU $ED24     ;Print a hex line
Ø8ØØ                                     ;# in decimal
ØØB7              CHRGOT   EQU $ØØB7     ;Get curr byte
Ø8ØØ                                     ;w/o inc TXTPTR
ØØB1              CHRGET   EQU $ØØB1     ;Inc TXTPTR and
Ø8ØØ                                     ;get next byte
```

```
                 ;
                 ;**ZERO PAGE LOCATIONS**
                 ;
ØØØ7              MAXX     EQU $ØØØ7     ;Loop ctrl for
Ø8ØØ                              ;transfLINBUF to new prog
ØØØ5              OLDBEG   EQU $ØØØ5     ;Ptr to last EOS
Ø8ØØ                                     ;in orig prog
nØØ4              LASTX    EQU $ØØØ4     ;Ptr to last EOS
Ø8ØØ                                     ;in LINBUF
ØØØ3              NEWPTR+1 EQU $ØØØ3
ØØØ2              NEWPTR   EQU $ØØØ2     ;Ptr to curr posn
Ø8ØØ                                     ;in compr prog
ØØØ1              IFFLAG   EQU $ØØØ1     ;Flag set when IF
Ø8ØØ                                     ;found in line
ØØØØ              ERRORS   EQU $ØØØØ     ;Flag for errors
Ø8ØØ                                     ;during PASS #1
ØØØA              LSTEOS   EQU $ØØØA     ;Last EOS token
Ø8ØØ                                     ;$ØØ or $FF
ØØØ9              OLDEOP+1 EQU $ØØØ9
ØØØ8              OLDEOP   EQU $ØØØ8     ;Value of EPROG
Ø8ØØ
```

Editor's Note: This program improves on programs previously done by: Barton M. Bauers (MICRO 52:89); Peter J.G. Meyer (MICRO 55:26).

Requirements:
Apple II or Apple II Plus; 48K and Applesoft BASIC in ROM

I had just finished writing a large, well-commented Applesoft program which was part of a major System I was working on. Unfortunately, when I came to test it, there was not enough room for its several large arrays and various string variables, and the program would not run. Coincidentally, on that same day, I purchased the September 1982 edition of MICRO magazine and was excited to see that it contained an article by Barton M. Bauers, giving a source listing of a machine language routine which compressed Applesoft programs. I eagerly hurried home, read the article and proceeded to key it into my Apple. I tested it on several small programs first and found that it seemed to work as described, so I set about running COMPRESS on my large program. Much to my dismay, COMPRESS aborted with ERROR #3 which meant that the Called Line Number Table had overflowed and so I couldn't use it! Not only does Barton Bauers' program impose a limit of 256 called line numbers, but it doesn't even check for duplicates, so for anything but a very small program the table soon fills up and overflows. One of the major reasons for wanting to compress the Applesoft code cannot be accommodated! Also, Mr. Bauers' program contains an error. Applesoft allows a statement of the form:

```
1ØØ NEXT I,J,K
```

Mr. Bauers' COMPRESS reduces this to:

```
1ØØ NEXT
```

instead of:

```
1ØØ NEXT :NEXT :NEXT
```

introducing a logic error into your Applesoft program!

Not being able to COMPRESS my large program, I resorted to removing all the REMs manually and finally, after several hours work, my program was small enough to run. Unfortunately, my source version has suffered as it now lacked comments and was consequently difficult to read. I resolved that I would redesign and rewrite the compression routine and I hereunder present my results. I have called my routine CMPRSS because it will compress an Applesoft program even more than COMPRESS does; it also uses less RAM space.

What CMPRSS does

CMPRSS compresses an Applesoft program by:

(a) Concatenating as many statements as possible onto one line, thus eliminating many of the unreferenced line numbers

(b) Removing the text of REM statements and where possible the REM itself (in some instances even when a REM line is referenced)*

(c) Removing LETs

(d) Removing the variable names from NEXT statements (correctly!)

(e) Truncating variable names to a maximum of two characters*

* *Additional features not performed by COMPRESS.*

```
                                        ;at beg of PASS#2
ØØ51            LINNUM+1 EQU $ØØ51
ØØ5Ø            LINNUM   EQU $ØØ5Ø      ;Line num returnd
Ø8ØØ                                    ;by LINGET
ØØ67            TXTTAB   EQU $ØØ67      ;Ptr to start of
Ø8ØØ                                    ;Applesoft prog
ØØ6E            EARS+1   EQU $ØØ6E
ØØ6D            EARS     EQU $ØØ6D      ;Ptr to end of
Ø8ØØ                                    ;array space
ØØ6C            ARS+1    EQU $ØØ6C
ØØ6B            ARS      EQU $ØØ6B      ;Ptr to start of
Ø8ØØ                                    ;array space
ØØ6A            LOMEM+1  EQU $ØØ6A
ØØ69            LOMEM    EQU $ØØ69      ;Lomem pointer
ØØ68            TXTTAB+1 EQU $ØØ68
ØØ74            HIMEM+1  EQU $ØØ74
ØØ73            HIMEM    EQU $ØØ73      ;Himem pointer
ØØ9C            LSTLIN+1 EQU $ØØ9C
ØØ9B            LSTLIN   EQU $ØØ9B      ;Ptr to start of
Ø8ØØ               ;line found by FNDLIN
ØØAF            EPROG    EQU $ØØAF      ;Ptr to end of
Ø8ØØ                                    ;Applesoft prog
ØØB9            TXTPTR+1 EQU $ØØB9
ØØB8            TXTPTR   EQU $ØØB8      ;Ptr to current
Ø8ØØ                                    ;byte of program
ØØFD            LN2+1    EQU $ØØFD
ØØFC            LN2      EQU $ØØFC      ;Hex line number
Ø8ØØ                                    ;of undefnd line
ØØFB            LN1+1    EQU $ØØFB
ØØFA            LN1      EQU $ØØFA      ;Hex line number
Ø8ØØ                                    ;containing error
ØØF9            TOKEN    EQU $F9
Ø8ØØ                                    ;GOSUB,THEN token
ØØB8            OLDPTR   EQU $ØØB8      ;Ptr to curr posn
Ø8ØØ                                    ;in old program
ØØFC            TEMP     EQU $ØØFC      ;Holds EOS byte
Ø8ØØ               ;until put into LSTEOS
```

```
               ;
               ;**OTHER LOCATIONS**
               ;
Ø3DØ            DOSWS    EQU $Ø3DØ      ;DOS warmst vector
Ø3F5            BJP      EQU $Ø3F5      ;& vector
95ØØ            LINBUF   EQU $95ØØ      ;Base address of
Ø8ØØ                                    ;cmprssd ln buffer
```

```
               ;
               ;**CONSTANTS**
               ;
ØØØØ            ENDLIN   EQU $ØØ        ;Non-referenced
Ø8ØØ                                    ;line token
ØØ22            QUOTE    EQU $22        ;ASCII quote
ØØ2C            COMMA    EQU $2C        ;ASCII comma
ØØ3Ø            ZERO     EQU $3Ø        ;ASCII zero
ØØ39            NINE     EQU $39        ;ASCII '9'
ØØ3A            COLON    EQU $3A        ;ASCII ':'
ØØ41            LETTRA   EQU $41        ;ASCII 'A'
ØØ5A            LETTRZ   EQU $5A        ;ASCII 'Z'
ØØ82            NXTTOK   EQU $82        ;NEXT token
ØØAA            LETTOK   EQU $AA        ;LET token
ØØAB            GOTOTK   EQU $AB        ;GOTO token
ØØBØ            GOSBTK   EQU $BØ        ;GOSUB token
ØØAD            IFTOK    EQU $AD        ;IF token
ØØB2            REMTOK   EQU $B2        ;REM token
ØØC4            THENTK   EQU $C4        ;THEN token
ØØFF            REFLIN   EQU $FF        ;Referenced line
Ø8ØØ                                    ;token
Ø8ØØ                                    ;
```

```
9000                    ORG $9000
9000 A9 13     START    LDA #< BEGIN   ;Establish &
9002 8D F6 03           STA BJP1       ; vector
9005 A9 90              LDA #> BEGIN
9007 8D F7 03           STA BJP2
900A A9 00              LDA #< START   ;Reset HIMEM to
900C 85 73              STA HIMEM      ; protect CMPRSS
900E A9 90              LDA #> START
9010 85 74              STA HIMEM+1
9012 60                 RTS
9013 20 FB DA  BEGIN    JSR CRDO       ;Output CR to screen
9016 A9 62              LDA #< PASS1A  ;Print PASS #1
9018 A0 94              LDY #> PASS1A  ;message
901A 20 3A DB           JSR STROUT
901D A2 00              LDX #$00       ;Init error mess
901F 86 00              STX ERRORS
9021 20 97 D6           JSR STXTPT     ;Init TXTPTR
9024 20 B1 00  NXTLIN   JSR CHRGET     ;Get next byte
9027 A0 01              LDY #$01
9029 B1 B8              LDA (TXTPTR),Y  ;End-of-prog?
902B D0 27              BNE SAVLIN     ;No-so branch
902D 20 FB DA           JSR CRDO
9030 A9 6D              LDA #< PASS1B   ;Print End Pass1
9032 A0 94              LDY #> PASS1B
9034 20 3A DB           JSR STROUT
9037 20 FB DA           JSR CRDO       ;CR to screen
903A A5 00              LDA ERRORS     ;Any errors-Pass1
903C F0 10              BEQ PASS2      ;No-so Pass2
903E 20 FB DA           JSR CRDO
9041 A9 7C              LDA #< ERRMES   ;Print Not Com-
9043 A0 94              LDY #> ERRMES   ;pressed message
9045 20 3A DB           JSR STROUT
9048 20 03 91           JSR RESTOR     ;Remove $FF tokens
904B 4C D0 03           JMP DOSWS      ;BASIC via DOS
904E 20 22 91  PASS2    JSR SECOND     ;Perform Pass2
9051 4C D0 03           JMP DOSWS      ;BASIC via DOS
9054 C8        SAVLIN   INY            ;Save Line#
9055 B1 B8              LDA (TXTPTR),Y
9057 85 FA              STA LN1
9059 C8                 INY
905A B1 B8              LDA (TXTPTR),Y
905C 85 FB              STA LN1+1
905E A5 B8              LDA TXTPTR
9060 18                 CLC
9061 69 03              ADC #$03       ;Inc TXTPTR to
9063 85 B8              STA TXTPTR     ;first byte in
9065 90 02              BCC SCANLN     ;text of prog ln
9067 E6 B9              INC TXTPTR+1
9069 20 B1 00  SCANLN   JSR CHRGET     ;Search for End
906C C9 00              CMP #ENDLIN    ;of Line Token
906E F0 B4              BEQ NXTLIN     ;Unref and refnd
9070 C9 FF              CMP #REFLIN
9072 F0 B0              BEQ NXTLIN
9074 C9 C4              CMP #THENTK    ;THEN token?
9076 D0 0F              BNE NEXT       ;No-so branch
9078 A0 01              LDY #$01
907A B1 B8              LDA (TXTPTR),Y
907C 38                 SEC
907D E9 30              SBC #$30
907F C9 0A              CMP #$0A
9081 B0 E6              BCS SCANLN
9083 A9 C4              LDA #THENTK    ;Restore THEN
9085 D0 08              BNE STORE      ;token in accum
9087 C9 AB     NEXT     CMP #GOTOTK    ;GOTO token?
9089 F0 04              BEQ STORE      ;Yes-so branch
908B C9 B0              CMP #GOSBTK    ;GOSUB token?
908D D0 DA              BNE SCANLN     ;No-so branch
908F 85 F9     STORE    STA TOKEN      ;Save token
9091 20 B1 00  READLN   JSR CHRGET     ;Inc Ptrto ln#
```

How CMPRSS works

CMPRSS operates in two passes of your Applesoft program. The first pass consists of scanning the program for referenced line numbers which are found in the following Applesoft statement types:

```
GOTO
GOSUB
IF...THEN
ON...GOTO
ON...GOSUB
```

CMPRSS does not check the following commands for referenced line numbers:

LIST RUN DEL

These statements are not commonly used and can be adjusted manually after running CMPRSS if they should occur.

In this first pass, each time a line number is referenced, somehow it must be recorded so that when the Applesoft program is compressed during Pass #2, referenced line numbers will not be removed. Mr. Bauers' COMPRESS uses a Called Line Number Table which severely limits the number of referenced lines you can have in your program, especially as it does not check for duplicates. I have decided to use a method of recording a line number as being referenced which imposes no restriction upon the amount. It involves flagging the referenced lines within the Applesoft program itself. For example, take the following simple program:

```
10 INPUT J
20 IF J=0 THEN 50
30 PRINT J
40 GOTO 10
50 END
```

Each Applesoft program line is represented in memory as follows:
(a) Two bytes in lo-byte, hi-byte order which point to the beginning of the next Applesoft line in memory. This 2-byte address is in hexadecimal.
(b) Two bytes in lo-byte, hi-byte order representing the line number (in hexadecimal) of the Applesoft line.
(c) Following the initial 4 bytes of the line is the 'text' of the Applesoft line itself. All reserved words (commands) are represented in a single byte by a 'token'. For example, INPUT is

represented by the token $84 (adopting the usual convention of preceding a hexadecimal number with $). All tokens can be recognized as bytes with their high bit set (i.e., $80 or greater). Applesoft tokens range from $80 (END) to $EA (MID$). All the rest of the text line (which is not represented by an Applesoft token) is represented character by character by each character's ASCII code (including line numbers in GOTOs etc.). All spaces are eliminated by the Interpreter except those within quoted strings.

(d) The end of the Applesoft line is marked by a $00 byte. The hexadecimal representation of our sample program in memory thus would be as follows, starting at address $800:

```
$8ØØ ØØ Ø8 Ø8 ØA ØØ 84 4A ØØ
$8Ø8 14 Ø8 14 ØØ AD 4A DØ 3Ø
$81Ø 14 35 3Ø ØØ 1B Ø8 1E ØØ
$818 BA 4A ØØ 23 Ø8 28 ØØ AB
$82Ø 31 3Ø ØØ 29 Ø8 32 ØØ 8Ø
$828 ØØ ØØ ØØ
```

The end of the entire Applesoft program is marked by a sequence of three $00 bytes.

Because the end of each Applesoft line is marked by a $00 byte, there is also a $00 byte immediately preceding each following line. Note that there is also a $00 byte preceding the first line which usually begins at $801 in memory.

The method I have devised of flagging a referenced line is to set the $00 byte immediately preceding the referenced line to $FF (note that in a normal Applesoft program no byte is ever set to $FF so therre can be no confusion).

After Pass #1 through the sample program, it will look like this:

```
$8ØØ FF Ø8 Ø8 ØA ØØ 84 4A ØØ
$8Ø8 14 Ø8 14 ØØ AD 4A DØ 3Ø
$81Ø C4 35 3Ø ØØ 1B Ø8 1E ØØ
$818 B4 4A ØØ 23 Ø8 28 ØØ AB
$82Ø 31 3Ø FF 29 Ø8 32 ØØ 8Ø
$828 ØØ ØØ ØØ
```

During Pass #1, while CMPRSS is flagging all referenced lines with $FF tokens, it occurred to me that the routine might as well check that these line numbers actually exist and so I have incorporated Peter Meyer's GOTO/GOSUB checker from the December 1982 edition of MICRO. The

```
9Ø94 2Ø ØC DA            JSR LINGET    ;Read ln# and st
9Ø97 A5 5Ø               LDA LINNUM
9Ø99 A4 51               LDY LINNUM+1
9Ø9B 85 FC               STA LN2       ;Save LINNUM in
9Ø9D 84 FD               STY LN2+1     ;LN2
9Ø9F 2Ø 1A D6            JSR FNDLIN    ;Look for ln#
9ØA2 BØ 35               BCS CHKCOM    ;Found-so branch
9ØA4 E6 ØØ               INC ERRORS    ;Inc err count
9ØA6 2Ø FB DA  NOLINE    JSR CRDO
9ØA9 A5 FB               LDA LN1+1
9ØAB A6 FA               LDX LN1
9ØAD 2Ø 24 ED            JSR LINPRT    ;Print ln# w err
9ØBØ A5 F9               LDA TOKEN
9ØB2 C9 C4               CMP #THENTK   ;THEN token?
9ØB4 DØ Ø7               BNE NEXT1     ;No-so branch
9ØB6 A9 59               LDA #< THEN    ;Print THEN on
9ØB8 AØ 94               LDY #> THEN    ; screen
9ØBA 4C CC 9Ø            JMP PRINT
9ØBD C9 BØ     NEXT1     CMP #GOSBTK   ;GOSUB token?
9ØBF FØ Ø7               BEQ NEXT2     ;Yes-so branch
9ØC1 A9 46               LDA #< GOTO    ;Must have GOTO
9ØC3 AØ 94               LDY #> GOTO    ;so print GOTO
9ØC5 4C CC 9Ø            JMP PRINT     ;on screen
9ØC8 A9 4F     NEXT2     LDA #< GOSUB   ;Print GOSUB
9ØCA AØ 94               LDY #> GOSUB
9ØCC 2Ø 3A DB  PRINT     JSR STROUT    ;Print undefd
9ØCF A5 FD               LDA LN2+1     ;line #
9ØD1 A6 FC               LDX LN2
9ØD3 2Ø 24 ED            JSR LINPRT
9ØD6 4C DE 9Ø            JMP CHK1
9ØD9 A2 FF     CHKCOM    LDX #REFLIN   ;Put $FF in prog
9ØDB 2Ø FØ 9Ø            JSR WRTBYT    ;to flag ref ln
9ØDE 2Ø B7 ØØ  CHK1      JSR CHRGOT    ;Re-get curbyte
9ØE1 C9 2C               CMP #COMMA    ;Comma?
9ØE3 FØ AC               BEQ READLN    ;Yes-so branch
9ØE5 A5 B8               LDA TXTPTR    ;Dec TXTPTR in
9ØE7 DØ Ø2               BNE NEXT3     ;prep for CHRGET
9ØE9 C6 B9               DEC TXTPTR+1
9ØEB C6 B8     NEXT3     DEC TXTPTR
9ØED 4C 69 9Ø            JMP SCANLN
9ØFØ 18        WRTBYT    CLC           ;Put $ØØ or $FF
9ØF1 A5 9B               LDA LSTLIN    ;in byte preceed
9ØF3 69 FF               ADC #$FF      ;a partic Apple
9ØF5 85 9B               STA LSTLIN    ;soft line
9ØF7 A5 9C               LDA LSTLIN+1
9ØF9 69 FF               ADC #$FF
9ØFB 85 9C               STA LSTLIN+1
9ØFD 8A                  TXA           ;X-reg contains
9ØFE AØ ØØ               LDY #$ØØ      ;$ØØ or $FF
91ØØ 91 9B               STA (LSTLIN),Y
91Ø2 6Ø                  RTS
91Ø3 A5 67     RESTOR    LDA TXTTAB    ;All $ØØ to $FF
91Ø5 85 9B               STA LSTLIN    ;Init LSTLIN to
91Ø7 A5 68               LDA TXTTAB+1  ;start of prog
91Ø9 85 9C               STA LSTLIN+1
91ØB A2 ØØ     REST1     LDX #ENDLIN
91ØD 2Ø FØ 9Ø            JSR WRTBYT    ;Put $ØØ before
911Ø AØ Ø1               LDY #$Ø1      ;current line
9112 B1 9B               LDA (LSTLIN),Y  ;Load lo-byte
9114 AA                  TAX           ;of next line ptr trans
9115 C8                  INY           ;to X-Register
9116 B1 9B               LDA (LSTLIN),Y  ;load hi-byte
9118 85 9C               STA LSTLIN+1  ;Update LSTLIN
911A 86 9B               STX LSTLIN
911C 88                  DEY
911D B1 9B               LDA (LSTLIN),Y  ;End of Prog?
911F DØ EA               BNE REST1     ;No-so loop
9121 6Ø                  RTS
9122 2Ø FB DA  SECOND    JSR CRDO      ;Start of PASS2
```

```
9125 A9 8F              LDA #< PASS2A  ;Print PASS2 mes
9127 AØ 94              LDY #> PASS2A
9129 2Ø 3A DB           JSR STROUT
912C 2Ø FB DA           JSR CRDO
912F A9 FF              LDA #REFLIN   ;Init variables
9131 85 Ø4              STA LASTX
9133 A5 67              LDA TXTTAB
9135 85 9B              STA LSTLIN
9137 85 Ø2              STA NEWPTR
9139 85 B8              STA OLDPTR
913B A5 68              LDA TXTTAB+1
913D 85 9C              STA LSTLIN+1
913F 85 Ø3              STA NEWPTR+1
9141 85 B9              STA OLDPTR+1
9143 A5 AF              LDA EPROG
9145 85 Ø8              STA OLDEOP
9147 A5 BØ              LDA EPROG+1
9149 85 Ø9              STA OLDEOP+1
914B A9 ØØ              LDA #ENDLIN
914D 85 ØA              STA LSTEOS
914F 85 Ø1              STA IFFLAG
9151 2Ø Ø5 94           JSR DECOLD
9154 2Ø F7 93           JSR DECNEW
9157 2Ø C5 93           JSR GETLIN    ;Get 1st ln#
915A 2Ø 39 93           JSR NEWLIN    ;Init LINBUF
915D 2Ø 22 94  GETBYT   JSR GETOLD    ;Get next byte
916Ø C9 FF              CMP #REFLIN   ;EOLine Ref
9162 FØ Ø4              BEQ GB1       ;Yes-so branch
9164 C9 ØØ              CMP #ENDLIN   ;EOLine Unref
9166 DØ 17              BNE GB2       ;No-so branch
9168 85 ØA     GB1      STA LSTEOS    ;Recall last End
916A 85 FC              STA TEMP      ;of Statmt Token
916C 2Ø DØ 91           JSR EOL       ;Deal with EOL
916F 9Ø EC     GB1A     BCC GETBYT
9171 2Ø 6E 93           JSR EOP       ;Deal w EOProg
9174 A9 9A              LDA #< PASS2B  ;Print END PASS2
9176 AØ 94              LDY #> PASS2B
9178 2Ø 3A DB           JSR STROUT
917B 2Ø FB DA           JSR CRDO
917E 6Ø                 RTS
917F C9 3A     GB2      CMP #COLON    ;Colon?
9181 DØ Ø6              BNE GB3       ;No-so branch
9183 2Ø Ø2 92           JSR EOS       ;Deal w EOStmt
9186 4C 5D 91           JMP GETBYT    ;Get next byte
9189 C9 AA     GB3      CMP #LETTOK   ;LET token?
918B FØ DØ              BEQ GETBYT    ;Yes - ignore
918D C9 B2              CMP #REMTOK   ;REM token?
918F DØ Ø6              BNE GB4       ;No-so branch
9191 2Ø 1D 92           JSR REMARK    ;Deal with REM
9194 4C 6F 91           JMP GB1A      ;Check EOP
9197 C9 82     GB4      CMP #NXTTOK   ;NEXT token?
9199 DØ Ø6              BNE GB5       ;No-so branch
919B 2Ø 59 92           JSR NEXTX     ;Deal w NEXT
919E 4C 5D 91           JMP GETBYT
91A1 C9 22     GB5      CMP #QUOTE    ;Is it a quote?
91A3 DØ Ø6              BNE GB6       ;No-so branch
91A5 2Ø 85 92           JSR STRING    ;Deal with quote
91A8 4C 5D 91           JMP GETBYT
91AB 2Ø 2A 94  GB6      JSR LETTER    ;Is it a letter?
91AE BØ Ø6              BCS GB7       ;No-so branch
91BØ 2Ø A6 92           JSR VARIBL    ;Yes-must be var
91B3 4C 5D 91           JMP GETBYT
91B6 C9 AD     GB7      CMP #IFTOK    ;IF token?
91B8 DØ Ø8              BNE GB8       ;No-not special
91BA A4 Ø1              LDY IFFLAG    ;If IFFLAG isn't
91BC DØ Ø4              BNE GB8       ;Ø then leave
91BE A4 Ø4              LDY LASTX     ;Remem beg of IF
91CØ 84 Ø1              STY IFFLAG
91C2 2Ø 1B 94  GB8      JSR PUTBUF    ;Byte in LINBUF
91C5 9Ø 96              BCC GETBYT    ;LINBUF not full
```

process of Pass #1 goes something like this:

(a) Locate a GOTO, GOSUB or THEN token.

(b) Call the Applesoft Interpreter routine LINGET to get the 'decimalized' line number and convert it to hexadecimal.

(c) Call the Applesoft Interpreter routine FNDLIN to locate the line number in the Applesoft program.

(d) If it is found, store $FF in the byte immediately preceding the line; otherwise print an error message on the screen and set the error flag.

(e) Repeat until the end of the Applesoft program is reached.

Other Applesoft Interpreter routines used are:

CH-RGET	increment TXTPTR, the text pointer and load the next byte of the Applesoft program into the Accumulator.
CH-RGOT	same as CHRGET but does not increment TXTPTR.
STX-TPT	initialize TXTPTR to the byte immediately preceding the start of the Applesoft program in preparation for scanning through it.
CRDO	output a carriage return to the screen
STR-OUT	prints a text string to the screen (used for messages).
LINPRT	prints a two-byte hexadecimal number as a decimal number to the screen.

By using these routines, I was able to considerably reduce the amount of memory occupied by CMPRSS; it occupies 3 pages of memory less than COMPRESS and, in addition, it also checks for unreferenced line numbers.

If any unreferenced line numbers are encountered during Pass #1, the Applesoft program will not be compressed. CMPRSS cannot just return control to Applesoft however, because the Applesoft program will be sprinkled with $FF tokens. Before returning control to the Interpreter, a routine called RESTOR is executed which replaces all $FF bytes with $00 bytes. Return is then made via the DOS warm start vector at $3D0.

If no unreferenced line numbers are encountered, CMPRSS enters Pass #2 which is the compression phase. Our sample program, after compression will look like this:

```
10 INPUT J : IF J = 0 THEN 50
30 PRINT J : GOTO 10
50 END
```

which in memory will look like:

```
$800 00 10 08 0A 00 84 4A 3A
$808 AD 4A D0 30 C4 35 30 00
$810 1B 08 1E 00 BA 4A 3A AB
$818 31 30 00 21 08 32 00 80
$820 00 00 00
```

All $FFs have been replaced by $00 again. This program has been compressed by 8 bytes or 20% of the original size. Programs containing REMs and long variable names show much more spectacular reductions after compression.

Techniques used by CMPRSS for Compression

(a) Concatenation of statements and removal of line numbers.
As many statements as possible are concatenated onto each line (to a maximum of 255 characters per line). This often results in longer lines than can ever be keyed in manually through the keyboard. Referenced lines cannot be concatenated, so the process stops when an $FF token is encountered. Also, if an IF statement occurs in the Applesoft line, then the next line cannot be concatenated on the end or it will alter the logic flow of the program. E.g.,

```
100 IF A = B THEN A = A + 1
110 B = B + 1
```

cannot be compressed as:

```
100 IF A = B THEN A = A + 1 :
               B = B + 1
```

because in the original program, B = B + 1 is always performed regardless of the values of A and B, whereas in the "compressed" version B = B + 1 is only executed when A = B. This is of paramount importance. Take the following example from Mr. Bauers' article:

```
(1) 10 GOTO 50
    20 J = 5
    50 END
```

```
91C7 20 AE 93           JSR BAKTRK   ;LINBUF full so
91CA 20 39 93           JSR NEWLIN   ;backtrk, start
91CD 4C 5D 91           JMP GETBYT   ;new ln,nxt byte
91D0 C9 FF     EOL      CMP #REFLIN  ;Deal w EOL
91D2 D0 15              BNE EOL2     ;Ref line - No
91D4 A9 00     EOLX     LDA #ENDLIN  ;Yes Replace $FF
91D6 20 1B 94           JSR PUTBUF   ;w $00 in LINBUF
91D9 20 9B 93           JSR TRNBUF   ;Transfer LINBUF
91DC                                  ;to new program
91DC 20 C5 93  EOL0     JSR GETLIN   ;Get nxt Ap ln#
91DF B0 04              BCS EOL1     ;Branch if EOP
91E1 20 39 93           JSR NEWLIN   ;Newln in LINBUF
91E4 18                 CLC          ;Flag not EOP
91E5 20 D5 92  EOL1     JSR RESOLD   ;Rset OLDBEG ptr
91E8 60                 RTS
91E9 A5 01     EOL2     LDA IFFLAG   ;Force EOL?
91EB D0 E7              BNE EOLX     ;Yes-so branch
91ED A9 3A              LDA #COLON   ;Colon-mark EOS
91EF 86 04              STX LASTX    ;Updte LASTX ptr
91F1 20 1B 94           JSR PUTBUF   ;Colon in LINBUF
91F4 90 05              BCC EOL4     ;Not full-branch
91F6 CA        EOL3     DEX
91F7 A9 FF              LDA #REFLIN  ;Force EOL
91F9 D0 D5              BNE EOL      ;Always branch
91FB 20 C5 93  EOL4     JSR GETLIN   ;Get new ln#
91FE B0 F6              BCS EOL3     ;EOP - branch
9200 90 E3              BCC EOL1     ;Not EOP-branch
9202 86 04     EOS      STX LASTX    ;Deal w EOS
9204 20 1B 94           JSR PUTBUF   ;Updte LASTX ptr
9207 90 0C              BCC EOS1     ;LINBUF not full
9209 CA                 DEX
920A A9 00              LDA #ENDLIN  ;terminate ln
920C 20 1B 94           JSR PUTBUF   ;$00 in LINBUF
920F 20 9B 93           JSR TRNBUF
9212 20 39 93           JSR NEWLIN   ;Start a new line
9215                                  ;in LINBUF
9215 20 D5 92  EOS1     JSR RESOLD   ;Reset OLDBEG ptr
9218 A9 00              LDA #ENDLIN
921A 85 0A              STA LSTEOS   ;Set last EOS to
921C 60                 RTS          ;$00
921D 20 22 94  REMARK   JSR GETOLD   ;Deal with REM
9220 C9 FF              CMP #REFLIN  ;1st loop reading
9222 F0 08              BEQ REM1     ;bytes until EOL
9224 C9 00              CMP #ENDLIN  ;($00 or $FF) is
9226 D0 F5              BNE REMARK   ;reached
9228 A0 00              LDY #$00     ;Set Y-reg to re-
922A                                  ;member that
922A F0 02              BEQ REM2     ;$00 was EOL
922C A0 01     REM1     LDY #$01     ;or $FF was EOL
922E 85 FC     REM2     STA TEMP     ;Temp store EOL
9230 E0 04              CPX #$04     ;Is REM on sep ln?
9232 F0 04              BEQ REM3     ;Yes, so branch
9234 CA                 DEX          ;No, so drop REM
9235 4C D0 91           JMP EOL
9238 A5 0A     REM3     LDA LSTEOS   ;Is Rem referencd
923A D0 07              BNE REM4     ;Yes, so branch
923C C0 00              CPY #$00     ;Is nxt ln ref?
923E F0 11              BEQ REM5     ;No, so branch
9240 4C DC 91           JMP EOL0     ;Drop Rem line
9243 C0 00     REM4     CPY #$00     ;Is nxt ln ref?
9245 F0 0A              BEQ REM5     ;No, so branch
9247 A9 B2              LDA #REMTOK  ;Retain REM, put
9249 20 1B 94           JSR PUTBUF   ;token in LINBUF
924C A9 FF              LDA #REFLIN  ;Force EOL
924E 4C D0 91           JMP EOL
9251 A5 0A     REM5     LDA LSTEOS   ;Carry LSTEOS
9253 85 FC              STA TEMP     ;to next line
9255 20 C5 93           JSR GETLIN   ;Get nxt ln #
9258 60                 RTS
```

```
9259 20 1B 94  NEXTX   JSR PUTBUF   ;Deal w NEXT stm
925C 90 07             BCC NEXTA    ;NEXT token in
925E 20 AE 93          JSR BAKTRK   ;LINBUF, branch
9261                   ;if not full else backtrack
9261                   ;to previous EOS
9261 20 39 93          JSR NEWLIN
9264 60                RTS
9265 20 22 94  NEXTA   JSR GETOLD
9268 C9 FF             CMP #REFLIN
926A F0 15             BEQ NEXTB
926C C9 00             CMP #ENDLIN
926E F0 11             BEQ NEXTB
9270 C9 3A             CMP #COLON   ;EOS yet?
9272 F0 0D             BEQ NEXTB    ;Yes, so branch
9274 C9 2C             CMP #COMMA   ;More than one
9276                                ;var in NEXT?
9276 D0 ED             BNE NEXTA    ;No, so branch
9278 A9 3A             LDA #COLON   ;Write a : NEXT
927A 20 1B 94          JSR PUTBUF   ;for each comma
927D A9 82             LDA #NXTTOK  ;Load NEXT into
927F D0 D8             BNE NEXTX    ;Accum, always BR
9281 20 05 94  NEXTB   JSR DECOLD   ;Backstep OLDPTR
9284 60                RTS          ;and return
9285 20 1B 94  STRING  JSR PUTBUF   ;Deal w Quoted Str
9288 90 07             BCC COPYST   ;Put quote in LIN
928A                                ;BUF, BR not full
928A 20 AE 93  SBAK    JSR BAKTRK   ;Full, so backtrak
928D 20 39 93          JSR NEWLIN   ;to end of prev
9290 60                RTS          ;stm, start new
9291                                ;line in LINBUF
9291 20 22 94  COPYST  JSR GETOLD
9294 20 1B 94          JSR PUTBUF
9297 B0 F1             BCS SBAK     ;If LINBUF full BR
9299 CA                DEX
929A A9 22             LDA #QUOTE   ;Is char just
929C DD 00 95          CMP LINBUF,X ;placed a quote
929F F0 03             BEQ CLQUOT   ;Yes,so branch
92A1 E8                INX          ;Restore X-reg
92A2 D0 ED             BNE COPYST   ;Always branch
92A4 E8        CLQUOT  INX          ;Restore X-reg
92A5 60                RTS
92A6 20 1B 94  VARIBL  JSR PUTBUF   ;Truncate var
92A9                   ;name to maximum 2 chars
92A9 90 07             BCC VAR1     ;LINBUF not full BR
92AB 20 AE 93  VBAK    JSR BAKTRK
92AE 20 39 93          JSR NEWLIN
92B1 60                RTS
92B2 20 22 94  VAR1    JSR GETOLD   ;Get next byte
92B5 20 2A 94          JSR LETTER   ;Is it a letter?
92B8 90 05             BCC VAR2     ;Yes, so branch
92BA 20 38 94          JSR NUMBER   ;Is it a number?
92BD B0 12             BCS VAR4     ;No, so branch
92BF 20 1B 94  VAR2    JSR PUTBUF   ;Put 2nd char in
92C2 B0 E7             BCS VBAK     ;LINBUF,if full BR
92C4 20 22 94  VAR3    JSR GETOLD
92C7 20 2A 94          JSR LETTER
92CA 90 F8             BCC VAR3
92CC 20 38 94          JSR NUMBER
92CF 90 F3             BCC VAR3
92D1 20 05 94  VAR4    JSR DECOLD   ;Dec OLDPTR and
92D4 60                RTS          ;return
92D5 A4 01     RESOLD  LDY IFFLAG   ;Reset OLDBEG
92D7 D0 08             BNE RS1      ;ptr except in
92D9 A5 B8             LDA OLDPTR   ;mid of an IF
92DB 85 05             STA OLDBEG
92DD A5 B9             LDA OLDPTR+1
92DF 85 06             STA OLDBEG+1
92E1 60        RS1     RTS
```

His COMPRESS would leave this program as it is, because although line #20 is not referenced, he says that concatenating it onto line #10:

```
(ii) 10 GOTO 50 : J = 5
     50 END
```

would cause the J = 5 statement never to be executed. This is true, but in fact, if you carefully examine the original program, (i), you will see that it will not even be executed in the original! So the program at (ii) is perfectly acceptable because it behaves identically to the original. It is perhaps preferable to (i) because it emphasizes the "dead code". As soon as you see the J = 5 appended to the GOTO statement, you can see that there is something wrong. If there is no "dead code" in your program, then all lines following a terminal statement such as GOTO, RETURN, STOP or END will always be referenced and there is no need for CMPRSS to take any special action.

(b) Removal of REMs

It is important, especially in a large program, to liberally sprinkle the program with meaningful REMarks - it makes the program listing much easier to follow. But REM statements are included in a program for documentation purposes only and serve no useful purpose during execution. In fact, the text of a REM occupies many valuable bytes of memory and often is assigned a line number of its own so that, apart from the text of the REM (one byte per character), an additional four bytes for the line number and link bytes, one byte for the REM token and one byte for the end-of-statement token are wasted. If the REM statement occupies a line of its own, then CMPRSS will remove it entirely if it is not referenced. If it is referenced but the following line is not referenced, the REM line is also removed as shown below:

```
50 GOSUB 1000
...
...
1000 REM THIS IS A SUBROUTINE
1010 A = 10
...
...
1090 RETURN
```

If line #1010 is not referenced, it does not matter whether it has line #1010 or line #1000, so the REM will be completely removed and the unreferenced line, A = 10, will be given the line number of the referenced REM. E.g.,

```
1ØØØ A = 1Ø
```

This does not alter the performance of the program and saves 6 bytes more than Mr. Bauers' COMPRESS which would compress the same statements as:

```
1ØØØ REM
1Ø1Ø A = 1Ø
```

The only time that a REM token has to remain in the program is when it is a referenced REM and the following line is also referenced. E.g.,

```
15 GOSUB 5ØØ
...
...
5ØØ REM THIS IS A REM
51Ø INPUT X,Y
52Ø IF X = Ø OR Y = Ø THEN 51Ø
53Ø RETURN
```

This would compress to:

```
15 GOSUB 5ØØ
...
...
5ØØ REM
51Ø INPUT X,Y : IF X = Ø OR
                Y = Ø THEN 51Ø
53Ø RETURN
```

If the REM is at the end of a multistatement line, it is always removed completely and, if possible, other lines will be concatenated in its place. E.g.,

```
1ØØ X1 = X : REM SAVE X-COORDINATE
11Ø Y1 = Y : REM SAVE Y-COORDINATE
12Ø INPUT X,Y
```

would compress as:

```
1ØØ X1 = X : Y1 = Y : INPUT X,Y
```

a very spectacular compression of the original 68 bytes into 21 bytes! This is 70% compression.

(c) Removal of LETs

Because the two statements, LET A = B and A = B mean exactly the same thing, CMPRSS removes the unnecessary LET token, saving one byte.

```
92E2 2Ø FB DA   SUMARY  JSR CRDO        ;Print result of
92E5 A9 A6              LDA #<MESS1      ;cmpress to scrn
92E7 AØ 94              LDY #>MESS1      ;prnt orig lngth
92E9 2Ø 3A DB           JSR STROUT
92EC 38                 SEC
92ED A5 Ø8              LDA OLDEOP
92EF E5 67              SBC TXTTAB
92F1 AA                 TAX
92F2 A5 Ø9              LDA OLDEOP+1
92F4 E5 68              SBC TXTTAB+1
92F6 2Ø 24 ED           JSR LINPRT
92F9 2Ø 2E 93           JSR PRT1A
92FC A9 CØ              LDA #<MESS2      ;prnt lngth of
92FE AØ 94              LDY #>MESS2      ;compressd prog
93ØØ 2Ø 3A DB           JSR STROUT
93Ø3 38                 SEC
93Ø4 A5 AF              LDA EPROG
93Ø6 E5 67              SBC TXTTAB
93Ø8 AA                 TAX
93Ø9 A5 BØ              LDA EPROG+1
93ØB E5 68              SBC TXTTAB+1
93ØD 2Ø 24 ED           JSR LINPRT
931Ø 2Ø 2E 93           JSR PRT1A
9313 A9 D7              LDA #<MESS3      ;prnt # of bytes
9315 AØ 94              LDY #>MESS3      ;compressed
9317 2Ø 3A DB           JSR STROUT
931A 38                 SEC
931B A5 Ø8              LDA OLDEOP
931D E5 AF              SBC EPROG
931F AA                 TAX
932Ø A5 Ø9              LDA OLDEOP+1
9322 E5 BØ              SBC EPROG+1
9324 2Ø 24 ED           JSR LINPRT
9327 2Ø 2E 93           JSR PRT1A
932A 2Ø FB DA           JSR CRDO
932D 6Ø                 RTS
932E A9 E7      PRT1A   LDA #<MESS1A    ;prnt the word
933Ø AØ 94              LDY #>MESS1A    ;bytes after the
9332 2Ø 3A DB           JSR STROUT      ;above 3 messge
9335 2Ø FB DA           JSR CRDO
9338 6Ø                 RTS
9339 A9 ØØ      NEWLIN  LDA #ENDLIN     ;Start a new ln
933B 85 Ø1              STA IFFLAG      ;in LINBUF, 1st
933D                    ;reset OLDBEG and IFFLAG
933D 2Ø D5 92           JSR RESOLD
934Ø 2Ø 5F 93           JSR WRTLNK      ;Write the link
9343 A5 Ø2              LDA NEWPTR      ;bytes at beg
9345                    ;of last compressed line
9345 85 9B              STA LSTLIN      ;Remember positn
9347 A5 Ø3              LDA NEWPTR+1    ;of strt of new
9349 85 9C              STA LSTLIN+1    ;ln just being
934B                                     ;commenced
934B 2Ø F7 93           JSR DECNEW      ;reset NEWPTR
934E A2 Ø2              LDX #$Ø2        ;Write nxt ln #
935Ø A5 FA              LDA LN1         ;at start of LINBUF
9352 9D ØØ 95           STA LINBUF,X
9355 E8                 INX
9356 86 Ø4              STX LASTX       ;Init LASTX for
9358 A5 FB              LDA LN1+1       ;new line
935A 9D ØØ 95           STA LINBUF,X
935D E8                 INX
935E 6Ø                 RTS
935F AØ ØØ      WRTLNK  LDY #$ØØ        ;Write link bytes
9361 2Ø E9 93           JSR INCNEW      ;at start of last
9364 A5 Ø2              LDA NEWPTR      ;compressed line
9366 91 9B              STA (LSTLIN),Y
9368 C8                 INY
9369 A5 Ø3              LDA NEWPTR+1
936B 91 9B              STA (LSTLIN),Y
```

```
936D 60               RTS
936E 20 5F 93   EOP    JSR WRTLNK     ;Deal with EOP
9371 20 97 D6          JSR STXTPT     ;Put $00 before
9374 A9 00             LDA #$00       ;1st byte of new
9376 A8                TAY            ;prog in case a
9377 91 B8             STA (TXTPTR),Y ;Write two
9379 91 02             STA (NEWPTR),Y ;extra $00
937B 20 E9 93          JSR INCNEW      ;bytes to new
937E 91 02             STA (NEWPTR),Y ;prog (3 in a
9380 20 E9 93          JSR INCNEW     ;row is EOP)
9383 A5 02             LDA NEWPTR
9385 85 AF             STA EPROG      ;Set new EOP ptr
9387 85 69             STA LOMEM      ;Set new LOMEM
9389 85 6B             STA ARS        ;Set new strt of
938B                                  ;array space
938B 85 6D             STA EARS       ;Set new end of
938D A5 03             LDA NEWPTR+1    ;array space
938F 85 B0             STA EPROG+1
9391 85 6A             STA LOMEM1
9393 85 6C             STA ARS+1
9395 85 6E             STA EARS+1
9397 20 E2 92          JSR SUMARY     ;Print results
939A 60                RTS            ;of compression
939B CA         TRNBUF DEX            ;Transfer LINBUF
939C                   ;to New Program Area
939C 86 07             STX MAXX       ;Store max loop
939E A2 00             LDX #$00       ;Reset X-reg for
93A0                                  ;transfer loop
93A0 BD 00 95   LOOP1  LDA LINBUF,X   ;Load next byte
93A3                                    ;from LINBUF
93A3 20 13 94          JSR PUTNEW     ;Trans to new
93A6 E8                INX            ;program area
93A7 E4 07             CPX MAXX       ;Loop complete?
93A9 F0 F5             BEQ LOOP1      ;No, so do again
93AB 90 F3             BCC LOOP1      ;No, so do again
93AD 60                RTS
93AE A6 01      BAKTRK LDX IFFLAG     ;Backtrk to prev
93B0                   ;EOS or start of IF statemt
93B0 D0 02             BNE BK1
93B2 A6 04             LDX LASTX      ;Reset X-reg to
93B4 A9 00      BK1    LDA #ENDLIN    ;prev EOS
93B6 20 1B 94          JSR PUTBUF
93B9 A5 05             LDA OLDBEG
93BB 85 B8             STA OLDPTR
93BD A5 06             LDA OLDBEG+1
93BF 85 B9             STA OLDPTR+1
93C1 20 9B 93          JSR TRNBUF
93C4 60                RTS
93C5 A0 02      GETLIN LDY #$02       ;Get ln # of
93C7 B1 B8             LDA (OLDPTR),Y  ;curr old ln
n3C9 F0 1C             BEQ GET1       ;If hibyte of link-
93CB C8                INY            ;byte pair is zero
93CC                                  ;then this is EOP
93CC B1 B8             LDA (OLDPTR),Y  ;Get lobyte
93CE 85 FA             STA LN1        ;remember it
93D0 C8                INY            ;Update last
93D1                                  ;EOS byte
93D1 B1 B8             LDA (OLDPTR),Y ;Get hibyte
93D3 85 FB             STA LN+1       ;remember it
93D5 A5 FC             LDA TEMP       ;Update last
93D7                                  ;EOS byte
93D7 85 0A             STA LSTEOS     ;Get OLDPTR just
93D9 20 F0 93          JSR INCOLD     ;before 1st byte
93DC 20 F0 93          JSR INCOLD     ;of actual Apple
93DF 20 F0 93          JSR INCOLD     ;soft line
93E2 20 F0 93          JSR INCOLD
93E5 18                CLC            ;Flag not EOP
93E6 60                RTS
93E7 38         GET1   SEC            ;Flag EOP
```

(d) Removal of Variable Names from NEXT Statements
Not only does the removal of the variable name(s) associated with a NEXT token save memory, but it also enables the Applesoft interpreter to execute the FOR..NEXT loop(s) faster, because it obviates the need for it to check that the variable name refers to the currently active FOR. CMPRSS correctly performs this removal even in the instance where more than one FOR..NEXT loop terminates on the same statement:

```
100 NEXT I1, I2
```

CMPRSS will transform this into:

```
100 NEXT : NEXT
```

saving one byte for each character of each variable name removed.

(e) Truncation of Variable Names To a Maximum of 2 Characters
No longer is it necessary for you to name all your variables with meaningless names like A$,C1%, Q2 etc. to save space. You can give your variables longer, more meaningful names like AMOUNT, NAME$ etc. and retain these in the listable 'source' version for ease of understanding what the program is doing. But the Applesoft interpreter only recognizes the first 2 characters of a variable name, so variables AMOUNT and AMT would be identical as far as Applesoft is concerned. It will only recognize the AM. CMPRSS uses this fact to reduce your program as much as possible. AMOUNT becomes AM and NAME$ becomes NA$. The compressed version is hard to read, but you should never list the compressed version. It will certainly operate the same as the original, but much more efficiently. You should always keep two versions of your program, the original, readable version and the compressed one.

Executing CMPRSS

1. Type BRUN CMPRSS (RETURN). This will load CMPRSS at $9000 and reset HIMEM to protect itself. It also installs the '&' vector to enable CMPRSS to be easily run.
2. If your Applesoft program is already in memory, type & (RETURN) and your program will be compressed; otherwise key in or LOAD your Applesoft program from disk and then type & (RETURN). Compression takes a mere 5 seconds or so for the largest program.

It is important to note that you should always SAVE the "uncompressed" version BEFORE you run CMPRSS, or the valuable REMs and meaningful variable names will be lost forever.

If there are no non-existent line numbers, the display on the screen will look something like:

```
*** PASS 1 ***
*** END PASS 1 ***


*** PASS 2 ***


OLD PROGRAM LENGTH:  16224 BYTES
NEW PROGRAM LENGTH:    9528 BYTES
PROGRAM COMPRESSED BY: 6696 BYTES

*** END PASS 2 ***
```

If, however, non-existent line numbers have been encountered during Pass #1, they will be reported and your program will not be compressed. The display, in this case, will look something like this:

```
*** PASS 1 ***
8560 GOSUB4170
9000 GOTO3010
9050 THEN9095
***END PASS 1 ***


*** NOT COMPRESSED ***
```

The line numbers of the offending statements are 8560, 9000 and 9050. The non-existent lines are 4170, 3010 and 9095.

The program resides just below DOS from $9000 to $94FF and the space from $9500 to $95FF is used for the Compressed Line Buffer where the current compressed line is assembled before being written back into the Applesoft program.

Once CMPRSS is installed, your Applesoft programs may be LOADed, changed, SAVEd and CMPRSSed by merely keying & (RETURN). You can even run them and, provided that they never alter HIMEM, POKE any values into memory locations $9000 to $94FF, or alter the & vector, CMPRSS will remain unharmed and may be used again and again. If, however, you need the 1.5K bytes which CMPRSS occupies because you are running a very large program, you can reset HIMEM to just below DOS ($9600) and then, next time CMPRSS is required, you will have to BRUN it from disk again. MICRO

```
93E8 60                 RTS
93E9 E6 02     INCNEW   INC NEWPTR     ;Incr NEWPTR
93EB D0 02              BNE IN1
93ED E6 03              INC NEWPTR+1
93EF 60        IN1      RTS
93F0 E6 B8     INCOLD   INC OLDPTR     ;Incr OLDPTR
93F2 D0 02              BNE IN2
93F4 E6 B9              INC OLDPTR+1
93F6 60        IN2      RTS
93F7 18        DECNEW   CLC            ;Decr NEWPTR
93F8 A5 02              LDA NEWPTR
93FA 69 FF              ADC #$FF
93FC 85 02              STA NEWPTR
93FE A5 03              LDA NEWPTR+1
9400 85 03              ADC #$FF
9402 69 FF              STA NEWPTR+1
9404 60                 RTS
9405 18        DECOLD   CLC            ;Decr OLDPTR
9406 A5 B8              LDA OLDPTR
9408 69 FF              ADC #$FF
940A 85 B8              STA OLDPTR
940C A5 B9              LDA OLDPTR+1
940E 69 FF              ADC #$FF
9410 85 B9              STA OLDPTR+1
9412 60                 RTS
9413 20 E9 93  PUTNEW   JSR INCNEW     ;Store Accum in
9416 A0 00              LDY #$00       ;new prog area
9418 91 02              STA (NEWPTR),Y
941A 60                 RTS
941B 9D 00 95  PUTBUF   STA LINBUF,X   ;Put Accum into
941E E8                 INX            ;LINBUF
941F E0 FD              CPX #$FD       ;Set if LINBUF
9421 60                 RTS            ;is full
9422 20 F0 93  GETOLD   JSR INCOLD     ;Get a byte from
9425                                    ;the old prog
9425 A0 00     GOTOLD   LDY #$00
9427 B1 B8              LDA (OLDPTR),Y
9429 60                 RTS
942A C9 41     LETTER   CMP #LETTRA    ;Is byte a lettr
942C 90 06              BCC NOLETR     ;If < 'A' then
942E C9 5A              CMP #LETTRZ    ;not a letter
9430 90 04              BCC ISLETR     ;If < 'Z',is ltr
9432 F0 02              BEQ ISLETR     ;If = 'Z',is ltr
9434 38        NOLETR   SEC            ;Set carry,not a letter
9435 60                 RTS
9436 18        ISLETR   CLC            ;Clear carry, is letter
9437 60                 RTS
9438 C9 30     NUMBER   CMP #ZERO      ;Is byte number?
943A 90 06              BCC NONUMB     ;If < '0',not #
943C C9 39              CMP #NINE
943E F0 04              BEQ ISNUMB     ;If = '9',is #
9440 90 02              BCC ISNUMB     ;If < '9',is #
9442 38        NONUMB   SEC            ;Set carry,not a number
9443 60                 RTS
9444 18        ISNUMB   CLC            ;Clear carry, is number
9445 60                 RTS
9446 20 20 20  GOTO     ASC '   GOTO  '
944F 20 20 20  GOSUB    ASC '   GOSUB '
9458 20 20 20  THEN     ASC '   THEN '
9460 2A 2A 2A  PASS1A   ASC '*** PASS 1 '
946B 2A 2A 2A  PASS1B   ASC '*** END PASS1 '
9479 2A 2A 2A  ERRMES   ASC '*** NOT COMPRESSED '
948C 2A 2A 2A  PASS2A   ASC '*** PASS 2 '
9497 2A 2A 2A  PASS2B   ASC '*** END PASS 2 '
94A6 4F 4C 44  MESS1    ASC 'OLD PROGRAM LENGTH:   '
94BC 4E 45 57  MESS2    ASC 'NEW PROGRAM LENGTH:  '
94D1 50 52 4F  MESS3    ASC 'PROGRAM COMPRESSED BY '
94E7 20 20 42  MESS1A   ASC '  BYTES '
94EF                    END
```

feature

Save time and mathematical aggrevation with a compilation of defined functions in a very friendly program

USEFUL FUNCTIONS Part 2

by Paul Garrison

EDITOR'S NOTE

In last month's issue we printed a program that allowed you to easily access various defined functions. This saved time and aggravation when working with complicated mathematical formulas. As a continuation of this approach, we present the second of three programs which will put a host of valuable formulas and functions at your fingertips. Again we invite you to send in any defined functions you may be using that are not mentioned. The submissions we receive will be collected and published in a future issue.

PROGRAM #2

This program includes the formulas for trigonometric ratios, two formulas dealing with matters related to aviation (the effect of wind on ground speed and density altitude), the formulas for converting temperatures from Fahrenheit to Celsius and vice versa, plus the formulas that comprise Ohm's Law and determine the resistance factor of electrical wires, and finally the formula that determines future values based on compound interest, present value and the time span to be examined. The structure of the program is identical to the one described above.

```
1 REM FUNCTIONS (DELETE THOSE NOT USED IN A PROGRAM)
2 PI=3.14159
3 RAD=57.2958
47 DEF FNHYP(X,Y)=SQR(X↑2+Y↑2):                         REM FIND
HYPOTENUSE
48 DEF FNHX(H,Y)=SQR(H↑2-Y↑2):                          REM FIND SIDE
X,HORIZONAL
49 DEF FNVY(H,X)=SQR(H↑2-X↑2):                          REM FIND SIDE
Y,VERTICAL
5Ø DEF FNANGL(A)=9Ø-A:                                  REM FIND ANGLE A OR B
51 DEF FNX(H,A)=H*COS(A*(PI/18Ø)):                      REM FIND SIDE X
BY H
& A
52 DEF FNY(H,A)=H*SIN(A*(PI/18Ø)):                      REM FIND SIDE Y
BY H
& A
53 DEF FNB(X,Y)=(ATN(X/Y))*(18Ø/PI):                    REM FIND A OR B BY
X
& Y
6Ø DEF FNWC(WV,WD,MC,MV)=-1*WV*COS((WD-MC-MV)/RAD):     REM WIND
COMPONENT,AI CRAFT
61 DEF FNDENALT(PA,F)=(145426*(1-(((288.15-
PA*.ØØ1981)/288.15)↑5.2563/((273.15+F)/288.15))↑.235))
```

```
62                                                       REM DENSITY ALTITUDE
63 DEF FNFC(F)=(F-32)/1.8:                               REM DEG.F. TO DEG.C.
64 DEF FNCF(C)=(C*1.8)+32:                               REM DEG.C. TO DEG.F.
65 DEF FNVA(V,A)=V/A:                                    REM OHM=VOLT/AMPERE
66 DEF FNVO(V,O)=V/O:                                    REM AMP=VOLT/OHM
67 DEF FNAO(A,O)=A*O:                                    REM VOLT=AMP*OHM
68 DEF FNWR(M,L)=1Ø.4*L/M:                               REM WIRE RESISTENCE
69 DEF FNCP(PV,I,CP)=PV*(1+(I/1ØØ))↑CP:                    REM COMPOUND INTEREST1ØØ REM (PRO-
GRAM TITLE, AUTHOR)
11Ø REM (TYPE OF BASIC USED)
12Ø GOTO 2ØØ
13Ø ?"-------------------------------------------------------":RETURN
14Ø HOME:VTAB(1Ø):RETURN
15Ø ?:INPUT "Press >RETURN<  (Q to quit)   ",R$
155 IF R$="Q" THEN 16Ø ELSE RETURN
16Ø GOSUB 14Ø:GOSUB 13Ø:?TAB(33)"End.":GOSUB 13Ø:END
19Ø                                                      REM TESTING FUNCTIONS
2ØØ GOSUB 14Ø:?"Menu:":GOSUB 13Ø:?"Aviation functions:":GOSUB 13Ø
21Ø ?1,"Wind component"
22Ø ?2,"Density altitude"
222 ?3,"Convert degrees F. to degrees C."
224 ?4,"Convert degrees C. to degrees F.":GOSUB 13Ø
23Ø ?"Ratios for right triangles:":GOSUB 13Ø
24Ø ?5,"Find hypotenuse"
25Ø ?6,"Find horizontal side (X)"
26Ø ?7,"Find vertical side (Y)"
27Ø ?8,"Find angles A and B"
28Ø ?9,"Find two sides (X & Y) by hypotenuse & angle"
29Ø ?1Ø,"Find angles A and B by X and Y":GOSUB 13Ø
291 ?"Electrical:":GOSUB 13Ø
292 ?11,"Find ohms"
293 ?12,"Find amperes"
294 ?13,"Find volts"
295 ?14,"Find wire resistence":GOSUB 13Ø
296 ?15,"Compound interest":GOSUB 13Ø
3ØØ ?16,"Exit program":GOSUB 13Ø
31Ø INPUT "Which?         ",WHICH:GOSUB 14Ø
32Ø ON WHICH GOTO 4ØØ,5ØØ,6ØØ,7ØØ,2ØØØ,2Ø5Ø,21ØØ,215Ø,219Ø,228Ø,24ØØ,25ØØ,26ØØ,27ØØ,28ØØ,16Ø
4ØØ ?"Find wind component (effect on aircraft in flight)":GOSUB 13Ø
41Ø INPUT "Wind direction?                        ",WD
42Ø INPUT "Wind velocity? (knots)                 ",WV
43Ø INPUT "Magnetic course?                       ",MC
44Ø INPUT "Magnetic variation?  (E= - / W= +)     ",MV
45Ø X=FNWC(WV,WD,MC,MV):GOSUB 13Ø
46Ø ?"The wind component factor is          ";X:GOSUB 15Ø:GOTO 2ØØ
5ØØ ?"Find the density altitude":GOSUB 13Ø
51Ø INPUT "Pressure altitude?                     ",PA
52Ø INPUT "Temperature? (degrees centigrade)      ",F
53Ø X=FNDENALT(PA,F):GOSUB 13Ø
54Ø ?"The density altitude is ";X;" feet.":GOSUB 15Ø:GOTO 2ØØ
6ØØ ?"Convert degrees F. to degrees C.":GOSUB 13Ø
61Ø INPUT "Degrees F.?                            ",F
62Ø X=FNFC(F):GOSUB 13Ø
63Ø ?F;" degrees F. equal ";X;" degrees C":GOSUB 15Ø:GOTO 2ØØ
7ØØ ?"Convert degrees C. to degrees F.":GOSUB 13Ø
71Ø INPUT "Degrees C.?                            ",C
72Ø X=FNCF(C):GOSUB 13Ø
73Ø ?C;" degrees C. equal ";X;" degrees F.":GOSUB 15Ø:GOTO 2ØØ
2ØØØ ?"Find the length of the hypotenuse of a right triangle":GOSUB 13Ø
2Ø1Ø INPUT "Enter the horizontal length (X)       ",X
```

```
2020 INPUT "Enter the vertical length   (Y)       ",Y
2030 X=FNHYP(X,Y):GOSUB 130
2040 ?"The length of the hypotenuse is       ";X:GOSUB 150:GOTO 200
2050 ?"Find the length of the horizontal side (X) of a right triangle":GOSUB 130
2060 INPUT "Enter the vertical length   (Y)       ",Y
2070 INPUT "Enter the diagonal length (hypotenuse)",H
2080 X=FNHX(H,Y):GOSUB 130
2090 ?"The horizontal length is              ";X:GOSUB 150:GOTO 200
2100 ?"Find the length of the vertical side (Y) of a right triangle":GOSUB 130
2110 INPUT "Enter the horizontal length (X)       ",X
2120 INPUT "Enter the diagonal length (hypotenuse)",H
2130 XX=FNVY(H,X):GOSUB 130
2140 ?"The vertical length is                ";XX:GOSUB 150:GOTO 200
2150 ?"Find the angle opposite side X or Y in a right triangle":GOSUB 130
2160 INPUT "Enter degrees of one angle            ",A
2170 X=FNANGL(A):GOSUB 130
2180 ?"The other angle is ";X;" degrees":GOSUB 150:GOTO 200
2190 ?"Find the two other sides by hypotenuse and the angle"
2195 ?"between the hypotenuse and the horizontal side":GOSUB 130
2200 INPUT "Enter length of hypotenuse            ",H
2210 INPUT "Enter the degrees of the angle        ",A
2220 X=FNX(H,A):GOSUB 130
2225 XX=FNY(H,A)
2230 ?"The horizontal length is               ";X
2275 ?"The vertical side is                   ";XX:GOSUB 150:GOTO 200
2280 ?"Find the degrees of two angles by sides X and Y":GOSUB 130
2290 INPUT "Enter horizontal side (X)             ",X
2300 INPUT "Enter vertical side   (Y)             ",Y
2310 XX=FNB(X,Y):GOSUB 130
2320 ?"Angle A (opposite X) is ";XX;" degrees":BB=90-XX
2330 ?"Angle B (opposite Y) is ";BB;" degrees":GOSUB 150:GOTO 200
2400 ?"Find ohms by volts and amperes":GOSUB 130
2410 INPUT "Volts?                                ",V
2420 INPUT "Amperes?                              ",A
2430 X=FNVA(V,A):GOSUB 130
2440 ?X;" ohms":GOSUB 150:GOTO 200
2500 ?"Find amperes by volts and ohms":GOSUB 130
2510 INPUT "Volts?                                ",V
2520 INPUT "Ohms?                                 ",O
2530 X=FNVO(V,O):GOSUB 130
2540 ?X;" amperes":GOSUB 150:GOTO 200
2600 ?"Find volts by amperes and ohms":GOSUB 130
2610 INPUT "Amperes?                              ",A
2620 INPUT "Ohms?                                 ",O
2630 X=FNAO(O,A):GOSUB 130
2640 ?X;" volts":GOSUB 150:GOTO 200
2700 ?"Find wire resistence by length and mils":GOSUB 130
2710 INPUT "Length of wire (inches)               ",L
2720 INPUT "Diameter of wire (mils)               ",M
2730 X=FNWR(M,L):GOSUB 130
2740 ?"Resistence is ";X;" ohms":GOSUB 150:GOTO 200
2800 ?"Find future value based on interest and compounding periods":GOSUB 130
2810 INPUT "Present value?                              $",PV
2820 INPUT "Annual interest rate?                       %",I
2830 INPUT "Compounding periods (day/month/year)(D/M/Y) ",CP$
2840 IF CP$="D" THEN I=I/365.25
2850 IF CP$="M" THEN I=I/12
2860 INPUT "Period of how many years?                   ",CP
2870 IF CP$="D" THEN CP=CP*365.25
2875 IF CP$="M" THEN CP=CP*12
2880 X=FNCP(PV,I,CP):GOSUB 130
2890 ?"The future value is $";X:GOSUB 150:GOTO 200
```

MICRO

Commodore to Apple Cassette File Loader

by Art Matheny

Your Apple can read cassette files written by a Commodore VIC-20 or C64 computer with this assembly language program . The file is written into a sequential text file on the Apple's disk. Three types of files are discussed--data files, BASIC programs, and memory ranges.

Requires: Apple II with disk drive and optional printer, Commodore VIC-20 or C64 with C2N cassette drive.

I have a Commodore VIC-20 and a C64 as well as my trusty old Apple II. Of course I have a disk drive for the Apple, but for mass storage with the Commodores I use a C2N cassette tape drive ("Datassette") which works amazingly well. This article shows how the Apple can read cassette files written by either Commodore computer. The method described here can be used to transfer various kinds of data. For example, since I do not presently have an interface to connect my printer to my Commodores, I am using this utility to move BASIC programs to my Apple, where I can make hardcopy listings. It also saves a lot of retyping when I want to convert a Commodore BASIC program to Applesoft. Sorry, though, this program only goes one way. I have not yet taught the Apple to write cassette files that Commodore computers can read, but, with the information given here, I think such a program would not be very difficult to do.

The assembler listing of the main program is shown in Listing 1.

Listing 1

```
                ;COMMODORE-TO-APPLE CASSETTE FILE LOADER
                ;
                ;BY ART MATHENY
                ;
                ; Copyright © 1984
                ; The Computerist, Inc.
                ; Chelmsford, MA Ø1824
                ;
                ;RUNS ON APPLE II.
                ;LOADS A TEXT FILE FROM A
                ;CASSETTE TAPE WRITTEN BY A
                ;COMMODORE COMPUTER, AND SAVES
                ;IT AS AN APPLE DISK FILE.
                ;
                ;CONSTANTS
                ;
ØØØ6             SLOT     EQU 6     ;SLOT # FOR SAVING FILE
ØØØ1             DRIVE    EQU 1     ;DRIVE # FOR SAVING FILE
ØØCØ             BLOKLEN  EQU 192   ;# OF CHARS IN A BLOCK
ØØ1E             NAMLEN   EQU 3Ø    ;# OF CHARS IN FILE NAME
                ;
                ;PAGE Ø VARIABLES
                ;
ØØØ6             BYTE     EQU 6     ;BYTE NOW BEING READ
ØØØ7             TEMP     EQU 7     ;ZPAGE TEMP STORAGE
ØØØ8             PTR      EQU 8     ;POINTER INTO DATA BUFFER
ØØØA             ADR      EQU $A    ;ADDR OF MESSAGE TO PRINT
ØØØC             FMPL     EQU $C    ;FILE MGR PARMLIST POINTER
                ;
                ;PAGE 3 VARIABLES
                ;
Ø3ØØ             CHSUM    EQU $3ØØ  ;CHECK SUM BYTE
Ø3Ø1             PAR      EQU $3Ø1  ;PARITY
Ø3Ø2             KNT      EQU $3Ø2  ;BIT COUNTER
Ø3Ø3             SCAN     EQU $3Ø3  ;FLAG: DOING SECOND SCAN
Ø3Ø4             KDOWN    EQU $3Ø4  ;COUNT-DOWN COUNTER
Ø3Ø5             START    EQU $3Ø5  ;ADDR WHERE BLOCK STARTS
Ø3Ø7             FIN      EQU $3Ø7  ;ADDR WHERE BLOCK ENDS
                ;
                ;DOS SYSTEM CALLS
                ;
Ø3DC             LOCFPL   EQU $3DC  ;LOCATE PARMLIST ADDR
Ø3D6             DOSFM    EQU $3D6  ;DOS FILE MANAGER
                ;
                ;OTHER ADDRESSES
                ;
Ø8Ø1             LOMEM    EQU $8Ø1  ;START OF USABLE MEMORY
Ø8Ø6             NAME     EQU LOMEM+5   ;FILENAME LOCATION
Ø8C2             BODY     EQU LOMEM+BLOKLEN+1 ;START OF FILE
CØ6Ø             TAPEIN   EQU $CØ6Ø ;CASSETTE INPUT PORT
                ;
                ;ROM ROUTINES
                ;
```

Listing 1 *(continued)*

```
FC58                HOME     EQU $FC58 ;CLEAR TEXT SCREEN
FDDA                PRBYTE   EQU $FDDA ;PRINT A HEX BYTE
FDFØ                COUT1    EQU $FDFØ ;OUTPUT TO SCREEN
                   ;
                   ;-------------------------------
9ØØØ                         ORG $9ØØØ
                   ;-------------------------------
                   ;
                   ;SET IRQ MASK TO PREVENT INTERRUPTS
                   ;
9ØØØ 78             PROG     SEI
                   ;
                   ;PRINT HEADING
                   ;
9ØØ1 2Ø 58 FC                JSR HOME
9ØØ4 A9 A3                   LDA #MESG5
9ØØ6 85 ØA                   STA ADR
9ØØ8 A9 91                   LDA /MESG5
9ØØA 85 ØB                   STA ADR+1
9ØØC 2Ø 62 93                JSR PRMESG
                   ;
                   ;PUT 1ST BLOCK AT BEGINNING OF THE BUFFER
                   ;
9ØØF A9 Ø1                   LDA #LOMEM     ;FIN = LOMEM
9Ø11 8D Ø7 Ø3                STA FIN
9Ø14 A9 Ø8                   LDA /LOMEM
9Ø16 8D Ø8 Ø3                STA FIN+1
                   ;
                   ;-------------------------------
                   ;SET UP POINTERS FOR NEXT BLOCK
                   ;-------------------------------
                   ;
9Ø19 AD Ø7 Ø3       LOOP     LDA FIN        ;START = OLD FIN
9Ø1C 8D Ø5 Ø3                STA START      ;START —>
9Ø1F 18                      CLC            ; START OF BLOCK
9Ø2Ø 69 CØ                   ADC #BLOKLEN   ;FIN = START+BLOKLEN
9Ø22 8D Ø7 Ø3                STA FIN        ;FIN —>
9Ø25 AD Ø8 Ø3                LDA FIN+1      ; END OF BLOCK + 1
9Ø28 8D Ø6 Ø3                STA START+1
9Ø2B 69 ØØ                   ADC #Ø
9Ø2D 8D Ø8 Ø3                STA FIN+1
9Ø3Ø C9 9Ø                   CMP /PROG      ;BUFFER FULL
9Ø32 BØ 36                   BCS ERR9       ;IF SO, QUIT READING
                   ;
                   ;-------------------------------
                   ;READ A BLOCK
                   ;-------------------------------
                   ;
9Ø34 A9 ØØ                   LDA #Ø         ;SCAN=Ø:
9Ø36 8D Ø3 Ø3                STA SCAN       ; LOAD THE BLOCK
9Ø39 2Ø 19 92                JSR BLOCK
9Ø3C A9 Ø1                   LDA #1         ;SCAN=1:
9Ø3E 8D Ø3 Ø3                STA SCAN       ; VERIFY THE BLOCK
9Ø41 2Ø 19 92                JSR BLOCK
9Ø44 A9 2E                   LDA #'.'       ;PRINT A PERIOD
9Ø46 2Ø FØ FD                JSR COUT1
                   ;
                   ;-------------------------------
                   ;CHECK FOR END OF FILE
                   ;-------------------------------
                   ;
9Ø49 AD Ø5 Ø3                LDA START      ;PTR = START
9Ø4C 85 Ø8                   STA PTR
9Ø4E AD Ø6 Ø3                LDA START+1
9Ø51 85 Ø9                   STA PTR+1
9Ø53 AØ ØØ                   LDY #Ø         ;LOOK AT 1ST CHAR
9Ø55 B1 Ø8                   LDA (PTR),Y    ; OF BLOCK
9Ø57 C9 Ø5                   CMP #5         ;EOF MARKER
9Ø59 FØ 31                   BEQ EOFMARK    ;BRANCH IF SO
9Ø5B C9 Ø2                   CMP #2         ;DATA BLOCK
9Ø5D DØ BA                   BNE LOOP       ;BRANCH IF NOT
```

Apple has less than 48K of memory, move the origin down to fit the program below DOS, but start it at the beginning of a memory page. Moving the origin will change the machine code for every JSR and JMP.

There are three types of files which I would like to transfer--data files, BASIC programs, and memory ranges. It will be sufficient, though, to transfer data files because, as will be shown later, BASIC programs and memory dumps can both be converted into data files prior to the transfer.

Transfer of Data Files

With a Commodore computer, any kind of data can be written into a tape file. To see how this is done, let's work through a simple example. First put a scratch cassette in the C2N tape drive and either rewind it to the beginning or record the tape counter value. A filename must be selected, say "ANYFILE". A logical file number between 1 and 127 must also be selected. In the following example, the logical file number is 5:

```
OPEN 5,1,2,"ANYFILE"
```

The device number is 1, which denotes the cassette drive. The 2 indicates an intention to write to the file and to put an end-of-file marker at the end. Once the file has been opened, data can be written to it with PRINT # statements such as the following:

```
PRINT #5,"ANY CHARACTER STRING";CHR$(13)
FOR K=1 TO 1Ø : PRINT #5,K;CHR$(13)
: NEXT
```

Since more than one file can be open at once (i.e. on other devices), the logical file number, 5 in this example, must be specified. When the program is finished writing, it should close the file:

```
CLOSE 5
```

The logical file number used here indicates which file is to be closed. The data file on the tape is now ready for transfer.

Rewind the tape to the beginning of the file and move the tape to a tape player connected to the Apple's cassette input. Now BRUN the cassette file loader. Figure 1 shows the Apple's TV display after a successful load operation. The program prints a period for every "block" that it reads successfully. That lets you know that it is still working, which is a comfort when long files are being loaded.

```
TEXT FILE READER

ROLL TAPE
..END OF FILE
SAVING:
ANYFILE

DONE
```

Figure 1. Typical video display of CTACFL.

If anything goes wrong, the program prints an error message and executes a "break" instruction, thus leaving you in the monitor. To try again, rewind the tape and enter:

9000G

The most likely cause of any error is a misreckoning of the loudness control of the tape player. This is a very touchy setting, and it may take several trials to find the right spot. My advice is to start very loud and to work down in small increments. Other causes of error are less likely. It is possible that there may actually be bad data on the tape, in which case you have to go back to the Commodore and save the file again. Test the Commodore C2N tape drive by saving and then verifying any BASIC program. Maybe the tape medium is bad; try a different tape. If all else fails, try a different tape player, preferably one that is not so noisy.

Listing the File

The cassette file loader puts the data into a sequential text file on the disk. The program in Listing 2, called TEXTLISTER, can list this or any other sequential file. The output can be directed either to the TV or to a printer. RUN this program and give the name of the data file. Compare the output with what the original Commodore program wrote. Such data files can be used as input for Apple programs. See the chapter on sequential files in *The DOS Manual*.

TEXTLISTER replaces any unprintable characters by an "@" sign to show at least that there is a character present.

BASIC Programs

Although there are similarities in syntax between Commodore BASIC

Listing 1 *(continued)*

```
                   ;
                   ;SEARCH THE BLOCK FOR FILE TERMINATION BYTE
                   ;
905F A0 BF                   LDY #BLOKLEN-1
9061 B1 08          F1       LDA (PTR),Y
9063 F0 2D                   BEQ HOMERUN   ;FILE TERMINATION
9065 88                      DEY           ; BYTE = 0
9066 D0 F9                   BNE F1
9068 F0 AF                   BEQ LOOP      ;BRANCH ALWAYS
                   ;
                   ;*****<     END OF FILE     >*****
                   ;*****<  SAVE THE DATA ON DISK >*****
                   ;
                   ;------------------------------
                   ;PRINT "BUFFER FULL"
                   ;------------------------------
                   ;
906A A9 78          ERR9     LDA #MESG9
906C 85 0A                   STA ADR
906E A9 90                   LDA /MESG9
9070 85 0B                   STA ADR+1
9072 20 62 93                JSR PRMESG
9075 4C 9D 90                JMP FNAME
9078 C2 D5 C6       MESG9    ASC "BUFFER FULL"
9083 8D                      BYT $8D       ;<RETURN>
9084 D3 C1 D6                ASC "SAVING:"
908B 00                      BYT 0
                   ;
                   ;------------------------------
                   ;HIT EOF MARKER BLOCK
                   ;------------------------------
                   ;
908C A9 00          EOFMARK  LDA #0        ;INSERT ZERO
908E A0 01                   LDY #1        ; INTO DATA
9090 91 08                   STA (PTR),Y
                   ;
                   ;------------------------------
                   ;PRINT"END OF FILE"
                   ;------------------------------
                   ;
9092 A9 CC          HOMERUN  LDA #MESG6
9094 85 0A                   STA ADR
9096 A9 91                   LDA /MESG6
9098 85 0B                   STA ADR+1
909A 20 62 93                JSR PRMESG
                   ;
                   ;------------------------------
                   ;FIND FILE NAME
                   ;------------------------------
                   ;
909D A9 06          FNAME    LDA #NAME     ;ADR = NAME
909F 85 0A                   STA ADR       ;ADR -->
90A1 A9 08                   LDA /NAME     ; HEADER FILE NAME
90A3 85 0B                   STA ADR+1
                   ;
                   ;IS A FILENAME PRESENT
                   ;
90A5 A0 1D                   LDY #NAMLEN-1
90A7 B1 0A          FNAME1   LDA (ADR),Y
90A9 C9 20                   CMP #$20      ;SPACE
90AB D0 0B                   BNE FNAME2
90AD 88                      DEY
90AE 10 F7                   BPL FNAME1
                   ;
                   ;IF NOT, USE DEFAULT NAME
                   ;
90B0 A9 FB                   LDA #DFALT    ;ADR = DFALT
90B2 85 0A                   STA ADR
90B4 A9 91                   LDA /DFALT
90B6 85 0B                   STA ADR+1
```

Listing 1 *(continued)*

```
                    ;
                    ;PRINT THE FILENAME
                    ;
90B8 A0 00           FNAME2   LDY #0
90BA A2 1E                    LDX #NAMLEN
90BC B1 0A           FNAME3   LDA (ADR),Y
90BE 09 80                    ORA #$80        ;SET BIT 7
90C0 91 0A                    STA (ADR),Y
90C2 20 F0 FD                 JSR COUT1
90C5 C8                       INY
90C6 CA                       DEX
90C7 D0 F3                    BNE FNAME3
                    ;
                    ;-----------------------------
                    ;LOCATE PARMLIST
                    ;-----------------------------
                    ;
90C9 20 DC 03                 JSR LOCFPL
90CC 84 0C                    STY FMPL        ;FMPL-->
90CE 85 0D                    STA FMPL+1      ; FILE MGR PARMLIST
                    ;
                    ;-----------------------------
                    ;PUT FILE NAME IN PARMLIST
                    ;-----------------------------
                    ;
90D0 A5 0A                    LDA ADR
90D2 A0 08                    LDY #8
90D4 91 0C                    STA (FMPL),Y
90D6 A5 0B                    LDA ADR+1
90D8 C8                       INY
90D9 91 0C                    STA (FMPL),Y
                    ;
                    ;-----------------------------
                    ;OPEN THE OUTPUT FILE
                    ;-----------------------------
                    ;
90DB A9 01                    LDA #1          ;CALL TYPE 1 = OPEN
90DD A0 00                    LDY #0
90DF 91 0C                    STA (FMPL),Y
90E1 A9 00                    LDA #0
90E3 A0 02                    LDY #2
90E5 91 0C                    STA (FMPL),Y
90E7 C8                       INY
90E8 91 0C                    STA (FMPL),Y
90EA C8                       INY
90EB 91 0C                    STA (FMPL),Y
90ED A0 07                    LDY #7
90EF 91 0C                    STA (FMPL),Y    ;TEXT FILE
90F1 A9 01                    LDA #DRIVE
90F3 A0 05                    LDY #5
90F5 91 0C                    STA (FMPL),Y
90F7 A9 06                    LDA #SLOT
90F9 C8                       INY
90FA 91 0C                    STA (FMPL),Y
                    ;
                    ;PUT BUFFER ADDRESSES IN PARMLIST
                    ;
90FC A9 76                    LDA #WORKAREA
90FE A0 0C                    LDY #$C
9100 91 0C                    STA (FMPL),Y
9102 A9 93                    LDA /WORKAREA
9104 C8                       INY
9105 91 0C                    STA (FMPL),Y
9107 A9 A3                    LDA #SECTOR
9109 C8                       INY
910A 91 0C                    STA (FMPL),Y
910C A9 93                    LDA /SECTOR
910E C8                       INY
```

and Applesoft BASIC, most programs written for a Commodore computer will require extensive revisions before they will run on an Apple. The cassette file loader could save a lot of retyping, though, by moving programs verbatim from the Commodore to the Apple. First, the BASIC program must be converted to a data file so that it can be transferred. The procedure is straightforward:

1. LOAD the program into the Commodore in the usual way.

2. Remove the program tape and put in a "scratch" tape.

3. Enter the following commands in immediate execution mode:

```
OPEN 1,1,2,"FILENAME.TXT"
CMD 1
LIST
PRINT #1
CLOSE 1
```

This writes the program listing into a data file on the tape. It does not make a copy of the original BASIC file, but rather a replica of the program *listing* just as it would appear on the TV. Do not panic if the LIST step above takes 3 times as long as you would expect.

4. Rewind the scratch tape and physically move it to the Apple's cassette tape player.

5. BRUN the cassette file loader and play the file through.

6. You now have a text file on the disk called "FILENAME.TXT". TEXTLISTER can be used to list it. It can be edited with any text editor that can work with "T" type files. In this step it is only necessary to fix the syntax so that it looks like an Applesoft program. Delete the extraneous lines at the beginning and end of the file. Change every "SYS" to "CALL". Make any other changes needed to make it conform to legal Applesoft syntax. It is not essential for the program to be *logically* correct at this point. Save the edited file.

7. Go into Applesoft, give a NEW command if necessary and then (here comes the exciting part) EXEC the text file. This step enters the text file just as if you were typing the whole thing.

8. The program is now in memory, and you can LIST it. Give it a name and save it. As a convention, I use the same filename without the ".TXT" suffix. Note that this program now shows up as an "A" type file in the catalog.

9. This program can be worked just like any other Applesoft program, so do whatever it takes to get it running on the Apple.

Memory Dumps & Dissassembly

It is also possible to transfer a range of memory from a Commodore to an Apple. Again, the trick is to first generate a data file. The program in Listing 3 is a Commodore BASIC program which does this. The user is asked to specify the starting and ending addresses of the memory range as well as a file name for the tape file. It then PEEKs each byte of the range and writes that value (as decimal digits) into the tape file. This serves as a useful example of the procedure discussed above for creating a data file. It also serves as an example of a BASIC program that has been transferred to the Apple to get a hardcopy listing, but the listing shown here has been doctored slightly. (The word "CLR" in line 10 was inserted by hand.)

The memory range is written into a data file on the tape. The tape is transferred to the other tape player and loaded into the Apple by the cassette file loader. The data is then loaded into the Apple's memory by the Applesoft program in Listing 4. Note that it does not necessarily have to be loaded into the same address range from whence it came. Use BSAVE to save the memory range as a conventional "B" type file if you wish. The disassembler of the monitor or autostart ROM will work on this.

Commodore Tape Format

This part gets technical, so I am going to start by defining a few terms.

A *cycle* is a complete wave cycle (both half-cycles; for a square wave, both the down and the up phases).

The *duration* of a cycle is the total time spanned by a complete cycle (both half-cycles).

There are 3 kinds of *bits*, each consisting of 2 cycles of different durations. The following table gives approximate cycle durations in microseconds:

	1st cycle	2nd cycle
"1" BIT	500 µs	333 µs
"0" BIT	333	500
SYNC	667	500

Listing 1 *(continued)*

```
910F 91 0C                  STA (FMPL),Y
9111 A9 A3                  LDA #BUFFER
9113 C8                     INY
9114 91 0C                  STA (FMPL),Y
9116 A9 94                  LDA /BUFFER
9118 C8                     INY
9119 91 0C                  STA (FMPL),Y
911B A2 00                  LDX #0         ;NEW FILE IS OK
911D 20 D6 03               JSR DOSFM
9120 B0 6E                  BCS DOSERR
                 ;
                 ;-------------------------------
                 ;POSITION FILE AT START
                 ;-------------------------------
                 ;
9122 A9 0A                  LDA #$A        ;CALL TYPE $A =
9124 A0 00                  LDY #0         ; POSITION
9126 91 0C                  STA (FMPL),Y
9128 A9 00                  LDA #0
912A A0 04                  LDY #4
912C 91 0C                  STA (FMPL),Y
912E C8                     INY
912F 91 0C                  STA (FMPL),Y
9131 A2 01                  LDX #1
9133 20 D6 03               JSR DOSFM
9136 B0 58                  BCS DOSERR

                 ;
                 ;-------------------------------
                 ;WRITE THE DATA
                 ;-------------------------------
0                ;
9138 A9 04                  LDA #4         ;CALL TYPE 4 = WRITE
913A A0 00                  LDY #0
913C 91 0C                  STA (FMPL),Y
913E A9 01                  LDA #1         ;ONE BYTE AT A TIME
9140 C8                     INY
9141 91 0C                  STA (FMPL),Y
                 ;
                 ;INITIALIZE BUFFER POINTER TO
                 ;1ST BYTE OF ACTUAL DATA
                 ;
9143 A9 C2                  LDA #BODY      ;PTR —> BODY
9145 85 08                  STA PTR
9147 A9 08                  LDA /BODY
9149 85 09                  STA PTR+1
                 ;
                 ;SKIP EVERY 192ND BYTE (BLOCK-TYPE TOKENS)
                 ;
914B A2 BF        PRINT1    LDX #BLOKLEN-1
914D 8E 02 03               STX KNT        ;CHAR COUNTER
9150 A0 00        PRINT2    LDY #0
9152 B1 08                  LDA (PTR),Y
                 ;
                 ;WATCH FOR END OF FILE,
                 ;WHICH IS MARKED BY A ZERO BYTE
                 ;
9154 F0 21                  BEQ WRAPUP     ;BRANCH IF ZERO
9156 09 80                  ORA #$80       ;SET BIT 7
9158 A0 08                  LDY #8
915A 91 0C                  STA (FMPL),Y   ;BYTE TO BE WRITTEN
915C A2 01                  LDX #1
915E 20 D6 03               JSR DOSFM      ;< WRITE THE BYTE >
9161 B0 2D                  BCS DOSERR     ;BRANCH IF ERROR
                 ;
                 ;INCREMENT BUFFER POINTER
                 ;
9163 E6 08                  INC PTR
9165 D0 02                  BNE PRINT3
9167 E6 09                  INC PTR+1
```

Listing 1 *(continued)*

```
9169 CE Ø2 Ø3   PRINT3  DEC KNT         ;SKIP 1ST BYTE
916C DØ E2              BNE PRINT2      ;OF EACH BLOCK
916E E6 Ø8              INC PTR
917Ø DØ D9              BNE PRINT1
9172 E6 Ø9              INC PTR+1
9174 DØ D5              BNE PRINT1
9176 ØØ                 BRK
                ;
                ;-----------------------------
                ;CLOSE OUTPUT FILE
                ;-----------------------------
                ;
9177 A9 Ø2      WRAPUP  LDA #2          ;CALL TYPE 2 = CLOSE
9179 AØ ØØ              LDY #Ø
917B 91 ØC              STA (FMPL),Y
917D A2 Ø1              LDX #1
917F 2Ø D6 Ø3           JSR DOSFM
9182 BØ ØC              BCS DOSERR
                ;
                ;-----------------------------
                ;PRINT"DONE" AND EXIT TO BASIC
                ;-----------------------------
                ;
9184 A9 EØ              LDA #MESG7
9186 85 ØA              STA ADR
9188 A9 91              LDA /MESG7
918A 85 ØB              STA ADR+1
918C 2Ø 62 93           JSR PRMESG
918F 6Ø                 RTS             ;EXIT
                ;
                ;-----------------------------
                ;DOS ERROR
                ;-----------------------------
                ;
919Ø A9 EA      DOSERR  LDA #MESG8
9192 85 ØA              STA ADR
9194 A9 91              LDA /MESG8
9196 85 ØB              STA ADR+1
9198 2Ø 62 93           JSR PRMESG
919B AØ ØA              LDY #$A
919D B1 ØC              LDA (FMPL),Y    ;ERROR CODE
919F 2Ø DA FD           JSR PRBYTE      ;PRINT THE HEX CODE
91A2 ØØ                 BRK             ;ABANDON SHIP
                ;
                ;-----------------------------
                ;MESSAGES
                ;-----------------------------
                ;
91A3 AØ AØ AØ   MESG5   ASC "              "
91AF D4 C5 D8           ASC "TEXT FILE LOADER"
91BF 8D 8D 8D           BYT $8D,$8D,$8D
91C2 D2 CF CC           ASC "ROLL TAPE"
91CB ØØ                 BYT Ø
91CC C5 CE C4   MESG6   ASC "END OF FILE"
91D7 8D                 BYT $8D
91D8 D3 C1 D6           ASC "SAVING:"
91DF ØØ                 BYT Ø
91EØ 8D 8D      MESG7   BYT $8D,$8D
91E2 C4 CF CE           ASC "DONE"
91E6 8D 8D 8D           BYT $8D,$8D,$8D,Ø
91EA 8D         MESG8   BYT $8D
91EB C4 CF D3           ASC "DOS ERROR CODE:"
91FA ØØ                 BYT Ø
                ;
                ;-----------------------------
                ;DEFAULT FILE NAME (3Ø CHARS)
                ;-----------------------------
                ;
91FB C3 CF CD   DFALT   ASC "COMMODORE FILE "
92ØA AØ AØ AØ           ASC "               "
                ;
```

Note that the "1" and "0" bit have the same total duration. A byte of data is coded as follows:

sync bit
8 data bits (LSB first...MSB last)
parity bit

The parity bit is "1" if the byte parity is even and "0" if the parity is odd. Figure 2 shows a typical byte frame.

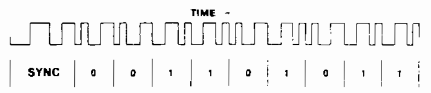


Figure 2. Example of tape format for a single byte. The SYNC bit is followed by 8 data bits with the least significant bit first. The value of this byte is thus $AC in hex. The last bit on the right is the parity bit. Since in this case the number of "1" bits is even, the parity is even, so the parity bit is "1". The parity bit helps to check for errors.

I will use the term "block" to describe the next level of structure. A block contains all the information in the cassette buffer, which is 192 bytes. The format of a block is as follows:

leader tone of continuous 333 microsecond cycles
9 count-down bytes, $89...$81
192 data bytes
checksum byte
a single 667 microsecond cycle
about 80 cycles of 333 microseconds (spacer)
9 count-down bytes, 9...1
data bytes (repeated)
checksum byte
a single 667 microsecond cycle
about 80 cycles of 333 microseconds (trailer)

The checksum byte is the EOR of all of the data bytes in the block.

A "file" is simply a sequence of blocks. The first block in the file is a header which contains the file name. The last block is a special End-Of-File marker block, although this can be omitted. The actual end of the file is indicated by a zero byte in the data after the last legitimate character in the final data block.

Overview of the Program

Toward the end of Listing 1 is a subroutine labeled "GETBIT". It watches the cassette input (TAPEIN) for two cycles (down, up, down, up).

The x-register measures the duration of the first cycle, and the y-register measures the duration of the second cycle. A comparison of the two tells whether it is a "1" or a "0". The bit is left in the carry flag so that it can easily be rotated into the data byte.

Obviously, the timing of this program is critical because the cycle durations are measured by counting trips through program loops. That is why the interrupt disable flag is set (SEI instruction) at the top of the program. However, any peripheral device which still slows down the 6502 will interfere with this program and must be removed.

The subroutine labeled "BLOCK" reads any block from a Commodore tape and adds it to a memory buffer. The memory buffer used here begins at $801 and extends to $8FFF. Since the data field is repeated on the tape, the program verifies that the second occurrence of the data matches what is in memory.

The end of the file is signaled by a zero byte in the data field. When the file is fully loaded, the program writes a "T" type file with the same name as it finds in the file header. If no name is found, the default name "COMMODORE FILE" is used.

This program uses the DOS File Manager for all disk operations. *Beneath Apple DOS* by Don Worth and Pieter Lechner explains in detail how to use the File Manager from assembly language.

Summary

Although there may be less cumbersome ways to transfer data between computers, I went with this method because it didn't cost me any money. One could call it a poor man's modem. The success of this program demonstrates the possibility of two other cheap tricks: (1) It should be possible for the Apple to write tape files that are readable by Commodore computers. (2) It should also be possible to have a direct link between the Commodore cassette interface and the Apple cassette interface. The read and write lines would, of course, be crossed over. In addition there would have to be a signal ground connection and a fourth connection from an annunciator output of the Apple's game port to the cassette sense input of the Commodore's cassette port. The latter connection would allow the Apple to simulate the button-down condition of the C2N tape drive.

Listing 1 *(continued)*

```
                  ;*****************
                  ;*               *
                  ;*  SUBROUTINES  *
                  ;*               *
                  ;*****************
                  ;
                  ;------------------------------
                  ;READ A BLOCK
                  ;------------------------------
                  ;
                  ;INITIALIZE POINTER & CHECKSUM
                  ;
9219 AD Ø5 Ø3      BLOCK    LDA START      ;PTR = START
921C 85 Ø8                  STA PTR        ;PTR -->
921E AD Ø6 Ø3               LDA START+1    ; START OF BLOCK
9221 85 Ø9                  STA PTR+1
9223 A9 ØØ                  LDA #Ø
9225 8D ØØ Ø3               STA CHSUM
                  ;
                  ;READ COUNT-DOWN BYTES
                  ;
9228 A9 Ø9                  LDA #9         ;9 COUNT-DOWN BYTES
922A 8D Ø4 Ø3               STA KDOWN      ;COUNTER
922D A2 Ø6         BLOCK1   LDX #6
922F 2Ø DA 92               JSR RDBYTE1
9232 A5 Ø6                  LDA BYTE
9234 29 7F                  AND #$7F       ;CLEAR BIT 7
9236 CD Ø4 Ø3               CMP KDOWN      ;IS IT CORRECT
9239 DØ 48                  BNE ERR4       ;IF NOT, THEN QUIT
923B CE Ø4 Ø3               DEC KDOWN
923E DØ ED                  BNE BLOCK1
924Ø A2 Ø6                  LDX #6
9242 DØ Ø2                  BNE BLOCK3     ;BRANCH ALWAYS
                  ;
                  ;READ DATA BYTES
                  ;
9244 A2 ØB         BLOCK2   LDX #11
9246 2Ø DA 92      BLOCK3   JSR RDBYTE1    ;< NEXT DATA BYTE >
9249 A5 Ø6                  LDA BYTE
924B 4D ØØ Ø3               EOR CHSUM      ;CHSUM =
924E 8D ØØ Ø3               STA CHSUM      ; EOR OF ALL DATA
9251 AØ ØØ                  LDY #Ø
9253 A5 Ø6                  LDA BYTE
9255 AE Ø3 Ø3               LDX SCAN       ;LOAD OR VERIFY
9258 FØ Ø6                  BEQ BLOCK4     ;BRANCH IF LOADING
925A D1 Ø8                  CMP (PTR),Y    ;VERIFY THIS CHAR
925C DØ 31                  BNE ERR2
925E FØ Ø4                  BEQ BLOCK5     ;BRANCH ALWAYS
926Ø 91 Ø8         BLOCK4   STA (PTR),Y    ;STORE THIS CHAR
9262 EA                     NOP            ;TIME DELAY
9263 EA                     NOP
9264 E6 Ø8         BLOCK5   INC PTR        ;INCREMENT
9266 DØ Ø2                  BNE BLOCK6     ; BUFFER POINTER
9268 E6 Ø9                  INC PTR+1
926A A5 Ø8         BLOCK6   LDA PTR        ;PTR < FIN
926C CD Ø7 Ø3               CMP FIN
926F A5 Ø9                  LDA PTR+1
9271 ED Ø8 Ø3               SBC FIN+1      ;IF SO,
9274 9Ø CE                  BCC BLOCK2     ; GET ANOTHER CHAR
                  ;
                  ;READ CHECKSUM BYTE
                  ;
9276 A2 ØB                  LDX #11
9278 2Ø DA 92               JSR RDBYTE1
927B A5 Ø6                  LDA BYTE
927D CD ØØ Ø3               CMP CHSUM      ;DOES IT CHECK
928Ø DØ 19                  BNE ERR3       ;IF NOT, THEN QUIT
9282 6Ø                     RTS
                  ;
                  ;ERROR TRAPS
                  ;
```

Listing 1 *(continued)*

```
9283 A9 C4     ERR4     LDA #MESG4
9285 85 ØA              STA ADR
9287 A9 92              LDA /MESG4
9289 85 ØB              STA ADR+1
928B 2Ø 62 93           JSR PRMESG
928E ØØ                 BRK
928F A9 A7     ERR2     LDA #MESG2
9291 85 ØA              STA ADR
9293 A9 92              LDA /MESG2
9295 85 ØB              STA ADR+1
9297 2Ø 62 93           JSR PRMESG
929A ØØ                 BRK
929B A9 B4     ERR3     LDA #MESG3
929D 85 ØA              STA ADR
929F A9 92              LDA /MESG3
92A1 85 ØB              STA ADR+1
92A3 2Ø 62 93           JSR PRMESG
92A6 ØØ                 BRK
92A7 D6 C5 D2  MESG2    ASC "VERIFY ERROR"
92B3 ØØ                 BYT Ø
92B4 C3 C8 C5  MESG3    ASC "CHECK-SUM ERROR"
92C3 ØØ                 BYT Ø
92C4 C3 CF D5  MESG4    ASC "COUNT-DOWN ERROR"
92D4 ØØ                 BYT Ø
92D5 CE Ø4 Ø3           DEC KDOWN
              ;
              ;------------------------------
              ;READ A BYTE
              ;------------------------------
              ;
              ;WAIT FOR SYNC BIT
              ;
92D8 A2 Ø2     RDBYTE   LDX #2
92DA 2Ø 55 93  RDBYTE1  JSR PULSE1
92DD EØ 43              CPX #$43      ;15ØØ HZ CYCLE
92DF 9Ø F7              BCC RDBYTE
92E1 EØ 56              CPX #$56
92E3 BØ F3              BCS RDBYTE
92E5 A2 Ø2              LDX #2        ;IF SO,
92E7 2Ø 55 93           JSR PULSE1    ; LOOK AT NEXT CYCLE
92EA EØ 3Ø              CPX #$3Ø      ;2ØØØ HZ CYCLE
92EC 9Ø EA              BCC RDBYTE
92EE EØ 43              CPX #$43
92FØ BØ E6              BCS RDBYTE
              ;
              ;DATA BITS
              ;
92F2 A9 ØØ              LDA #Ø
92F4 8D Ø1 Ø3           STA PAR       ;CLEAR PARITY COUNT
92F7 A9 Ø8              LDA #8        ;DO 8 BITS
92F9 8D Ø2 Ø3           STA KNT
92FC 2Ø 32 93  RDBYTE2  JSR GETBIT
92FF A5 Ø6              LDA BYTE
93Ø1 6A                 ROR A         ;ROTATE BIT
93Ø2 85 Ø6              STA BYTE      ; INTO BYTE
93Ø4 4D Ø1 Ø3           EOR PAR       ;EOR THIS BIT WITH
93Ø7 8D Ø1 Ø3           STA PAR       ;BIT 7 OF
93ØA CE Ø2 Ø3           DEC KNT       ; PARITY COUNT
93ØD DØ ED              BNE RDBYTE2
              ;
              ;CHECK PARITY
              ;
93ØF 2Ø 32 93           JSR GETBIT
9312 6A                 ROR A
9313 4D Ø1 Ø3           EOR PAR
9316 1Ø Ø1              BPL ERR1
9318 6Ø                 RTS
              ;
              ;PARITY ERROR
              ;
```

Listing 2

```
 1Ø HOME :
    PRINT "TEXTLISTER"
 2Ø PRINT :
    PRINT "BY ART MATHENY"
 3Ø PRINT
 4Ø PRINT TAB( 6);
     "THIS PROGRAM WILL LIST
    A"
 5Ø PRINT TAB( 8);
     "SEQUENTIAL TEXT FILE."
 6Ø PRINT
 7Ø INPUT "GIVE THE FILE
    NAME: ";F$
 8Ø PRINT
 9Ø PRINT "WHAT SLOT IS THE
    PRINTER IN (Ø FOR TV)":
    INPUT SLOT
1ØØ IF SLOT >= Ø AND SLOT
    <= 7 THEN 11Ø
1Ø5 PRINT "ENTER A NUMBER
    BETWEEN Ø AND 7.":
    GOTO 9Ø
11Ø PRINT
12Ø D$ = CHR$ (4):
    REM <CTRL-D>
13Ø ONERR GOTO 33Ø
14Ø PRINT D$;"PR #";SLOT
15Ø PRINT : PRINT : PRINT :
    PRINT
16Ø PRINT "LISTING OF FILE:
    ";F$
17Ø PRINT : PRINT
18Ø PRINT D$;"OPEN ";F$
19Ø PRINT D$;"READ ";F$
2ØØ :
21Ø REM
    ---------------------------
22Ø REM GET ONE CHARACTER
    AT A TIME
23Ø REM
    ---------------------------
24Ø :
25Ø GET A$:A = ASC (A$)
255 IF A > 31 THEN PRINT "
    "; CHR$ (128 + A);:
    GOTO 25Ø
26Ø IF A = 13 THEN PRINT "
    ": GOTO 25Ø
27Ø PRINT " @";: GOTO 25Ø:
    REM UNPRINTABLE CHAR
28Ø :
29Ø REM
    --------------------------
3ØØ REM ERROR HANDLING
    ROUTINE
31Ø REM
    --------------------------
32Ø :
33Ø PRINT : PRINT : PRINT :
    PRINT
34Ø PRINT D$;"CLOSE ";F$
35Ø PRINT D$;"PR #Ø"
36Ø END
```

Listing 3

```
 10 PRINT"{CLR}
    SAVE A RANGE OF MEMORY"
 20 PRINT"WHAT IS THE":
    INPUT"STARTING ADDRESS";
    K1
 30 PRINT"WHAT IS THE":
    INPUT"ENDING ADDRESS";
    K2
 40 PRINT"WHAT IS THE":
    INPUT"FILENAME";F$
 50 OPEN 1,1,2,F$
 60 PRINT#1,K1;CHR$(13)
 70 PRINT#1,K2;CHR$(13)
 80 FOR K=K1 TO K2
 90 PRINT#1,PEEK(K);
    CHR$(13)
100 NEXT
110 CLOSE 1
120 END
```

Listing 4

```
 10 TEXT : HOME
 20 PRINT "LOADING NUMERIC
    DATA FROM A TEXT FILE"
 30 PRINT "INTO A RANGE OF
    MEMORY."
 40 PRINT
 50 INPUT "WHAT IS THE
    FILENAME ";F$
 60 PRINT "WHAT IS THE
    STARTING ADDRESS (ENTER
    0"
 70 PRINT "TO PUT IT AT
    THE ORIGINAL ADDRESS)"
 80 INPUT A1
 90 PRINT  CHR$ (4);
    "OPEN ";F$
100 PRINT  CHR$ (4);
    "READ ";F$
110 INPUT K1: INPUT K2
120 IF A1 = 0 THEN A1 = K1
130 L = K2 - K1 + 1:
    A2 = A1 + L - 1
140 PRINT
150 PRINT "STARTING
    ADDRESS = ";A1
160 PRINT "ENDING ADDRESS
    = ";A2
170 PRINT "LENGTH = ";L
180 PRINT
190 FOR K = A1 TO A2
200 INPUT X: POKE K,X
210 NEXT K
220 PRINT  CHR$ (4);
    "CLOSE ";F$
230 END
```

MICRO

Listing 1 *(continued)*

```
9319 A9 25      ERR1     LDA #MESG1
931B 85 0A               STA ADR
931D A9 93               LDA /MESG1
931F 85 0B               STA ADR+1
9321 20 62 93            JSR PRMESG
9324 00                  BRK
9325 D0 C1 D2   MESG1    ASC "PARITY ERROR"
9331 00                  BYT 0
               ;
               ;-------------------------------
               ;READ A BIT
               ;-------------------------------
               ;
               ;SUBROUTINE RETURNS:
               ; X=DURATION OF 1ST PULSE
               ; Y=DURATION OF 2ND PULSE
               ; CARRY SET IFF X> Y
               ;
9332 A2 05      GETBIT   LDX #5
9334 E8         GETBIT1  INX
9335 AD 60 C0            LDA TAPEIN
9338 30 FA               BMI GETBIT1
933A E8         GETBIT2  INX
933B AD 60 C0            LDA TAPEIN
933E 10 FA               BPL GETBIT2
9340 A0 00               LDY #0
9342 C8         GETBIT3  INY
9343 AD 60 C0            LDA TAPEIN
9346 30 FA               BMI GETBIT3
9348 C8         GETBIT4  INY
9349 AD 60 C0            LDA TAPEIN
934C 10 FA               BPL GETBIT4
934E 84 07               STY TEMP
9350 E4 07               CPX TEMP
9352 60        ;         RTS
               ;-------------------------------
               ;READ A SINGLE PULSE
               ;-------------------------------
               ;
9353 A2 00      PULSE    LDX #0
9355 E8         PULSE1   INX
9356 AD 60 C0            LDA TAPEIN
9359 30 FA               BMI PULSE1
935B E8         PULSE2   INX
935C AD 60 C0            LDA TAPEIN
935F 10 FA               BPL PULSE2
9361 60                  RTS
               ;
               ;-------------------------------
               ;PRINT MESSAGES
               ;-------------------------------
               ;
9362 A0 00      PRMESG   LDY #0
9364 B1 0A      PRMESG1  LDA (ADR),Y
9366 F0 08               BEQ PRMESG2    ;BRANCH IF ZERO
9368 09 80               ORA #$80       ;SET BIT 7
936A 20 F0 FD            JSR COUT1
936D C8                  INY
936E D0 F4               BNE PRMESG1
9370 A9 8D      PRMESG2  LDA #$8D       ;RETURN CHAR
9372 20 F0 FD            JSR COUT1
9375 60                  RTS
               ;
               ;-------------------------------
               ;FILE MANAGER BUFFERS
               ;-------------------------------
               ;
9376            WORKAREA DFS 1
93A3            SECTOR   EQU WORKAREA+45
94A3            BUFFER   EQU SECTOR+256
9377                     END
```

Subscribe to MICRO...

Save 20% and we'll send you a BONUS GIFT

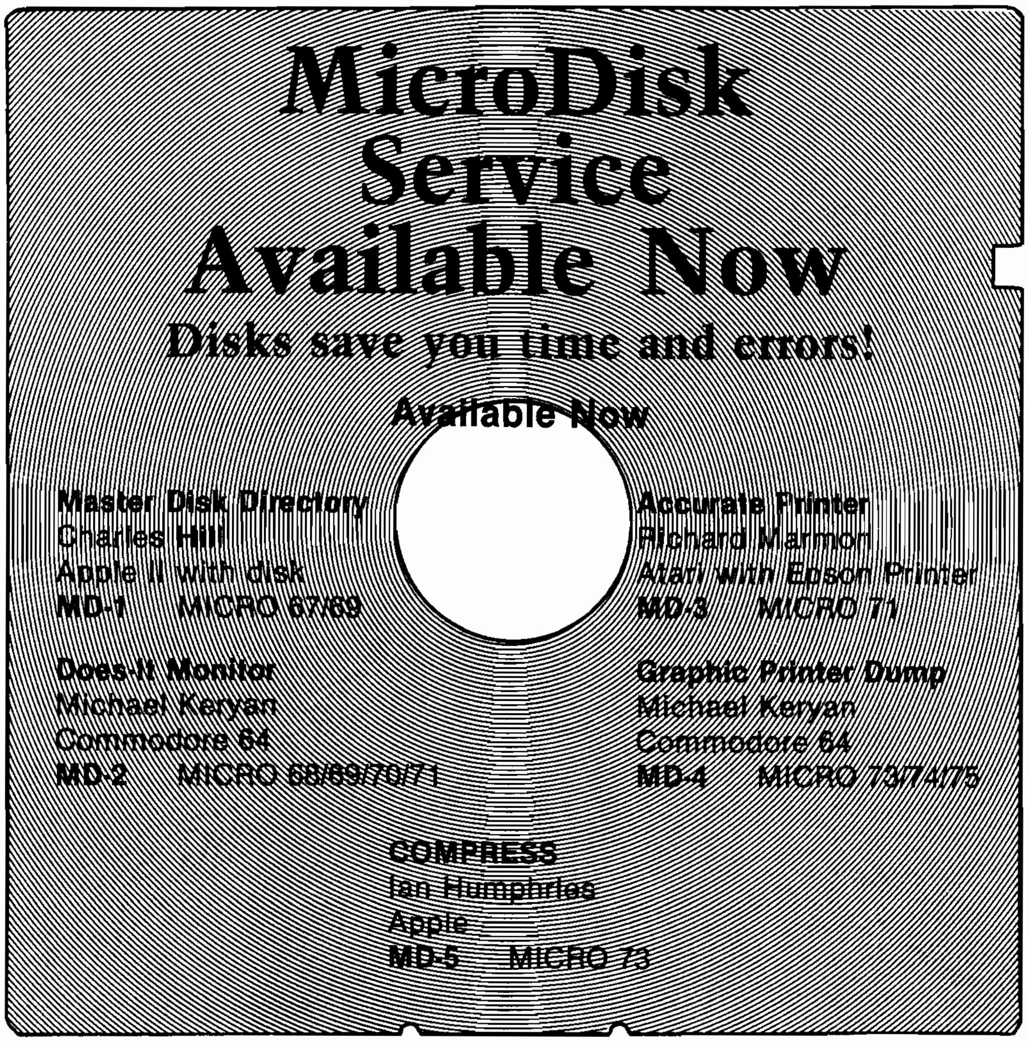
with your subscription!

Receive a coupon good for one MicroDisk of your choice. A $15.00 Value — FREE!

Each Disk contains a variety of programs from Micro, all entered and ready to run on your machine. Saves time, avoids errors.

This offer expires October 31, 1984

Fill out the attached card and mail today!



MicroDisk Service Available Now

Disks save you time and errors!

Available Now

Master Disk Directory
Charles Hill
Apple II with disk
MD-1 MICRO 67/69

Accurate Printer
Richard Marmon
Atari with Epson Printer
MD-3 MICRO 71

Does-It Monitor
Michael Keryan
Commodore 64
MD-2 MICRO 68/69/70/71

Graphic Printer Dump
Michael Keryan
Commodore 64
MD-4 MICRO 73/74/75

COMPRESS
Ian Humphries
Apple
MD-5 MICRO 73

Each diskette includes all of the programs in BASIC and/or Assembly Source, plus binary 'load-and-go' files. The price of only $15.00 includes shipping and handling.

SAVE 20%!

Use This Postage-Paid Card To Subscribe... Or Call Toll-Free 1-800-345-8112

(In PA 1-800-662-2444)

(MasterCard and VISA Accepted)

MICRO™ **SAVE 20%**

☐YES! Enter my subscription to MICRO for just $24.00 per year and send me a BONUS GIFT COUPON for one free MicroDisk.

☐ Renew my subscription to MICRO for just $24.00 and send me a BONUS GIFT COUPON for one free MicroDisk.

My mailing label number is MC__ __ __ __ __ __ __ __ __ __.

Please rush my subscription and the BONUS GIFT I've checked to:

Name ____________________

Address ____________________

City ____________ State ______ Zip ______

(Allow 6-8 weeks for delivery) Bonus Coupon offer expires Oct. 31, 1984

I'm paying by: ☐Check ☐MO ☐VISA ☐MC

Total Enclosed:$__________

Visa/MC #__________

Exp. Date:__________

MICRO™'s Newest Software

MicroDisks

Send me the following MicroDisks at $15.00 each prepaid

☐MD-1 Master Disk Directory
☐MD-2 Does'It Monitor
☐MD-3 Accurate Printer
☐MD-4 Graphic Printer Dump
☐MD-5 COMPRESS

MicroCalc

The Full Screen Calculator

Easier than spreadsheet programs, does calculations of unlimited length and complexity on screen, links screens, saves on disk or tape, provides formatted printer output. Contains a 48'page manual and diskette.

☐ MicroCalc for Commodore 64............$29.95
☐ MicroCalc for Apple II/IIe/IIc..............$29.95
☐ MicroCalc for Atari..............................$29.95

Please rush the Micro Software checked above to:

Name ____________________

Address ____________________

City ____________ State ______ Zip ______

MA residents add 5% sales tax.

I'm paying by: ☐ Check ☐ MO ☐ VISA ☐ MC

Total Enclosed: $ __________

Visa/MC # __________

Exp. Date: __________

MICRO™'s Best Sellers

Mastering Your VIC-20

Contains 8 projects and programs..................Utilities, games — even a version of "VisiCalc." All 8 programs on cassette to help you learn faster. Makes learning to program you VIC 20 fun.

☐ Mastering Your VIC 20...$19.95

What's Where in the Apple

Revised new edition with Apple IIe information added to original atlas and gazetter. All Apple users will find this book helpful in understanding their machine and essential for mastering it.

☐ What's Where in the Apple **@ $19.95**

Best Sellers for APPLE Users!

MICRO on the APPLE

Programming aids, utilities, games, enhancements. Together Volumes 1, 2, and 3 contain over 100 programs on diskette. Fully documented and explained.

☐ 3-Volume Gift-Boxed @ $59.95
☐ Vol.2☐ Vol.3 $24.95 ea.

Please rush the MICRO Books I've checked above to:

Name ____________________

Address ____________________

City ____________ State ______ Zip ______

MA residents add 5% sales tax.

I'm paying by: ☐ Check ☐ MO ☐ VISA ☐ MC

Total Enclosed: $ __________

Visa/MC # __________

Exp. Date: __________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 60, CHELMSFORD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

P.O. Box 6502
Chelmsford, MA 01824

SAVE 20%!!!!

Subscribe to

MICRO

Use This Postage Paid Card to Order the Next 12 Issues of MICRO and SAV

$6.00 Off Newsstand Price!

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 60, CHELMSFORD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

P.O. Box 6502
Chelmsford, MA 01824

A Feast O Computin Ideas...

New Boo From

MICRC

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 60, CHELMSFORD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

P.O. Box 6502
Chelmsford, MA 01824

Order Thes

Best Sellers

From

MICRC

▶BASIC Hex Loader▼

A short BASIC utility that loads DATA written in Hexidecimal notation. A special version for the C-64 generates the DATA statements.

by Robert M. Tripp

Requirements: Any BASIC

If you have an assembly listing or the hex dump of a machine language program, getting it to load with BASIC can be a real problem. BASIC likes to work only in decimal, so you must make the conversion from hex to decimal and then type in the DATA statements. For years, MICRO has had to 'waste space' providing both the 'useful' assembly listing and the 'necessary' decimal DATA statement form of the same information. If there was a simple way to input the natural hex information, then this additional dump would not be required.

One solution is presented here in Listing 1. It is a simple, short BASIC program that will load hexidecimal information. It is best understood through a brief example. Suppose that you have an assembly program that starts as follows:

```
Ø33C A5 7A       ENTER   LDA   TXTPTR
Ø33E 8D 7Ø Ø3            STA   TEMPLO
Ø341 A5 7B               LDA   TSTPTR1
```

and so forth. Normally you would have to convert the hex information: A5 7A 8D 70 03 A5 7B etc. into the decimal equivalents to generate the following DATA statement:

```
DATA 165,122,141,112,3,165,123
```

The HEX Loader lets you use a DATA statement of the form:

```
DATA "A57A8D7ØØ3A57B"
```

which is obviously much easier to generate.

Using Hex Loader

The first DATA statement must be the hex address at which the hex information is to start loading. The remaining DATA statements each consist of an ASCII string that contains the hex data, terminated by the non-hex ASCII pair "YY". The end of hex information is indicated by the non-hex ASCII pair "XX". For example:

```
1ØØØØ DATA "Ø33C"
1ØØ1Ø DATA "A57A8D7ØØ3A57BYY"
1ØØ2Ø DATA "8D71Ø3A9ØØ857AA9Ø2XX"
```

The program was written to fit neatly between lines 10 and 20 of your typical BASIC program. You may want to change line 14 so that it performs a **GOTO** when done loading instead of the current **END**. That is the only change that should be required to add this utility to your programs.

Hex DATA Generator

The second listing is a special program for the Commodore 64 that generates the BASIC DATA statements from information already existing in memory. You may already have the information in memory from an assembly, from entering it through a monitor, or as the result of running a program. You specify the BASIC line number to start using for the DATA statements and the memory start and ending addresses. The program automatically generates all of the DATA statements required by the Hex Loader and then automatically deletes itself, leaving just the Hex Loader and the DATA statements. It is really pretty neat — and fun to watch in operation, since most of the action is on the screen. And, it can save you a lot of time.

Listing 1

```
1Ø REM HEX LOADER  R.M.TRIPP
11 READ X$:Z=LEN(X$):GOSUB 17:
   MS=X:Z=2
12 READ HX$:J=1
13 X$=MID$(HX$,J,2)
14 IF X$="XX" THEN END
15 IF X$="YY" THEN GOTO 12
16 GOSUB 17:POKE MS,X:MS=MS+1:
   J=J+2:GOTO 13
17 X=Ø:FOR I=1 TO Z:
   Y=ASC(MID$(X$,I,1)):
   IF Y>57 THEN Y=Y-7
18 Y=Y-48:X=X*16+Y:NEXT:RETURN
```

Listing 2

```
 1 REM HEX MAKER  R.M. TRIPP
 2 Z=4:INPUT "{CLEAR}BASIC LINE
   NUMBER: ";LN
 3 INPUT "HEX START ADDR: ";X$:
   MS$=X$:GOSUB 3Ø:MS=X
 4 INPUT "HEX LAST ADDR: ";X$:
   GOSUB 3Ø:ME=X
 5 PRINT "{CLEAR}";
   MID$(STR$(LN),2);" DATA ";
   CHR$(34);MS$;CHR$(34):
   LN=LN+1Ø:K=1:GOTO 7
 6 PRINT "{CLEAR}";:K=Ø
 7 FOR I=K TO 6:
   PRINT MID$(STR$(LN),2);
   " DATA ";CHR$(34);
 8 FOR J=ØTO1Ø:X=PEEK(MS):
   GOSUB 5Ø:PRINT HL$;:MS=MS+1
 9 IF MS> ME THEN PRINT "XX";-
   :I=6:
   J=11
1Ø NEXT J:PRINT "YY";CHR$(34):
   LN=LN+1Ø
11 NEXT I:PRINT"LN=";LN;":
   MS=";MS;":ME=";ME
12 IF MS> ME THEN PRINT"{DOWN2}
   GOTO 14":GOTO 16
13 PRINT "{DOWN2}GOTO 6":GOTO 16
14 PRINT "{CLEAR}";:FORI=1TO8:
   PRINT I:NEXT:PRINT "GOTO 15":
   GOTO 16
15 PRINT "{CLEAR}";:FORI=9TO16:
   PRINT I:NEXT
16 POKE 631,19:FOR I=1 TO 9:
   POKE 631+I,13:NEXT:
   POKE 198,1Ø:END
2Ø REM HEX LOADER  R.M.TRIPP
21 READ X$:Z=LEN(X$):GOSUB 3Ø:
   MS=X:PRINT "{CLEAR}LOADING
   FROM ";X$;" TO ";:Z=2
22 READ HX$
23 FOR J=1 TO 99 STEP 2:
   X$=MID$(HX$,J,2)
24 IF X$="XX" THEN MS=MS-1:
   GOSUB 4Ø:PRINT MS$:END
25 IF X$="YY" THEN J=99:GOTO 27
26 GOSUB 3Ø:POKE MS,X:MS=MS+1
27 NEXT:GOTO 22
3Ø X=Ø:FOR I=1 TO Z:
   Y=ASC(MID$(X$,I,1)):
   IF Y> 57 THEN Y=Y-7
31 Y=Y-48:X=X*16+Y:NEXT:RETURN
4Ø X=INT(MS/256):GOSUB 5Ø:
   MS$=HL$:X=INT(MS-X*256):
   GOSUB 5Ø:MS$=MS$+HL$:RETURN
5Ø H=INT(X/16):L=INT(X-H*16):
   IF H> 9 THEN H=H+7
51 IF L> 9 THEN L=L+7
52 HL$=CHR$(H+48)+CHR$(L+48):
   RETURN
```

Circles for the Commodore 64

by Lester Cain

An interesting mathematical way to plot circles on the C-64

Editor's Note: For easy method of entering hex object into a BASIC program, see Hex Loader, by R. Tripp, page 65.

The programs contained in this article will give a theory behind creating circles on a Commadore 64 specifically, but generally on any 6502 computer with HiRes capabilities. Also, it gives necessary code to implement circles in a game or business type analysis.

Let us first discuss the problems associated with creating a circle in a HiRes environment. In an 8 bit screen memory each memory address is made up of bytes containing 8 bits in some kind of sequential fashion. Unfortunately, most of the more popular computers do not do this in the same way. Therefore, a universal method has been developed to visualize this screen memory in a way common to all configurations. This universal way of looking at a graphics screen is referred to as World Coordinates X and Y, taken from common graphing methods, where X is the horizontal axis and Y is the vertical axis. The problem then is to draw a circle in this X and Y environment. Using the X and Y outlook, the only time the actual screen layout comes into effect is when actually setting the computed bit at its computed spot in the maze.

```
                          ; CIRCLE DRAWING ROUTINES
                          ; PLOTS HIRES CIRCLE ON THE
                          ; COMMODORE 64.
                          ;
                          ; CODE BY: LESTER CAIN
                          ;
                          ; EXTERNAL GLOBL VARIABLES
                          ;
Ø33C                       ST       EQU $33C
Ø33C                       X1LO     EQU ST
Ø33D                       X1HI     EQU ST+1
Ø33E                       Y1LO     EQU ST+2
Ø344                       CXLO     EQU ST+8
Ø345                       CXHI     EQU ST+9
Ø346                       CY       EQU ST+1Ø
Ø347                       RAD      EQU ST+11
Ø348                       MODE     EQU ST+12
ØØBØ                       SPLO     EQU $BØ
ØØB1                       SPHI     EQU SPLO+1
                          ;
CØØØ                                ORG $CØØØ
                          ;
                          ; CIRCLE: PLOT A CIRCLE IN HIRES
                          ;   ENTRY CONDITIONS:
                          ;     CX AND CY SET BY CALLING
                          ;     RADIUS SET IN GLOBL RAD
                          ;   EXIT CONDITIONS:
                          ;     CIRCLE IS DRAWN IN HIRES
                          ;
CØØØ AD 47 Ø3              CIRCLE   LDA RAD        ;FETCH RADIUS
CØØ3 8D 79 CØ                       STA DX         ;SAVE AS FIRST DX
CØØ6 A8                             TAY            ;COPY RAD TO Y
CØØ7 2Ø 2C C1                       JSR MULT8      ;AND SQUARE IT
CØØA 8D 77 CØ                       STA RSQLO      ;SAVE FOR COMP.
CØØD 8C 78 CØ                       STY RSQHI      ;AND THE HI BYTE
CØ1Ø A9 ØØ                          LDA #$Ø        ;ZERO DY
CØ12 8D 7A CØ                       STA DY
CØ15 2Ø 7B CØ                       JSR COMPXY     ;PLOT 1ST 4 DOTS
CØ18 EE 7A CØ              LOOP     INC DY         ;LEG +1
CØ1B AD 79 CØ                       LDA DX
CØ1E CD 7A CØ                       CMP DY         ;45 DEGREES YET
CØ21 3Ø ØC                          BMI LOOP1      ;PLOT OTHER HALF
CØ23 AC 7A CØ                       LDY DY         ;COMP OTHER LEG
CØ26 2Ø FE CØ                       JSR COMLEG     ;PLOT ANOTHER 1
CØ29 2Ø 7B CØ                       JSR COMPXY
CØ2C 18                             CLC            ;FORCED JUMP
CØ2D 9Ø E9                          BCC LOOP
                          ;
CØ2F AD 47 Ø3              LOOP1    LDA RAD        ;GET THE RADIUS
CØ32 8D 7A CØ                       STA DY
CØ35 A9 ØØ                          LDA #$ØØ       ;ZERO DX
CØ37 8D 79 CØ                       STA DX
CØ3A 2Ø 7B CØ                       JSR COMPXY     ;COMPUTE THIS BAT.
CØ3D EE 79 CØ              LOOP2    INC DX         ;INC Y
CØ4Ø AD 7A CØ                       LDA DY         ;CHECK FOR = WILL
CØ43 CD 79 CØ                       CMP DX         ;MEAN CIRCLE COMP
CØ46 3Ø 2D                          BMI DONE       ;CIRCLE DON
CØ48 AD 79 CØ                       LDA DX         ;SWAP FUNCTIONS
CØ4B 8D 76 CØ                       STA TEMP
CØ4E AD 7A CØ                       LDA DY
CØ51 8D 79 CØ                       STA DX
CØ54 AD 76 CØ                       LDA TEMP
CØ57 8D 7A CØ                       STA DY         ;SWAP DONE
CØ5A AC 7A CØ                       LDY DY         ;COMP. OTHER LEG
CØ5D 2Ø FE CØ                       JSR COMLEG     ;COMPUTE NEW LENG
CØ6Ø 8D 76 CØ                       STA TEMP
```

```
CØ63 AD 7A CØ          LDA DY
CØ66 8D 79 CØ          STA DX
CØ69 AD 76 CØ          LDA TEMP
CØ6C 8D 7A CØ          STA DY
CØ6F 2Ø 7B CØ          JSR COMPXY    ;COMPUTE NEW SET
CØ72 18                CLC
CØ73 9Ø C8             BCC LOOP2
CØ75 6Ø       DONE     RTS           ;RETURN TO CALL
CØ76 ØØ       TEMP     BYT Ø         ;TEMP STORAGE
CØ77 ØØ       RSQLO    BYT Ø
CØ78 ØØ       RSQHI    BYT Ø
CØ79 ØØ       DX       BYT Ø
CØ7A ØØ       DY       BYT Ø
             ;
             ; COMPXY: COMPUTES X,Y COORDINATES
             ;   IN EACH QUADRANT FROM DX,DY
             ; ENTRY CONDITIONS:
             ;   DX, DY COMPUTED BY CALLING PROGRAM
             ; EXIT CONDITIONS:
             ;   A DOT IS PLOTTED IN EACH
             ;   OF THE FOUR QUADRANTS
             ;   X1L AND Y1L ARE THE OFFSET IN 1
             ;   X2L AND Y1L ARE THE OFFSET IN 2
             ;   X2L AND Y2L ARE THE OFFSET IN 3
             ;   X1L AND Y2L ARE THE OFFSET IN 4
             ;
CØ7B AD 45 Ø3  COMPXY  LDA CXHI      ;HI CENTER
CØ7E 8D F9 CØ          STA X1H       ;RT QUADS.
CØ81 8D FB CØ          STA X2H       ;LT QUADS.
CØ84 AD 44 Ø3          LDA CXLO      ;CENTER LO
CØ87 18                CLC
CØ88 6D 79 CØ          ADC DX
CØ8B 8D F8 CØ          STA X1L
CØ8E 9Ø Ø8             BCC CIP1      ;NO OVERFLOW
CØ9Ø AD 45 Ø3          LDA CXHI      ;IS HI ON
CØ93 DØ Ø3             BNE CIP1      ;SKIP INCREM.
CØ95 EE F9 CØ          INC X1H       ;UP RT HI+1
             ;
CØ98 AD 44 Ø3  CIP1    LDA CXLO      ;CENTER X
CØ9B 38                SEC           ;-DX
CØ9C ED 79 CØ          SBC DX
CØ9F 8D FA CØ          STA X2L       ;NEW PLOT X LO
CØA2 BØ Ø3             BCS CIP2      ;NO BORROW
CØA4 CE FB CØ          DEC X2H       ;HIBYTE OF X-1
             ;
CØA7 AD 46 Ø3  CIP2    LDA CY        ;CENTER Y
CØAA 18                CLC           ;+ DY
CØAB 6D 7A CØ          ADC DY
CØAE 8D FC CØ          STA Y1L       ;NEW PLOT Y LO
             ;
CØB1 AD 46 Ø3          LDA CY        ;CENTER Y AG.
CØB4 38                SEC           ;-DY
CØB5 ED 7A CØ          SBC DY
CØB8 8D FD CØ          STA Y2L       ;LO VALUE NEW Y
             ;
             ; TRANSFER NEW VALUES TO X AND Y COORDINATES
             ; AND PLOT THE FOUR NEW POINTS.
             ;
CØBB AD F8 CØ          LDA X1L       ;UPPER RT. QD.
CØBE 8D 3C Ø3          STA X1LO      ;NEW X LO
CØC1 AD F9 CØ          LDA X1H       ;NOW HI VAL.
CØC4 8D 3D Ø3          STA X1HI
CØC7 AD FC CØ          LDA Y1L       ;NOW DO Y
CØCA 8D 3E Ø3          STA Y1LO      ;ONLY LO VAL.
CØCD 2Ø 78 C1          JSR PLOTXY    ;PLOT UP RT.
             ;
CØDØ AD FA CØ          LDA X2L       ;GET NEW X
CØD3 8D 3C Ø3          STA X1LO      ;Y DOES NOT
CØD6 AD FB CØ          LDA X2H       ;CHANGE
```

```
CØD9 8D 3D Ø3              STA X1HI      ;THIS TIME
CØDC 2Ø 78 C1              JSR PLOTXY    ;PLOT UP LT
                  ;
CØDF AD FD CØ              LDA Y2L       ;CHANGE Y THIS
CØE2 8D 3E Ø3              STA Y1LO      ;TIME
CØE5 2Ø 78 C1              JSR PLOTXY    ;PLOT LWR LT.
                  ;
CØE8 AD F8 CØ              LDA X1L       ;CHANGE X THIS
CØEB 8D 3C Ø3              STA X1LO      ;TIME
CØEE AD F9 CØ              LDA X1H       ;SO WE CAN
CØF1 8D 3D Ø3              STA X1HI
CØF4 2Ø 78 C1              JSR PLOTXY    ;PLOT LWR RT.
CØF7 6Ø                    RTS           ;RETURN
CØF8 ØØ            X1L     BYT Ø
CØF9 ØØ            X1H     BYT Ø
CØFA ØØ            X2L     BYT Ø
CØFB ØØ            X2H     BYT Ø
CØFC ØØ            Y1L     BYT Ø
CØFD ØØ            Y2L     BYT Ø

                  ;COMLEG: COMPUTES UNKNOWN LEG OF TRIANGLE.
                  ;ENTRY CONDITIONS:
                  ; RADIUS (HYPOTENUSE) IN ACC., DY IN Y.
                  ;EXIT CONDITIONS:
                  ; DX CONTAINS THE OTHER LEG.
                  ;
CØFE 98            COMLEG  TYA           ;GET DY
CØFF 2Ø 2C C1              JSR MULT8     ;DY*DY
C1Ø2 8D 2A C1              STA TEDYL     ;RETURN LO BYTE
C1Ø5 8C 2B C1              STY TEDYH     ;Y HAS HI BYTE
C1Ø8 AD 77 CØ              LDA RSQLO     ;R LO
C1ØB 38                    SEC
C1ØC ED 2A C1              SBC TEDYL     ;R LO -DY LO
C1ØF 8D 2A C1              STA TEDYL
C112 AD 78 CØ              LDA RSQHI     ;R HI
C115 FØ Ø6                 BEQ XY1       ;NO HI BYTE
C117 ED 2B C1              SBC TEDYH     ;R HI -DY LO
C11A 8D 2B C1              STA TEDYH
C11D AD 2A C1      XY1     LDA TEDYL     ;(R)-(DY)LO
C12Ø AC 2B C1              LDY TEDYH     ;HI BYTE
C123 2Ø 45 C1              JSR SQRT      ;SQRT OF
C126 8D 79 CØ              STA DX        ;SAVE FOR DX
C129 6Ø                    RTS           ;(R)-(DY)
C12A ØØ            TEDYL   BYT Ø
C12B ØØ            TEDYH   BYT Ø
                  ;
                  ; MULT8: 8 BITS BY 8 BITS
                  ; ENTRY CONDITIONS:
                  ; MULTIPLICAND IN Y, MULTIPLIER IN ACC.
                  ; EXIT CONDITIONS:
                  ; LO BYTE IN ACC. HI BYTE IN Y
                  ;
ØØAC               ANSLO   EQU $AC
ØØAD               PLIER   EQU ANSLO+1
ØØAE               CAND    EQU ANSLO+2
                  ;
C12C 85 AD         MULT8   STA PLIER     ;SAVE MULTIPIER
C12E 84 AE                 STY CAND      ;SAVE MULTICAND
C13Ø A9 ØØ                 LDA #$ØØ      ;INIT FIRST VALUE
C132 AØ Ø8                 LDY #$Ø8      ;COUNTER 8 BITS
C134 46 AD         MUL1    LSR ALIER     ;TST NEXT BIT
C136 9Ø Ø3                 BCC MUL2      ;IF OFF ROUND
C138 18                    CLC
C139 65 AE                 ADC CAND      ;IF ON, ADD
C13B 6A            MUL2    ROR A         ;SHIFT ANSWER 1
C13C 66 AC                 ROR ANSLO
C13E 88                    DEY           ;DEC POS. COUNTER
C13F DØ F3                 BNE MUL1      ;LOOP 8 TIMES
```

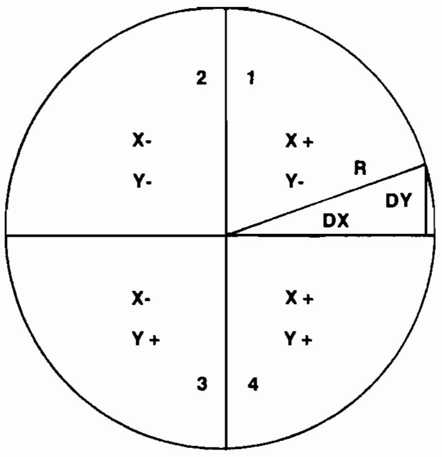


Figure 1

Refer to Figure 1 as this discussion proceeds. The first step will be to define the center of the circle, referred to as CX and CY. Any value will do for a starter, of course assuming it will fit into the screen limitations. Let it be CX = 100 and CY = 100 for an even set of figures to add to and subtract from. Pick out a nice radius for the circle, say R = 50. Divide the circle into 4 quadrants and picture inside each quadrant a right triangle. One side will be DX, the other side DY and the hypotenuse is the Radius. The first point(s) to plot will be on the Radius. No problem so far; the first four points are just + or - from the center of the circle. But this is the end of the easy part. To compute the next point, add one to the value DY and using the pythagorean theorem, compute DX. This formula says the unknown leg is equal to the square root of the (hypotenuse sq. - the known leg sq.). Since this value is the same in all of the 4 quadrants, only one computation is needed. Depending on which quadrant the point is in will determine whether the values DX and DY are added to or subtracted from the center CX,CY values. In quadrant 1, DX is positive and DY is negative. Figure 1 gives each quadrant DX and DY values. To get the circumference point in terms of X and Y, the DY and DX values will be algebraically added to the CX and CX center for each point on the circle. Now, it is time to call the plotting routine 4 times, once for each quadrant. Also this is where the plotting routine is more or less machine dependent.

Continue incrementing DY until it is > = DX. This will plot half of the circle from the horizontal axis right and left. When this point is reached, to make the circle come together in a neat fashion, it is necessary to swap DX and DY and plot from the top and bottom towards the already plotted portion of the circle. Continuing the plot without the swap will leave gaps at the vertical axis, because DY has become larger than DX, stretching integer arithmetic beyond its limits of accuracy.

Listing 1 is the Basic loader to load the machine code into memory. Type it in carefully and save often, especially before trying to run it. The last 39 bytes is a screen clear routine. Listing 2 is a short demo to exercise the code. Type it and save it also. Run the loader first, then the demo routine. If all the data statements were correct, the demo will draw four sets of circles converging at a peak in the center of the screen. These two routines are limited to the Commodore 64 HiRes screen.

Some explanation of the Demo is in order to explain how to use the Circle function.

Line 130	Sets up the storage in the cassette buffer and equates the variables of the circle parameters. CL is the center, X value lo, Ch X value hi.
Line 140	Video chip address, CY is storage for center of the circle.
Line 150	Turns on the HiRes 0 $2000 and clears it.
Line 160	Sets the mode bit to draw.

```
C141 A8                      TAY           ;Y=HI BYTE
C142 A5 AC                   LDA ANSLO     ;A=LO BYTE
C144 60                      RTS
                   ;
                   ; SQRT: 16 BIT SQUARE ROOT
                   ; ENTRY CONDITIONS:
                   ; LO BYTE IN ACC., HI BYTE IN Y
                   ;
                   ; EXIT CONDITIONS:
                   ; SQRT OF NO. IN ACC.
                   ;
00AC                LO       EQU $AC
00AD                HI       EQU LO+1
00AE                LO1      EQU LO+2
00AF                HI1      EQU LO+3
                   ;
C145 85 AC          SQRT     STA LO        ;SAVE LO BYTE
C147 84 AD                   STY HI        ;SAVE HI BYTE
C149 A2 01                   LDX #$01      ;START WITH FIRST 1
C14B 86 AE                   STX LO1
C14D CA                      DEX           ;SUBTRACTION REG
C14E 86 AF                   STX HI1       ;SQRT =0
C150 38             LOP      SEC
C151 A5 AC                   LDA LO        ;SAVE REM IN Y
C153 A8                      TAY
C154 E5 AE                   SBC LO1       ;SUB ODD FROM LO
C156 85 AC                   STA LO        ;ONE REM
C158 A5 AD                   LDA HI        ;SUB 1 FROM HI
C15A E5 AF                   SBC HI1
C15C 85 AD                   STA HI        ;HI REMAINDER
C15E 90 0D                   BCC DNE       ;- RESULT
C160 E8                      INX
C161 A5 AE                   LDA LO1       ;ADD 1 + CARRY
C163 69 01                   ADC #1
C165 85 AE                   STA LO1
C167 90 E7                   BCC LOP       ;NO NEED TO UP HI
C169 E6 AF                   INC HI1       ;HI SUB +1
C16B D0 E3                   BNE LOP
C16D 86 AC          DNE      STX LO        ;CHECK FOR ROUND
C16F C4 AC                   CPY LO        ;REM< N
C171 90 02                   BCC RETS
C173 E6 AC                   INC LO        ;ROUND UP
C175 A5 AC          RETS     LDA LO        ;PUT SQRT IN ACC.
C177 60                      RTS

                   ;
                   ;PLOTXY: PLOTTING ROUTINE
                   ;USED IN GRAPHICS HIRES MODE
                   ;ENTRY CONDITIONS:
                   ; MODE IS SET TO 0,1,2
                   ; X IS IN X1LO AND X1HI
                   ; Y IS IN Y1LO AND Y2HI
                   ;EXIT CONDITIONS:
                   ; 1 BIT IS SET FOR X,Y IN HIRES SCREEN
                   ;
C178 AD 3C 03       PLOTXY   LDA X1LO      ;LINE=XAND7
C17B 48                      PHA
C17C 29 07                   AND #$07
C17E 8D 32 C2                STA LINE
C181 68                      PLA
C182 29 F8                   AND #$F8      ;STRIP X OF LO 3 BITS
C184 85 B0                   STA SPLO      ;INITIAL POINT
C186 AD 3D 03                LDA X1HI
C189 85 B1                   STA SPHI      ;HI BYTE
C18B AD 3E 03                LDA Y1LO
C18E 29 07                   AND #$07      ;STRIP Y OF HI 5 BITS
C190 18                      CLC
C191 65 B0                   ADC SPLO      ;AND ADD TO INIT.
C193 85 B0                   STA SPLO
```

```
C195 A9 00              LDA #$00      ;ADD IN ANY CARRY
C197 65 B1              ADC SPHI
C199 85 B1              STA SPHI
C19B AD 3E 03           LDA Y1LO      ;ROW=INT(Y/8)
C19E 4A                 LSR A
C19F 4A                 LSR A
C1A0 4A                 LSR A
C1A1 A8                 TAY           ;GIVES INDEX
C1A2 C0 19              CPY #25       ;DISALLOW OUTSIDE
C1A4 10 2C              BPL RETP      ;GRAPHICS RANGE
C1A6 B9 00 C2           LDA COLTAB,Y  ;GET LO OFFSET
C1A9 18                 CLC
C1AA 65 B0              ADC SPLO      ;ADD TO LO 3 OF Y
C1AC 85 B0              STA SPLO      ;AND INITIAL POINT
C1AE B9 19 C2           LDA ROWTAB,Y  ;GET HI OFFSET
C1B1 65 B1              ADC SPHI
C1B3 85 B1              STA SPHI
                 ;
C1B5 AD 48 03    DETMOD LDA MODE      ;MODE 0,1,2
C1B8 F0 19              BEQ ANDBIT    ;CLEAR WITH AND
C1BA C9 02              CMP #2
C1BC F0 29              BEQ XORBIT    ;CLR OR SET?
C1BE C9 01              CMP #1
C1C0 D0 10              BNE RETP      ;BAD VALUE
                 ;
C1C2 98          SETBIT TYA           ;SAVE Y
C1C3 48                 PHA
C1C4 AC 32 C2           LDY LINE      ;INDEX
C1C7 B9 F8 C1           LDA BITTAB,Y  ;BIT VALUES
C1CA A0 00              LDY #$00
C1CC 11 B0              ORA (SPLO),Y  ;SET SPEC. BIT
C1CE 91 B0              STA (SPLO),Y
C1D0 68                 PLA           ;RESTORE Y
C1D1 A8                 TAY
C1D2 60          RETP   RTS
                 ;
C1D3 98          ANDBIT TYA
C1D4 48                 PHA
C1D5 AC 32 C2           LDY LINE
C1D8 A9 FF              LDA #$FF      ;USE RECIPROCAL
C1DA 38                 SEC
C1DB F9 F8 C1           SBC BITTAB,Y  ;OF SET FUNCTIONS
C1DE A0 00              LDY #0
C1E0 31 B0              AND (SPLO),Y
C1E2 91 B0              STA (SPLO),Y
C1E4 68                 PLA
C1E5 A8                 TAY
C1E6 60                 RTS
                 ;
C1E7 98          XORBIT TYA           ;XOR WILL ALLOW
C1E8 48                 PHA           ;WRITING AND
C1E9 AC 32 C2           LDY LINE      ;ERASING OVER
C1EC B9 F8 C1           LDA BITTAB,Y  ;OTHER GRAPHIC
C1EF A0 00              LDY #$00      ;VALUES
C1F1 51 B0              EOR (SPLO),Y
C1F3 91 B0              STA (SPLO),Y
C1F5 68                 PLA
C1F6 A8                 TAY
C1F7 60                 RTS
                 ;
                 ; TABLE OF BIT VALUES TO SET IN A
                 ; BYTE INDEXED BYT VALUE FOUND IN LINE
                 ;
C1F8 80 40 20    BITTAB BYT $80,$40,$20
C1FB 10 08 04           BYT $10,$08,$04
C1FE 02 01              BYT $02,$01
                 ;
                 ; LO BYTE VALUES SCREEN ADDRESSES
                 ; TOP TO BOTTOM ASSUMING STARTING
                 ; ON AN EVEN BOUNDARY
                 ;
```

Line 170	Initial values of Radius, center X and Y.
Line 180-250	Draws the four sets of circles.
Line 260	Kills some time, changes background color and starts over again.
Line 280	If CX is > 255 then make low value -255 and sets hi X to 1.
Line 290	Poke the Center value of Circle to area for the machine code to use. Set the Radius and draw the circle.
Line 310	Resets the screen to normal LoRes mode and quits. GOTO 310 after a break to reset.
Line 320	Call screen clear routine.

Three parameter are necessary to draw the circle:
1) A Center X lo and Center X hi (0-320). CL and CH in Demo.
2) A Center Y lo value. (0-200). CY in Demo.
3) A radius (0-255). R in Demo.

The circle will wrap around on the X axis and will clip at Y greater than 200 or less than 0 on the Y axis. Funny things happen if the Y value exceeds a value or 200, so the routine will clip for you.

I have included the assembly language source for assembly buffs and for added explanation of the theory. All the routines with the exception of PLOTXY should be adaptable to any machine with HiRess capabilities.

CIRCLE is the master routine. It squares the Radius and saves it for the remaining computations, and plots the first four dots. At LOOP DY is incremented and checked if > = DX, if not the next four points are computed and plotted. When the test passes, LOOP1 swaps DX and DY. The plot direction here is from vertical axis, right and left. When DX becomes = DY, the circle is complete and a return is made.

COMPXY does the adding and subtracting of DX and DY from the center point. After each quadrant is computed, the new X and Y values are set to on by calling the plotting routine.

COMLEG finds the unknown value DX using the Pythagorean formula, the Radius squared is computed in CIRCLE.

MULT8 is an 8 bit multiply routine. An 8 bit multiply was chosen due to speed, and anything over 255 would be out of range of most screen displays, since this would only be half of the total in the Circle.

SORT returns an 8 bit square root of the unknown leg of the right triangle. Final value is rounded towards the integer value the remainder is closest to.

PLOTXY is the machine dependent routine made to work on the Commodore 64's HiRes screen. Basically it uses the formula from the Programmer's Reference for setting a bit on the HiRes screen. Where it deviates is the final way it determines the byte on the screen. The mode of plotting the bit is determined from the value in The Globl MODE. The bit can be set with an OR, cleared with an AND or toggled with an XOR. The XOR will allow an object to be drawn on top of another and then erased, leaving the object underneath undisturbed. However, the XOR doesn't work very well on the circle, due to an occasional overlap of bits at the meeting point of the circle halves. Look over this routine as it can be used to plot a bit at X and Y from any kind of function (circle, line, rectangle, etc.).

CLEAR clears the HiRes screen and sets screen color to the value found at Address 02, poked here by the Basic Demo.

MICRO™

```
C2ØØ ØØ 4Ø 8Ø   COLTAB   BYT $Ø,$4Ø,$8Ø
C2Ø3 CØ ØØ 4Ø            BYT $CØ,$Ø,$4Ø
C2Ø6 8Ø CØ ØØ            BYT $8Ø,$CØ,$Ø
C2Ø9 4Ø 8Ø CØ            BYT $4Ø,$8Ø,$CØ
C2ØC ØØ 4Ø 8Ø            BYT $Ø,$4Ø,$8Ø
C2ØF CØ ØØ 4Ø            BYT $CØ,$Ø,$4Ø
C212 8Ø CØ ØØ            BYT $8Ø,$CØ,$Ø
C215 4Ø 8Ø CØ            BYT $4Ø,$8Ø,$CØ,$Ø
                ;
                ; HI BYTE VALUES
                ; TABLE ASSUMES HIRES STARTS ' $2ØØØ
                ;
C219 2Ø 21 22   ROWTAB   BYT $2Ø,$21,$22,$23,$25
C21E 26 27 28            BYT $26,$27,$28,$2A
C222 2B 2C 2D            BYT $2B,$2C,$2D,$2F
C226 3Ø 31 32            BYT $3Ø,$31,$32,$34
C22A 35 36 37            BYT $35,$36,$37,$39
C22E 3A 3B 3C            BYT $3A,$3B,$3C,$3E
                ;
C232 ØØ         LINE     BYT Ø          ;LO 3 BITS
                ;
                ;CLEAR : CLEAR HIRES SCREEN ' $2ØØØ
                ;
C233 A9 2Ø      CLEAR    LDA #$2Ø       ;NUMBER OF PAGES
C235 AA                  TAX            ;SET UP SCREEN
C236 85 B1               STA SPHI       ; ADDRESS
C238 A9 ØØ               LDA #$ØØ
C23A 85 BØ               STA SPLO
C23C AØ ØØ      CLR      LDY #$ØØ
C23E 91 BØ      CLR1     STA (SPLO),Y
C24Ø C8                  INY
C241 DØ FB               BNE CLR1
C243 E6 B1               INC SPHI
C245 CA                  DEX            ;DO 2Ø PAGES
C246 DØ F4               BNE CLR
C248 A5 Ø2               LDA $Ø2        ;VALUE POKED IN
                                        ; FROM BASIC
C24A 9D ØØ Ø4   COLOR    STA $Ø4ØØ,X    ;FIRST PAGE OF
C24D 9D ØØ Ø5            STA $Ø5ØØ,X    ; LO RES SCREEN
C25Ø 9D ØØ Ø6            STA $Ø6ØØ,X
C253 9D ØØ Ø7            STA $Ø7ØØ,X
C256 CA                  DEX
C257 DØ F1               BNE COLOR
C259 6Ø                  RTS
C25A                     END
```

```
1ØØ REM — CIRCLE DEMO —
11Ø REM — CIRCLE ROUTINE RESIDENT —
12Ø REM — @ $CØØØ
13Ø TS=828:CL=TS+8:CH=TS+9:RAD=TS+11:MODE=TS+12
14Ø V=53248:CY=TS+1Ø:POKE 2,1
15Ø POKE V+17,59:POKE V+24,24:GOSUB 32Ø
16Ø POKE MODE,1
17Ø R=4Ø:CX=1ØØ:Y=1ØØ
18Ø FOR I=1 TO 12:C1=Ø
19Ø GOSUB 28Ø:CX=CX+5:R=R-3:NEXT:C2=CX:R1=R
2ØØ CX=CX+5:FOR I=1 TO 12:C1=Ø
21Ø GOSUB 28Ø:CX=CX+5:R=R+3:NEXT
22Ø R=R1:CX=C2:FOR I=1 TO 12:C1=Ø
23Ø GOSUB 28Ø:Y=Y-5:R=R+3:NEXT
24Ø Y=1ØØ:R=R1:CX=C2:FOR I=1 TO 12:C1=Ø
25Ø GOSUB 28Ø:Y=Y+5:R=R+3:NEXT
26Ø GOSUB 33Ø:A=A+1:IF A> 31 THEN A=1
27Ø POKE 2,A:GOSUB 32Ø:GOTO 17Ø
28Ø CS=CX:IF CX> 255 THEN CX=CX-255:C1=1
29Ø POKE CL,CX:POKE CH,C1:POKE CY,Y:POKE RAD,R:SYS 49152:
    CX=CS:RETURN
31Ø POKE V+17,27:POKE V+24,21:END
32Ø SYS 49715:RETURN
33Ø FOR T=1 TO 3ØØØ:NEXT T:RETURN
```

Graphicom and the Koalapad

by John Steiner

Chicago Rainbowfest

Over a year has gone by since the first Color Computer only show, Rainbowfest. Since that first show in Chicago, there have been several around the country, most have been too far away for me to attend. I am looking forward to traveling to Chicago again for the next Rainbowfest.

At the last show, I enjoyed meeting many of the people who have made the Color Computer one of the most expandable and usable computers on the market. Also, many people who have written powerful software were in attendance. This show should be no different; if you can attend, please look for me and say hello.

Graphicom and the Koalapad

This month, I must comment in more detail about one of the best graphic oriented programs I have seen for the Color Computer, Graphicom. Yes, Graphicom is fun for the kids to play with and also interesting, but don't dismiss it as another toy program. For example, I have two practical and useful applications. I use it to create logos and designs for my company products. In addition, I use it to draw and print schematic diagrams. There are many other applications that relate to graphics in a practical business and personal sense.

Drawing with Graphicom requires a single joystick and two fire buttons. One option, however, is to use a Koalapad, modified to fit the Color Computer. For those of you who may be unaware, the Koalapad is a small drawing tablet that plugs into the joystick port of several different types of computers. There are versions for the Apple, Atari, Commodore, IBM PC, and other personal computers, and it comes withth software that allows the use of this sophisticated digitizer.

Koala Industries, however, has not seen fit to make a version of the Koalapad for the CoCo. The enterprising people at Cheshire Cat Software (creators of Graphicom) have included modification instructions to enable the use of the Koalapad with their software. After following these instructions, I found the pad to be a useful tool for other joystick applications as well. Essentially, the pad is an unusual joystick. If nothing is being pressed on the face of the pad, the joystick port returns coordinates of 32,32 (the joystick is centered). If you use a finger or other object to press on the face of the pad, the joystick port reports the coordinates of the location of the pressure on the pad. Moving the finger, or the wood "pencil" that comes with the pad, will cause the joystick coordinates to change in relation to the new location. The result of all this is that the modified Koalapad can be used anywhere you can use a standard joystick.

This new application of a joystick intrigued me, and I have found other joystick software that can use the Koalapad to better advantage than a standard joystick. It occurred to me that other people might be interested in using the Koalapad for use with Graphicom, or for other purposes. I contacted Bob Rosen of Spectrum Projects, publisher of the Graphicom program, and he gave me permission to pass along the modification instructions to you.

The modification instructions are for the Atari version of the Koalapad. I don't know how much difference there is between versions, so you might be sure to get the Atari version. The pad retails for around $100.00, but I have seen them on sale for less than $80.00. In addition to the pad, you will need a six conductor cable, two 1 Megohm resistors, and one or two din plugs that fit the joystick port. A 9 to 12 volt supply is also required.

Figure one contains a circuit board layout of the pad. It is easy to interpret the drawing, once you take the screws out of the bottom of the pad. By the way, there is one screw underneath the label that is stuck to the bottom of the pad. Removing this screw will void your warranty on the pad, so you might want to have the store you purchased the pad from check the pad to make sure it is a working unit before you take it apart.

From the diagram in figure 1, the six wires are connected as follows:

Step 1 to pin 1 of the right joystick din plug.
Step 2 to pin 2 of the right joystick din plug.
Step 3 to pin 4 of the left joystick din plug. (See next paragraph).
Step 4 to pin 4 of the right joystick din plug.
Step 5 to pin 3 of the right joystick din plug and minus of the 8.3 volt supply.
Step 6 to positive of the 8.3 volt supply.

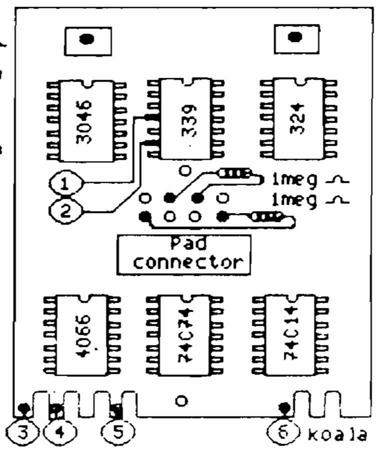
Figure 1

Modification of Atari Koala Pad for CoCo joystick Port of TRS80 COLOR COMPUTOR.

Merely add 6 wires to the board, solder in two resistors and add 8.3 V power supply.

1) Hook up X axis to pin 4 of 339 chip.
2) hook up Y axis to pin 6 of 339 chip.
3) Connect left fire button.
4) Connect right fire button.
5) Connect to ground
6) Connect 8.3 volts

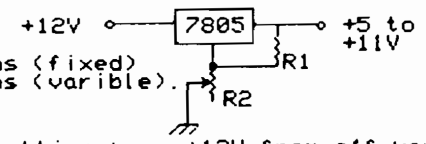
Add two 1 meg ohm resistors as shown



The Koalapad has two "fire" buttons on the top of the pad. The right joystick and fire button connections are hooked to a single din plug. The left fire button is connected to the other din plug for use with Graphicom. I preferred to have only one fire button hooked up to the pad, thus allowing me to have a standard joystick, or remote footswitch in the left joystick port. With Graphicom, the left joystick is not used, only the left fire button. If you are using the tablet with other software, you may want the flexibility of having a joystick and Koalapad in either port at the same time.

Figure two is a schematic of a simple 8.3 volt regulator that is used to obtain power for the Koalapad. The manual states that the 8.3 volts there is quite critical, so they recommend regulating it. Because I was in a hurry to see how it worked, and had an old nine volt AC power supply sitting around (one of those that contain a small transformer that plugs into the wall, and a small cord that ran to a nine volt battery snap), I used it. I found that the load on the Koalapad pulled the 9 volt supply down to 8.45 volts. The pad seems to work fine. I would, however, follow their recommendations on regulating the supply, if you plan on heavy duty use of the pad.

This simple schematic shows how to use a 7805 voltage regulator as a **variable** voltage regulator that gets a +12 volt input, and yields a voltage tha can be adjusted to the apprximately +8.3 volts required by the CoCo Koala Pad mod.



R1 1000 ohms (fixed)
R2 1000 ohms (varible).

I suggest getting your +12V from off your Color Compter's motherboard, and running it to your pad via the center pin of the joystick connector. You'll have to disconnect this pin from ground and then connect it to +12V on your CoCo board.

Figure 2

Figures one and two were both created using Graphicom, by the people at Cheshire Cat Software, and are reprinted from page 32 of the Graphicom software manual by permission of Spectrum Projects. These two illustrations should give you an idea of the usefulness and power of the Graphicom software.

After these simple modifications, plug the pad into the right joystick port, and run the following test program.

```
10 CLS
20 A=JOYSTK(0):B=JOYSTK(1)
30 PRINT@224,A,B
40 GOTO 20
```

When you run the program, it should print 32 SPACES 32 on the screen, indicating the two values being read in from the right joystick port. Use the wood pencil to touch the very upper left hand corner of the pad. The numbers should change to 0,0. If you press on the lower right hand corner, it should return 63,63. Moving the stick on the pad should cause the numbers to change with respect to the position of the stick.

I have had a lot of fun with the pad, and pass this information along to those of you who like to experiment with hardware. The process is fairly simple. If you try the modification, and have any problems, you may give me a call in the evening at 701-281-0549. I will try to help. Have fun, and if you develop any software that uses the pad, let me know. The pad is a useful, and interesting accessory for the CoCo.

MICRO

microbes

A Note to Our Readers: In the last issue (Micro 72:26) we printed an article on a Better Random Number Generator. Due to problems with our typesetting equipment, when we transferred the text and program, all of the special symbols such as plus signs, equal signs, greater than, less than, etc. were missing. This was brought to our attention by the authors after the issue was already printed. To correct this problem, we are listing the appropriate changes for the text and reprinting the entire program (minus the hex listing, since it was correct). We are sorry for any inconvenience this may have caused and assure you that the problem has been rectified. Thanks.

In the text wherever R[I1], R[I2],..., R[IK], R[N1], etc. appear there should be a plus sign between the letters and numbers in the brackets — R[I+1], R[I+2], etc.

Page 28, 2nd para., should read R[I+1] = R[I] + 1

Page 29 , last para., should read (R[N]/m)

Page 31, under Combination of RNG's, 2nd para., should read RANDOM = XRAN [Y * 100]

Page 32, 1st para., should read RAN = USR(SELECT)

Page 32, 2nd column, 4th para., should read
(A+B) mod C = (A mod C + B mod C) mod C

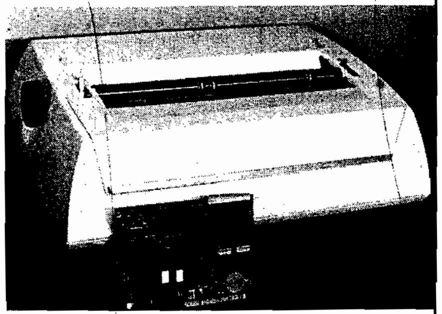
```
      12345  OLDRAN
    X 11111  MULT
     |12345
    1|2345
   12|345
  123|45
 1234|5
 ----------
 1371|65295  PROD
 *********************
 *                        *
 *  A BETTER RANDOM NUMBER GENERATOR *
 *          FOR APPLESOFT         *
 *                        *
 *         COPYRIGHT 1984         *
 *      THE COMPUTERIST INC.      *
 *      ALL RIGHTS RESERVED       *
 *                        *

**********************
;
; TO USE THE RNG SUBROUTINE, YOU MUST
; SET UP THE USR FUNCTION.
; SEE EDITORIAL NOTE
;
;  LOAD IN PARAMETERS FOR THE RNG'S
;
; Z: RAN=(31415938565*OLD+246Ø7)MOD2Ø
;
 ZADD     BYT $ØØ,$ØØ,$ØØ,$67,$27
 ZMULT    BYT $Ø7,$5Ø,$89,$2E,$Ø5
 ZRAN     BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ
; Y: RAN=(84134532Ø5*OLD+99991)MOD2Ø
;
 YADD     BYT $ØØ,$ØØ,$Ø1,$86,$97
 YMULT    BYT $Ø1,$F5,$7B,$1B,$95
 YRAN     BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ
; X: RAN = (27182819621*OLD+3)MOD2Ø
;
 XADD     BYT $ØØ,$ØØ,$ØØ,$ØØ,$Ø3
 XMULT    BYT $Ø6,$54,$38,$E9,$25
 XRAN     BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ
; ADD LOOKUP TO BASE LOCS FOR
; PARAMETER ADDRESSES FOR CURRENT RNG.
 LOOKUP   BYT $Ø4,$13,$22      ; Z, Y, X
;
 XYORZ    BYT $ØØ         ; WHICH GENERATOR
 YTEMP    BYT $ØØ         ; Y-REG ON ENTRY
 XTEMP    BYT $ØØ         ; X-REG ON ENTRY
 MULT     BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ
 OLDRAN   BYT $ØØ,$ØØ,$ØØ,$ØØ,$ØØ
;
 RNG      PHP             ; SAVE EVERYTHING
          STX XTEMP
          STY YTEMP
          JSR SIGN        ; SEE EDITOR'S NOTE FOR
                          ; SIGN ROUTINE
                          ; FAC HOLDS S OF USR(S)
;                           PUT FF IN A IF S<Ø,
;                           PUT Ø IF Ø, 1 IF S>Ø
          TAX             ; FROM THIS
          INX             ; DECIDE WHICH RNG
          LDY LOOKUP,X    ; VIA LOOKUP TABLE AND
          STY XYORZ       ; SAVE IT FOR LATER
;
; NOW THAT WE KNOW WHICH GENERATOR, MOVE
; ITS CONSTANTS TO THE TEMP LOCS.
;
          LDX #$Ø4        ; LOOP TO TRANSFER
 TRNSFR   LDA ADDBAS,Y    ; RNG'S VALS TO
;                           STANDARD LOCS, I.E.
          STA NEWRAN,X    ; ADD CONST TO NEWRAN,
          LDA MULBAS,Y    ; MULT CONST
          STA MULT,X      ; TO MULT,
          LDA LSTBAS,Y    ; LAST RND VAL FROM
          STA OLDRAN,X    ; THIS RNG TO OLDRAN
          DEY
          DEX             ; 5 BYTES DONE
```

```
          BPL TRNSFR      ; IF NO, DO NEXT
;                           IF YES, MULTIPLY.
          LDX #$Ø4        ; INDEX # OF BYTES
          STX BYTCNT      ; KEEP TRACK OF # BYTES
;                           DEALT WITH SO FAR
 NXTBYT   LDA MULT,X      ; LEAST SIGNIF BYTE
          STA MULTMP
          LDY #$Ø7        ; COUNT # BITS
 MULPLY   LSR MULTMP      ; GET LEAST SIG BIT.
          BCC SHIFT       ; BIT=Ø DON'T ADD.
          CLC             ; BIT SET, SO ADD
 ADD      LDA OLDRAN,X    ; OLDRAN TO NEWRAN.
          ADC NEWRAN,X
          STA NEWRAN,X
          DEX             ; ALL BYTES DONE
          BPL ADD         ; NO ADD NEXT
          CLC             ; YES, SO PREPARE TO
;                           SHIFT OLDRAN (IE
;                           MULT * 2). DROP LAST
;                           CARRY AS IT IS
;                           Ø MOD2Ø ANYWAY.
 SHIFT    LDX BYTCNT      ; # BYTES TO SHIFT
 SHFTIT   ROL OLDRAN,X
          DEX             ; BYTE LEFT
          BPL SHFTIT      ; YES, SHIFT IT.
          LDX BYTCNT      ; RECOVER # BYTES.
          DEY             ; MORE BITS LEFT
;                           IN THIS BYTE
          BPL MULPLY      ; YES, MULT BY NEXT.
          DEC BYTCNT      ; NO, DONE A BYTE.
          LDX BYTCNT      ; ANY BYTES LEFT
          BPL NXTBYT      ; YES MULT BY IT.
;
          LDY XYORZ       ; DONE. PUT THE
          LDX #$Ø4        ; NEW RND INTO THE
 MOVRAN   LDA NEWRAN,X    ; RESPECTIVE RNG'S
          STA LSTBAS,Y    ; LAST RAN STORAGE.
          DEY
          DEX             ; MORE TO MOVE
          BPL MOVRAN      ; YES, DO.
;
; DONE. NOW TO NORMALIZE FAC, ALIAS NEWRAN.
;
          LDY #$28        ; $28 (4Ø) BITS IN FAC.
 NRMLIZ   LDA NEWRAN      ; FIND HIGHEST SET.
          ROL             ; # SIGNIFICANT =
;                           28 - # NOT SET
          BCS BITSET      ; LEAVE WHEN TOP BIT FOUND
          ROL NEWRAN+4    ; NOT FOUND YET, SO
          ROL NEWRAN+3    ; GET RID OF THE Ø
          ROL NEWRAN+2    ; BIT AT THE TOP.
          ROL NEWRAN+1    ; Y WILL KEEP TRACK
          ROL NEWRAN      ; OF # OF BITS LEFT.
          DEY             ; ANY LEFT
          BNE NRMLIZ      ; YES, KEEP LOOKING
;                           NO, ALL DONE.
          DEY             ; PROTECT AGAINST
;                           DIVIDE BY Ø.
 BITSET   LDA #$ØØ        ; PUT Ø IN FAC'S
          STA NEWRAN+4    ; SIGN BYTE.
          TYA             ; GET # SIG BITS
          CLC             ; PUT IN FAC'S +$8Ø
          ADC #$58        ; FORMAT: $58+$28=$8Ø.
          STA RANEXP      ; PUT IN EXPONENT
;                           BYTE AND DONE.
          LDY YTEMP       ; SO, UNSAVE
          LDX XTEMP       ; EVERYTHING
          PLP             ; AND
          RTS             ; SAY GOODBYE.
          END
```

Name:	**Printerface Intelligent Interface**
Hardware:	Printers: Diablo Hytype I, Hytype II, DEC LQP-01, Xerox

Description: This unique printer interface board is installed in the printer rather than the computer, and upgrades an older printer to perform like the best Daisy Wheel printers. Model DT150 and DT151A intelligent interfaces snap into place without modifying the printer and provide all standard configurations, including RS232 serial, Centronics parallel, IEEE488, and Current loop.

Features include automatic bidirectional printing, microspace, proportional spacing, bold facing, auto centering, variable pitch, self test and debug modes. Accessories available include a 16K buffer memory and a front control panel for 16 functions.



Price:	$395.00
Contact:	Kuzara International 7770 Vickers, Suite 105 San Diego, CA 92111 619/569-9107

Name:	**MasterType**
Hardware:	Apple, Atari, Commodore-64

Description: "Mastertype" is the best-selling educational software program, having sold over 150,000. It teaches typing and keyboard skills through an exciting arcade game format, and is now the first software program designed to teach Dvorak keyboard skills on the Apple IIc. The new version has been enhanced with HiRes graphics, scoring retention, and, in addition to the 18 lessons on the standard QWERTY keyboard, five lessons on the Dvorak keyboard.

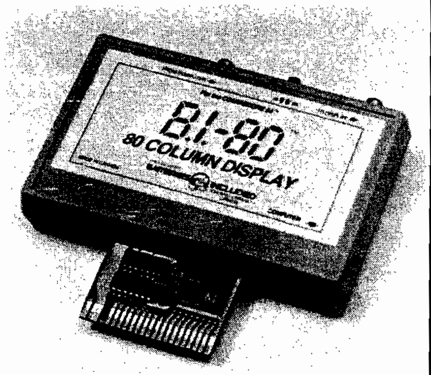
The Dvorak keyboard increases speed and comfort because the most frequently used keys are placed on the "home row" beneath the typists strongest fingers. It is beginning to gain wide acceptence.

Price:	$39.95
Contact:	Scarborough Systems 25 North Boardway Tarrytown, NY 10591 914/332-4545

Name:	**B.I.-80 Column Adaptor**
System:	Commodore-64

Description: A high-quality 80 column plug-in module that eliminates the problems of snow, fuzziness, hashing and interference. It gives optimum clarity, even with a full screen of characters, and can easily switch from 40 to 80 column display at any time.

B.I.-80 can be used with Commodore color monitors 1701 and 1703, or with any monochrome video monitor. It is self-initializing, with complete 80 column operating system and BASIC 4.0 language built in. Comes with one year warranty, and full documentation, including a description of the BASIC 4.0 language.



Price:	$
Contact:	Batteries Included 186 Queen Street West Toronto, ON m5v 1z1 Canada 416/596-1405

Name:	**Decisions**
System:	Atari
Memory:	48K

Description: A new program that provides assistance on making a logical choice among several alternatives, for both home and business use. The program is flexible enough to analyze any multiple choice decision. Features such as fully prompted inputs, help screens, rapid re-analysis and thorough reference manual make it easy to use. Graphic output screens are easily interpreted and a hard copy record is provided to users with an 80-column printer.

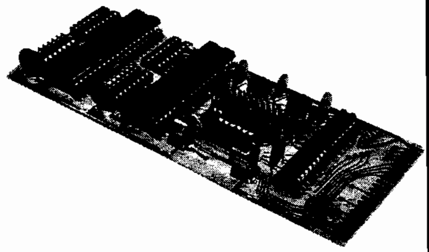
The program uses logical analysis based on scientific principles. It is available either on 5 1/4" disk or cassette tape. Available at some dealers or by mail order.

Price:	$37.50
Contact:	Lateral Software P.O. Box 605 Stanton, CA 90680 714/826-3970

Name: **Interface Adapter Board**
System: Commodore 64

Description: The 6522 VIA (Versatile Interface Adapter) input/output chip interface adpater board allows 6522 programming techniques, covered in many available books, to be applied to the C-64 for real-time control applications. It allows full use of the IRQ interrupt and, when combined with the C-64's memory capacity, provides a powerful development system and controller in one package. Extensive application notes and programming examples are included.

Each board includes two 6522s, with total of four 8-bit bidirectional I/O ports, eight handshake lines, four 16-bit timer/counters. Up to four Model 64IF22 boards can be connected, providing 16 8-bit ports.



Price: $169.00 for first; $149 for each extra
Contact: Schnedler Systems
1501 N. Ivanhoe,
Dept. NR
Arlington, VA 22205
703/237-4796

Name: **Language Development Software**
System: Apple II/IIe (Atari coming soon)
Hardware: One disk drive

Description: Currently available languages in this product line include Spanish, French, German, Italian, Biblical Hebrew, Modern Hebrew and Arabic. In the near future, Latin, Russian, Polish, Swedish, and Classical Greek will also be available. All programs teach 1000 of the most common words in the target language. When words have more than one meaning, the program allows for these other meanings, along with English translation. A "Teach Yourself Book"is included in the package for additional information.

Each language program is menu-driven with sequential review, random review and quiz options. The software gives instant feedback, tests, and percentage of correctness through interactive learning.

Price: $56.95
Contact: Soflight Software
2223 Encinal Station
Sunnyvale, CA 94087
408/735-0871

Name: **Apple SourceLink**
System: Apple II, IIe, II Plus
Memory: Minimum 48K

Description: Communications software designed to supplement the use of The Source by personal computer owners. It is compatible with the new Apple modem, as well as the Hayes and Transend modem products. The software includes automatic dial-up and sign-on procedure for Telenet, Uninet and Sourcenet networks, simultaneous capture of data from The Source into the Apple memory or disks, including a capture editor and simplified transfer of data from disks to The Source. An additional feature allows Apple and IBM users to automatically access any number of pre-determined services and databases once online.

Contact: The Source
1616 Anderson Road
McLean VA 22102
703/734-7500

Name: **ScreenShooter**
Hardware: CRT

Description: A simple way to take photos and slides of a computer CRT using Polaroid 600 High Speed color film, Polachrome 35mm instant slide film, or conventional 35mm color or black and white films. The outfit includes a Polaroid One Step 600 Camera, CRT hood, CRT hood adapter, diopter lens and 35mm SLR camera bracket.

When using the Polaroid One Step, camera exposure is automatic. You place the Screenshooter against the computer screen, view the image through the camera and click away. When using a 35mm SLR camera, the camera's built-in metering system is used to find the exposure. Screenshooter comes with a lifetime warranty (the camera has a one year warranty).

Price: $169.00
Contact: NPC Photo Division
1238 Chestnut Street
Newton, MA 02164
617/969-4522

Name: **Bug Off!**
System: Apple II or IIe
Memory: 64K
Language: Pascal 1.1 or 1.2

Description: A powerful tool that saves time in testing and debugging Apple II Pascal programs. The easily installed package runs at near-execution speed and is totally interactive. The command screen gives you complete control and lets you build and use your own macros. Stored debugging commands let you start where you left off and you can insert breakpoints wherever you want them.

This package comes with a guarantee of total refund if you are not satisfied and return it within 30 days of shipment.

Price: $49.96
Contact: First Byte
2845 Temple Avenue
LongBeach, CA 90806
213/595-7006

Name: **Fit and Trim**
System: Apple II/IIe
Memory: 64K RAM
Hardware: 1-2 disk drives, printer optional

Description: This educational and counseling program for weight control features two units. The first Educational unit provides general information on eating and activity changes needed for weight loss, suggesting goals for aerobic, muscle building and other activities. The Counseling unit has Weight Review (projections, current weight and change progress displays), Eating Review (analysis of food you eat, showing calories and problem foods with recommended changes), and Exercise Review (analysis of activities with weekly exercise suggestions).

Five week histories can be summarized and recommendations for weight change can be printed. Capacity is 80 individuals per diskette. The program can be copied and is modifiable.

Price: $39.95
Contact: Andent, Inc
1000 North Avenue
Waukegan, IL 60085
312/223-5077

Name: **SpellPack**
System: Commodore-64

Description: This powerful program teaches your C-64 to spell and checks an entire document in 2-4 minutes. It contains a dictionary of over 20,000 of the most commonly used English words, and allows you to expand this by 5,000 specialized terms.

Each word is compared to the dictionary and those not found are highlighted in context, right on the screen. If the word is misspelled, it can be edited and instantly added to the dictionary. If it is correct but not listed, it can also be added immediately. It accelerates the page rate of checking so that a one page document may take two minutes to check, but a five pager may only take three minutes. Additions and corrections are made with single key command. SpellPack works with most major word processing programs.

Price: $
Contact: Batteries Included
186 Queen Street West
Toronto, ON m5v 1z1
Canada
416/596-1405

Name: **4 in 1**
System: Apple

Description: An enhanced database management system that simplifies record-keeping at home or business by handling four separate functions: word processing, list and label making, calculations and data management.

Major data processing operations are combined in a single program so there is no need to change disks mid-project. For example, 4 in 1 can perform calculations on defined fields, then merge those fields plus the results into forms or letters created with the word processor. Current tab stops and margin settings are indicated onscreen, as are menu options, prompt messages and system operating messages.

Price: $129.95
Contact: Softsmith Corp.
1431 Doolittle Drive
San Leandro, CA 94577
415/487-5900

Name: **Digital TLC-1**
Hardware: Any RS232 devices

Description: This is a three port active switch that lets any two RS232 devices share a third and also communicate with each other. Any transmission format at any rate up to 19,200 baud can be accommodated and all connections are made via a six button control panel with out switching transients.

Proper connection between the transmitted and received data pins is fully resolved with the TLC-1 for any combination of Data Communication Equipment and Data Terminal Equipment. Permitting 64 possible connection combinations, all data paths are monitored by six LEDs.

Price: $245
Contact: Digital Laboratories, Inc.
600 Pleasant St.
Watertown, MA 02172
617/924-1680

Name: **Intec 300 Modem**
System: Apple II/IIPlus/IIe, TRS80 Model 3/4, IBM PC

Description: A new auto dial/auto answer modem featuring software and essential phone-computer interface connections to function with several computers. Also provided is easy to follow, detailed documentation.

Features include data capture direct to disk file as well as memory buffer, 255 number auto-dialing telephone directory with auto redial of last phone number, non-ASCII file transfer, optional add/delete of linefeeds, transmission of true break signal, and many more.

Price: $189.00
Contact: Intec Corp.
West Bloomfield, MI

Title: **The RS-232 Solution**
Author: Joe Campbell
Price: $16.95
Publisher: Sybex Computer Books

The problem of interfacing your computer with any RS-232-C peripheral is covered in this book. Using tools that total less than $15.00, the reader is instructed how to measure logic levels and conduct other tests. The results of these tests are then taken to derive a specification for a cable, thus making the correct connections. There are ample diagrams and illustrations explaining the basics and beyond, of serial interfacing. The author's 'fool-proof' method is illustrated with real case studies. Case studies include SB80/ADDS, N*/OKI, KayPro/Epson, Osb/TnT, and IBM/NEC. In addition to printers, the interfacing of modems, terminals, and plotters is also explained.

Title: **The Elements of Friendly Software Design**
Author: Paul Heckel
Price: $8.95
Publisher: Warner Books

Taking the approach that software is a communication craft, the author draws upon a variety of innovators in this area. Citing such greats as Walt Disney, George Orwell and Leonardo Da Vinci, the idea of visuality and clear communication in software is emphasized. All of the elements of friendly software design are covered from the perspective of both the user and designer. Attention is given to what the user expects, perceives, feels and thinks; all lending to a better understanding and foundation from which to design software. Prototypes and innovations are examined. Points are supported with a variety of pictures, illustrations, etc. Thirty principles of software design are given in addition to seven traps that catch experienced designers.

Title: **The BBC Microcomputer for Beginners**
Authors: Seamus Dunn & Valerie Morgan
Price: $13.95
Publisher: Prentice/Hall International

This book covers the in's and out's of the BBC Microcomputer, more popularly known as the Acorn; both models, A and B, are covered. In addition to noting the various characteristics and options available on the BBC microcomputer, programming in BASIC is also covered. In this vain, the book guides the reader in a learning by doing process. Carefully sequenced programs take the user through a variety of programming 'musts', including: conditionals, loops, file management, functions, strings, formatting, graphics, color, and sound. The approach is that of structured programming. The marriage of programming skills and knowledge of the machine are integral to the book as a whole. There are examples and sample programs to aid the reader in learning both the BBC microcomputer and structured programming using BASIC. At the end of each chapter are problems, happily at the back of the book answers are also provided.

Title: **Microprogrammers Market 1984**
Author: Marshall Hamilton
Price: $13.50
Publisher: Tab Books

Basically a sourcebook for programmers looking to sell their program ideas, this listing covers hundreds of companies. The information provided on each publisher includes: Company name, address, telephone number, president, submission contact, microcomputer systems covered, age of the company, company's publishing track record, what they are looking for, payment methods, how and when submissions should be handled, response time, current program sources, what types of programs are now being sold and how they are marketed. In addition the author provides a number of valuable tips on writing, submitting and selling. Listings are broken down into Business/Industry, Educational/Tutorial, Games, Home Use, and Utilities.

Title: **How to Make Love to A Computer**
Author: Dr. Maurice K. Byte
Price: $3.95
Publisher: Pocket Books

For those who are really into their computer this book is a must. Learn the heretofore unspoken secrets of how to make love to your computer. Every aspect is touched upon in this Kama Sutra of computer love making. From the first meeting to that special night together, all of the in's and out's of computer romance are examined. Sexual fears, tips from pros, computerotica, and the joy of programming are a few of the many areas this book covers. Complete with photographs, this is not a book for children.

Title: **The Illustrated dBase II Book**
Author: Russell A. Stultz
Price: $16.95
Publisher: Spectrum Books

A reference/tutorial for the popular dBase II software program by Ashton-Tate. The author uses modules to teach the reader how to use dBase II. With the aid of examples and illustrations the beginning programmer is guided through the world of database management. Descriptions of dBase II files, how they are stored, displayed, printed and edited are included. The experienced programmer will find that this can be used as a handy reference; educators will also find the concise text helpful. The modules are alphabetically organized, with a good index offering further reference support. All the reader needs is dBase II, and 8- or 16-bit microcomputer with at least 64K RAM, a disk drive, and a printer.

MICRO Program Listing Conventions

Commodore

LISTING	C64 KEYBOARD
Commands	
{CLEAR}	■ ^ CLR
{HOME}	■ HOME
{INSERT}	■ ^ INST
{DOWN}	■ CRSR DOWN
{UP}	■ ^ CRSR UP
{RIGHT}	■ CRSR RIGHT
{LEFT}	■ ^ CRSR LEFT
Colors	
{BLACK}	■ CTRL 1 BLK
{WHITE}	■ CTRL 2 WHT
{RED}	■ CTRL 3 RED
{CYN}	◣ CTRL 4 CYN
{PURPLE}	■ CTRL 5 PUR
{GREEN}	■ CTRL 6 GRN
{BLUE}	■ CTRL 7 BLU
{YELLOW}	■ CTRL 8 YEL
{RVS}	■ CTRL 9 RVS ON
{RVSOFF}	■ CTRL 0 RVS OFF
{ORANGE}	■ = 1
{BROWN}	■ = 2
{GREY 1}	■ = 3
{GREY 1}	■ = 4
{GREY 2}	■ = 5
{LT GREEN}	■ = 6
{LT BLUE}	■ = 7
{GREY 3}	■ = 8
Functions	
{F1}	■ f1
{F2}	■ ^ f2
{F3}	■ f3
{F4}	■ ^ f4
{F5}	■ f5
{F6}	■ ^ f6
{F7}	■ f7
{F8}	■ ^ f8
Special Characters	
{PI}	π ^ Pi Char
{POUND}	£ Pound Sign
{UP ARROW}	↑ Up Arrow
{BACK ARROW}	← Back Arrow

Atari

Conventions used in ATARI Listings.

Normal Alphanumeric appear as UPPER CASE:
SAMPLE
Reversed Alphanumeric appear as lower case:
yES (y is reversed)
Special Control Characters in quotes appear as:
{command} as follows:

Listing	Command	ATARI Keys
{UP}	Cursor Up	↑ ESC/CTRL -
{DOWN}	Cursor Down	↓ ESC/CTRL =
{LEFT}	Cursor Left	← ESC/CTRL +
{RIGHT}	Cursor Right	→ ESC/CTRL *
{CLEAR}	Clear Screen	↖ ESC/CLEAR
{BACK}	Back Space	◀ ESC/BACK S
{TAB}	Cursor to Tab	▶ ESC/TAB
{DELETE LINE}	Delete Line	■ ESC/SHIFT DELETE
{INSERT LINE}	Insert Line	■ ESC/SHIFT INSERT
{CLEAR TAB}	Clear Tab Stop	■ ESC/CTRL TAB
{SET TAB}	Set Tab Stop	■ ESC/SHIFT TAB
{BEEP}	Beep Speaker	■ ESC/CTRL 2
{DELETE}	Delete Char.	■ ESC/CTRL BACK S
{INSERT}	Insert Char.	■ ESC/CTRL INSERT
{CTRL A}	Graphic Char. where A is any Graphic Letter Key	▙ CTRL A

Non-Keyboard Commands

{DIS=}	CHR$(8)
{ENB=}	CHR$(9)
{LOWER CASE}	CHR$(14)
{UPPER CASE}	CHR$(142)
{^RETURN}	CHR$(142)
{DEL}	CHR$(20)
{SPACE}	CHR$(160)

Notes:

1. ^ represents SHIFT KEY
2. = represents Commodore key in lower left corner of keyboard
3. CTRL represents CTRL key
4. Graphics characters represented in Listing by keystrokes required to generate the character
5. A number directly after a {SYMBOL} indicates multiples of the SYMBOL: {DOWN6} would mean DOWN 6 times

Advertiser's Index

Andent, Inc.	77
Batteries Included	75,77
Computer Mail Order	Ins Frt Cvr,1
Computerose	Ins Back Cvr
Digital Laboratories	77
F. Ashton	18
First Byte	76
Indus-Tool	9
Intec Corp.	77
Kuzara International	75
Lateral Software	75
MICRO Magazine	64,67,Ins Back Cvr
NPC Photo Division	76
Protecto	39,40,41
Safeware	14
Scarborough Systems	75
Schnedler Systems	7,76
Soflight Software	76
Softsmith Corp.	77
Specialty Electronics	9
The Source	76
Totl Software	31
Transactor	3

Next Month In MICRO

Features

The UCSD P-System — This is a more powerful operating system than MS-DOS and the 8088, and, on a 68000 machine, a very fast one, too. Reviews of six 68000 machines are included.

Constructing 3-D Mazes — The program actually gives you rat's-eye views of the maze corridors — and all in 3 1/2K of RAM.

Graphics Print for C64 — The third part of this series adds a program that loads graphic files from a number of popular graphic programs, displays them and dumps them to a printer.

Atari/Epson Character Printing — The Atari puts a tremendous variety of graphic characters on screen; this program allows even custom characters to be put on paper.

Hilister — The second of a two-part series, this covers moving around within a program listing.

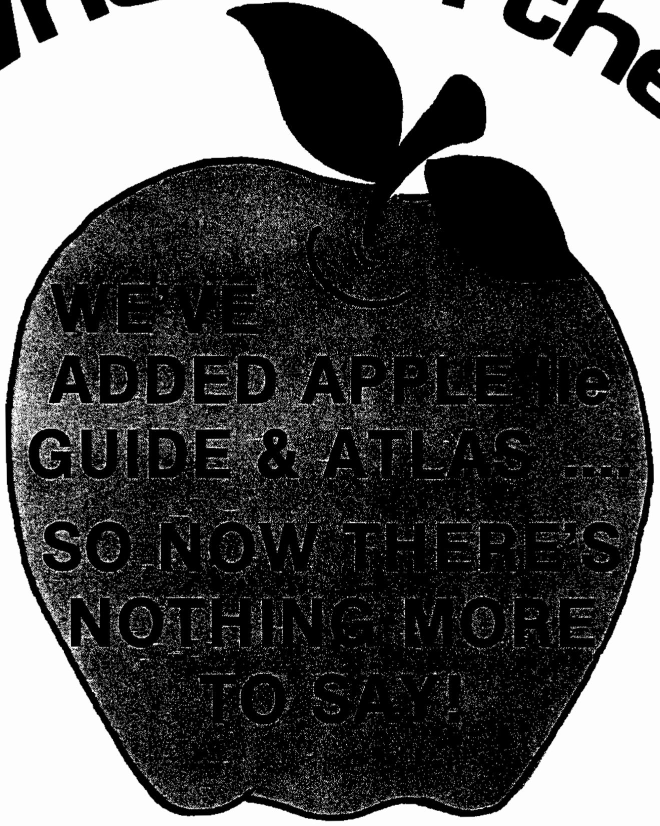
Alter T & S — Dump, in hex, any sector on a diskette with Commodore format and then modify any byte in that sector without the loss of other data.

Plus More...

Departments

Reviews in Brief
Spotlight
Software/Hardware Catalogs
New Publications
Interface Clinic
Lyte Bytes

What's Where in the APPLE



This famous book now contains the most comprehensive description of firmware and hardware ever published for *the whole Apple II family*. A new section with guide, atlas and gazeteer now provides Apple IIe specific information.

- Gives names and locations of various **Monitor, DOS, Integer BASIC** and **Applesoft** routines and tells what they're used for
- Lists **Peeks, Pokes** and **Calls** in over 2000 memory locations
- Allows easy movement between BASIC and Machine Language
- Explains how to use the information for easier, better, faster software writing

This expanded edition is available at the new low price of only $19.95

For the 35,000 people who already own previous editions, the IIe Appendix is available separately for just $5.00.

Please send me:

_______ What's Where in the Apple @ $19.95 ea. _______
(Plus $2.00 per copy shipping/handling)

_______ Apple *IIe* Appendix @ $5.00 ea. _______
(includes shipping charges)

Mass residents add 5% sales tax $ _______

Total Enclosed $ _______

For faster service
Phone 617/256-3649

____
Name

____
Address

____
City State Zip

Signature _______________________

☐ Check ☐ VISA ☐ MasterCard

Acct # _______________ Expires _________

MICRO, P.O. Box 6502, Chelmsford, MA 01824

Take an active role in your child's development.

Parenting. The most important and rewarding endeavor you'll ever undertake. Gaze into your child's eyes. They're capturing all the wonders of the world around him, and looking to you for guidance.

Now you can gain a unique insight into your child's world with Childpace™ — an amazing new Child Development Program for ages 3 to 60 months.

Share the precious firsts.

When will your baby dazzle you with his first spontaneous smile? Stand alone? Take that first wobbly step?

The first five years are filled with continual growth and change. And questions. So even if your child's a toddler, you're still looking for answers. When will he start dressing himself? When should those random scribbles turn into distinctive shapes?

Compare apples-to-apples.

Childpace lets you evaluate your child's dexterity, language and social skills in the privacy of your own home. You enter information into Childpace, then he attempts tasks that are appropriate for his age group.

Childpace assesses his skill level based on extensive research, not the biased opinions of friends or relatives. Childpace uses your child's chronological (actual) age.

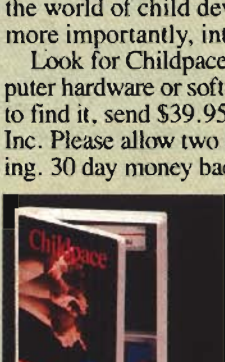
Grow with your child.

As your child grows, the tasks change to match his newly acquired skills. So Childpace is just as valuable for a 48-month old child as for an infant. Childpace can even evaluate up to 16 different children, and keep permanent records on each of them. Snapshots record your child's physical growth, but Childpace documents his or her actual development.

Track your child's progress, and help him develop specific skills. Childpace also contains warning signals to alert you to potential developmental problems at an early age, before they hold your child back. An ounce of prevention pays off.

Childpace. A fascinating glimpse into the world of child development. And more importantly, into *your child's* world.

Look for Childpace at your local computer hardware or software store. If unable to find it, send $39.95 to Computerose, Inc. Please allow two weeks for processing. 30 day money back guarantee.



$39.95 suggested retail price

Childpace is available for the Commodore 64,® IBM PC,® IBM PC Jr.,® Atari 800,® Apple II,® and Radio Shack Color Computer®*

*Each is a registered trademark of the respective manufacturer.

Computerose We're programming for life. TM

2012 East Randol Mill Road Suite 223
Arlington, TX 76011 (817) 277-9153