

APPLE ASSEMBLY LINES

DEC 87 - MAY 88

Apple

\$1.80



Assembly

Line

Volume 8 -- Issue 3

December, 1987

Peeking Inside AppleWorks 1.3	2
Subroutine Call Parameter Passage	2
String Handling Subroutines	2
Screen Dump PLUS!	13
It's 1988, and ProDOS Thinks It's 1982	24

About Back Issues...

Don't be bashful, just tell me what you need! I have copies of all back issues in stock, at \$1.80 each, \$18 per volume, or \$120 for all seven of the completed volumes. Remember, our volume-year runs from October through September, and I started in 1980.

I also have Quarterly Disks for the entire period, \$15 each or four for \$60. Each Quarterly Disk covers a calendar quarter, so just specify which months you need. Starting with January 1986 I went to Monthly Disks, so you can get any individual month for only \$5 from then till now. Each Quarterly or Monthly Disk contains all of the source code printed in the AAL issues it covers, in the format for the S-C Macro Assembler. Early ones are all DOS, later are split with both DOS and ProDOS directories on the same disk. And since sometime last year we have also been including all of the text files for the articles themselves, as a service to readers who like to have the Echo Speech Synthesizer (or other brands) read it all to them.

Toward a New Standard Assembly Language...

Randy Hyde, who you may remember as author of the Lisa 6502 Assembler, is attempting to organize interested parties to produce a new definitive 65816 assembly language standard. He claims the existing standard, based on Orca, is confusing, overly complex, and idiosyncratic; a new standard could allow assemblers with more power than the Microsoft 8086 assembler to be written for the 65816. Right now Randy is collecting ideas and contacting key individuals (such as the authors of the various existing 65816 assemblers and 65816 books, and the chip's designer), and planning for a conference at WDC in Arizona some time this summer. If you are interested in participating in any way, write to him: Randall Hyde, 65C816 Standards, 2271 Indian Horse Drive, Norco, CA 91760.

Peeking Inside AppleWorks 1.3
 + Subroutine Call Parameter Passage
 + String Handling Subroutines.....Bob Sander-Cederlof

There are a lot of useful subroutines inside AppleWorks. I have been looking at a raw disassembly, and have learned a few new tricks. Even though AppleWorks is ProDOS-based, the subroutines are general enough that you can use them in your own code in any operating system.

All my observations are based on version 1.3, as that is the only one I have. Meanwhile, Apple has moved on to version 2.0 and turned it all over to Claris. That is all right, because I am not proposing that we use the subroutines by loading AppleWorks and calling them; I am proposing that we copy the code or some modification of it into our own programs.

When you boot AppleWorks 1.3 the first thing it does is to copy the APLWORKS.SYSTEM image down from \$2000 to \$1000. I simply loaded it there from inside the S-C assembler with "BLOAD APLWORKS.SYSTEM,TSYS,A\$1000". Then I printed out a huge listing with the monitor's "L" command, and went to work with a pencil. I don't even know what section of AppleWorks I am looking at yet, but it is chock full of interesting code I can use.

The first thing I noticed was that a lot of code did not disassemble correctly: The "L" command went weird after a lot of JSR's. It seems the author liked to call subroutines with parameters in data form following the JSR, like ProDOS MLI calls. In most cases (all I could find) there are either two, four, or six bytes of parameters after these JSR's. The subroutines all call, in turn, on a magic little subroutine which copies the parameter bytes to a standard area in page zero, starting at \$9A. This GET.x.PARMS subroutine also updates the return address on the stack so that, when the parameterized call is completed, execution will resume after the parameter bytes.

The GET.x.PARMS subroutine is shown as I found it inside AppleWorks 1.3, in lines 1300-1580. I used the .PH \$18AD line to make it look exactly the same as the AppleWorks image. The code looked a little fluffy to me, so I wrote my own version (which is shorter and swifter); you can see it in lines 1830-2070.

The AppleWorks version is evidently one of the busiest pieces of code in the system. I say that, because the author chose to poll the keyboard inside GET.x.PARMS. Line 1550 calls the POLL.KEYBOARD subroutine, which I show with comments in lines 1600-1810. I left this out of my rendition of GET.x.PARMS, because I am building a little package of routines for my own use. I included the listing here because I thought you might like to see how it is done. Notice the buffer holds only ten characters, as written.

Notice that there are three entry points to the GET.x.PARMS subroutine. The first copies four bytes following the JSR to

S-C Macro Assembler Version 2.0	\$150
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners	\$50
ProDOS Upgrade Kit for Version 2.0 DOS owners	\$50
Source Code of S-C Macro 2.0 (DOS or ProDOS)	without source code \$30, with source code \$50	
S-C DisAssembler (ProDOS only)	without source code \$30, with source code \$50	
RAK-Ware DISASM (DOS only)	\$20
ProVIEW (ProDOS only)	\$49
Full Screen Editor for S-C Macro (with complete source code)	without source code \$20, with source code \$50	
S-C Cross Reference Utility	\$50
S-C Word Processor (with complete source code)	\$50
DP18 and DPPF, including complete source and object code	\$50
ES-CAPE (Extended S-C Applesoft Program Editor), including Version 2.0 With Source Code	\$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)	\$50
Bag of Tricks 2 (Quality Software)	(\$20.00)	\$45
S-C Documentor (complete commented source code of Applesoft ROMs)	\$50
Cross Assemblers for owners of S-C Macro Assembler	\$32.50 to \$70	each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051, 8085, 1802/4/5, PDP-11, GI1650/70, Mitsubishi 50740, others)	
Beagle Bros. Applesoft Compiler (ProDOS only)	(\$74.00) \$65 *
Copy II Plus (Central Point Software)	(\$30.00) \$30 *
AAL Quarterly Disks	each \$15, or any four for \$45

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Vinyl disk pages, 6"x8.5", hold two disks each	10 for \$6 *
Diskette Mailing Protectors (hold 1 or 2 disks)	40 units each or \$20 per 100
(Cardboard folders designed to fit 6"x9" Envelopes.)	6 units each
Envelopes for Diskette Mailers	
Sider 20 Meg Hard Disk, includes controller & software	(\$800) \$550 +
Sider 40 Meg Hard Disk, includes controller & software	(\$900) \$660 +
Minuteman 250 Uninterruptible Power Supply	(\$350) \$320 +
Minuteman 300 Uninterruptible Power Supply	(\$500) \$490 +
65802 Microprocessor, 4 MHz (Western Design Center)	(\$170) \$170 *
quikLoader EPROM System (SCRG)	(\$142.50) \$140 *
PROMGRAMER (SCRG)	
"Exploring the Apple IIgs"	Gary B. Little	(\$22.00) \$21 *
"Apple IIgs Technical Reference"	Michael Fischer	(\$19.00) \$19 *
"65816/65802 Assembly Language Programming"	Michael Fischer	(\$21.00) \$20 *
"Programming the 65816"	David Eyes & Ron Lichty	(\$22.00) \$21 *
"Apple //e Reference Manual"	Apple Computer	(\$24.00) \$23 *
"Apple //c Reference Manual"	Apple Computer	(\$24.00) \$23 *
"ProDOS-8 Technical Reference Manual"	Apple Computer	(\$23.00) \$27 *
"ProDOS-16 Technical Reference Manual"	Apple Computer	(\$24.00) \$27 *
"Apple IIgs Firmware Reference"	Apple Computer	(\$24.00) \$23 *
"Apple IIgs Hardware Reference"	Apple Computer	(\$24.00) \$23 *
"ProDOS Inside and Out"	Dennis Doms & Tom Weishaar	(\$16.00) \$16 *
"DOSTalk Scrapbook"	Tom Weishaar & Bert Kersey	(\$14.00) \$14 *
"Beneath Apple ProDOS"	Don Worth & Pieter Lechner	(\$19.00) \$18 *
"Beneath Apple DOS"	Don Worth & Pieter Lechner	(\$19.00) \$18 *
"Inside the Apple //c"	Gary B. Little	(\$19.00) \$18 *
"Inside the Apple //e"	Gary B. Little	(\$19.00) \$18 *
"Understanding the Apple //e"	Jim Sather	(\$24.00) \$23 *
"Understanding the Apple II"	Jim Sather	(\$22.00) \$21 *
"Apple II+/IIe Troubleshooting & Repair Guide"	Brenner	(\$19.00) \$18 *
"Assembly Language for Applesoft Programmers"	Finley & Myers	(\$18.00) \$18 *
"Now That You Know Apple Assembly Language"	Jules Gilder	(\$19.00) \$18 *
"Enhancing Your Apple II, vol. 1"	Don Lancaster	(\$15.00) \$15 *
"Enhancing Your Apple II, vol. 2"	Don Lancaster	(\$17.00) \$17 *
"Assembly Cookbook for the Apple II/IIe"	Don Lancaster	(\$21.00) \$20 *
"Microcomputer Graphics"	Roy E. Myers	(\$14.00) \$14 *
"Assembly Lines -- the Book"	Roger Wagner	(\$19.00) \$12 *

* These items add \$2 for first item, \$.75 for each additional item for US shipping.
 + Inquire for shipping cost.
 Customers outside USA inquire for postage needed.
 Texas residents please add 8% sales tax to all orders.
 << Master Card, VISA, Discover and American Express >>

S-C Software Corporation
 2331 Gus Thomasson #125
 DALLAS, TX 75228
 Phone 214-324-2050



\$9A...9D; the second copies only two bytes; and the third copies any number, which you specify in the A-register. I used a memory search to uncover all the calls on this third entry, and I only found calls which wanted to copy six bytes. There may be others, hidden in other sections of the AppleWorks code.

My more efficient rendition saves one byte by using the BIT opcode (\$2C) to skip over the two-byte LDA #2 instruction (see lines 1370 and 1870). I also save by pushing the byte count on the stack instead of saving it in RAM: the storage location is saved, and the PLA is two bytes shorter than a STA. However, I have to pull the byte back off the stack at line 2050, so the net saving is only two bytes. Line 2060, the LDY #0, can be deleted and save another two bytes (Y is already 0, in order for the loop in lines 2010-2040 to terminate). I don't need it because I have not called POLL.KEYBOARD. By the way, we do want to be sure that Y=0 here, because a lot of the subroutines depend on it. If we don't make it one of the functions of GET.x.PARMS, we will have to add a line to most of the subroutines which call GET.x.PARMS.

Inside AppleWorks a lot of string processing goes on. All of the strings I have observed are stored in memory as a length byte followed by the string data bytes. The maximum string will have 255 data bytes. A byte count equal to \$00 represents a null string. Lines 2980-3130 are my own code, simply to illustrate how you might code a subroutine to display a string stored in this fashion. Lines 3640-3690 show some strings built by the assembler. In my DEMO code, starting at line 3470, I called on DISPLAY.STRING to print these strings.

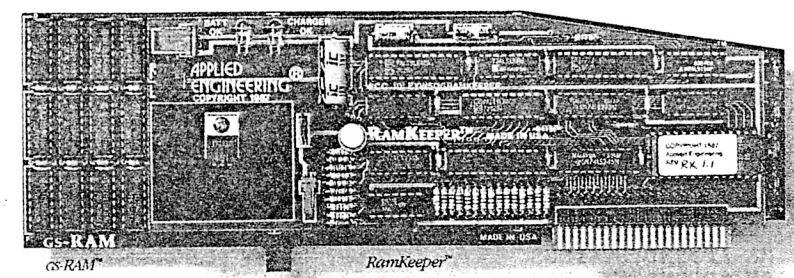
Lines 2080-2970 show my renditions of four subroutines I found inside AppleWorks, which I have named MOVE.STRING, COMPARE.STRING, APPEND.STRING, and FILTER.LC.TO.UC. These are not identical to the AppleWorks code, as I found some easy improvements here and there. I showed in comment lines where you can find these subroutines inside the AppleWorks 1.3 image. My DEMO program, lines 3470 to the end, shows some of these in action.

Not all of the subroutines which I found have to do with strings. Lines 3140-3410 show MOVE.BLOCK, which does the equivalent of a monitor "M" command. The three parameters are the destination starting address, the source starting address, and the number of bytes to be moved. (I say "moved", perhaps I should really say "copied".) A similar subroutine which starts immediately after this one, at \$1BAC, moves a block of memory "up", by starting at the last byte of the source block and moving backwards. This gives the ability to move a block of memory up to an overlapping area, without clobbering the data.

At the beginning of the code at \$1000 there is a JMP table (a long series of JMP xxxx instructions, one right after the other) which is evidently used when other segments want to use some of these general subroutines. Each of the other segments also begins with a JMP table. This is a good scheme for joining together pieces of a large system, and is easy to do. I find it a lot handier than the use of a Link Editor approach,

RamKeeper™

For the "Instant On" Apple IIGs.



Permanent Storage with an "Electronic Hard Disk"

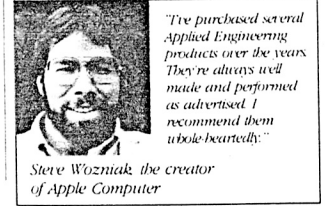
Now when you turn on your IIGs your favorite program can appear on screen in just a few seconds! With RamKeeper, your IIGs memory card will retain stored programs and stored data while your IIGs is turned off. RamKeeper allows you to divide your IIGs memory into part "electronic hard disk" and part RAM for your programs workspace—in almost any way you want and at anytime you want. GS-RAM, GS-RAM Plus, Apple IIGs memory card and most other IIGs memory cards are compatible with RamKeeper.

Supports Up to Two IIGs Memory Cards at the Same Time

If you bought your IIGs with Apple's memory card and later wished you had the GS-RAM, no problem. RamKeeper will support both cards plugged into RamKeeper simultaneously!

How it Works

Just unplug your IIGs memory card



from your computer, plug your IIGs memory card into RamKeeper, plug RamKeeper into the IIGs memory slot. If you have another IIGs memory card, an additional card socket on RamKeeper will accommodate your second card. That's all there is to it!

Reliability from the Most Experienced

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple so you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, reliability is everything! That's why Applied Engineering uses the more dependable Gel-Cells instead of Ni-Cad batteries (if Ni-Cad's aren't discharged periodically, they lose much of their capacity). RamKeeper has close to 6 times (about 6 hours) the "total power failure" back-up time of other systems. When power returns, RamKeeper automatically recharges the battery to a full charge. With power from your wall outlet, RamKeeper will back-up your IIGs memory cards RAM indefinitely.

RamKeeper Has and Does It All!

- Allows instant access to your programs without slow disk delays
- Configure Kilobytes or Megabytes of instant ROM storage for your favorite programs

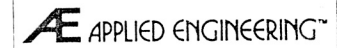
- Reduces power strain to your internal IIGs' power supply
- Contains back-up status L.E.D.'s
- Can support up to two IIGs memory cards simultaneously
- Supports both 256K installed memory chip boards like GS-RAM and the Apple IIGs Memory Expansion Card as well as 1 MEG installed memory chip boards like GS-RAM Plus
- 5-year hassle-free warranty
- 15 day money back guarantee
- Proudly made in the U.S.A.
- RamKeeper comes complete with battery, software and documentation

Only \$179.00!

(GS-RAM card shown in photo not included)

Order Your RamKeeper Today!

See your dealer or call (214) 241-6060, 9:00-11:00 CST, 7 days a week, or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 if outside U.S.A.



The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

Prices subject to change without notice

as is used under ProDOS-16. On the other hand, subroutines such as these I have shown in this article are the very type you might want to keep in assembled form as relocatable, linkable, object files, on a library ready to be used by all your future code. Even better, they are the type of subroutines I wish were in the //gs tool boxes, in ROM. And there, they would best be called via a JMP table, for efficiency. These routines are too short to afford the tremendous overhead of "real" toolbox calls.

The handiest way to use subroutines like these would involve writing macros for the calls. For example, here are some macro definitions for MOVE.STRING, APPEND.STRINGS, and MOVE.BLOCK:

```
.MA MOVE.STRING
JSR MOVE.STRING
.DA |1,|2
.EM

.MA APPEND.STRINGS
JSR APPEND.STRINGS
.DA |1,|2
.EM

.MA MOVE.BLOCK
JSR MOVE.BLOCK
.DA |1,|2,|3
.EM
```

These are simple enough, but they can make coding a program easier. If you are error prone, or simply enjoy being cautious, you might add code to the definitions to check for the correct number of macro parameters. For example, MOVE.BLOCK requires three parameters:

```
.MA MOVE.BLOCK
.DO |#=3
JSR MOVE.BLOCK
.DA |1,|2,|3
.ELSE
```

==> ERROR: WRONG NUMBER OF PARAMETERS. You have |#, I need 3.
.FIN
.EM

The line starting "==" will not be assembled as long as you have 3 parameters. If you have some other number, that line will cause an assembly error, since it starts with an illegal character. This will make it display during assembly, so you can catch and correct your call.

For example, if I try to assemble the following line:

```
1140 >MOVE.BLOCK BUFFER,SIZE
```

you will get the message:

```
>1140 ==> ERROR: WRONG NUMBER OF PARAMETERS. You have 2, I need 3.
```

Did you know you could do that? I wasn't sure, but I tried it and it works.

```
1000 *SAVE S.AW.SUBS
1010 *-----
98- 1020 PNTR .EQ $98,99
9A- 1030 P0 .EQ $9A
9B- 1040 P1 .EQ $9B
9C- 1050 P2 .EQ $9C
9D- 1060 P3 .EQ $9D
9E- 1070 P4 .EQ $9E
9F- 1080 P5 .EQ $9F
AO- 1090 COUNT .EQ $A0
-----
FD8E- 1110 MON.CROUT .EQ $FD8E
FDED- 1120 MON.COUT .EQ $FDED
-----
1130 *
1140 * GET PARS
1150 * (A) = # bytes of parameter info
1160 * Copy the bytes to $9A, 9B, ... etc.
1170 * Update Return Address
1180 * Poll Keyboard for Type-Ahead
1190 * Set Y=0, clobbers A and X
1200 *
1210 * For Example:
1220 * JSR subroutine
1230 * .DA parm1,parm2
1240 * <return here>
1250 *
1260 * subroutine JSR GET.4.PARMS
1270 *
1280 * RTS
1290 *
1300 *-----
1310 * The following code is as it exists in AppleWorks 1.3
1320 * .PH $18AD
1330 *-----
18AD- 1340 AW.GET.PARM.TEMP .BS 1
1350 *-----
18AE- A9 04 1360 AW.GET.4.PARMS LDA #4
18B0- D0 02 1370 BNE AW.GET.A.PARMS
18B2- A9 02 1380 AW.GET.2.PARMS LDA #2
18B4- A8 1390 AW.GET.A.PARMS TAY
18B5- 8D AD 18 1400 STA AW.GET.PARM.TEMP
18B8- BA 1410 TSX
18B9- BD 03 01 1420 LDA $0103,X
18BC- 85 98 1430 STA PNTR
18BE- 18 1440 CLC
18BF- 6D AD 18 1450 ADC AW.GET.PARM.TEMP
18C2- 9D 03 01 1460 STA $0103,X
18C5- BD 04 01 1470 LDA $0104,X
18C8- 85 99 1480 STA PNTR+1
18CA- 69 00 1490 ADC #0
18CC- 9D 04 01 1500 STA $0104,X
18CF- B1 98 1510 LDA (PNTR),Y
18D1- 99 99 00 1520 STA PNTR+1,Y
18D4- 88 1530 DEY
18D5- D0 F8 1540 BNE .1
18D7- 20 A7 1F 1550 JSR POLL.KEYBOARD
18DA- A0 00 1560 LDY #0
18DC- 60 1570 RTS
1580 .EP
```

```

1590 * POLL KEYBOARD
1600 *-----
1610
1620 .PH $1FA7
1630 POLL.KEYBOARD
1FA7- AD 00 CO 1640 LDA $C000 ANY KEY PRESSED?
1FAA- 10 24 1650 BPL .3 ...NO, RETURN NOW
1FAC- 8D 10 CO 1660 STA $C010 ...YES, CLEAR STROBE
1FAF- AE 61 CO 1670 LDX $C061 OPEN APPLE PRESSED?
1FB2- 30 07 1680 BMI .1 ...YES
1FB4- AE 62 CO 1690 LDX $C062 SOLID APPLE PRESSED?
1FB7- 30 02 1700 BMI .1 ...YES
1FB9- 29 7F 1710 AND #$7F ...NO APPLES, SO CLEAR BIT 7
1FBB- AE 84 11 1720 .1 LDX $1184 <<KEY.BUFFER.INDEX>>
1FBE- 9D 7A 11 1730 STA $117A,X <<KEY.BUFFER>>
1FC1- E8 1740 INX
1FC2- E0 0A 1750 CPX #10 AT END OF BUFFER YET?
1FC4- 90 02 1760 BCC .2 ...NO END YET
1FC6- A2 00 1770 LDX #0 ...END, SO WRAP AROUND
1FC8- EC 85 11 1780 .2 CPX $1185 <<KEY.BUFFER.OUTDEX>>
1FCB- F0 03 1790 BEQ .3 BUFFER IS FULL
1FCD- 8E 84 11 1800 STX $1184 <<KEY.BUFFER.INDEX>>
1FD0- 60 1810 .3 RTS
1820 .EP
1830 *-----
1840 * My More Efficient Version
1850 *-----
085A- A9 04 1860 GET.4.PARMS LDA #4
085C- 2C 1870 .HS 2C SKIP NEXT 2 BYTES
085D- A9 02 1880 GET.2.PARMS LDA #2
085F- A8 1890 GET.A.PARMS TAX
0860- 48 1900 PHA
0861- BA 1910 TSX
0862- BD 04 01 1920 LDA $0104,X GET RETURN ADDR-LO
0865- 85 98 1930 STA PNTR KEEP FOR VECTOR
0867- 18 1940 CLC
0868- 7D 01 01 1950 ADC $0101,X ADD # BYTES
086B- 9D 04 01 1960 STA $0104,X UPDATE RETURN ADDR-LO
086E- BD 05 01 1970 LDA $0105,X GET RETURN ADDR-HI
0871- 85 99 1980 STA PNTR+1 SAVE FOR VECTOR
0873- 69 00 1990 ADC #0
0875- 9D 05 01 2000 STA $0105,X UPDATE RETURN ADDR-HI
0878- B1 98 2010 .1 LDA (PNTR),Y USING VECTOR, COPY PARMS
087A- 99 99 00 2020 STA PNTR+1,Y ...TO 9A,9B,etc.
087D- 88 2030 DEY
087E- D0 F8 2040 BNE .1
0880- 68 2050 PLA GET LENGTH OFF STACK
0881- A0 00 2060 LDY #0
0883- 60 2070 RTS
2080 *-----
2090 * MOVE STRING
2100 * JSR MOVE.STRING
2110 * .DA destination,source
2120 * (at $1EF8 in AppleWorks 1.3)
2130 *-----
2140 MOVE.STRING
0884- 20 5A 08 2150 JSR GET.4.PARMS
0887- B1 9C 2160 LDA (P2),Y COPY THE LENGTH BYTE
0889- 91 9A 2170 STA (P0),Y
088B- F0 08 2180 BEQ .2 STRING IS EMPTY
088D- A8 2190 TAY LENGTH TO Y
088E- B1 9C 2200 .1 LDA (P2),Y
0890- 91 9A 2210 STA (P0),Y
0892- 88 2220 DEY
0893- D0 F9 2230 BNE .1
0895- 60 2240 .2 RTS
2250 *-----
2260 * COMPARE TWO STRINGS
2270 * JSR COMPARE.STRINGS
2280 * .DA str1,str2
2290 * return Carry Clear if str1 < str2; else Carry Set
2300 * (at $1ED9 in AppleWorks 1.3)
2310 *-----
2320 COMPARE.STRINGS
0896- 20 5A 08 2330 JSR GET.4.PARMS
0899- B1 9A 2340 LDA (P0),Y GET LENGTH OF SHORTER STRING
089B- D1 9C 2350 CMP (P2),Y
089D- 90 02 2360 BCC .1

```

SPECIAL !!! EXPANDED RAM/ROM BOARD: \$39.00

Similar to our \$30 RAM/ROM dev board described below. Except this board has two sockets to hold your choice of 2-2K RAM, 2-2K ROM or even 2-4K ROM for a total of 8K. Mix RAM and ROM too. Although Apple limits access to only 2K at a time, soft switches provide convenient socket selection. Hard switches control defaults.

IMPROVED !!! II IN A MAC (ver 2.0): \$75.00

Now includes faster graphics, UniDisk support and more! Bi-directional data transfers are a snap! This Apple II emulator runs DOS 3.3/PRODOS (including 6502 machine language routines) on a 512K MAC or MACPLUS. All Apple II features are supported such as HI/LO-RES graphics, 40/80 column text, language card and joystick. Also included: clock, RAM disk, keyboard buffer, on-screen HELP, access to the desk accessories and support for 4 logical disk drives. Includes 2 MAC diskettes (with emulation, communications and utility software, plus DOS 3.3 and PRODOS system masters, including Applesoft and Integer BASIC) and 1 Apple II diskette.

SCREEN.GEN: \$35.00

Develop HI-RES screens for the Apple II on a Macintosh. Use MACPAINT (or any other application) on the MAC to create your Apple II screen. Then use SCREEN.GEN to transfer directly from the MAC to an Apple II (with SuperSerial card) or IIC. Includes Apple II diskette with transfer software plus fully commented SOURCE code.

MIDI-MAGIC for Apple //c: \$49.00

Compatible with any MIDI equipped music keyboard, synthesizer, organ or piano. Package includes a MIDI-out cable (plugs directly into modem port - no modifications required!) and 6-song demo diskette. Large selection of digitized QRS player-piano music available for 19.00 per diskette (write for catalog). MIDI-MAGIC compatible with Apple II family using Passport MIDI card (or our own input/output card w/drum sync for only \$99.00).

FONT DOWNLOADER & EDITOR: \$39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. Special functions (like expanded, compressed etc.) supported. Includes HIRES screen editor to create custom fonts and special graphics symbols. For Apple II, II+, //e. Specify printer: Apple Imagewriter, Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/85, or Okidata 92/192. * FONT LIBRARY DISKETTE #1: \$19.00 contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

DISASM 2.2e : \$30.00 (\$50.00 with SOURCE Code)

Use this intelligent disassembler to investigate the inner workings of Apple II machine language programs. DISASM converts machine code into meaningful, symbolic source compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even provides label substitution. Address-based triple cross reference generator included. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his ASSEMBLY COOKBOOK.

The 'PERFORMER' CARD: \$39.00 (\$59.00 with SOURCE Code)

Converts a 'dumb' parallel printer I/F card into a 'smart' one. Simple command menu. Features include perforation skip, auto page numbering with date & title, large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), Okidata 82A/83A with Okigraph & Okidata 92/93.

'MIRROR' ROM: \$25.00 (\$45.00 with SOURCE Code)

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support.

RAM/ROM DEVELOPMENT BOARD: \$30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

C-PRINT For The APPLE //c: \$69.00

Connect standard parallel printers to an Apple //c serial port. Separate P/S included. Just plug in and print!

Unless otherwise specified, all Apple II diskettes are standard (not copy protected!) 3.3 DOS.

Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders OK.

RAK-WARE 41 Ralph Road W. Orange NJ 07052 (201) 325-1885



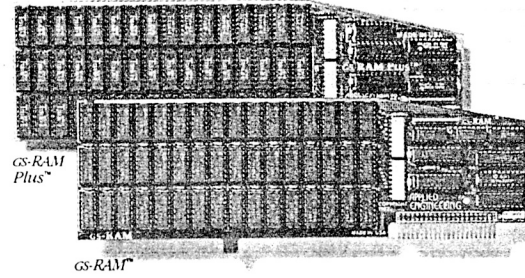
```

089F- B1 9C 2370 LDA (P2),Y
08A1- AA 2380 TAX
08A2- FO 0A 2390 BEQ .3 LENGTH OF SHORTEST TO X-REG
08A4- C8 2400 .2 INY SHORTEST IS NULL
08A5- B1 9A 2410 LDA (P0),Y COMPARE BODY OF STRINGS
08A7- D1 9C 2420 CMP (P2),Y
08A9- DO 09 2430 BNE .4 NOT SAME, SO CARRY GIVES RELATION
08AB- CA 2440 DEX
08AC- DO F6 2450 BNE .2 MORE TO COMPARE
08AE- AO 00 2460 .3 LDY #0 STRINGS MATCH TO END OF SHORTEST
08B0- B1 9A 2470 LDA (P0),Y COMPARE ON BASIS OF LENGTH
08B2- B1 9C 2480 LDA (P2),Y
08B4- 60 2490 .4 RTS
-----
2500
2510 * APPEND TWO STRINGS
2520 * JSR APPEND.STRINGS
2530 * .DA stringA,stringB
2540 * (at $1341 in AppleWorks 1.3)
-----
2550
2560 APPEND.STRINGS
08B5- 20 5A 08 2570 JSR GET.4.PARMS
08B8- B1 9A 2580 LDA (P0),Y GET LENGTH OF STRING A
08BA- 48 2590 PHA SAVE IT
08BB- 18 2600 CLC
08BC- 71 9C 2610 ADC (P2),Y ADD LENGTH OF STRING B
08BE- 91 9A 2620 STA (P0),Y MAKES LENGTH OF COMBINED STRING
08C0- 68 2630 PLA GET LENGTH OF STRING A AGAIN
08C1- 18 2640 CLC
08C2- 65 9A 2650 ADC P0 BUMP POINTER TO END OF STRING A
08C4- 85 9A 2660 STA P0
08C6- 90 02 2670 BCC .1
08C8- E6 9B 2680 INC P1
08CA- B1 9C 2690 .1 LDA (P2),Y LENGTH OF STRING B
08CC- FO 08 2700 BEQ .3 ...NULL, SO FINISHED
08CE- A8 2710 TAY
08CF- B1 9C 2720 .2 LDA (P2),Y
08D1- 91 9A 2730 STA (P0),Y
08D3- 88 2740 DEY
08D4- DO F9 2750 BNE .2
08D6- 60 2760 .3 RTS
-----
2770
2780 * FILTER LOWER CASE to UPPER CASE in a STRING
2790 * JSR FILTER.LC.TO.UC
2800 * .DA string
2810 * (at $1BF in AppleWorks 1.3)
-----
2820
2830 FILTER.LC.TO.UC
08D7- 20 5D 08 2840 JSR GET.2.PARMS
08DA- B1 9A 2850 LDA (P0),Y GET LENGTH OF STRING
08DC- FO 12 2860 BEQ .3 NULL STRING
08DE- A8 2870 TAY LENGTH TO Y-REG
08DF- B1 9A 2880 .1 LDA (P0),Y
08E1- C9 61 2890 CMP #'a'
08E3- 90 08 2900 BCC .2
08E5- C9 7B 2910 CMP #'z'+1
08E7- B0 04 2920 BCS .2
08E9- 29 DF 2930 AND #$DF TURN OFF LOWER/CASE BIT
08EB- 91 9A 2940 STA (P0),Y
08ED- 88 2950 .2 DEY
08EE- DO EF 2960 BNE .1 ...MORE BYTES
08F0- 60 2970 .3 RTS
-----
2980
2990 * DISPLAY STRING
3000 * JSR DISPLAY.STRING
3010 * .DA string.address
-----
3020
3030 DISPLAY.STRING
08F1- 20 5D 08 3040 JSR GET.2.PARMS
08F4- B1 9A 3050 LDA (P0),Y GET LENGTH
08F6- FO 0C 3060 BEQ .2 ...NULL STRING
08F8- 85 AO 3070 STA COUNT
08FA- C8 3080 .1 INY
08FB- B1 9A 3090 LDA (P0),Y
08FD- 20 ED FD 3100 JSR MON.COUNT
0900- C4 AO 3110 CPY COUNT
0902- 90 F6 3120 BCC .1
0904- 60 3130 .2 RTS

```

For Those Who Want the Most. From Those Who Make the Best. GS-RAM™

Now expand the IIcs™ RAM and ROM with up to 8 MEG of "Instant On" memory with the all new GS-RAM!



GS-RAM has an all new design. A design that delivers higher performance including increased speed, greater expandability, and improved software.

More Sophisticated, Yet Easier to Use

GS-RAM works with all IIcs software. In fact any program that runs on Apple's smaller memory card runs on the GS-RAM. But with GS-RAM you can have more memory, improved performance, and almost unlimited expansion capabilities. We've designed the new GS-RAM to be easier to use too—you don't have to adjust the size of your RAM disk every time you use a DMA device. No other RAM card with more than 4 banks of memory installed can make the same claim.

More than Just Hardware

Each GS-RAM and GS-RAM Plus includes the most powerful set of IIcs software enhancements available anywhere. In fact, our nearest competitor offers only a fraction of the invaluable programs that we include with each GS-RAM card. This software includes the most powerful disk-caching program available, the GS-RAM Cache. The Cache will make most of your applications run up to 7 times faster. Also included is a diagnostic utility that lets you test your GS-RAM by graphically showing the location of any bad or improperly installed RAM chips. And for AppleWorks users, we give you our exclusive Expander program that dramatically enhances both the capabilities and speed of AppleWorks.

Making AppleWorks Even Better

Applied Engineering's Expander program eliminates AppleWorks internal memory limits allowing it to recognize up to 8 megabytes of desktop workspace. You can increase the limits from only 7,250 lines to 22,600 lines in the word processor and from 6,350 records to 22,600 records in the database. The Expander allows all of AppleWorks, including print functions, to automatically load into RAM. The clipboard size will increase from 255 to 2,042 lines maximum. GS-RAM will automatically segment larger files so you can save them onto multiple floppies. And

GS-RAM provides a built-in print buffer that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

Grow by Kilobytes or Megabytes

We offer GS-RAM in two configurations so you can increase your memory 256K at a time (GS-RAM) or a megabyte at a time (GS-RAM Plus). Both are IIcs compatible and both come with our powerful enhancement software. GS-RAM can hold up to 1.5 MEG of 256K chips and GS-RAM Plus can hold up to 6 MEG using 1 MEG chips. And since both use standard RAM chips (not high-priced SIMMs), you'll find expanding your GS-RAM or GS-RAM Plus easy, convenient, and very economical. For further expansion, you can plug a 2 MEG "piggyback" card into the GS-RAM's expansion port for up to 3.5 MEG of total capacity. Or up to a whopping 8 MEG on GS-RAM Plus. If a GS-RAM owner outgrows 3.5 MEG, he can easily upgrade to GS-RAM Plus for a nominal charge.

Permanent Storage for an "Instant On" Apple

With our RamKeeper™ back-up option, your GS-RAM or GS-RAM Plus will retain both programs and data while your IIcs is turned off! Now when you turn your IIcs back on, your favorite software is on your screen in under 4 seconds! With RamKeeper you can divide your IIcs memory into part "electronic hard disk" and part extended RAM. Even change the memory boundaries at any time—and in any way—you want. Because



Steve Wozniak, the creator of Apple Computer

"In quality, performance, compatibility, expandability and support, Applied Engineering's GS-RAM and GS-RAM Plus are number one."

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple, you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, Reliability is everything. That's why Applied Engineering uses state-of-the-art "GEL-CELL's" instead of Ni-Cad batteries (if Ni-Cads aren't discharged periodically, they lose much of their capacity). RamKeeper has about 6 hours of "total power failure" back-up time. That's 6 times the amount of other systems. But with power from your wall outlet, RamKeeper will back-up GS-RAM, GS-RAM Plus, or most other IIcs memory cards indefinitely. Should you ever have a "total power failure," RamKeeper switches to its 6-hour battery. When power returns, RamKeeper will automatically recharge the battery to full power. RamKeeper incorporates a dual-rate charger, status LED's, and advanced power reducing circuitry. RamKeeper comes complete with battery, software, and documentation.

GS-RAM's Got it All!

- 5-year warranty — parts & labor
- 6 RAM banks (most cards have 4)
- Memory expansion port
- ROM expansion port
- Ultra-fast disk caching on ProDOS 8 AND ProDOS 16.
- Expands AppleWorks internal limits
- Includes hi-res self test
- No soldered-in RAM chips
- Expandable to 8 MEG
- No configuration blocks to set
- RamKeeper back-up option allows permanent storage of programs & data
- 15-day money-back guarantee
- Proudly made in the U.S.A.

GS-RAM with 256K	\$189
GS-RAM with 512K	\$259
GS-RAM with 1 MEG	\$399
GS-RAM with 1.5 MEG	\$539
GS-RAM with 2.5 to 3.5 MEG	CALL
GS-RAM Plus with 1-8 MEG	CALL
RamKeeper Option	\$179

Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside U.S.A.

AE APPLIED ENGINEERING™
The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006
Prices subject to change without notice

GS-RAM, GS-RAM Plus and RamKeeper are trademarks of Applied Engineering. Other brands and product names are registered trademarks of their respective holders.

```

3140 *-----*
3150 * MOVE MEMORY BLOCK
3160 * JSR MOVE.BLOCK
3170 * .DA destination,source,num.bytes
3180 * (at $1B84 in AppleWorks 1.3)
3190 *-----*
0905- A9 06 3200 MOVE.BLOCK
0907- 20 5F 08 3210 LDA #6 GET 6 PARM BYTES
090A- A5 9F 3220 JSR GET.A.PARMS
090C- F0 0F 3230 LDA P5 GET NUMBYTES-HI (# FULL PAGES)
3240 BEQ .2 ...NO FULL PAGES TO MOVE
3250 *---Move Full Pages---
090E- B1 9C 3260 .1 LDA (P2),Y
0910- 91 9A 3270 STA (P0),Y
0912- C8 3280 INY
0913- D0 F9 3290 BNE .1 ...UNTIL FULL PAGE MOVED
0915- E6 9B 3300 INC P1 SOURCE-HI
0917- E6 9D 3310 INC P3 DESTINATION-HI
0919- C6 9F 3320 DEC P5 # FULL PAGES LEFT
091B- D0 F1 3330 BNE .1 ...STILL MORE
3340 *---Move Partial Page---
091D- C4 9E 3350 .2 CPY P4 FINISHED PARTIAL PAGE?
091F- F0 07 3360 BEQ .3 ...YES
0921- B1 9C 3370 LDA (P2),Y
0923- 91 9A 3380 STA (P0),Y
0925- C8 3390 INY
0926- D0 F5 3400 BNE .2 ...ALWAYS
0928- 60 3410 .3 RTS
3420 *-----*
3430 *
3440 * DEMONSTRATION OF SOME STRING SUBROUTINES
3450 *
3460 *-----*
0929- 20 F1 08 3470 DEMO JSR DISPLAY.STRING
092C- 55 09 3480 .DA STR.A
092E- 20 8E FD 3490 JSR MON.CROUT
0931- 20 F1 08 3500 JSR DISPLAY.STRING
0934- 67 09 3510 .DA STR.B
0936- 20 8E FD 3520 JSR MON.CROUT
0939- 20 84 08 3530 JSR MOVE.STRING
093C- 79 09 67 3540 .DA STR.C,STR.B
093F- 09 3550 JSR DISPLAY.STRING
0940- 20 F1 08 3560 .DA STR.C
0943- 79 09 3570 JSR MON.CROUT
0945- 20 8E FD 3580 JSR APPEND.STRING
0948- 20 B5 08 3590 .DA STR.C,STR.A
094B- 79 09 55 3600 JSR DISPLAY.STRING
094E- 09 3610 .DA STR.C
094F- 20 F1 08 3620 RTS
0952- 79 09 3630 *-----*
0954- 60 3640 STR.A .DA #SZ.A
0955- 11
0956- D4 C8 C9
0959- D3 A0 C9
095C- D3 A0 D3
095F- D4 D2 C9
0962- CE C7 A0
0965- C1 AE 3650 .AS -/THIS IS STRING A./
11- 3660 SZ.A .EQ #-STR.A-1
0967- 11 3670 STR.B .DA #SZ.B
0968- D4 C8 C9
096B- D3 A0 C9
096E- D3 A0 D3
0971- D4 D2 C9
0974- CE C7 A0
0977- C2 AE 3680 .AS -/THIS IS STRING B./
11- 3690 SZ.B .EQ #-STR.B-1
0979- 3700 STR.C .BS 80
3710 *-----*

```

Enclosed is SCRNDUMP.PLUS, an enhancement to Steve Knouse's Generic Screen Dump in Apple Assembly Line, September 1983. The main enhancements are 40/80/Lores capability and, via conditional assembly, either a DOS 3.3 or a ProDOS version. The Lores capability is modified from a routine by R.M. Mottola in Nibble/#3/p.18.

The idea is to squeeze in as many features as possible and still have a utility that will fit in good old page 3 space (\$300.3CF). However, there are times you have something else in page 3 and need your screendump utility elsewhere. This is easy to do just by reassembling the screendump at another address. However, I'm lazy, and when I need a screendump utility installed I don't want to have to hunt for my assembler disk. So this version is self-relocating in that you can BLOAD and CALL to install it at any address where you have 284 bytes (\$11C) free.

This is too big to fit in page 3, so I borrowed Bill Morgan's idea from AAL/Nov.82, to use the upper part of page 2. The input trap and dump portion of SCRNDUMP.PLUS fits in page 3, while the installation takes the top of page 2. Since installation is a one-time affair, it's disposable, although it will remain there if you don't make too heavy use of the input buffer.

In addition to Steve Knouse and R.M. Mottola, I also learned and borrowed from Bill Parker's EPSON TEXT SCREEN DUMP, 1982 and Gary Little's ProDOS DUMP# utility in A+/Jul.85/p.69. I give credit to Roger Wagner's "Assembly Lines: The Book" for relocation ideas. There are a lot of neat techniques in this short utility, most of course 'borrowed' from other sources.

Now for some comments on why I chose BLOAD & CALL logic. You can install the dump via BRUN in immediate mode in DOS 3.3. However, readers of AAL know (AAL/JUN.86, AUG.86, SEP.86) that there are problems in DOS 3.3 with BRUN from within an Applesoft program. So I chose BLOAD & CALL to avoid the extra code you'd need to solve the problem. The BLOAD and CALL 41876 approach of AAL/AUG.86 is a good alternative to BRUN.

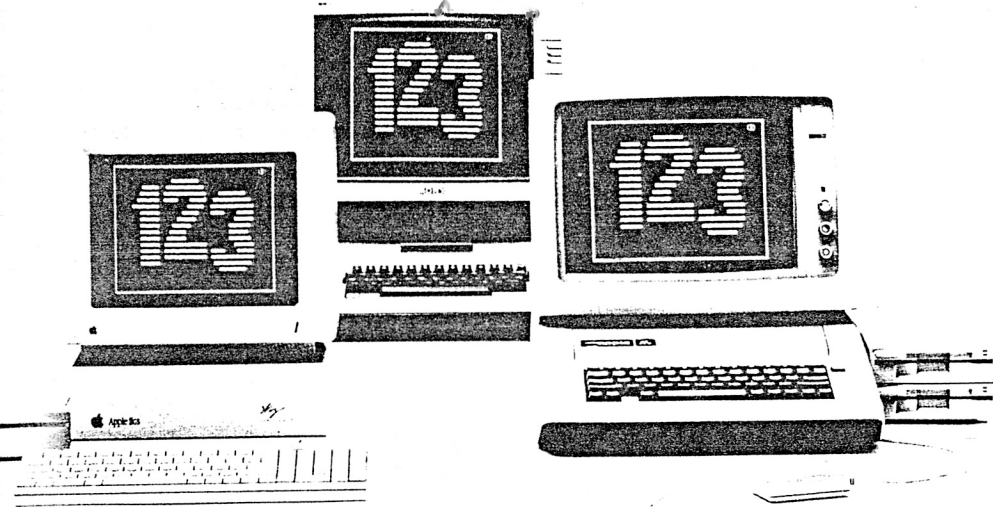
However, 3.3 DOSologists need not be ashamed, for ProDOS has its quirks also! See Call-Apple/Apr.84/p.39/Cecil Fretwell for details, but in ProDOS you must choose between a BLOAD & CALL or BRUN approach even for immediate mode! Replace Lines 2120 & 2130 by

```

2120 STX KSWL
2130 STY KSWH
2134 RTS
2138 .BS 2 (filler)

```

for BRUN logic. Again, I chose BLOAD & CALL logic for both DOS 3.3 and ProDOS to be consistent. Also, you can CALL 692 (or 2B4G from monitor) to rehook if you haven't made too heavy use of the input buffer.

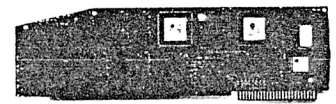


Now Apple speaks IBM. Three times faster than IBM.

Introducing PC Transporter™, the Apple® II expansion board that lets you run MS-DOS programs. Now your Apple II can run over 10,000 programs you could never use before. Like Lotus® 2-3,® MultiMate,® dBASE III,® Lotus,® Even Flight Simulator,® With PC Transporter, MS-DOS programs run on your Apple II the way they do on IBM® PCs or compatibles. With one important difference, PC Transporter runs most of those programs *three times faster* than an IBM PC/XT. Plus, to speed through number-crunching tasks, you can use our optional 8087-2 math coprocessor chip. It plugs into a socket on the PC Transporter.

Less expensive than an IBM clone.

Sure, a stripped-down IBM



clone costs about the same as the PC Transporter. But the peripherals it takes to get the clone up and running make the clone cost about three times what our American-made card costs.

You don't have to buy new hardware to use PC Transporter. **Works with the hardware you already own.**

With PC Transporter, MS-DOS programs see your Apple hardware as IBM hardware. You can use the same hardware you have now.

With IBM software, your Apple hardware works just like IBM hardware. Including your drives, monitors, printers, printer cards, clock cards and serial clocks.

You can use your IIe® or IIgs™ keyboard with IBM software. Or use our optional IBM-style keyboard (required for the II Plus).

You can use your Apple mouse. Or an IBM compatible serial mouse.

Plenty of power.

PC Transporter gives you as much as 640K of user RAM and 128K of system RAM in the IBM mode.

PC Transporter also is an Apple expansion card, adding up to 768K of extra RAM in the Apple mode. The Apple expansion alone is a \$300 value.

Easy to install.

You can install PC Transporter in about 15 minutes, even if you've never added an expansion board. You don't need special tools. Simply plug it into an Apple expansion slot (1 through 7 except 3), connect a few cables and a disk drive, and go!



PC Transporter taps into the world's largest software library. Now your Apple can run most of the IBM software you use at work. And it opens a new world of communications programs, games and bulletin boards.

A universal disk drive controller.

PC Transporter supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes. You'll shift instantly between Apple ProDOS and IBM MS-DOS.

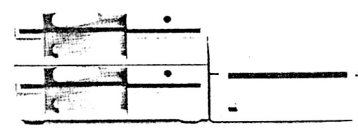
You'll need our versatile 5.25" 360K drive system to run IBM applications from 5.25" floppy disks. Use your Apple 5.25" drive for Apple 5.25" disks.

An Apple Disk 3.5 Drive will support the new 3.5" disks whether they're IBM MS-DOS formatted or Apple ProDOS formatted. The PC Transporter acts like an Apple Disk 3.5 Drive disk controller for IIgs, IIe, and II Plus users.

PC Transporter supports up to 5 drives in a number of combinations.

For example, you can connect a 5.25 Applied Engineering 360K dual-drive system directly to the card. Then plug two daisy-chained Apple 3.5 Drives (not the Apple UniDisk 3.5) to the dual-drive system. For a fifth drive, use a ProDOS file as an IBM hard disk.

PC Transporter controls Apple and IBM compatible disk drives. It supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes.



Versatile data storage.

PC Transporter reads MS-DOS and translates it into Apple native ProDOS. You can store IBM programs and data on any ProDOS storage device including the Apple 3.5 Drive, Apple UniDisk™ 3.5, Apple 5.25" drive, SCSI or ProDOS compatible hard drives. (You can use the Apple UniDisk 3.5 with its own controller card for storing programs and data, but not for directly booting an IBM formatted disk.)

You can even use our 360K PC compatible drive for ProDOS.

Make your Apple speak IBM.

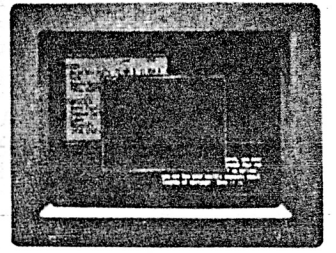
PC Transporter memory choices.

RAM in Apple mode:	RAM in IBM mode:	Price:
384K	256K	\$489.00
512K	384K	529.00
640K	512K	569.00
768K	640K	609.00

Note: The IBM mode is 128K less because the PC Transporter uses 128K for system memory.

- IIgs Installation Kit** 49.00
- IIe/II Plus Installation Kit** 39.00
- PC Transporter Accessories**
- 5.25" IBM Format 360K Drive Systems**
- Single-Drive System** 269.00
- Dual-Drive System** 399.00
- Half-Height Drive** 135.00
(add to Single-Drive system to make Dual-Drive)
- IBM-Style Keyboard** 139.00
(required for Apple II Plus. Requires IBM Keyboard Cable.)
- IBM Keyboard Cable** 34.00
- Sony RGB Monitor** 499.00
- Analog RGB Cable** 39.00
(for use with Sony monitor)
- Digital RGB Cable** 39.00
(for use with Sony monitor)
- Digital RGB Adapter** 24.00
- Colorswitch** 44.00
(included with IIgs Installation Kit)
- 128K ZIP** 40.00
- PC Transporter Memory Expansion** per set
- Chip Set**
- 8087-2 Math Co-processor Chip** 229.00
- Heavy Duty Power Supply** 69.00
(IIe and II Plus only)

See your dealer or call or send check or money order to Applied Engineering. MasterCard, VISA and COD welcome. Texas residents add 6 1/4% sales tax.



PC Transporter produces better IBM graphics than IBM. Analog is sharper than digital. So with an analog RGB monitor, PC Transporter's CGA graphics and text are superior to IBM's digital display — even while running IBM software!

And, you can also use an Apple composite monitor in IBM text or graphics mode.

storage and a 143K Apple 5.25" drive for MS-DOS storage.

Created by Apple's original designers.

The brains behind PC Transporter were also behind your Apple II.

The PC Transporter design team includes the former project managers for the creation of the Apple IIe and IIc. The co-designer of the Apple II disk controller. And the first full-time Apple programmer and author of the ProDOS operating system.

So you know the PC Transporter and your Apple were made for each other.

Support and service from the leader in Apple add-ons.

Applied Engineering sells more Apple peripheral boards than anyone else — including Apple Computer. So you know we'll be around after the sale.

PC Transporter comes with a 15-day money back guarantee. If you're not fully satisfied after using it, return it for a full refund. PC Transporter also comes with a 1-year warranty.

How to get your PC Transporter today.

See your dealer. Or call Applied Engineering any day between 9 a.m. and 11 p.m. CST at 214-241-6060.

AE Applied Engineering
The Apple enhancement experts.
P.O. Box 798, Carrollton, TX 75006
214-241-6060
A Division of AE Research Corporation

to describe the utility in more detail. Lines 1940-1950 find where it has been BLOADED. I think it's a good idea to inhibit interrupts while getting information from the stack. Then lines 1640-1720 modify (SETRAP+1) and (SETRAP+3) to point at TRAP. Lines 1730-1900 put the current INPUT address at TRAP+1, TRAP+2 and TRAP+8. TRAP+10: the first so we can 'daisy-chain' our input TRAP at the end of the current INPUT routine, and the second so we can terminate via CTRL-T to remove our INPUT trap. Lines 1910-1980 set default parameters at locations 6-9 and \$ED. If you BLOAD without CALL-ing, you must set these parameters yourself. As far as I can tell, there are 11 page 0 locations not used by Applesoft, Integer Basic, DOS 3.3, ProDOS (including interrupt handlers), or the System Monitor. They are 6-9, \$1E, \$EB-EF, and \$F9.

Anyway, 6 (PRFLAG) is used as an enable/disable (1/0) flag- for when you want to disable dumping without removing the input trap. 7 (WINTOP) and 8 (WINBOT) are used to set a 'window' to dump if you don't want to dump the whole screen. Here the top line is 0 and bottom is 23 (\$17). Note that I couldn't fit left and right margin windows in. If you want a routine that does both 40 and 80 column dumps you'll have to be careful in adding left and right margins. \$ED (FLAG) is used to flag if a dump has been done via CTRL-P from the TRAP (value \$FF) or just a CALL 802 (JSR \$322) (value 0).

SETRAP, Lines 2050-2150, hooks TRAP into DOS or ProDOS.

TRAP, our input 'filter', does its thing by first daisy-chaining to the current input routine and letting it do all the dirty work. Only then does TRAP check for either CTRL-T (Terminate TRAP and reset input hooks) or CTRL-P (Print the screen). If neither, exit to caller of input routine. If a CTRL-P, Set FLAG (\$ED) to \$FF to show we came thru TRAP. This avoids a JSR DUMP, which would not be relocatable. You could have a JSR DUMP with address filled in during the first portion of the utility just like SETRAP+1, +3 are filled in from Lines 1640-1720. That way you wouldn't need FLAG at all, but the approach takes 3 more bytes. Roger Wagner's book has other ways to do a relocatable JSR, but remember that we want a DUMP that can be called either directly or from TRAP.

Note that lines 2280 and 2290 inside TRAP avoid another ProDOS 'gotcha'. A JMP \$3D0 (DOS.WARM) in ProDOS (Q from IIgs Monitor) returns to immediate mode but erases Applesoft variables, unlike the friendly DOS 3.3! Thanks to Steve and Marsha Meuse for this one (Nibble/Nov.85/p.20). So from ProDOS JSR RSTINT and JSR BASIC2 exit without erasing variables. BASIC2 (\$E003) is the address a CTRL-C <Return> from monitor uses. RSTINT (\$9A17) is the same in BASIC.SYSTEM versions 1.0 and 1.1, the only ones so far. Future versions may change, so beware.

For the DUMP itself, we disable interrupts and save the A-X-Y registers, the cursor location CH (so after the dump is done you end up with the cursor where it was when you started), and the output hooks, in lines 2370-2490.

Lines 2510 and 2520 test the enable flag and exit if it's zero. Lines 2530-2650 simulate a PRN#SLOT, and kill video echo. Change SLOT in Line 1060 if printer isn't in slot 1, and change NOAIO to 0 in Line 1080 if you have an AIO interface, which is code from Steve Knouse.

Lines 2670-2720 start the dump with screenline specified by WINTOP. In Lines 2730-2820 we handle 80-column dumps. This is good only for //e or //c or IIgs using firmware 80-column routines. There are too many video interface cards that work in different ways to support them all. Some cards have their own dump utilities, so you may not be out of luck if you have a non-standard interface. In line 2810 I use \$FF to flag the X-register if 80-columns are active. Otherwise, lines 2840-2870 in MORCHR will end with X in range 0-23. That way in Lines 3090 and 3100 if X gets incremented to 0 then branch back to MORCHR to get even column character, so 2*40=80 characters/line get dumped. This trick of using a register value out of the range you would normally get in the rest of your code is a handy way of saving bytes.

In Line 2840 PRFN gets tested. 0->LORES, and 1->TEXT. If LORES, still only do a mixed mode dump of 20 LORES lines and 4 TEXT lines. Replace Line 2880 by 2 NOP's (\$EA) to do full-screen LORES. Note that if you're in 80-column mode, the odd columns are dumped as text regardless of PRFN, since 2820 skips MORCHR. So this dump doesn't do double-LORES. Also, your LORES-creating program should start with TEXT:HOME:GR since TEXT:HOME clears even 80-columns to spaces, whereas GR only clears 40 columns to nulls. If you just do GR, some text may be left in the aux memory text pages, and you'll see it when you do a LORES dump.

So a LORES dump in 80-column mode still dumps only 40 column LORES, spaced out (if you did TEXT:HOME:GR) to match the 4 lines of 80-column text at the bottom.

In Lines 2890-3010 a quasi-40-by-40 resolution for LORES is done by using a space for both LORES blocks off, a caret for top LORES block on, an underline for bottom LORES block on, and an X for both LORES blocks on. This should work for most printers, but feel free to use other characters or symbols, especially if your printer has graphics. The resulting dump is good for monochrome LORES pictures, and especially Bar Charts. Multi-colored LORES and even abstract monochrome patterns lose a lot in the 'translation' to print.

Note especially how 2910 turns off the lo-nibble, while the exclusive-or in 2930 turns off the hi-nibble and turns back on the lo-nibble. Exclusive-or is a handy opcode to learn, and to save bytes by using. Also, the BIT opcode is used in Lines 2960, 2980, and 3000 as a 2-byte NOP to skip the following 2 bytes. BIT changes the N,V, and Z flags, but since 3020 does another CMP anyway, this doesn't matter here. Another 'gotcha' is to avoid inadvertently toggling a soft-switch in the \$C000-C0FF range via BIT. I did that once and it took me a long time to figure out!

Lines 3020-3050 make sure a value of \$A0 or greater is sent to printer, to avoid control or inverse or flashing characters. Line 3060 then masks off the MSB to avoid graphics in EPSON printers. Lines 3080-3190 send the character to the printer and do loop checking for 80-columns, line length, and WINBOT screen checking.

Lines 3210-3410 restore everything saved upon entry to DUMP. If FLAG was set to show we came thru TRAP, it's cleared in case next time we don't. Besides, if we came thru TRAP, we're in input mode and need to JMP RDKEY to get the next keystroke. If we didn't come thru TRAP, we can just exit via RTS.

Some Demonstrations

So much for the dump. Now it's time for some demo programs. The following, which I call CHARDEMO, shows the active character set, including MouseText if you're in 80-column mode. //e's, c's, and IIGs's have different character sets for 40 and 80 column modes, and you'll see that most printers match the 40-column character set best for INVERSE and FLASHing characters. So what you dump isn't always what you see! If you avoid MouseText and INVERSE and FLASH, you'll do all right except that most printers use code \$7F=127 as 'delete previous char' instead of a checkerboard. So if you don't see a tilde (code \$7E=126) or a checkerboard, you'll know what happened.

```

5 REM CHARDEMO
6 PRINT CHR$(4)"BLOAD SCRNDUMP.PLUS"
7 CALL 692: REM INSTALL HOOKS
10 INVERSE
20 PRINT CHR$(27);"@ABCDEFGHIJKLMNOPS
   TUVWXYZ[\]^_"; CHR$(24)
30 NORMAL
40 GOSUB 1000
50 INVERSE : GOSUB 1000: NORMAL
60 FLASH : GOSUB 1000: NORMAL
70 END
1000 FOR I = 32 TO 127: PRINT CHR$(I);: NEXT
1010 PRINT : RETURN

```

TESTGR does a LORES dump by CALLing DUMP directly. Try in 40 and 80-column mode to see the difference.

```

5 TEXT : HOME : GR
10 COLOR= 15
20 FOR I = 0 TO 39
30 VLIN I,39 AT I: NEXT
40 PRINT "THIS IS A TEST."
50 POKE 9,0: CALL 802: POKE 9,1
60 REM ASSUNES M/L BLOADED & PARAMETERS SET

```

TEST40 and TEST80 merely show that no top lines or characters get dropped.

```

5 TEXT : HOME
10 FOR I = 1 TO 22
20 PRINT "LINE NUMBER ";I;: HTAB 16
25 PRINT "THIS IS A TEST."
30 NEXT

```

```

5 TEXT : HOME
10 FOR I = 1 TO 22
20 PRINT "LINE NUMBER ";I;: HTAB 16
25 PRINT "THIS IS A TEST.";
26 FOR J = 1 TO 40: PRINT "A";: NEXT : PRINT
30 NEXT

```

LORES.PIC, adapted from a David Thornburg program to do LORES and not HIRES patterns, allows you to experiment with patterns. Try Inputs x=0,y=0,s=field size=10, and 2 colors to see an abstract monochrome pattern-it loses a lot in the translation to print.

```

5 REM LORES.PIC/DAVID THORNBURG/A+/DEC.86&JUN.87
6 REM ASSUMES M/L BLOADED & PARAMETERS SET
7 POKE 9,0: REM FOR LORES DUMP
10 PRINT "ENTER STARTING X-VALUE ";
20 INPUT A
30 PRINT "ENTER STARTING Y-VALUE ";
40 INPUT B
50 PRINT "ENTER FIELD SIZE ";
60 INPUT S
70 PRINT "ENTER NUMBER OF COLORS (2-16) ";
80 INPUT N
90 TEXT : HOME : GR
100 FOR I = 0 TO 39
110 FOR J = 0 TO 39
120 X = A + (S * (I - 20) / 100)
130 Y = B + (S * (J - 20) / 100)
140 Z = 10 * SIN (X * X) + 10 * SIN (Y * Y)
160 COLOR=(Z - (N * INT (Z / N)))
170 PLOT I,J
180 NEXT J
190 NEXT I
200 CALL 802: POKE 9,1: REM DUMP, THEN BACK TO TEXT
DEFAULT

```

Try changing Line 140 to Z= X*X + Y*Y and use inputs 0,0,25,2 to see circles. If the pattern isn't too abstract, the resulting dump is ok. Also try 140 Z = X*Y and inputs 0,0,20,2 to see hyperbolas. It's a neat program to play with, but unfortunately abstract and/or multi-colored patterns go past the capabilities of the simplistic LORES dump here.

```

1000 *SAVE S.SCRNDUMP.PLUS
1010 -----
1020 * See AAL/SEP.83/P.22-24/Steve Knouse
1030 * and NIBBLE/#3/P.19&40/R.M. Mottola
1040 * ...modified 5/10/87 Louis Pitz
1050 *-----
01- 1060 SLOT .EQ 1 PRINTER SLOT#
01- 1070 DOS .EQ 1 DOS3.3 ACTIVE
01- 1080 NOAIO .EQ 1 GENERIC INTERFACE

```

```

1100 *-----
06- 1110 PRFLAG .EQ $6 0->DISABLE, 1->ENABLE
07- 1120 WINTOP .EQ $7 TOP OF WINDOW TO PRINT
08- 1130 WINBOT .EQ $8 BOTTOM
09- 1140 PRFN .EQ $9 0->LORES, 1->TEXT
EB- 1150 PTR .EQ $EB USE FOR RELOCATING
24- 1160 CH .EQ $24
23- 1170 BASL .EQ $28
36- 1180 CSWL .EQ $36
37- 1190 CSWH .EQ $37
38- 1200 KSWL .EQ $38
39- 1210 KSWH .EQ $39
ED- 1220 FLAG .EQ $ED
EE- 1230 LINCTR .EQ $EE
1240 *-----
0100- 1250 STACK .EQ $100 USE FOR RELOCATING
1260 *-----
03D0- 1270 DOS.WARM .EQ $3D0
03EA- 1280 DOS.HOOK .EQ $3EA
AA55- 1290 DOSKSW .EQ $AA55 DOS ACTIVE->TRUE KSW
BE32- 1300 VECTIN .EQ $BE32 PRODOS ACTIVE->TRUE KSW
9A17- 1310 RSTINT .EQ $9A17
E003- 1320 BASIC2 .EQ $E003
1330 *-----

```

EnterSoft:

Basic-like macros which make the complex simple. Don't re-write that multiplication routine for the hundredth time! Get EnterSoft instead! Do 8/16/32/64 bit Math/Input-Output/Graphics simply without all of the hassles. These routines are a must for the serious programmer who doesn't want to spend all of his/her time trying to re-invent the wheel. DOS 3.3 Version=\$30.00, ProDos Version=\$30.00, BOTH for only \$50.00. GET YOURS TODAY.

A Shape Table Program:

For once! A shape table program which is logically organized into its componet parts. Each section resides in its own program. The editor, disk access, HI-Res section; each section is separate. Written almost entirely in Basic, it is easily modified. Not copyprotected! Put them on a Hard Disk, Ram Drive, anywhere! DOS 3.3 Version=\$20.00, ProDos Version=\$20.00, BOTH for \$30.00!

```

*****
Send Check or Money Order To:
*****
Mark Manning
c/o Simulacron I/Baggy Game
P.O. Box 58598
Webster, TX 77598
*****
ProDos Upgrade for DOS 3.3
EnterSoft Owners - $20.00
*****
Thanks for the letters -
Keep Writing!
*****

```

```

0578- 1340 NOVID .EQ $578
C01F- 1350 COL80 .EQ $C01F
C054- 1360 REGRAM .EQ $C054
C055- 1370 AUXRAM .EQ $C055
FBB3- 1380 IDBYTE .EQ $FBB3
FBC1- 1390 BASCAL .EQ $FBC1
FC22- 1400 VTAB .EQ $FC22
FD0C- 1410 RDKEY .EQ $FD0C
FD1B- 1420 KEYIN .EQ $FD1B
FD8E- 1430 CROUT .EQ $FD8E
FDED- 1440 COUT .EQ $FDED
FE95- 1450 OUTPRT .EQ $FE95
FF58- 1460 RTRN .EQ $FF58
1470 *-----
1480 * First find out 'WHERE AM I?'
1490 * See "Assembly Lines: The Book", chapter 14
1500 * by Roger Wagner
1510 *-----
1520 .OR $2B4 FIT $2B4.3CF A$2B4,L$11C [A692,L284]
1530 .TF SCRNDUMP.PLUS
02B4- 08 1540 DISINT PHP SAVE INTERRUPT STATUS
02B5- 78 1550 SEI DISABLE INTERRUPTS
02B6- 20 58 FF 1560 START JSR RTRN PUT START+2 ON STACK
02B9- BA 1570 TSX NOW GET STACK POINTER
02BA- BD 00 01 1580 LDA STACK,X GET (START+2)-HI BYTE
02BD- 85 EC 1590 STA PTR+1
02BF- CA 1600 DEX
02C0- BD 00 01 1610 LDA STACK,X GET (START+2)-LO BYTE
02C3- 85 EB 1620 STA PTR
02C5- 28 1630 PLP RESTORE INTERRUPT STATUS
02C6- A0 42 1640 LDY #SETRAP-START-1 OFFSET (SETRAP+1)-(START+2)
02C8- 18 1650 CLC
02C9- 69 4C 1660 ADC #TRAP-START-2 OFFSET (TRAP)-(START+2)
02CB- 91 EB 1670 STA (PTR),Y PUT IN SETRAP+1
02CD- A5 EC 1680 LDA PTR+1
02CF- 69 00 1690 ADC #0 IN CASE CROSS PAGE BOUNDARY
02D1- C8 1700 INY
02D2- C8 1710 INY
02D3- 91 EB 1720 STA (PTR),Y PUT IN SETRAP+3
02D5- AD 55 AA 1730 .DO DOS IF DOS ACTIVE
1750 LDA DOSKSW SAVE INPUT HOOKS INSIDE TRAP
1760 .ELSE IF PRODOS
1770 LDA VECTIN
1780 .FIN
02D8- A0 54 1780 LDY #TRAP+6-START OFFSET (TRAP+8)-(START+2)
02DA- 91 EB 1790 STA (PTR),Y
02DC- A0 4D 1800 LDY #TRAP-START-1 OFFSET (TRAP+1)-(START+2)
02DE- 91 EB 1810 STA (PTR),Y
02E0- C8 1820 INY
1830 .DO DOS IF DOS ACTIVE
02E1- AD 56 AA 1840 LDA DOSKSW+1
1850 .ELSE IF PRODOS
1860 LDA VECTIN+1
1870 .FIN
02E4- 91 EB 1880 STA (PTR),Y
02E6- A0 56 1890 LDY #TRAP+8-START OFFSET (TRAP+10)-(START+2)
02E8- 91 EB 1900 STA (PTR),Y
02EA- A2 00 1910 LDX #0 SET DEFAULT VALUES
02EC- 86 07 1920 STX WINTOP TOP OF SCREEN=LINE 0
02EE- 86 ED 1930 STX FLAG CLEAR FLAG AT START
02F0- E8 1940 INX
02F1- 86 06 1950 STX PRFLAG ENABLE ROUTINE
02F3- 86 09 1960 STX PRFN SET FOR TEXT
02F5- A2 17 1970 LDX #23 SET BOTTOM SCREEN
02F7- 86 08 1980 STX WINBOT TO LINE 23
1990 *-----
2000 * NOW FOR SETRAP -SET INPUT HOOKS
2010 * TO POINT AT 'TRAP' OR INPUT FILTER
2020 * NOTE HOW FIRST SECTION MODIFIES (SETRAP+1)
2030 * AND (SETRAP+3) TO POINT AT TRAP!
2040 *-----
02F9- A2 04 2050 SETRAP LDX #TRAP
02FB- A0 03 2060 LDY /TRAP
2070 .DO DOS IF DOS ACTIVE
02FD- 86 38 2080 STX KSWL
02FF- 84 39 2090 STY KSWH
0301- 4C EA 03 2100 JMP DOS.HOOK
2110 .ELSE PRODOS
2120 STX VECTIN
2130 STY VECTIN+1
2140 RTS
2150 .FIN

```

It's 1988, and ProDOS Thinks it's 1982....Bob Sander-Cederlof

If you are still using ProDOS 1.1.1, and you have some sort of clock card such as Thunderclock, TimeMaster, or any other "standard" ProDOS clock, you have a problem. Apple built this bug into ProDOS, and they came out with the new versions (they call it ProDOS-8 version 1.4 now) just in time.

In my article about the clock driver in the November 1983 issue of AAL (pages 25-28), I discussed the problem. It seemed a little more remote at the time. Apple based ProDOS on the Thunderclock, even though that device does not keep track of the year. The ProDOS clock driver reads the Month, Day, and Day of Week information and does some arithmetic to determine which of six years could produce that day of week on the corresponding month and day. ProDOS 1.1.1 and earlier versions could produce dates from 1982 through 1987. When 1988 rolled around a few weeks ago, hundreds of thousands of Applers around the world slipped back in time to 1982.

And it is not funny! Some programs will not let you operate if the dates are not correct!

Well, there are at least four ways around the problem. You can remove your clock card, and type the date in manually wherever it is really needed. Not very nice.

Or, you can get the up-to-date version of ProDOS, now called ProDOS-8 Version 1.4. You can get it, and then you can copy it to every floppy (both 3 1/2 and 5 1/4), to every RamFactor, to every hard disk in sight. This is tedious, but it is the best resolution. If you have a friendly dealer, you can get it from the IIgs system disk. But don't copy the file named PRODOS from this disk (that is only a loader now). Instead, copy the file named P8 from the subdirectory SYSTEM. P8 is a longer file than version 1.1.1 of ProDOS was, so if you use BSAVE to put it on your disks be sure to specify the L parameter. Something like this should do the trick:

Boot any ProDOS disk, preferably one with version 1.4 so the correct dates will get into the file directories you are updating. Get into the S-C Macro Assembler or Applesoft. With the latest IIgs system disk in your drive, type:

```
BLOAD SYSTEM/P8,TSYS,A$2000
```

Now put the disk you want to update into a drive, and type the following. You may want to include slot and drive parameters, or set the prefix to the appropriate value for a ram disk or hard disk.

```
UNLOCK PRODOS
BSAVE PRODOS,TSYS,A$2000,L$3C7D
LOCK PRODOS
```

A third approach saves you a trip to the dealer. You can simply PATCH the copies of ProDOS version 1.1.1 to give you the correct year. When you BLOAD the file named PRODOS at \$2000, the six-year table is at \$4F76. If you look there now you will find the following bytes:

```
4F76: 54 54 53 52 57 56 55
```

These correspond to the years 1984, 1984, 1983, 1982, 1987, 1986, and 1985. Notice that 1984, being a leap year, takes up two of the values. Patch these seven bytes, using the monitor, as follows:

```
4F76:5A 59 58 58 57 56 5B
```

The table now includes the years from 1986 through 1991. If you want 1992 in there also, substitute 5C where I have 57 and 56 above. Both 1988 and 1992 are leap years, so they both take two table positions. When ProDOS 1.4 was released it was still 1987, so there was not room for 1992 in the table.

A fourth possible solution was suggested by reader Garth O'Donnell. You can replace the clock driver inside ProDOS with one that reads the year directly from your clock card! This is what happens when you boot Version 1.4 in a IIgs, because P8 senses that you are in a IIgs and plugs in a different driver. But if you are still using an older Apple, as most of us are, you can modify the PRODOS file to load an intelligent driver for your own clock card. Of course, if you are using a Thunderclock, the driver with the above patches is the best you can do. But if you have a TimeMaster, as Garth does, you can use a program like he wrote.

I decided to try my hand at modifying the standard clock driver so that it uses the year information in the TimeMaster. The following program is derived directly from the standard driver, with as few modifications as possible. It still resides in the ProDOS SYS file at \$4F00, but it is a lot shorter. (Maybe you can think of something useful to do with the extra 45 bytes!) It still depends upon the standard ProDOS loader to plug in the actual slot number in lines 1260 and 1310. The major change I made was to call on the ":" instead of the "#" mode. The "#" mode is a ThunderClock mode, which does not return the year. The ":" mode is a TimeMaster mode, which does return the year.

If you have an Applied Engineering Serial Pro card, which includes a TimeMaster compatible clock, you can use the driver I wrote by making the single change as shown in the comments on line 1090. Or, maybe you could use those extra 45 bytes for a subroutine that would check which clock is in the slot and make the appropriate changes at run time.

```

4F00- 1180 .TF B.CLOCK.DRIVER
      1000 *SAVE S.CLOCK.1988
      1010 *-----
      1020 * IF THE PRODOS BOOT RECOGNIZES A TIMEMASTER,
      1030 * A "JMP $D742" IS INSTALLED AT $BF06 AND
      1040 * THE SLOT ADDRESS IS PATCHED INTO THE FOLLOWING
      1050 * CODE AT SLOT.A AND SLOT.B BELOW.
      1060 *-----
      1070 * DEFINE CLOCK ENTRY POINT
      1080 *-----
C108- 1090 CLOCK .EQ $C108 <<<USE $C11D FOR AE SERIAL PRO>>>
      1100 *-----
BF90- 1110 DATE .EQ $BF90 $BF91 = YYYYYYYM
      1120 * $BF90 = MMMDDDDD
BF92- 1130 TIME .EQ DATE+2 $BF93 = 00HHHHH
      1140 * $BF92 = 00MMMM
0538- 1150 MODE .EQ $5F8-$C0 TIMEMASTER MODE IN SCREEN HOLE
      1160 *-----
      1170 .OR $4F00
4F00- 1180 .TF B.CLOCK.DRIVER
      1190 .PH $D742
      1200 *-----
      1210 PRODOS.TIMEMASTER.DRIVER
D742- AE 50 D7 1220 LDX SLOT.B $CN
D745- BD 38 05 1230 LDA MODE,X SAVE CURRENT TIMEMASTER MODE
D748- 48 1240 PHA
D749- A9 BA 1250 LDA #":" SEND ":" TO TIMEMASTER
D74B- 20 0B C1 1260 JSR CLOCK+3 SELECT TIMEMASTER MODE
D74D- 1270 SLOT.A .EQ #-1
      1280 *-----
      1290 * READ TIME & DATE INTO $200...$211 IN FORMAT:
      1300 *-----
D74E- 20 08 C1 1310 JSR CLOCK
D750- 1320 SLOT.B .EQ #-1

```

```

1330 *-----
      1340 CONVERT ASCII VALUES TO BINARY
      1350 * $3E -- MINUTE
      1360 * $3D -- HOUR
      1370 * $3C -- YEAR
      1380 * $3B -- DAY OF MONTH
      1390 * $3A -- MONTH
      1400 *-----
D751- 18 1410 CLC
D752- A2 04 1420 LDX #4
D754- A0 0C 1430 LDY #12
D756- B9 03 02 1440 LDA $203,Y POINT AT MINUTE
D759- 29 0F 1450 AND #0F TEN'S DIGIT
D75B- 85 3A 1460 STA $3A IGNORE TOP BIT
D75D- 0A 1470 ASL #2 MULTIPLY DIGIT BY TEN
D75E- 0A 1480 ASL #4
D75F- 65 3A 1490 ADC $3A #5
D761- 0A 1500 ASL #10
D762- 79 04 02 1510 ADC $204,Y ADD UNIT'S DIGIT
D765- 38 1520 SEC
D766- E9 B0 1530 SBC #$B0 SUBTRACT ASCII ZERO
D768- 95 3A 1540 STA $3A,X STORE VALUE
D76A- 88 1550 DEY BACK UP TO PREVIOUS FIELD
D76B- 88 1560 DEY
D76C- 88 1570 DEY
D76D- CA 1580 DEX BACK UP TO PREVIOUS VALUE
D76E- 10 E6 1590 BPL .1 ...UNTIL ALL 5 FIELDS CONVERTED
      1600 *-----
      1610 PACK MONTH AND DAY OF MONTH,
      1620 *-----
D770- A8 1630 TAX MONTH (1...12)
D771- 4A 1640 LSR 00000ABC--D
D772- 6A 1650 ROR D00000AB--C
D773- 6A 1660 ROR CD00000A--B
D774- 6A 1670 ROR BCD00000--A
D775- 05 3B 1680 ORA $3B MERGE DAY OF MONTH
D777- 8D 90 BF 1690 STA DATE SAVE PACKED DAY AND MONTH
      1700 *-----
D77A- A5 3C 1710 LDA $3C YEAR
D77C- 2A 1720 ROL MERGE TOP MONTH BIT
D77D- 8D 91 BF 1730 STA DATE+1 YYYYYYYM
      1740 *-----
D780- A5 3D 1750 LDA $3D GET HOUR
D782- 8D 93 BF 1760 STA TIME+1
D785- A5 3E 1770 LDA $3E GET MINUTE
D787- 8D 92 BF 1780 STA TIME
D78A- 68 1790 PLA RESTORE TIMEMASTER MODE
D78B- AE 50 D7 1800 LDX SLOT.B GET $CN FOR INDEX
D78E- 9D 38 05 1810 STA MODE,X
D791- 60 1820 RTS

```

PROGRAMMER

Applied Engineering is seeking an experienced 6502 and 65816 machine language programmer. 2 years minimum programming experience is required. We offer an exciting opportunity for the experienced programmer to take his skills to the limit. Applied Engineering offers an excellent compensation package including paid vacations, 11 paid holidays per year, health insurance program and more.

Applied Engineering's location in the suburbs of north Dallas offers a "buyer's market" for housing, as well as excellent schools, shopping and entertainment.

Successful applicant should have heavy machine language experience on the Apple IIe and IIgs as well as familiarity with AppleWorks. We're the best at what we do; if you are too, please send your resume to:

Applied Engineering
P.O. Box 5100
Carrollton, TX 75011
Attn: Personnel

WANTED

PROMGRAMER™

Hardware design by Bob Bice

Software by Bob Sander-Cedeno

The PROMGRAMER is an inexpensive EPROM (Erasable Programmable Read Only Memory) programmer for the APPLE II+, and IIe computers. The unit plugs into any slot of the computer, and allows the user to program any standard 5 volt, 27 series EPROM. Although not intended as a production tool, the ease of use allows rapid programming, copying, duplication, or modification of EPROMs.

FEATURES:

- Programs all single-voltage 27XXX series EPROMs from the 2708 to the 27512.
- Also does CMOS versions of above EPROMs.
- Choice of standard or fast programming algorithm.
- Disk-based software allows easy customization.
- Source code included (S-C Assembler).
- Files can be saved under standard DOS 3.3.
- On-board voltage generator--no messy wires or transformers.
- Zero-insertion force (ZIF) socket for ease in changing EPROMs.
- Complete--no personality modules needed.
- Slot independent for maximum flexibility.

SOUTHERN CALIFORNIA RESEARCH GROUP

S-C Software Corporation

2331 Gus Thomasson, Suite 125, P.O. Box 280300, Dallas, Texas 75228 (214) 324-2050

Our software is not only unlocked and fully copyable...we often provide the complete source code on disk, allowing you to understand how it works and make your own personal extensions.

S-C Cross Reference Utility: A support program which works with the S-C Macro Assembler to generate an alphabetized listing of all labels in an assembly language source program, showing with each label the line number where it is defined and all the line numbers containing references to that label. Works with multi-file source programs. The ProDOS version automatically follows the .IN or .INB ("include file") directives and assigns a code-letter to each included file in the cross reference listing. We include the complete source code for this amazingly fast program, in both DOS and ProDOS versions, for only \$50.

S-C DisAssembler: An intelligent two-pass disassembler which works in conjunction with the ProDOS version of the S-C Macro Assembler. Converts files of 6502 or 65C02 machine language programs into meaningful source code files complete with labels. Driven by a script you write in a convenient "disassembly" language, allowing you to define label names of your choice, process all or part of input files, and many other options. A built-in option produces comment lines before each label showing all reference points to that label. Comes with complete commented source code and easy-to-understand documentation, for only \$50.

Laumer Research Full Screen Editor: Integrates with the built-in line-oriented editor in the S-C Macro Assembler to provide a powerful full-screen editor for your assembly language source files. Drivers for Videx, STB80, and Apple //e (and //c) 80-column boards are included, as well as a 40-column version. Comes with complete source code, with both DOS and ProDOS versions, for only \$49.

ProVIEW, by Doug McComsey: A professional tool for "zapping" ProDOS disks and memory. On an Apple //c, //e, or //gs in 80-column mode ProVIEW gives you a 256-byte window into RAM, ROM, ProDOS files, and disk blocks. You can examine, modify, and update any portion of RAM, a file, or a disk block. Operates from a series of menu screens much like Appleworks, with complete HELP available at every step. Only \$20!

S-C Word Processor: The one we use for manuals, letters, our monthly newsletter, and whatever. 40-columns only (II and II+ require lower-case display mod and shift-key mod). Works with standard DOS text files, but at super fast speeds (100 sectors in 7 seconds). No competition for Word Perfect, but you get complete source code! \$50, DOS only. We also will include at no extra charge two additional disks. One contains the source and object code for an 80-column //e, //c, //gs. We use this one, but it has a few cosmetic un-features. Sorry, still only in DOS 3.3. The other disk contains the source and object code for a 40-column version which runs under ProDOS, on any ProDOS machine. We use both of these, but they are not as polished as the original. All for only \$50!

S-C Double Precision Supplement for Applesoft: Two complete packages for the price of one! Only \$50 gets you the complete commented source code for both DFPF and DP18, with documentation. DFPF gives you 21-digit floating point precision for INPUT, PRINT, and +*/ operations. DP18 is more complete, offering all of the math operations and functions, formatted I/O conversions, and 18-digit decimal floating point. DP18 is ideal for writing programs which deal with money. Both packages use the "&" command in Applesoft to add the new capabilities, without losing any of the old. DP18 includes both DOS and ProDOS versions, DFPF is DOS only. Detailed internal documentation for DP18 was published in 12 consecutive issues of "Apple Assembly Line", and is available for an additional \$18.

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$18 per year in the USA, sent Bulk Mail; add \$3 for First Class postage in USA, Canada, and Mexico; add \$14 postage for other countries. Back issues are \$1.80 each (other countries inquire for postage). A subscription to the newsletter and the Monthly Disk containing all source code is \$64 per year in the US, Canada and Mexico, and \$87 to other countries.

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)

Apple

\$2.40



Assembly

Line

Volume 8 -- Issue 4

January, 1988

Peeking Inside AppleWorks 1.3	2
Interpretive String Display Subroutine	2
Overhauling the S-C Program Selector	21
Special Version of S-C Macro for Huge Symbol Tables	31
"IIgs Toolbox Reference" Now Available	32

New Subscription Rates

For the first time since January of 1984, we are going to have to increase our subscription rates. The Post Office is raising the postage again, for the third time since then, and we have to respond. Of course postage is not the only expense that has increased, just the most recent, and most noticeable. Here are the old and new rates for a year's subscription:

Newsletter Only, 12 issues:	Current	New
Bulk Mail (USA only)	\$18	---
First Class Mail (USA, Canada, Mexico)	\$21	\$24
All Other Countries	\$32	\$36
Newsletter plus the Monthly Disk, 12 issues:		
USA, Canada, Mexico	\$64	\$64
All Other Countries	\$87	\$90

You will notice that the Bulk Mail option is being phased out, as I am planning to mail by First Class to all USA subscribers. The reliability of Bulk Mail has been entirely too erratic, and increasingly so.

You will also notice that the price of a subscription including disk, delivered in the USA, Canada, or Mexico, has not increased at all. Maybe now is the time to upgrade, and save yourself a lot of typing?

The new prices go into effect for new subscriptions as of March 15, 1988. Renewals at the old prices (but no more bulk mail) will continue to be accepted through the 4th of April.

At least twice in the last eight years I have published fancy message display subroutines. In the original Volume 1, Number 1, way back in October, 1980, I gave a really nice 40-column version. That one was actually used in a slightly different form inside a system that we developed for the American Heart Association for teaching Cardio-Pulmonary Resuscitation (CPR).

In the April 1987 issue I published an 80-column version for the //e and //c, which had similar but more powerful capabilities.

Last month I started revealing some of the code from inside AppleWorks (version 1.3). I covered parameter passing and some string handling subroutines, plus block move. I also described and printed the subroutine which polls the keyboard, so that you can type before being asked. POLL.KEYBOARD is called from all over everywhere, just to be sure no characters are lost. I mention that, because there is a call to it from the code I am unveiling this month. In the listing which follows, I have put a dummy POLL.KEYBOARD subroutine which simply does an RTS. In the real AppleWorks code, DISPLAY.STRING calls the real POLL.KEYBOARD.

As I mentioned last month, I am NOT showing in these pages the exact code you would find inside AppleWorks. Since my purpose is not to document AppleWorks, but rather to cull out generally useful code which we can adapt and use, I have rearranged and modified a little. My versions are, in general, shorter and faster. Maybe whoever is currently maintaining AppleWorks at Claris will notice and use these improvements.

In case you ARE interested in documenting AppleWorks, or just want to see what I changed, I have included comment lines with each segment of code which show what the corresponding address was inside AppleWorks 1.3. As was true last month, all the code shown here is found in the main segment called APLWORKS.SYSTEM, which begins at \$1000 once it is up and running.

I will begin with a general description of features. DISPLAY.STRING began at \$14D1, and there are numerous calls to it in the code. There is also a JMP vector to it in the JMP table which begins at \$1000; if you find any JSR \$1015 instructions in any of the other segments, they are calling this DISPLAY.STRING subroutine. Unlike all of the subroutines I discussed last month, this subroutine does not expect to find parameters following the JSR which called it. Instead, it expects the length of the string to be in the A-register, and the address of the string to be in locations \$80,81. (There is another subroutine which uses the parameter-passing protocol to display a string which starts with a length byte; it simply sets up \$80, \$81, and the A-register and calls DISPLAY.STRING. You can find it at \$2093, with a JMP vector at \$1087.)

S-C Macro Assembler Version 2.0	DOS \$100, ProDOS \$100, both for \$120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 Owners	\$20
ProDOS Upgrade Kit for Version 2.0 DOS owners	\$30
Source Code of S-C Macro 2.0 (DOS or ProDOS)	each, additional \$100
S-C DisAssembler (ProDOS only)	without source code \$30, with source \$50
RAK-Ware DIASAM (DOS only)	without source code \$30, with source \$50
ProVIEW (ProDOS only)	\$20
Full Screen Editor for S-C Macro (with complete source code)	\$49
S-C Cross Reference Utility	without source code \$20, with source \$50
S-C Word Processor (with complete source code)	\$50
DP18 and DPFF, including complete source and object code	\$50
ES-CAPE (Extended S-C Applesoft Program Editor), including Version 2.0 With Source Code	\$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)	\$30
Bag of Tricks 2 (Quality Software)	(\$49.95) \$45
S-C Documentor (complete commented source code of Applesoft ROMs)	\$50
Cross Assemblers for owners of S-C Macro Assembler	\$32.50 to \$50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051, 8085, 1802/4/5, FDP-11, GI1650/70, Mitsubishi 50740, others)	
Beagle Bros. Applesoft Compiler (ProDOS only)	(\$74.95) \$65
Copy II Plus (Central Point Software)	(\$39.95) \$30
AAL Quarterly Disks	each \$15, or any four for \$45

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Vinyl disk pages, 6"x8.5", hold two disks each	10 for \$6
Diskette Mailing Protectors (hold 1 or 2 disks)	40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.)	or \$25 per 100
Envelopes for Diskette Mailers	6 cents each

Sider 20 Meg Hard Disk, includes controller & software	(\$695) \$550 +
Sider 40 Meg Hard Disk, includes controller & software	(\$995) \$860 +

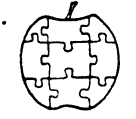
Minuteman 250 Uninterruptible Power Supply	(\$359) \$320 +
Minuteman 300 Uninterruptible Power Supply	(\$549) \$490 +

65802 Microprocessor, 4 MHz (Western Design Center)	\$25
quikLoader EPROM System (SCRG)	(\$179) \$170
PROMGRAMMER (SCRG)	(\$149.50) \$140

"Exploring the Apple IIgs"	Gary B. Little	(\$22.95) \$21
"Apple IIgs Technical Reference"	Michael Fischer	(\$19.95) \$19
"65816/65802 Assembly Language Programming"	Michael Fischer	(\$21.95) \$20
"Programming the 65816"	David Eyes & Ron Lichty	(\$22.95) \$21
"Apple //e Reference Manual"	Apple Computer	(\$24.95) \$23
"Apple //c Reference Manual"	Apple Computer	(\$24.95) \$23
"ProDOS-8 Technical Reference Manual"	Apple Computer	(\$29.95) \$27
"ProDOS-16 Technical Reference Manual"	Apple Computer	(\$29.95) \$27
"Apple IIgs Firmware Reference"	Apple Computer	(\$24.95) \$23
"Apple IIgs Hardware Reference"	Apple Computer	(\$24.95) \$23
"ProDOS Inside and Out"	Dennis Doms & Tom Weishaar	(\$16.95) \$16
"DOSstalk Scrapbook"	Tom Weishaar & Bert Kersey	(\$14.95) \$14
"Beneath Apple ProDOS"	Don Worth & Pieter Lechner	(\$19.95) \$18
"Beneath Apple DOS"	Don Worth & Pieter Lechner	(\$19.95) \$18
"Inside the Apple //c"	Gary B. Little	(\$19.95) \$18
"Inside the Apple //e"	Gary B. Little	(\$19.95) \$18
"Understanding the Apple //e"	Jim Sather	(\$24.95) \$23
"Understanding the Apple II"	Jim Sather	(\$22.95) \$21
"Apple II+/IIe Troubleshooting & Repair Guide"	Brenner	(\$19.95) \$18
"Assembly Language for Applesoft Programmers"	Finley & Myers	(\$18.95) \$18
"Now That You Know Apple Assembly Language"	Jules Gilder	(\$19.95) \$18
"Enhancing Your Apple II, vol. 1"	Don Lancaster	(\$15.95) \$15
"Enhancing Your Apple II, vol. 2"	Don Lancaster	(\$17.95) \$17
"Assembly Cookbook for the Apple II/IIe"	Don Lancaster	(\$21.95) \$20
"Microcomputer Graphics"	Roy E. Myers	(\$14.95) \$14
"Assembly Lines -- the Book"	Roger Wagner	(\$19.95) \$12

* These items add \$2 for first item, \$.75 for each additional item for US shipping.
 + Inquire for shipping cost.
 Customers outside USA inquire for postage needed.
 Texas residents please add 8% sales tax to all orders.
 << Master Card, VISA, Discover and American Express >>

S-C Software Corporation
 2331 Gus Thomasson #125
 DALLAS, TX 75228
 Phone 214-324-2650



DISPLAY.STRING does not use any Apple firmware at all. The display techniques used here work faster than the firmware, because they are dedicated to 80-columns and do not have to retain any compatibility with older machines. If you remember that the original Apple //e firmware scrolled the 80-column screen with a slow zigzag motion, you can see why Rupert Lissner decided to code his own.

The characters within the string to be displayed consist of control codes and displayable characters. Displayable characters include the full upper- and lower-case alphabet, numbers, and punctuation signs; all of these can be displayed in both normal and inverse mode. All 32 "mouse-text" characters can also be displayed, although the only one I have noticed in quickly scanning through the AppleWorks messages is the open-apple.

Inverse and normal mode is controlled by a flag byte, which I have called INVERSE.FLAG in my code. It is located at \$14D0 in AppleWorks. If that byte contains \$00, characters will display in inverse; if \$80, normal. A pair of control codes lets you switch INVERSE.FLAG back and forth from within a string, or you can directly set it between calls to DISPLAY.STRING.

The following table shows the hex values for the various character groups as interpreted by DISPLAY.STRING:

00-1F	Control Codes
20-7F	96 ASCII Characters
80-9F	32 Mouse Text Characters

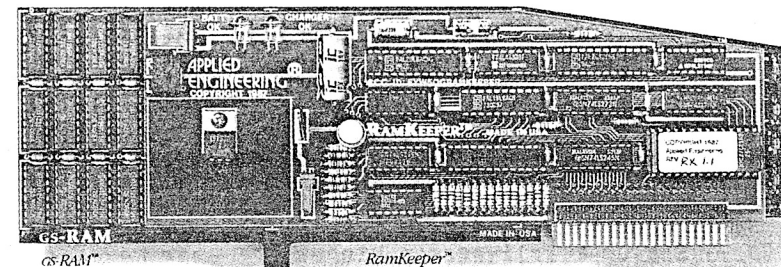
The codes 20-7F display in either normal or inverse, depending on INVERSE.FLAG, as described above. Codes C0-DF duplicate 80-9F, displaying the mouse text characters; codes A0-BF and E0-FF both display the 32 lower-case characters in inverse mode.

There are a couple of un-features in DISPLAY.STRING. There is a JMP \$1815 instruction located at \$1815 inside AppleWorks 1.3. This, of course, hangs up the Apple. The only way out is by hitting RESET. DISPLAY.STRING goes to this HANG.UP code under some circumstances. Since it is a deadly trap, I assume the author of AppleWorks used to have some debugging code there. It gets called from all over, when errors occur that are programming bugs. There is even a JMP vector to it in the JMP table, at \$101B! I have left the jumps and branches to HANG.UP in my version, but you might want to modify them to do something reasonable if you are going to use DISPLAY.STRING yourself.

The other un-feature is what happens if you try to print past the right edge of the text window. You might think it would automatically wrap to the next line, like the standard Apple firmware does; no, it just piles up the characters at the end of the line like old manual typewriters used to do. Possibly you might consider a third un-feature to be the limitation of 255 characters in a string, but this is easy enough to work around. My demonstration program, included at the end of the following listing, shows one way.

RamKeeper™

For the "Instant On" Apple IIGs.



Permanent Storage with an "Electronic Hard Disk"

Now when you turn on your IIGs your favorite program can appear on screen in just a few seconds! With RamKeeper, your IIGs memory card will retain stored programs and stored data while your IIGs is turned off. RamKeeper allows you to divide your IIGs memory into part "electronic hard disk" and part RAM for your programs workspace—in almost any way you want and at anytime you want. GS-RAM, GS-RAM Plus, Apple IIGs memory card and most other IIGs memory cards are compatible with RamKeeper.

Supports Up to Two IIGs Memory Cards at the Same Time

If you bought your IIGs with Apple's memory card and later wished you had the GS-RAM, no problem. RamKeeper will support both cards plugged into RamKeeper simultaneously!

How it Works

Just unplug your IIGs memory card



Steve Wozniak, the creator of Apple Computer

"I've purchased several Applied Engineering products over the years. They're always well made and performed as advertised. I recommend them wholeheartedly."

from your computer, plug your IIGs memory card into RamKeeper, plug RamKeeper into the IIGs memory slot. If you have another IIGs memory card, an additional card socket on RamKeeper will accommodate your second card. That's all there is to it!

Reliability from the Most Experienced

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple so you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, reliability is everything! That's why Applied Engineering uses the more dependable Gel-Cells instead of Ni-Cad batteries (if Ni-Cad's aren't discharged periodically, they lose much of their capacity). RamKeeper has close to 6 times (about 6 hours) the "total power failure" back-up time of other systems. When power returns, RamKeeper automatically recharges the battery to a full charge. With power from your wall outlet, RamKeeper will back-up your IIGs memory cards RAM indefinitely.

RamKeeper Has and Does It All!

- Allows instant access to your programs without slow disk delays
- Configure Kilobytes or Megabytes of instant ROM storage for your favorite programs

- Reduces power strain to your internal IIGs power supply
- Contains back-up status LED's
- Can support up to two IIGs memory cards simultaneously
- Supports both 256K installed memory chip boards like GS-RAM and the Apple IIGs Memory Expansion Card as well as 1 MEG installed memory chip boards like GS-RAM Plus
- 5-year hassle-free warranty
- 15 day money back guarantee
- Proudly made in the U.S.A.
- RamKeeper comes complete with battery, software and documentation

Only \$179.00!

(GS-RAM card shown in photo not included)

Order Your RamKeeper Today!

See your dealer or call (214) 241-6060, 9:00-11:00 CST, 7 days a week, or send check or money order to Applied Engineering MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 if outside U.S.A.

AE APPLIED ENGINEERING™

The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

Prices subject to change without notice

There are 17 control codes interpreted by DISPLAY.STRING, and room for adding 15 more. Most of these are single byte codes, but two are followed by parameter bytes. Here is a table of the codes:

```
01 -- Clear from cursor to end of line.
02 -- Clear entire line cursor is on.
03 -- Home (go to 0,0 and clear window).
04 -- Clear from cursor to end of window.
05xxyy -- Move cursor to column xx, row yy of window
          (HANG.UP if string ends with no xyy).
06 -- Move cursor left (HANG.UP if beyond window).
07 -- Move cursor right (HANG.UP if beyond window).
08 -- Move cursor up, scroll if already at top.
09 -- Move cursor down, scroll if already at bottom.
0A -- Set inverse mode.
0B -- Set normal mode.
0C -- Store current cursor position as bottom-right
      corner of the window.
0D -- Move cursor to beginning of current line.
0E -- Store current cursor position as top-left
      corner of the window.
0F -- Set up a full-screen window.
10 -- Beep! (the AppleWorks low-tone bell).
1lxx -- Slide the screen sideways xx columns.
        If xx>0 slide to right; if xx<0, slide left.
```

You can see that setting windows is fairly easy. Use code 05xxyy to position the cursor where you want the bottom-right corner of the new window to be, and then use code 0C to store it; then use 05xxyy to position the cursor where you want the top-left corner of the new window to be, and then use code 0E to make the new window. Since all 05xxyy moves are relative to the current window, you need to set the new bottom-right corner first. (You should now have a clue how AppleWorks nests the file folders on the screen.)

Since AppleWorks does not use any of the Apple firmware, it is also not tied to the standard page-zero locations for window info and cursor position. DISPLAY.STRING uses \$10 through \$13 to store the window definition. It is not the same as Apple's window definition bytes at \$20-\$23. Apple's firmware uses a starting column and a width, whereas AppleWorks uses a starting column and an ending column. The bytes are in a different order, too. See lines 1030-1130 for DISPLAY.STRINGs page zero-usage. The two bytes defined in lines 1120-1130, at \$F0 and \$F1, are the ones AppleWorks uses. However, you could change those two lines to share \$18 and \$19 with the labels defined on lines 1090-1100, if you wish.

Lines 1150-1320 define two macros, one for storing a byte on the screen and one for picking a byte off the screen. There is another macro definition in lines 4820-4840, for the function vector table. I decided to put these in as macros to shorten the program listing so it would fit in this issue of AAL. It also makes the code in the left-right scroll subroutine easier to follow. My code inside these macros is different from the

code in AppleWorks: it is 'shorter,' and on the average one cycle SLOWER. I more than made up for the speed loss in other places, though.

You will find the main body of DISPLAY.STRING in lines 1390-1920. Lines 1470-1480 are curious. They cause the entire call to DISPLAY.STRING to be ignored if the contents of \$A4 is non-zero. I don't know why or when AppleWorks would use this. Probably you will want to delete these two lines if you use a revision of DISPLAY.STRING in your own programs. In that case, you would want to substitute a LDA #0 instruction, so that line 1490 would store a zero as the beginning position in the string to be displayed. Line 1500 calls the POLL.KEYBOARD subroutine, which as I mentioned above is just a dummy routine in this listing. You will probably want to delete this line too.

By the time we get to line 1520, everything is set up. A pointer to the first character of the string is in \$80,81; POSITION.IN.STRING holds the index relative to that pointer for the next character to be processed; INVERSE.FLAG is either 00 or 80; and BYTES.IN.STRING is set to the string length. We come back to line 1520 for each character in the string, except for the parameter bytes on control codes 05 and 11.

Lines 1520-1540 test to see if we have finished the string, and go to an RTS if so. Lines 1550-1670 pick up the next character, and decide how to process it according to the range: values 80-FF are treated as mouse text, even though we only expect 80-9F for these; values 00-1F are control (function) codes; and values 20-7F are regular ASCII characters.

Mouse text characters are really put on the screen by using values from \$40 to \$5F, so lines 1680-1700 do the honors.

Regular ASCII characters are EORed with the INVERSE.FLAG in line 1600. If the result is negative, we have a "normal" character; if positive, "inverse". Normal characters are ready now to display, but inverse take some care. The range 40-5F should print as letters rather than mouse text, so they are mapped down to 00-1F in lines 1620-1670.

Control codes are handled in lines 1830-1920. This is not the same way AppleWorks did it. AppleWorks used the trick of pushing the table address on the stack and doing an RTS to effectively JMP to the function code; then each function code processor finished by doing a JMP \$14E1 (my line 1520, label .1). My code effectively does a JSR to the function code, so each processor can finish by doing an RTS. This saves space, but is a tiny bit slower. However, I made up for the speed loss by eliminating one unnecessary range check. AppleWorks tested the function code range to be sure it was no larger than \$11; if it was larger, AppleWorks jumped to HANG.UP. My code gives the same results, by merely extending the function code table in lines 4870-5190 to include all 32 vectors. The extra 28 bytes of table are more than saved elsewhere. By making the function code processors into subroutines which end with an RTS I have made them accessible to JSR calls from anywhere. This could save even more space.



SPECIAL !!! EXPANDED RAM/ROM BOARD: \$39.00

Similar to our \$30 RAM/ROM dev board described below. Except this board has two sockets to hold your choice of 2-2K RAM, 2-2K ROM or even 2-4K ROM for a total of 8K. Mix RAM and ROM too. Although Apple limits access to only 2K at a time, soft switches provide convenient socket selection. Hard switches control defaults.

IMPROVED !!! [I IN A MAC (ver 2.0): \$75.00

Now includes faster graphics, UniDisk support and more! Bi-directional data transfers are a snap! This Apple II emulator runs DOS 3.3/PRODOS (including 6502 machine language routines) on a 512K MAC or MACPLUS. All Apple II features are supported such as HI/LO-RES graphics, 40/80 column text, language card and joystick. Also included: clock, RAM disk, keyboard buffer, on-screen HELP, access to the desk accessories and support for 4 logical disk drives. Includes 2 MAC diskettes (with emulation, communications and utility software, plus DOS 3.3 and PRODOS system masters, including Applesoft and Integer BASIC) and 1 Apple II diskette.

SCREEN.GEN: \$35.00

Develop HI-RES screens for the Apple II on a Macintosh. Use MACPAINT (or any other application) on the MAC to create your Apple II screen. Then use SCREEN.GEN to transfer directly from the MAC to an Apple II (with SuperSerial card) or IIC. Includes Apple II diskette with transfer software plus fully commented SOURCE code.

MIDI-MAGIC for Apple //c: \$49.00

Compatible with any MIDI equipped music keyboard, synthesizer, organ or piano. Package includes a MIDI-out cable (plugs directly into modem port - no modifications required!) and 6-song demo diskette. Large selection of digitized QRS player-piano music available for 19.00 per diskette (write for catalog). MIDI-MAGIC compatible with Apple II family using Passport MIDI card (or our own input/output card w/drum sync for only \$99.00).

FONT DOWNLOADER & EDITOR: \$39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. Special functions (like expanded, compressed etc.) supported. Includes HIRES screen editor to create custom fonts and special graphics symbols. For Apple II, II+, IIe. Specify printer: Apple Imagewriter, Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/85, or Okidata 92/192.

* **FONT LIBRARY DISKETTE #1: \$19.00** contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

DISASM 2.2e : \$30.00 (\$50.00 with SOURCE Code)

Use this intelligent disassembler to investigate the inner workings of Apple II machine language programs. DISASM converts machine code into meaningful, symbolic source compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even provides label substitution. Address-based triple cross reference generator included. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his ASSEMBLY COOKBOOK.

The 'PERFORMER' CARD: \$39.00 (\$59.00 with SOURCE Code)

Converts a 'dumb' parallel printer I/F card into a 'smart' one. Simple command menu. Features include perforation skip, auto page numbering with data & title, large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), Okidata 82A/83A with Okigraph & Okidata 92/93.

'MIRROR' ROM: \$25.00 (\$45.00 with SOURCE Code)

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support.

RAM/ROM DEVELOPMENT BOARD: \$30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

C-PRINT For The APPLE //c: \$69.00

Connect standard parallel printers to an Apple //c serial port. Separate P/S included. Just plug in and print!

Unless otherwise specified, all Apple II diskettes are standard (not copy protected!) 3.3 DOS.

Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders OK.

RAK-WARE 41 Ralph Road W. Orange NJ 07052 (201) 325-1885



You can see that to add more functions you merely have to write the processing subroutines and enter the vectors in the function code table using the >VEC macro. I can think of several neat additions. For example, I might use code 00 to initialize full screen, home, normal mode all in one code. It might also be useful to add a code to draw a file folder in the current window. A single code could shrink the window one notch, clear it, and draw a folder. Another code could pop back out to the next larger window. Let your imagination and creativity loose!

I wrote a little demonstration program, shown in lines 6040-6270. This program steps through a list of strings. Each string starts with a length byte. When a length byte of 00 is found, the demonstration stops. I have listed the demonstration strings in raw form to save paper, in lines 6290-6970. The demonstration is not too fancy, but it is fun. I display some characters, then move them around the screen in all four directions, with intermittent beeps to slow it down enough to see. Then I wipe it clean and display the entire character set.

Let me know how you like this series of articles, and what kind of uses you find for the code. If you come up with some really great new function codes for DISPLAY.STRING, send them in and we'll share them in future issues.

```
1010 #SAVE AW.SUBS.2
1020 -----
10- 1030 AW.LEFT .EQ $10 DEFINES CURRENT WINDOW
11- 1040 AW.TOP .EQ $11 "
12- 1050 AW.RIGHT .EQ $12 "
13- 1060 AW.BOTTM .EQ $13 "
14- 1070 AW.CH .EQ $14 CURSOR HORIZONTAL
15- 1080 AW.CV .EQ $15 CURSOR VERTICAL
16- 1090 AW.BASE .EQ $16,17
18- 1100 AW.BASE2 .EQ $18,19
1110 -----
FO- 1120 SHUFFLE.SOURCE .EQ $F0
F1- 1130 SHUFFLE.DEST .EQ $F1
1140 -----
1150 .MA ST.SCRN
1160 LSR LSB into Carry
1170 TAY Index into Y-reg
1180 PLA Get char again
1190 BCS :1 ...Odd character, in Main RAM
1200 STA $C055 ...Even character, in Aux RAM
1210 :1 STA (AW.BASE),Y
1220 STA $C054
1230 .EM
1240 -----
1250 .MA LD.SCRN
1260 LSR LSB into Carry
1270 TAY Index into Y-reg
1280 BCS :1 ...Odd character, in Main RAM
1290 STA $C055 ...Even character, in Aux RAM
1300 :1 LDA (AW.BASE),Y
1310 STA $C054
1320 .EM
1330 -----
0800- 1340 INVERSE.FLAG .BS 1 00=display chars inverse, 80=display
0801- 1350 BYTES.IN.STRING .BS 1 chars normal
0802- 1360 POSITION.IN.STRING .BS 1
0803- 1370 SCROLL.DIRECTION .BS 1
```

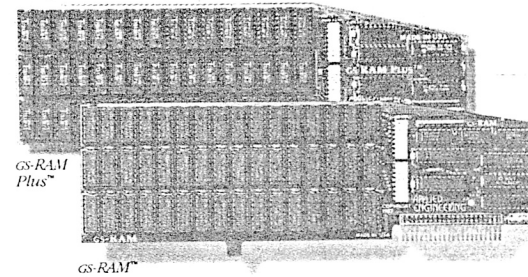
```

1380 * DISPLAY A STRING WITH FUNCTION CODES
1390 * (A) = # BYTES IN STRING
1400 * ($80,$1) = Address of string
1410 * ($A4) = flag: if 00 display, else ignore.
1420 * at $14D1 in AppleWorks 1.3
1430 * -----
1440 *
1450 DISPLAY.STRING
0804- 8D 01 08 1460 STA BYTES.IN.STRING
0807- A5 A4 1470 LDA $A4 If non-zero, don't display anything
0809- D0 5B 1480 BNE .99 ...to an RTS
080B- 8D 02 08 1490 STA POSITION.IN.STRING
080E- 20 3A 0A 1500 JSR POLL.KEYBOARD GET KEY IF ANY
1510 * -----
0811- AC 02 08 1520 .1 LDY POSITION.IN.STRING
0814- CC 01 08 1530 CPY BYTES.IN.STRING
0817- B0 4D 1540 BCS .99 ...to an RTS
0819- EE 02 08 1550 INC POSITION.IN.STRING
081C- E1 80 1560 LDA ($80),Y GET CHAR FROM STRING
081E- 30 15 1570 BMI .2 ...80-FF, Mouse char for screen
0820- C9 20 1580 CMP #$20
0822- 90 32 1590 BCC .5 ...00-1F
0824- 4D 00 08 1600 EOR INVERSE.FLAG
0827- 30 10 1610 BMI .3 ...Normal Char
0829- C9 40 1620 CMP #$40 ...Inverse char
082B- 90 0C 1630 BCC .3 ...00-3F, inverse A-Z, 0-9, punctuation
082D- C9 60 1640 CMP #$60
082F- B0 08 1650 BCS .3 ...60-7F, inverse a-z, punctuation
0831- 29 BF 1660 AND #$BF map 40-5F into 00-1F, more A-Z
0833- 90 04 1670 BCC .3 ...always
1680 *---Display a Mouse Char---
0835- 29 7F 1690 .2 AND #$7F ...map into 40-5F, mouse char range
0837- 09 40 1700 ORA #$40
1710 *---Display Char in A-register---
0839- 48 1720 .3 PHA Save the character
083A- 20 3B 0A 1730 JSR BASE.CALC.CV
083D- A5 14 1740 LDA AW.CH Char position on line
083F- 1750 >ST.SCRN Store character on screen
1760 * -----
084C- A6 14 1770 LDX AW.CH
084E- E0 4F 1780 CPX #79 End of line yet?
0850- B0 BF 1790 BCS .1 ...yes, stick there, pile em' up
0852- E6 14 1800 INC AW.CH ...no, advance CH
0854- D0 BB 1810 BNE .1 ...always
1820 * -----
0856- 20 5C 08 1830 .5 JSR .6 CALL THE FUNCTION
0859- 4C 11 08 1840 JMP .1 ...ALWAYS
1850 * -----
085C- 0A 1860 .6 ASL Convert code to index
085D- AA 1870 TAX
085E- BD FB 09 1880 LDA FUNTEL+1,X
0861- 48 1890 PHA
0862- BD FA 09 1900 LDA FUNTEL,X
0865- 48 1910 PHA
0866- 60 1920 RTS
1930 * -----
1940 * FUNCTION 03 -- CLEAR ENTIRE WINDOW, CURSOR TO TOP-LEFT
1950 * at $154A in AppleWorks 1.3
1960 * -----
1970 FUN.HOME
0867- A5 10 1980 LDA AW.LEFT
0869- 85 14 1990 STA AW.CH
086B- A5 11 2000 LDA AW.TOP
086D- 85 15 2010 STA AW.CV
2020 * -----
2030 * FUNCTION 04 -- CLEAR REST OF WINDOW
2040 * at $1552 in AppleWorks 1.3
2050 * -----
2060 FUN.CLR.CH.TO.BOS
086F- A5 14 2070 LDA AW.CH SAVE CH AND CV
0871- 48 2080 PHA
0872- A5 15 2090 LDA AW.CV
0874- 48 2100 PHA
0875- 20 5D 0A 2110 .1 JSR CLR.CH.TO.EOL
0878- A5 15 2120 LDA AW.CV
087A- C5 13 2130 CMP AW.BOTTOM
087C- B0 08 2140 BCS .2 ...THAT WAS THE BOTTOM LINE
087E- E6 15 2150 INC AW.CV
0880- A5 10 2160 LDA AW.LEFT
0882- 85 14 2170 STA AW.CH
0884- 90 EF 2180 BCC .1 ...ALWAYS
0886- 68 2190 .2 PLA

```

For Those Who Want the Most. From Those Who Make the Best. GS-RAM™

Now expand the IIgs™ RAM and ROM with up to 8 MEG of "Instant On" memory with the all new GS-RAM!



GS-RAM has an all new design. A design that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

More Sophisticated, Yet Easier to Use

GS-RAM works with all IIgs software. In fact any program that runs on Apple's smaller memory card runs on the GS-RAM. But with GS-RAM you can have more memory, improved performance, and almost unlimited expansion capabilities. We've designed the new GS-RAM to be easier to use too—you don't have to adjust the size of your RAM disk every time you use a DMA device. No other RAM card with more than 4 banks of memory installed can make the same claim.

More than Just Hardware

Each GS-RAM and GS-RAM Plus includes the most powerful set of IIgs software enhancements available anywhere. In fact, our nearest competitor offers only a fraction of the invaluable programs that we include with each GS-RAM card. This software includes the most powerful disk-caching program available, the GS-RAM Cache. The Cache will make most of your applications run up to 7 times faster!

Also included is a diagnostic utility that lets you test your GS-RAM by graphically showing the location of any bad or improperly installed RAM chips. And for AppleWorks users, we give you our exclusive Expander program that dramatically enhances both the capabilities and speed of AppleWorks.

Making AppleWorks Even Better

Applied Engineering's Expander program eliminates AppleWorks internal memory limits allowing it to recognize up to 8 megabytes of desktop workspace. You can increase the limits from only 7,250 lines to 22,600 lines in the word processor and from 6,350 records to 22,600 records in the database. The Expander allows all of AppleWorks, including print functions, to automatically load into RAM. The clipboard size will increase from 255 to 2,042 lines maximum. GS-RAM will automatically segment larger files so you can save them onto multiple floppies. And

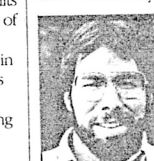
GS-RAM provides a built-in print buffer that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

Grow by Kilobytes or Megabytes

We offer GS-RAM in two configurations so you can increase your memory 256K at a time (GS-RAM) or a megabyte at a time (GS-RAM Plus). Both are IIgs compatible and both come with our powerful enhancement software. GS-RAM can hold up to 1.5 MEG of 256K chips and GS-RAM Plus can hold up to 6 MEG using 1 MEG chips. And since both use standard RAM chips (not high-priced SIMM's), you'll find expanding your GS-RAM or GS-RAM Plus easy, convenient, and very economical. For further expansion, you can plug a 2 MEG "piggyback" card into the GS-RAM's expansion port for up to 3.5 MEG of total capacity. Or up to a whopping 8 MEG on GS-RAM Plus. If a GS-RAM owner outgrows 3.5 MEG, he can easily upgrade to GS-RAM Plus for a nominal charge.

Permanent Storage for an "Instant On" Apple

With our RamKeeper™ back-up option, your GS-RAM or GS-RAM Plus will retain both programs and data while your IIgs is turned off! Now when you turn your IIgs back on, your favorite software is on your screen in under 4 seconds! With RamKeeper you can divide your IIgs memory into part "electronic hard disk" and part extended RAM. Even change the memory boundaries at any time—and in any way—you want. Because



Steve Wozniak, the creator of Apple Computer

"In quality, performance, compatibility, expandability and support, Applied Engineering's GS-RAM and GS-RAM Plus are number one."

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple, you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, Reliability is everything. That's why Applied Engineering uses state-of-the-art "GEL-CELL's" instead of Ni-Cad batteries (if Ni-Cads aren't discharged periodically, they lose much of their capacity). RamKeeper has about 6 hours of "total power failure" back-up time. That's 6 times the amount of other systems. But with power from your wall outlet, RamKeeper will back-up GS-RAM, GS-RAM Plus, or most other IIgs memory cards indefinitely. Should you ever have a "total power failure," RamKeeper switches to its 6-hour battery. When power returns, RamKeeper will automatically recharge the battery to full power. RamKeeper incorporates a dual-rate charger, status LED's, and advanced power reducing circuitry. RamKeeper comes complete with battery, software, and documentation.

GS-RAM's Got it ALL!

- 5-year warranty — parts & labor
- 6 RAM banks (most cards have 4)
- Memory expansion port
- ROM expansion port
- Ultra-fast disk caching on ProDOS 8 AND ProDOS 16.
- Expands AppleWorks internal limits
- Includes hi-res self test
- No soldered-in RAM chips
- Expandable to 8 MEG
- No configuration blocks to set
- RamKeeper back-up option allows permanent storage of programs & data
- 15-day money-back guarantee
- Proudly made in the USA.

GS-RAM with 256K	\$189
GS-RAM with 512K	\$259
GS-RAM with 1 MEG	\$399
GS-RAM with 1.5 MEG	\$539
GS-RAM with 2.5 to 3.5 MEG	CALL
GS-RAM Plus with 1-8 MEG	CALL
RamKeeper Option	\$179

Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside USA.

AE APPLIED ENGINEERING™
The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006
Prices subject to change without notice.

GS-RAM, GS-RAM Plus and RamKeeper are trademarks of Applied Engineering. Other brands and product names are registered trademarks of their respective holders.

```

887- 85 15 2200 STA AW.CV
889- 68 2210 PLA
88A- 85 14 2220 STA AW.CH
88C- 60 2230 RTS
2240 *-----*
2250 * FUNCTION 05 -- GO TO X,Y IN THIS WINDOW
2260 * Equivalent to HTAB X, VTAB Y
2270 * 05 XX YY in string
2280 * at $1576 in AppleWorks 1.3
2290 *-----*
2300 FUN.GOTO.XY
2310 LDY POSITION.IN.STRING
2320 LDA ($80),Y
2330 INY
2340 CPY BYTES.IN.STRING
2350 BCS FUN.HANG.UP
2360 ADC AW.LEFT
2370 STA AW.CH
2380 LDA ($80),Y
2390 ADC AW.TOP
2400 STA AW.CV
2410 INY
2420 STY POSITION.IN.STRING
2430 RTS
2440 *-----*
2450 * FUNCTION 06 -- BACK UP CURSOR
2460 * at $1593 in AppleWorks 1.3
2470 *-----*
2480 FUN.CURSOR.LEFT
2490 LDA AW.CH
2500 CMP AW.LEFT
2510 BEQ FUN.HANG.UP
2520 DEC AW.CH
2530 RTS
2540 *-----*
2550 * at $1599 and $1815 in AppleWorks 1.3
2560 FUN.HANG.UP JMP FUN.HANG.UP
2570 *-----*
2580 * FUNCTION 07 -- CURSOR RIGHT
2590 * at $15A1 in AppleWorks 1.3
2600 *-----*
2610 FUN.CURSOR.RIGHT
2620 LDA AW.CH
2630 CMP AW.RIGHT
2640 BEQ FUN.HANG.UP
2650 INC AW.CH
2660 RTS
2670 *-----*
2680 * FUNCTION 08 -- CURSOR UP (SCROLL DOWN IF NECESSARY)
2690 * at $15AB in AppleWorks 1.3
2700 *-----*
2710 FUN.CURSOR.UP
2720 LDA AW.CV
2730 CMP AW.TOP
2740 BEQ .1 ...ALREADY AT TOP, SCROLL DOWN
2750 DEC AW.CV
2760 RTS
2770 .1 LDA AW.BOTTOM
2780 LDX #-1
2790 BNE SCROLL ...ALWAYS
2800 *-----*
2810 * FUNCTION 09 -- CURSOR DOWN (SCROLL UP IF NECESSARY)
2820 * at $15BC in AppleWorks 1.3
2830 *-----*
2840 FUN.CURSOR.DOWN
2850 LDA AW.CV
2860 CMP AW.BOTTOM
2870 BEQ .1 ...ALREADY AT BOTTOM, SCROLL UP
2880 INC AW.CV
2890 RTS
2900 .1 LDA AW.TOP
2910 LDX #1
2920 SCROLL
2930 STX SCROLL.DIRECTION 01 IF UP, FF IF DOWN
2940 PHA SAVE LINE NUMBER
2950 JSR BASE.CALC.A
2960 .1 LDA AW.BASE
2970 STA AW.BASE2
2980 LDA AW.BASE+1
2990 STA AW.BASE2+1
3000 PLA GET LINE NUMBER AGAIN
3010 CMP AW.CV IS IT THE LAST LINE?

```

```

08EA- F0 2A 3020 BEQ CLR.LINE
08EC- 18 3030 CLC
08ED- 6D 03 08 3040 ADC SCROLL.DIRECTION
08FO- 48 3050 PHA SAVE SOURCE LINE NUMBER
08F1- 20 3D 0A 3060 JSR BASE.CALC.A
08F4- A5 10 3070 LDA AW.LEFT
08F6- AA 3080 TAX
08F7- 4A 3090 LSR
08F8- A8 3100 TAX
08F9- B0 0F 3110 BCS .3 START WITH ODD COLUMN
08FB- 8D 55 03 3120 .2 STA $C055 EVEN COLUMNS IN AUX MEM
08FE- B1 16 3130 LDA (AW.BASE),Y
0900- 91 18 3140 STA (AW.BASE2),Y
0902- 8D 54 03 3150 STA $C054 BACK TO MAIN MEM
0905- E4 12 3160 CPX AW.RIGHT
0907- F0 D6 3170 BEQ .1 ...END OF THIS LINE
0909- E8 3180 INX
090A- B1 16 3190 .3 LDA (AW.BASE),Y
090C- 91 18 3200 STA (AW.BASE2),Y
090E- E4 12 3210 CPX AW.RIGHT
0910- F0 CD 3220 BEQ .1 ...END OF THIS LINE
0912- E8 3230 INX
0913- C8 3240 INY
0914- D0 E5 3250 BNE .2 ...ALWAYS
3260 *-----*
3270 * FUNCTION 02 -- CLEAR ENTIRE CURRENT LINE
3280 * at $1540 in AppleWorks 1.3
3290 *-----*
3300 FUN.CLR.LINE
3310 LDA AW.LEFT
3320 STA AW.CH
3330 *-----*
3340 * FUNCTION 01 -- CLEAR REST OF CURRENT LINE
3350 * at $1544 in AppleWorks 1.3
3360 *-----*
3370 FUN.CLR.CH.TO.EOL
3380 JMP CLR.CH.TO.EOL
3390 *-----*
3400 * FUNCTION 0A -- INVERSE
3410 * at $160B in AppleWorks 1.3
3420 *-----*
3430 FUN.INVERSE
3440 LDA #$00
3450 .HS 2C SKIP OVER LDA #$80
3460 *-----*
3470 * FUNCTION 0B -- NORMAL
3480 * at $160F in AppleWorks 1.3
3490 *-----*
3500 FUN.NORMAL
3510 LDA #$80
3520 STA INVERSE.FLAG
3530 RTS
3540 *-----*
3550 * FUNCTION 0C -- DEFINE BOTTOM RIGHT CORNER
3560 * at $1617 in AppleWorks 1.3
3570 *-----*
3580 FUN.CORNER.BR
3590 LDA AW.CH
3600 LDX AW.CV
3610 SET.CORNER.BR
3620 STA AW.RIGHT
3630 STX AW.BOTTOM
3640 RTS
3650 *-----*
3660 * FUNCTION 0D -- CURSOR TO BEGINNING OF LINE
3670 * Equivalent to RETURN without LINEFEED
3680 * at $1622 in AppleWorks 1.3
3690 *-----*
3700 FUN.CURSOR.BOL
3710 LDA AW.LEFT
3720 STA AW.CH
3730 RTS
3740 *-----*
3750 * FUNCTION 0E -- DEFINE TOP LEFT CORNER
3760 * at $1629 in AppleWorks 1.3
3770 *-----*
3780 FUN.CORNER.TL
3790 LDA AW.CH
3800 LDX AW.CV
3810 STA AW.LEFT
3820 STX AW.TOP
3830 RTS
0916- A5 10 3300
0918- 85 14 3320
091A- 4C 5D 0A 3380
091D- A9 00 3440
091F- 2C 3450
0920- A9 80 3510
0922- 8D 00 08 3520
0925- 60 3530
0926- A5 14 3590
0928- A6 15 3600
092A- 85 12 3620
092C- 86 13 3630
092E- 60 3640
092F- A5 10 3710
0931- 85 14 3720
0933- 60 3730
0934- A5 14 3790
0936- A6 15 3800
0938- 85 10 3810
093A- 86 11 3820
093C- 60 3830

```

```

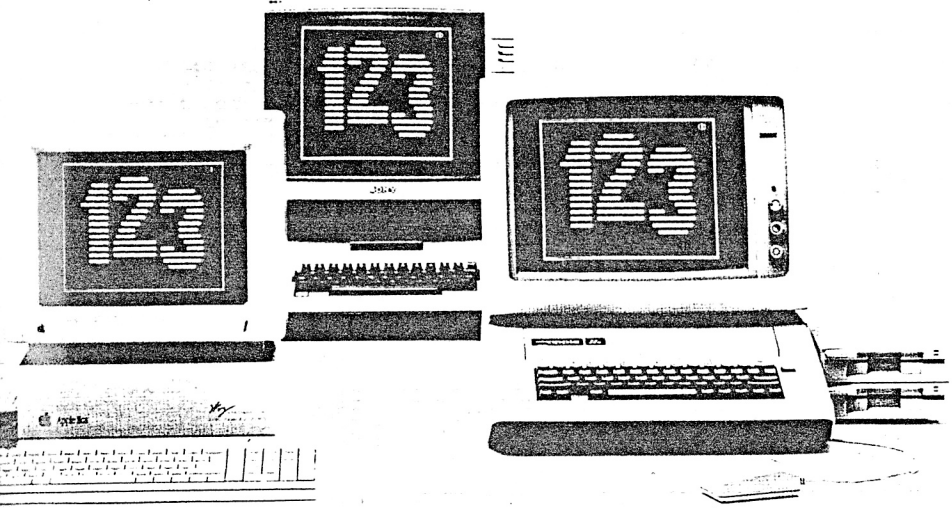
3840 *-----
3850 * FUNCTION OF -- SET FULL SCREEN WINDOW
3860 * at $1634 in AppleWorks 1.3
3870 *-----
3880 FUN.FULL.SCREEN
3890 LDA #0
3900 STA AW.TOP
3910 STA AW.LEFT
3920 LDA #79
3930 LDX #23
3940 BNE SET.CORNER.BR ...ALWAYS
3950 *-----
3960 * FUNCTION 10 -- "BEEP!"
3970 * at $1645 in AppleWorks 1.3
3980 *-----
3990 FUN.BEEP
4000 LDY #149
4010 .1 LDA #149
4020 .2 SEC
4030 SBC #1
4040 SEC
4050 SEC
4060 BNE .2 11*149-1 = 1638 CYCLES IN INNER LOOP
4070 LDA $C030 TOGGLE SPEAKER
4080 DEY
4090 BNE .1 1649 CYCLES BETWEEN CLICKS
4100 LDA #1 DELAY ABOUT 1/10 SECOND
4110 JMP DELAY.TENTHS
4120 *-----
4130 * FUNCTION 11 -- SHUFFLE SCREEN LEFT OR RIGHT
4140 * Following byte is scroll distance and direction:
4150 * If positive, SCROLL RIGHT; else SCROLL LEFT.
4160 * at $165E in AppleWorks 1.3
4170 *-----
4180 FUN.SHUFFLE
4190 LDY POSITION.IN.STRING
4200 LDA ($80),Y IF positive, SCROLL RIGHT; else SCROLL LEFT
4210 STA SCROLL.DIRECTION
4220 INC POSITION.IN.STRING
4230 LDA AW.TOP POINT TO TOP LINE FIRST
4240 .1 PHA
4250 JSR BASE.CALC.A
4260 LDA AW.RIGHT
4270 BIT SCROLL.DIRECTION
4280 BPL .2
4290 LDA AW.LEFT
4300 .2 STA SHUFFLE.DEST
4310 SEC
4320 SBC SCROLL.DIRECTION
4330 STA SHUFFLE.SOURCE
4340 JSR SCROLL.LEFT.OR.RIGHT
4350 PLA
4360 CMP AW.BOTTOM
4370 BEQ .3
4380 CLC
4390 ADC #1
4400 BNE .1
4410 .3 RTS
4420 *-----
4430 SCROLL.LEFT.OR.RIGHT
4440 LDA SCROLL.DIRECTION
4450 BPL .6 ...shuffle right
4460 BMI .2 ...shuffle left
4470 *----Scroll Left-----
4480 .1 INC SHUFFLE.DEST
4490 INC SHUFFLE.SOURCE
4500 .2 LDA #" " blank, in case off edge
4510 LDY SHUFFLE.SOURCE
4520 CPY AW.RIGHT
4530 BCC .3
4540 BNE .4 ...off the edge, use a blank
4550 .3 TYA Get source pointer
4560 >LD.SCRN Get character from screen
4570 .4 PHA Save the character
4580 LDA SHUFFLE.DEST destination pointer
4590 >ST.SCRN Store the character on the screen
4600 LDA SHUFFLE.DEST destination pointer
4610 CMP AW.RIGHT was it the right edge?
4620 BNE .1 ...no, keep shuffling
4630 RTS

```

```

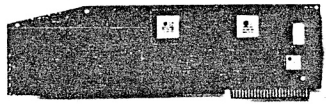
4650 DEC SHUFFLE.DEST
4660 DEC SHUFFLE.SOURCE
4670 .6 LDA #" " blank, in case off edge
4680 LDY SHUFFLE.SOURCE
4690 BMI .7 ...off edge, use blank
4700 CPY AW.LEFT
4710 BCC .7
4720 TYA Get source pointer
4730 >LD.SCRN Get character from screen
4740 .7 PHA Save the character
4750 LDA SHUFFLE.DEST destination pointer
4760 >ST.SCRN Store the character on the screen
4770 LDA SHUFFLE.DEST destination pointer
4780 CMP AW.LEFT was it the left edge?
4790 BNE .5 ...no, keep shuffling
4800 RTS
4810 *-----
4820 .MA VEC
4830 .DA FUN.]1-1
4840 .EM
4850 *-----
4860 * at $1779 in AppleWorks 1.3
4870 FUN.TBL
4880 >VEC HANG.UP 00
4890 >VEC CLR.CH.TO.EOL 01
4900 >VEC CLR.LINE 02
4910 >VEC HOME 03
4920 >VEC CLR.CH.TO.EOS 04
4930 >VEC GOTO.XY 05
4940 >VEC CURSOR.LEFT 06
4950 >VEC CURSOR.RIGHT 07
4960 >VEC CURSOR.UP 08
4970 >VEC CURSOR.DOWN 09
4980 >VEC INVERSE 0A
4990 >VEC NORMAL 0B
5000 >VEC CORNER.BR 0C
5010 >VEC CURSOR.BOL 0D
5020 >VEC CORNER.TL 0E
5030 >VEC FULL.SCREEN 0F
5040 >VEC BEEP 10
5050 >VEC SHUFFLE 11
5060 >VEC HANG.UP 12
5070 >VEC HANG.UP 13
5080 >VEC HANG.UP 14
5090 >VEC HANG.UP 15
5100 >VEC HANG.UP 16
5110 >VEC HANG.UP 17
5120 >VEC HANG.UP 18
5130 >VEC HANG.UP 19
5140 >VEC HANG.UP 1A
5150 >VEC HANG.UP 1B
5160 >VEC HANG.UP 1C
5170 >VEC HANG.UP 1D
5180 >VEC HANG.UP 1E
5190 >VEC HANG.UP 1F
5200 *-----
5210 POLL.KEYBOARD RTS (at $1FA7 in AppleWorks 1.3)
5220 *-----
5230 * Calculate Address of Screen Line
5240 * at $1717 in AppleWorks 1.3
5250 *-----
5260 BASE.CALC.CV
5270 LDA AW.CV
5280 BASE.CALC.A
5290 CMP #$$F <<<filled in with current line number>>>
5300 CURR.LINE .EQ #-1
5310 BEQ .2
5320 STA CURR.LINE
5330 LSR 0000ABCD--E
5340 AND #03 000000CD--E
5350 ORA #04 000001CD--E
5360 STA AW.BASE+1
5370 LDA CURR.LINE
5380 AND #18 000AB000--E
5390 BCC .1 E=0
5400 ADC #7F E00AB000
5410 .1 STA AW.BASE
5420 ASL 00AB0000
5430 ASL 0AB00000
5440 ORA AW.BASE EABAB000
5450 STA AW.BASE
5460 .2 RTS

```



Now Apple speaks IBM. Three times faster than IBM.

Introducing PC Transporter™, Apple® II expansion card that lets you run MS-DOS programs.



Your Apple II can run 1000 programs you could never do before. Like Lotus® MultiMate® dBASE III and Flight Simulator®. PC Transporter, MS-DOS programs run on your Apple II as if they were on IBM® PCs or compatibles. With one important difference: PC Transporter runs those programs three times faster than an IBM PC/XT®. To speed through number-crunching tasks, you can use optional 8087-2 math coprocessor chip. It plugs into an expansion slot on the PC Transporter. It's as expensive as an IBM clone.

clone costs about the same as the PC Transporter. But the peripherals it takes to get the clone up and running make the clone cost about three times what our American-made card costs. You don't have to buy new hardware to use PC Transporter. Works with the hardware you already own. With PC Transporter, MS-DOS programs see your Apple hardware as IBM hardware. You can use the same hardware you have now. With IBM software, your Apple hardware works just like IBM hardware. Including your drives, monitors, printers, printer cards, clock cards and serial clocks.

You can use your IIe® or IIgs™ keyboard with IBM software. Or use our optional IBM-style keyboard (required for the II Plus). You can use your Apple mouse. Or an IBM compatible serial mouse.

Plenty of power. PC Transporter gives you as much as 640K of user RAM and 128K of system RAM in the IBM mode. PC Transporter also is an Apple expansion card, adding up to 768K of extra RAM in the Apple mode. The Apple expansion alone is a \$300 value.

Easy to install. You can install PC Transporter in about 15 minutes, even if you've never added an expansion board. You don't need special tools. Simply plug it into an Apple expansion slot (1 through 7 except 3), connect a few cables and a disk drive, and go!



PC Transporter taps into the world's largest software library. Now your Apple can run most of the IBM software you use at work. And it opens a new world of communications programs, games and bulletin boards.

A universal disk drive controller. PC Transporter supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes. You'll shift instantly between Apple ProDOS and IBM MS-DOS.

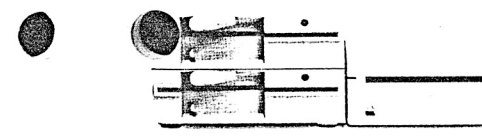
You'll need our versatile 5.25" 360K drive system to run IBM applications from 5.25" floppy disks. Use your Apple 5.25" drive for Apple 5.25" disks.

An Apple Disk 3.5 Drive will support the new 3.5" disks whether they're IBM MS-DOS formatted or Apple ProDOS formatted. The PC Transporter acts like an Apple Disk 3.5 Drive disk controller for IIgs, IIe, and II Plus users.

PC Transporter supports up to 5 drives in a number of combinations.

For example, you can connect a 5.25 Applied Engineering 360K dual-drive system directly to the card. Then plug two daisy-chained Apple 3.5 Drives (not the Apple UniDisk 3.5) to the dual-drive system. For a fifth drive, use a ProDOS file as an IBM hard disk.

PC Transporter controls Apple and IBM compatible disk drives. It supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes.



Versatile data storage. PC Transporter reads MS-DOS and translates it into Apple native ProDOS. You can store IBM programs and data on any ProDOS storage device including the Apple 3.5 Drive, Apple UniDisk™ 3.5, Apple 5.25" drive, SCSI or ProDOS compatible hard drives. (You can use the Apple UniDisk 3.5 with its own controller card for storing programs and data, but not for directly booting an IBM formatted disk.) You can even use our 360K PC compatible drive for ProDOS.

Make your Apple speak IBM.

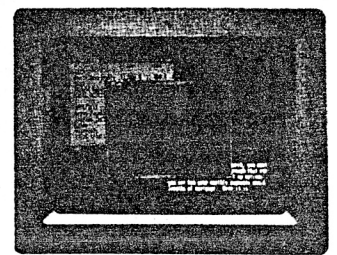
PC Transporter memory choices.

RAM in Apple mode:	RAM in IBM mode:	Price:
384K	256K	\$489.00
512K	384K	529.00
640K	512K	569.00
768K	640K	609.00

Note: The IBM mode is 128K less because the PC Transporter uses 128K for system memory.

- IIgs Installation Kit 49.00
- IIe/II Plus Installation Kit 39.00
- PC Transporter Accessories
- 5.25" IBM Format 360K Drive Systems
- Single-Drive System 269.00
- Dual-Drive System 399.00
- Half-Height Drive 135.00 (add to Single-Drive system to make Dual-Drive)
- IBM-Style Keyboard 139.00 (required for Apple II Plus. Requires IBM Keyboard Cable.)
- IBM Keyboard Cable 34.00
- Sony RGB Monitor 499.00
- Analog RGB Cable 39.00 (for use with Sony monitor)
- Digital RGB Cable 39.00 (for use with Sony monitor)
- Digital RGB Adapter 24.00
- ColorSwitch 44.00 (included with IIgs Installation Kit)
- 128K ZIP 40.00
- PC Transporter Memory Expansion Chip Set per set
- 8087-2 Math Co-processor 229.00
- Chip 69.00
- Heavy Duty Power Supply (IIe and II Plus only)

See your dealer or call or send check or money order to Applied Engineering, MasterCard, VISA and COD welcome. Texas residents add 6 1/4% sales tax.



PC Transporter produces better IBM graphics than IBM. Analog is sharper than digital. So with an analog RGB monitor, PC Transporter's CGA graphics and text are superior to IBM's digital display — even while running IBM software! And, you can also use an Apple composite monitor in IBM text or graphics mode.

storage and a 143K Apple 5.25" drive for MS-DOS storage. Created by Apple's original designers.

The brains behind PC Transporter were also behind your Apple II. The PC Transporter design team includes the former project managers for the creation of the Apple IIe and IIc. The co-designer of the Apple II disk controller. And the first full-time Apple programmer and author of the ProDOS operating system.

So you know the PC Transporter and your Apple were made for each other.

Support and service from the leader in Apple add-ons.

Applied Engineering sells more Apple peripheral boards than anyone else — including Apple Computer. So you know we'll be around after the sale.

PC Transporter comes with a 15-day money back guarantee. If you're not fully satisfied after using it, return it for a full refund. PC Transporter also comes with a 1-year warranty.

How to get your PC Transporter today.

See your dealer. Or call Applied Engineering any day between 9 a.m. and 11 p.m. CST at 214-241-6060.

AE Applied Engineering
The Apple enhancement experts.
P.O. Box 798, Carrollton, TX 75006
214-241-6060
A Division of AE Research Corporation


```

5470 *-----
5480 * Blank Line from Cursor to End of Line
5490 * at $173A in AppleWorks 1.3
5500 *-----
5510 CLR.CH.TO.EOL
OA5D- A5 12 5520 LDA AW.RIGHT
OA5F- 4A 5530 LSR
OA60- 8D 99 OA 5540 STA N.OVER.2
OA63- 2A 5550 ROL GET AW.RIGHT AGAIN
OA64- 38 5560 SEC
OA65- E9 01 5570 SBC #1
OA67- 4A 5580 LSR
OA68- 8D 9A OA 5590 STA N.MINUS.1.OVER.2
OA6B- 20 3B OA 5600 JSR BASE.CALC.CV
OA6E- A5 14 5610 LDA AW.CH
OA70- 4A 5620 LSR
OA71- 48 5630 PHA
5640 *---Clear Even Columns---
5650 TAY
5660 LDA # " "
5670 STA $C055 INTO AUX MEM
5680 BCC .2
5690 .1 INY
5700 .2 CPY N.OVER.2
5710 BEQ .3
5720 BCS .4
5730 .3 STA (AW.BASE),Y
5740 JMP .1
5750 .4 STA $C054 BACK TO MAIN MEM
5760 *---Clear Odd Columns---
5770 PLA
5780 TAY
5790 LDA # " "
5800 .5 STA (AW.BASE),Y
5810 INY
5820 CPY N.MINUS.1.OVER.2
5830 BCC .5
5840 BEQ .5
5850 RTS
5860 *-----
5870 N.OVER.2 .BS 1
5880 N.MINUS.1.OVER.2 .BS 1
5890 *-----
5900 * DELAY ABOUT (A) TENTHS OF A SECOND
5910 * at $1FD1 in AppleWorks 1.3
5920 *-----
5930 DELAY.TENTHS
5940 TAY
5950 .1 LDX #100 ...ABOUT 100 MILLISECONDS
5960 .2 CLC
5970 .3 ADC #1
5980 BCC .3 ...LOOP IS 256*5 CYCLES
5990 DEX
6000 BNE .2 (5*256+6)*100 = 128600 CYCLES
6010 DEY
6020 BNE .1 128606*A CYCLES
6030 RTS
6040 *-----
6050 * Some Demonstrations
6060 *-----
6070 T
OAAA- A9 CE 6080 LDA #MY.STRINGS
OAAC- A2 OA 6090 LDX /MY.STRINGS
OAAE- 85 80 6100 .1 STA $80
OAB0- 86 81 6110 STX $81
OAB2- A0 00 6120 LDY #0
OAB4- B1 80 6130 LDA ($80),Y
OAB6- F0 15 6140 BEQ .99 ...FINISHED
OAB8- 48 6150 PHA SAVE LENGTH
OAB9- E6 80 6160 INC $80
OABB- D0 02 6170 BNE .2
OABD- E6 81 6180 INC $81
OABF- 20 04 08 6190 .2 JSR DISPLAY.STRING
OAC2- 18 6200 CLC
OAC3- 68 6210 PLA GET LENGTH
OAC4- 65 80 6220 ADC $80 ADD TO POINTER
OAC6- A6 81 6230 LDX $81
OAC8- 90 E4 6240 BCC .1
OACA- E8 6250 INX
OACB- B0 E1 6260 BCS .1 ...ALWAYS
OACD- 60 6270 .99 RTS

```

```

6280 *-----
6290 MY.STRINGS
6300 .DA #Z1
6310 A1 .HS OF.03.OB Full Sern, Home, Normal
6320 .AS /ABCDECDFGHIJKLMNOPQRSTUVWXYZ 1@#%&'()*+=[\;:~',.<.>?{|}/
6330 .HS OA.OD.09 Inverse, RETURN, LINEFEED
6340 .AS /ABCDECDFGHIJKLMNOPQRSTUVWXYZ 1@#%&'()*+=[\;:~',.<.>?{|}/
6350 Z1 .EQ *-A1
6360 *-----
6370 .DA #Z2
6380 A2 .HS OB.OD.09 Normal, RETURN, LINEFEED
6390 .AS "abcdefghijklmnopqrstuvwxy / 0123456789 _ "
6400 .HS OA.OD.09 Inverse, RETURN, LINEFEED
6410 .AS "abcdefghijklmnopqrstuvwxy / 0123456789 _ "
6420 Z2 .EQ *-A2
6430 *-----
6440 .DA #Z3
6450 A3 .HS OB.OD.09 Normal, RETURN, LINEFEED
6460 .AS -/ABCDECDFGHIJKLMNOPQRSTUVWXYZ[\] /
6470 .HS 05.00.00 GO TO 0,0
6480 .HS 10.08.08.08.08.08.08.08 Beep, 7 cursor ups
6490 .HS 10.08.08.08.08.08.08.08 Beep, 7 cursor ups
6500 .HS 05.00.17 GO TO 0,23
6510 .HS 10.09.09.09.09.09.09.09 Beep, 7 cursor downs
6520 .HS 10.09.09.09.09.09.09.09 Beep, 7 cursor downs
6530 Z3 .EQ *-A3
6540 *-----
6550 .DA #Z4
6560 A4 .HS 10.11.08 Beep, Shuffle Right 8
6570 .HS 10.11.F8 Beep, Shuffle Left 8
6580 .HS 10.11.01.11.01.11.01.11.01.11.01.11.01.11.01
6590 .HS 10.11.FF.11.FF.11.FF.11.FF.11.FF.11.FF.11.FF
6600 .HS 10.11.F8
6610 .HS 10.11.08
6620 .HS 10.11.28
6630 .HS 10.11.D8
6640 Z4 .EQ *-A4
6650 *-----
6660 .DA #Z5
6670 A5 .HS 03 HOME
6680 .HS 20.21.22.23.24.25.26.27.28.29.2A.2B.2C.2D.2E.2F
6690 .HS 30.31.32.33.34.35.36.37.38.39.3A.3B.3C.3D.3E.3F.OD.09
6700 .HS 40.41.42.43.44.45.46.47.48.49.4A.4B.4C.4D.4E.4F
6710 .HS 50.51.52.53.54.55.56.57.58.59.5A.5B.5C.5D.5E.5F.OD.09
6720 .HS 60.61.62.63.64.65.66.67.68.69.6A.6B.6C.6D.6E.6F
6730 .HS 70.71.72.73.74.75.76.77.78.79.7A.7B.7C.7D.7E.7F.OD.09
6740 .HS 0A INVERSE
6750 .HS 20.21.22.23.24.25.26.27.28.29.2A.2B.2C.2D.2E.2F
6760 .HS 30.31.32.33.34.35.36.37.38.39.3A.3B.3C.3D.3E.3F.OD.09
6770 .HS 40.41.42.43.44.45.46.47.48.49.4A.4B.4C.4D.4E.4F
6780 .HS 50.51.52.53.54.55.56.57.58.59.5A.5B.5C.5D.5E.5F.OD.09
6790 .HS 60.61.62.63.64.65.66.67.68.69.6A.6B.6C.6D.6E.6F
6800 .HS 70.71.72.73.74.75.76.77.78.79.7A.7B.7C.7D.7E.7F.OD.09
6810 .HS 0B NORMAL
6820 Z5 .EQ *-A5
6830 *-----
6840 .DA #Z6
6850 A6 .HS OD.09 CRLF
6860 .HS 80.81.82.83.84.85.86.87.88.89.8A.8B.8C.8D.8E.8F
6870 .HS 90.91.92.93.94.95.96.97.98.99.9A.9B.9C.9D.9E.9F.OD.09
6880 .HS A0.A1.A2.A3.A4.A5.A6.A7.A8.A9.AA.AC.AD.AE.AF
6890 .HS B0.B1.B2.B3.B4.B5.B6.B7.B8.B9.BA.BB.BC.BD.BE.BF.OD.09
6900 .HS C0.C1.C2.C3.C4.C5.C6.C7.C8.C9.CA.CB.CC.CD.CE.CF
6910 .HS D0.D1.D2.D3.D4.D5.D6.D7.D8.D9.DA.DB.DC.DD.DE.DF.OD.09
6920 .HS E0.E1.E2.E3.E4.E5.E6.E7.E8.E9.EA.EB.EC.ED.EE.EF
6930 .HS F0.F1.F2.F3.F4.F5.F6.F7.F8.F9.FA.FB.FC.FD.FE.FF.OD.09
6940 Z6 .EQ *-A6
6950 *-----
6960 .HS 00
6970 *-----

```

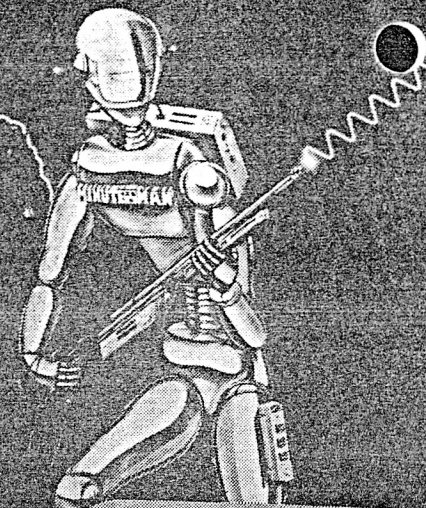
MINUTEMANTM

UNINTERRUPTIBLE POWER SUPPLIES

PROTECTION FROM:

- ★ BROWNOUTS
- ★ BLACKOUTS
- ★ OVER VOLTAGE
- ★ SPIKES
- ★ SURGES
- ★ EMI/RFI

NEW!
MINUTEMANTM
1600 WATT
 Sine Wave Output
 One msec Transfer Time
OVER VOLTAGE
PROTECTION



- Sine Wave Output 1 msec maximum switching time*
- Order - ship same day
- Full one year warranty

* 250 Watt and 500 Watt units offer 4 msec switching time; PWM waveform.

250 WATT (120V)	300 WATT (120V)	500 WATT (120V)	600 WATT (120V)	1200 WATT (120V)
\$359⁰⁰ Suggested Retail UL Approved	\$549⁰⁰ Suggested Retail	\$699⁰⁰ Suggested Retail	\$899⁰⁰ Suggested Retail	\$1499⁰⁰ Suggested Retail

230-V Units Also Available

1455 LeMay Drive
 Carrollton, Texas 75007

PARA SYSTEMS, INC.

Telephone:
 (214) 446-7363

We are dealers for MinuteMan UPS

250w for \$320, 300w for \$490

S-C Software Corporation
 2331 Gus Thomasson #125
 DALLAS, TX 75228
 Phone 214-324-2050

Buy it from us and save!

Overhauling the S-C Program Selector.....Bob Sander-Cederlof

About a year and a half ago, in the July 1986 issue of Apple Assembly Line, I published my replacement for the ProDOS "QUIT" code. It turned out to be a very popular article and program, and various updates and corrections were printed in the August, September, October, and December issues that same year.

In review, the QUIT code is a 768-byte program inside ProDOS-8 which is executed when you leave a system program, such as FILER, AppleWorks, BASIC.SYSTEM (Applesoft), Copy II Plus, the S-C Macro Assembler, and so on. When you QUIT you are really executing the MLI Quit call (\$65), which copies those 768 bytes down to RAM starting at \$1000 and jumps there. Apple's built-in QUIT code is about as user-friendly as an angry porcupine: if you haven't MEMORIZED the volume names and file names of all your system programs, you almost always end up resetting and re-booting instead of filling in the blanks.

My 1986 Program Selector exactly fits in the 768-byte space Apple's version occupies in the ProDOS system file. It puts a menu of online volumes on the screen, allowing you to select one with the arrow keys and RETURN. Once you select a volume, you see a menu of the SYS and DIR files in the main directory of that volume. Using the arrows and RETURN you can either start up a SYS program or select a subdirectory. If you choose a subdirectory, you get a menu of SYS and DIR files inside it. Hitting ESCAPE always takes you back to the Volume Menu. As I wrote the Program Selector, it requires an Apple //e, //c, or //gs, and works in 80-columns.

The modifications already published include fixing one bug, making it work with a Videx-compatible 80-column card for Apple II Plus owners, following Apple's published spec's for substitute QUIT-code, and making it begin with the menu bar on something other than the first volume in the menu.

Quite a few readers of AAL are now using this Program Selector. Some have gone the extra mile by adapting the S-C Program Selector to their own preferences. Jim Hammond (of FastFind SUPER INDEX) liked it so well he turned it into a product which he sells (with my permission) as "STARTER/QUITTER". Larry Skutchan (a blind user who adapted the S-C Word Processor into a talking version) adapted it to work with the Echo Speech Synthesizer. Brooke Boering (creator of CeeMac and Fire Organ) put in a feature allowing you to limit the Volume Menu to a particular disk drive. Brooke's ideas are what led me to try to improve and upgrade my program.

The main computer here at the office is an Apple //e with a 10-meg Sider (ProDOS sees it as two drives in slot 7), two standard Apple floppies in slot 6, a RamFactor card in slot 4 (simulates one drive), a 1-meg RamWorks card in the AuxSlot (simulates a drive in slot 3, Drive 2), and a 3.5 inch drive in slot 2 (ProDOS thinks there are two drives there, even though I only have one). When I type BYE or in some other way select the ProDOS QUIT code, my S-C Program Selector takes over. The old version seemed to go away and die for several seconds while

ProDOS did a complete ONLINE check to find out what volume if any was mounted in each and every drive. If my hard disk is not turned on, that takes several seconds for the firmware to timeout. If no floppies are in the 5.25 drives, they go through spinning, re-calibration, and the works before giving up.

It finally dawned on me that what I wanted was to direct the Program Selector to only try the slot and drive I booted from. Most likely if I booted from it, there is some kind of volume there. Of course I still wanted the capability of seeing every volume, but I did not want to waste all that time EVERY time!

Brooke told me six months or more ago that I could plug a drive ID into my ONLINE call (line 2760 of the original program) and it would limit the display to that one volume. It turned out to be a little harder than that, but I did get that to work. But I could not decide on just one slot and drive. I wanted that byte to be set up by ProDOS at boot-time to point to the booting drive.

I remembered that the ProDOS startup code began by plugging the boot drive ID into its own ONLINE call, which it uses to get the Volume Name. I looked up the code in the Supplement to "Beneath Apple ProDOS", and found it. In ProDOS 1.1.1 it is done by the first two instructions at \$2000; in ProDOS 1.2, 1.3, and 1.4 it starts six bytes later at \$2006. The first instruction is "LDA \$43", which picks up the drive ID (slot number times 16) that was used to load the ProDOS or P8 file. The second one is "STA \$21FE" for ProDOS 1.1.1, and "STA" somewhere else for later versions. I decided I could patch into that "STA" instruction a call to a piece of patch code which would not only do the patched-over STA for Apple, but also an additional one for me. More on this later, when we get into the code for my new Program Selector.

Anyway, the Program Selector now comes up only showing the Volume Name of the volume currently in the drive ProDOS was booted from. If that is the volume I want, I just hit RETURN and see the menu of SYS and DIR files on that volume. If I booted from RamFactor, this all happens at blinding speed.

If the boot-drive is not the drive I want, I can type a digit and select a different drive or all drives. Typing a digit 1 through 7 changes to drive 1 of that slot and displays the Volume Name in that drive. Typing the digit "8" changes to drive 2 in the same slot. Why 8? Why not? It made the code shorter, and it worked. Typing "0" changes back to the old way, making a menu of all online volumes. If you select a drive which has no volume mounted, you will get an empty menu. No problem, just select a different drive or all-drives, and continue.

I also made various other improvements here and there, such as making sure that text mode with a full-screen window is selected. I had to revise the "help" message at the bottom of the menu display to include information on the digits 0-8.

A major constraint in adding new features was that I wanted to retain the advantage of fitting inside the 768-byte hole in the ProDOS file. In developing the new version I decided not to worry about size too much until all the new features were working. I tested them by BRUNning the code at \$1000, instead of going through the process of putting it into ProDOS every time. Then when it was all ready, I started looking for ways to shrink the code and make it fit in only 768 bytes.

It ran about 32 bytes over, so I needed a lot of shrinking. For some reason I don't remember, back in 1986 I decided to keep a lot of variables out of page zero. There is no requirement to do this, so I moved these variables and saved almost all the bytes I wanted. All instructions referencing these variables shrank from three to two bytes. The rest of the savings were found by careful study of the code. If you compare the new listing which follows with the one I published in 1986 you can find the tricks I pulled. The new version, even with all the new features, is now shorter than the original! There are actually four unused bytes!

I also wrote a program to automatically install the new Program Selector inside the ProDOS file. Well, almost automatically. You still have to BLOAD ProDOS or BLOAD P8, and UNLOCK the file if it is LOCKed. Then you "-" or BRUN my INSTALL.QUITTER program, and it automatically does the installation. If successful, you get a nice message to that effect; then you have to BSAVE the image and re-LOCK it. I thought it would be too dangerous to make all of the above entirely automatic. If my installer made a mistake.... So, I left the crucial part manual.

The auto-installer does differentiate between ProDOS 1.1.1 and the later versions. It makes the boot-drive patch at either \$2002 or \$2008, depending on where it finds the STA instruction. And if it cannot find that instruction, it tells you so and quits. The Program Selector image is copied either to \$5700 (for ProDOS 1.1.1) or \$5900 (for later versions).

The code for the auto-installer is executed when you BRUN INSTALL.QUITTER. Lines 1460-2270 are the installation code, and lines 2280 to the end are the Program Selector image. Lines 1490-1710 try to determine which version of ProDOS, if any, is in memory starting at \$2000. If it finds a recognizable version, it sets up various pointers according to the version. If not, it prints out the long message from lines 2160-2220.

Lines 1720-1800 copy a JSR to my patch code over the top of the STA xxxx instruction which starts at either \$2002 or \$2008. It also modifies my patch code to include exactly the correct STA xxxx instruction which we are patching over. The patch code is at the very end of the Program Selector image, in lines 5970-6020. Later, when this patched ProDOS file is booted or otherwise executed, my patch code will install the boot drive ID into the ONLINE call block at line 5880.

Lines 1810-1950 copy the Program Selector image into the ProDOS image, at either \$5700 or \$5900 depending on version. Assuming we got this far, lines 1970-1990 will print out the "SUCCESSFUL" message.

```

1000 *SAVE NEW.QUIT.CODE
1010 *-----
1020 *   Installation:
1030 *   1. BLOAD PRODOS,TSYS,A$2000
1040 *   2. BRUN INSTALL,QUITTER
1050 *   3. BSAVE PRODOS,TSYS,A$2000
1051 *-----
1052 *           .MA ASC           Macro to shorten listing
1053 *           .AS  -"j1"
1054 *           .EM
1060 *-----
1070 *           .DUMMY
1080 *           .OR 0000
0000- 1090 BPNTR .BS 2
0002- 1100 SPNTR .BS 2
0004- 1110 DPNTR .BS 2
0006- 1120 DIR.INDEX .BS 1
0007- 1130 DIR.START .BS 1
0008- 1140 MAX.DIRPNT .BS 1
0009- 1150 SEL.LINE .BS 1
000A- 1160 MAX.LINE .BS 1
000B- 1170 UNIT .BS 1
000C- 1180 LENGTH .BS 1
000D- 1190 CURTYP .BS 1
000E- 1200 CURBLK .BS 1
1210 *-----
0800- 1220 .OR $800
OC00- 1230 OPNBUF .BS 1024
      1240 DIRBUF .BS 512
      1250 .ED

```

EnterSoft:

Basic-like macros which make the complex simple. Don't re-write that multiplication routine for the hundredth time! Get EnterSoft instead! Do 8/16/32/64 bit Math/Input-Output/Graphics simply without all of the hassles. These routines are a must for the serious programmer who doesn't want to spend all of his/her time trying to re-invent the wheel. DOS 3.3 Version=\$30.00, ProDos Version=\$30.00, BOTH for only \$50.00. GET YOURS TODAY.

A Shape Table Program:

For once! A shape table program which is logically organized into its componet parts. Each section resides in its own program. The editor, disk access, Hi-Res section; each section is separate. Written almost entirely in Basic, it is easily modified. Not copyprotected! Put them on a Hard Disk, Ram Drive, anywhere! DOS 3.3 Version=\$20.00, ProDos Version=\$20.00, BOTH for \$30.00!

<p>Send Check or Money Order To:</p> <p>Mark Manning c/o Simulacron I/Baggy Game P.O. Box 58598 Webster, TX 77598</p>	<p>ProDos Upgrade for DOS 3.3 EnterSoft Owners - \$20.00</p> <p>Thanks for the letters - Keep Writing!</p>
---	--

```

0280- 1260 *-----
2000- 1270 PATHNAME .EQ $280
2023- 1280 BUFFER .EQ $2000
2024- 1290 ENTTLEN .EQ BUFFER+$23   ENTRY LENGTH
      1300 ENTCNT .EQ BUFFER+$24   # ENTRIES PER BLOCK
      1310 *-----
25-   1320 CV .EQ $25
32-   1330 INVFLG .EQ $32
      1340 *-----
FB2F- 1350 INIT .EQ $FB2F
FC58- 1360 HOME .EQ $FC58
FC9C- 1370 CLRECL .EQ $FC9C
FD8E- 1380 COUT .EQ $FD8E
FE80- 1390 CROUT .EQ $FE80
FE84- 1400 SETINV .EQ $FE80
      1410 SETNORM .EQ $FE84
      1420 *-----
BF00- 1430 MLI .EQ $BF00
BF58- 1440 BITMAP .EQ $BF58
      1450 *-----
      1460 .OR $1000
1000- 1470 .TF INSTALL.QUITTER
      1480 *-----
      1490 INSTALL.QUITTER
1000- A2 57 1500 LDX /$5700   WHERE IMAGE IS IN 1.1.1
1002- A9 20 1510 LDA /$2000   WHERE LDA $43 IS IN 1.1.1
1004- 85 01 1520 STA BPNTR+1
1006- A0 00 1530 LDY #0
1008- 84 04 1540 STY DPNTR
100A- AD 00 20 1550 LDA $2000   SEE IF PRODOS IMAGE IS HERE
100D- C9 4C 1560 CMP #$4C   IF "JMP" THEN PROBABLY 1.4
100F- D0 04 1570 BNE .1   ...PROBABLY 1.1.1
1011- A0 06 1580 LDY #6   WHERE LDA $43 IS IN 1.4
1013- A2 59 1590 LDX /$5900   WHERE IMAGE IS IN 1.4
1015- 84 00 1600 .1 STY BPNTR   POINT AT LDA $43
1017- 86 05 1610 STX DPNTR+1   POINT AT IMAGE OF QUITTER
1019- E8 1620 INX
101A- E8 1630 INX
101B- 8E EC 13 1640 STX QPATCH+2 ADDRESS IN IMAGE OF "ONLINE+1"
101E- 8E FO 10 1650 STX JSR.QPATCH+2 ADDRESS IN IMAGE OF QPATCH
1021- A0 02 1660 LDY #2
1023- B9 EB 10 1670 .2 LDA PRODOS.ID.STRING,Y
1025- D1 00 1680 CMP (BPNTR),Y
1028- D0 34 1690 BNE .99   ...NEITHER, QUIT.
102A- 88 1700 DEY
102B- 10 F6 1710 BPL .2
1720 *---Trust we have ProDOS image---
102D- A0 02 1730 LDY #2   POINT AT STA XXXX
102F- B1 00 1740 LDA (BPNTR),Y
1031- 99 EB 13 1750 .3 STA QPATCH-2+3,Y
1034- B9 EC 10 1760 LDA JSR.QPATCH-2,Y
1037- 91 00 1770 STA (BPNTR),Y
1039- C8 1780 INY
103A- C0 05 1790 CPY #5
103C- 90 F1 1800 BCC .3
1810 *---Copy Quitter into ProDOS----
103E- A9 F1 1820 LDA #QUIT.IMAGE
1040- 85 02 1830 STA SPNTR
1042- A9 10 1840 LDA /QUIT.IMAGE
1044- 85 03 1850 STA SPNTR+1
1046- A2 03 1860 LDX #3   COPY 3 PAGES
1048- A0 00 1870 LDY #0
104A- B1 02 1880 .4 LDA (SPNTR),Y
104C- 91 04 1890 STA (DPNTR),Y
104E- C8 1900 INY
104F- D0 F9 1910 BNE .4
1051- E6 03 1920 INC SPNTR+1
1053- E6 05 1930 INC DPNTR+1
1055- CA 1940 DEX
1056- D0 F2 1950 BNE .4
1960 *---Successful, say so and end---
1058- A0 00 1970 LDY #IQ.GOOD
105A- 20 68 10 1980 JSR IQ.PRINT
105D- 60 1990 RTS   FINISHED
2000 *---No ProDOS, or already patched---
105E- A0 27 2010 .99 LDY #IQ.BAD
1060- 20 68 10 2020 JSR IQ.PRINT
1063- 60 2030 RTS
      2040 *-----

```

```

4- 20 ED FD 2050 IQ.OUT JSR COUT
7-  C8      2060 INY
      2070 IQ.PRINT
8-  B9 6E 10 2080 LDA IQ,Y
B-  D0 F7      2090 BNE IQ.OUT
D-  60      2100 RTS
-----
E-  2110 IQ .EQ *
E-  2120 IQ.GOOD .EQ *-IQ
E-  2130 >ASC "SUCCESSFULLY INSTALLED NEW QUIT CODE."
      2140 .HS 8D00
3-  8D 00      2150 IQ.BAD .EQ *-IQ
      2160 >ASC "EITHER PRODOS NOT HERE,"
      2170 .HS 8D
      2180 >ASC "OR NOT VERSION 1.1.1 OR 1.4,"
      2190 .HS 8D
      2200 >ASC "OR ALREADY INSTALLED QUIT CODE."
      2210 .HS 8D00
      2220
      2230
      2240 PRODOS.ID.STRING
EB- A5 43 8D      2250 .HS A5.43.8D
      2260 JSR.QPATCH
      2270 JSR QPATCH.EP
      2280
      2290 QUIT.IMAGE
      2300 .PH $1000
      2310
      2320
      2330 QUITTER
GC-  D8      2340 CLD REQUIRED BY "STANDARDS"
      2350 LDA $C082 MOTHERBOARD RQMS
      2360 JSR INIT TEXT MODE, FULL SCREEN WINDOW
      2370 JSR SETNORM
      2380 LDX #$16
      2390 LDA #0 PREPARE VIRGIN BITMAP
      2400 STA BITMAP,X
      2410 STX BITMAP+$17 LAST TIME STORES $01, LOCK OUT $BFOO PAGE
      2420 DEX
      2430 BNE .1
      2440 LDA #$CF
      2450 STA BITMAP
      2460
      2470
      2480
      2490
      2500
      2510
      2520
      2530
      2540
      2550
      2560
      2570
      2580
      2590
      2600
      2610
      2620
      2630
      2640
      2650
      2660
      2670
      2680
      2690
      2700
      2710
      2720
      2730
      2740
      2750
      2760
      2770
      2780
      2790
      2800
      2810
      2820
      2830
      2840
      2850
      2860
      2870

```

```

*---LIST THE FILENAMES-----
7A- A8      2890 TAX " " Y=0
      84 07      2900 STY DIR.START
      84 09      2910 STY SEL.LINE
      20 01 12 2920 JSR DISPLAY.FILES
      A0 11      2930 LDY #Q.VHELP
      20 68 12 2940 JSR MSG
      20 9D 11 2950 JSR GET.KEY
      90 F1      2960 BCC .6 ...ARROW KEYS
      DO 8E      2970 BNE .2 ...ESCAPE KEY
      A0 10      2980 LDY #$10
      B1 02      2990 LDA (SPNTR),Y GET FILE TYPE
      10 B8      3000 BPL .4 DIRECTORY ($OF)
      3010
      3020 *---SYS FILE, LOAD & EXECUTE-----
      C6 F2 12 3030 JSR MLI SET PREFIX
      20 A5 10 3040 .DA #$C6,PATH
      B0 03      3050 JSR READ.THE.FILE
      4C 00 20 3060 BCS .7 ...ERROR IN READING
      4C 00 10 3070 JMP BUFFER
      4C 00 10 3080 JMP QUITTER
      3090
      3100 READ.THE.FILE
      B1 02      3110 LDY #0 APPEND CURRENTLY SELECTED NAME
      29 0F      3120 LDA (SPNTR),Y GET LENGTH OF NAME
      85 0C      3130 AND #$OF
      AE 80 02 3140 STA LENGTH
      A9 2F      3150 LDX PATHNAME CURRENT LENGTH
      E8      3160 LDA #'/'
      C8      3170 INX
      9D 80 02 3180 STA PATHNAME,X
      B1 02      3190 LDA (SPNTR),Y
      C6 0C      3200 DEC LENGTH
      10 F5      3210 BPL .1
      8E 80 02 3220 STX PATHNAME
      20 00 BF 3230 JSR MLI OPEN THE FILE
      C8 E4 12 3240 .DA #$C8,OPEN
      B0 19      3250 BCS RF.ERR
      AD E9 12 3260 LDA O.REF FILE REFERENCE NUMBER
      8D EB 12 3270 STA R.REF
      20 00 BF 3280 JSR MLI READ THE WHOLE FILE
      CA EA 12 3290 .DA #$CA,READ
      90 05      3300 BCC CLOSE.ALL.FILES
      C9 4C      3310 CMP #$4C IS IT JUST EOF?
      38      3320 SEC
      DO 06      3330 BNE RF.ERR ...NO
      20 00 BF 3340 CLOSE.ALL.FILES
      CC DE 12 3350 JSR MLI CLOSE THE FILE
      60      3360 .DA #$CC,CLOSE
      3370 RF.ERR RTS
      3380
      3390 SCAN.DIRECTORY
      85 0D      3400 STA CURTYP TYPE WE ARE COLLECTING
      A9 00      3410 LDA #0 START WITH FIRST BLOCK
      85 0E      3420 STA CURBLK
      A9 04      3430 LDA #BUFFER+4 FIRST 4 BYTES OF BLOCK SKIPPED
      85 04      3440 STA DPNTR
      18      3450 CLC COMPUTE PAGE OF PNTR
      A9 20      3460 LDA /BUFFER+4
      65 0E      3470 ADC CURBLK
      85 05      3480 STA DPNTR+1
      AD 24 20 3490 LDA ENTCNT
      85 0C      3500 STA LENGTH
      3510
      3520
      3530
      3540
      3550
      3560
      3570
      3580
      3590
      3600
      3610
      3620
      3630
      3640
      3650
      3660
      3670
      3680
      3690
      3700
      3710
      3720
      3730
      3740
      3750
      3760
      3770
      3780
      3790
      3800
      3810
      3820
      3830
      3840
      3850
      3860
      3870
      3880
      3890
      3900
      3910
      3920
      3930
      3940
      3950
      3960
      3970
      3980
      3990

```

```

3680 *---ADVANCE TO NEXT ENTRY-----
1117- 18 3690 .4 CLC
1118- A5 04 3700 LDA DPNTR
111A- 6D 23 20 3710 ADC ENTLEN
111D- 85 04 3720 STA DPNTR
111F- 90 02 3730 BCC .5
1121- E6 05 3740 INC DPNTR+1
1123- C6 0C 3750 .5 DEC LENGTH AT END OF BLOCK YET?
1125- D0 D1 3760 BNE .2 ...NO, CONTINUE IN BLOCK
1127- 18 3770 CLC
1128- A5 0E 3780 LDA CURBLK
112A- 69 02 3790 ADC #2
112C- CD F1 12 3800 CMP ACTLEN+1
112F- 90 B5 3810 BCC .1 ...YES, READ NEXT BLOCK
1131- 60 3820 RTS
-----
3830 *
3840 DISPLAY.VOLUMES
1132- 20 43 12 3850 JSR SETUP.DISPLAY.LOOP
1135- A9 20 3860 LDA /BUFFER
1137- 85 01 3870 STA BPNTR+1
1139- A9 00 3880 LDA #BUFFER
113B- 85 00 3890 .1 STA BPNTR
113D- A0 00 3900 LDY #0
113F- B1 00 3910 LDA (BPNTR),Y
1141- F0 35 3920 BEQ .3 ...END OF LIST
1143- 29 0F 3930 AND #%OF
1145- F0 2A 3940 BEQ .3 ...NO VOLUME HERE
-----
1147- 20 51 12 3950 *
3960 JSR CHECK.FOR.SEL.LINE
3970 *
114A- B1 00 3980 LDA (BPNTR),Y GET UNIT NUMBER
114C- 4A 3990 LSR ISOLATE SLOT NUMBER
114D- 4A 4000 LSR
114E- 4A 4010 LSR
114F- 4A 4020 LSR
1150- 29 07 4030 AND #7
1152- 09 B0 4040 ORA #0"
1154- 20 ED FD 4050 JSR COUT PRINT SLOT NUMBER
1157- A9 AF 4060 LDA #"/"
1159- 20 ED FD 4070 JSR COUT
115C- B1 00 4080 LDA (BPNTR),Y GET UNIT NUMBER AGAIN
115E- 0A 4090 ASL SET CARRY IF DRIVE 2
115F- A9 B1 4100 LDA #1" ASSUME DRIVE 1
1161- 69 00 4110 ADC #0 CHANGE TO 2 IF TRUE
1163- 20 ED FD 4120 JSR COUT
1166- A9 A0 4130 LDA # " PRINT TWO SPACES
1168- 20 ED FD 4140 JSR COUT
116B- 20 ED FD 4150 JSR COUT
116E- 20 79 11 4160 JSR PRINT.BPNTR.NAME
-----
1171- 18 4170 *
4180 .3 CLC POINT TO NEXT VOLUME NAME
1172- A5 00 4190 LDA BPNTR
1174- 69 10 4200 ADC #16
1176- 90 C3 4210 BCC .1 STILL IN SAME PAGE
1178- 60 4220 .5 RTS
-----
4230 *
4240 PRINT.BPNTR.NAME
1179- A0 00 4250 LDY #0
117B- B1 00 4260 LDA (BPNTR),Y GET NAME LENGTH
117D- 29 0F 4270 AND #%OF
117F- AA 4280 TAX
1180- C8 4290 .1 INY PRINT THE VOLUME OR FILE NAME
1181- B1 00 4300 LDA (BPNTR),Y
1183- 09 80 4310 ORA #80
1185- 20 ED FD 4320 JSR COUT
1188- CA 4330 DEX
1189- D0 F5 4340 BNE .1
-----
4350 *
118B- A9 A0 4360 .2 LDA # " PRINT TRAILING BLANKS
118D- 20 ED FD 4370 JSR COUT
1190- C8 4380 INY
1191- C0 10 4390 CPY #16
1193- 90 F6 4400 BCC .2
1195- 20 84 FE 4410 JSR SETNORM NORMAL MODE NOW
1198- E6 0A 4420 INC MAX.LINE COUNT THE LINE
119A- 4C 8E FD 4430 JMP CROUT
-----
4440 *

```

```

119D- A4 09 4460 LDY SEL.LINE CURRENT BRIGHT LINE
119F- AD 00 CO 4470 .1 LDA $C000 READ KEY FROM KEYBOARD
11A2- 10 FB 4480 BPL .1
11A4- 8D 10 CO 4490 STA $C010 CLEAR THE STROBE
11A7- C9 B0 4500 CMP #B0 CHECK FOR "0"... "7"
11A9- 90 0F 4510 BCC .12
11AB- C9 B8 4520 CMP #B8
11AD- F0 25 4530 BEQ .25
11AF- B0 EE 4540 BCS .1
11B1- 0A 4550 ASL
11B2- 0A 4560 ASL
11B3- 0A 4570 ASL
11B4- 0A 4580 ASL LEAVE SLOT#16 IN A, CARRY SET
11B5- 8D E1 12 4590 .11 STA ONLINE+1 CHANGE ONLINE CALL
11B8- A9 9B 4600 LDA #9B SIMULATE AN <ESCAPE>
11BA- C9 8D 4610 .12 CMP #8D
11BC- F0 15 4620 BEQ .2 <RETURN>
11BE- C9 88 4630 CMP #88 <-
11C0- F0 19 4640 BEQ .3
11C2- C9 95 4650 CMP #95 -->
11C4- F0 26 4660 BEQ .7
11C6- C9 8A 4670 CMP #8A DOWN ARROW
11C8- F0 22 4680 BEQ .7
11CA- C9 8B 4690 CMP #8B UP ARROW
11CC- F0 0D 4700 BEQ .3
11CE- C9 9B 4710 CMP #9B ESCAPE
11D0- D0 CD 4720 BNE .1 GET ANOTHER CHARACTER
11D2- 0A 4730 ASL ...SET .NE. and CARRY
11D3- 60 4740 .2 RTS
11D4- AD E1 12 4750 .25 LDA ONLINE+1 TRY DRIVE 2
11D7- 09 80 4760 ORA #80 ...ALWAYS
11D9- D0 DA 4770 BNE .11
-----
4780 *---<UP OR LEFT ARROW>-----
11DB- 18 4790 .3 CLC
11DC- 88 4800 DEY IS CURRENT BRIGHT LINE TOP LINE?
11DD- 10 21 4810 BPL .8 ...NOT TOP LINE
11DF- A4 07 4820 LDY DIR.START ARE WE DISPLAYING THE FIRST ONE?
11E1- F0 05 4830 BEQ .5 ...YES
11E3- C6 07 4840 DEC DIR.START ...NO, MOVE TOWARD FIRST LINE
11E5- A0 00 4850 .4 LDY #0 MAKE FIRST LINE BRIGHT
11E7- 60 4860 RTS
11E8- A4 0A 4870 .5 LDY MAX.LINE MAKE LAST LINE BRIGHT
11EA- 88 4880 DEY
11EB- 60 4890 RTS
-----
4900 *---<DOWN OR RIGHT ARROW>-----
11EC- C8 4910 .7 INY MOVE TOWARD LAST LINE
11ED- C4 0A 4920 CPY MAX.LINE BEYOND END OF SCREEN?
11EF- 90 0F 4930 BCC .8 ...NO
11F1- A5 08 4940 LDA MAX.DIRPNT ...YES, CHECK IF SHOWING LAST LINE
11F3- E9 11 4950 SBC #17
11F5- 90 EE 4960 BCC .4 ...YES
11F7- C5 07 4970 CMP DIR.START
11F9- 90 EA 4980 BCC .4 ...YES
11FB- E6 07 4990 INC DIR.START ...NO, MOVE TOWARD LAST LINE
11FD- A4 09 5000 LDY SEL.LINE
11FF- 18 5010 CLC
1200- 60 5020 .8 RTS
-----
5030 *
5040 DISPLAY.FILES
1201- 20 43 12 5050 JSR SETUP.DISPLAY.LOOP
1204- A5 07 5060 LDA DIR.START
1206- 85 06 5070 STA DIR.INDEX
1208- 20 37 12 5080 JSR CLEAR.LINE.OR.PRINT.MORE.MSG
-----
5090 *
120B- A6 06 5100 .1 LDX DIR.INDEX
120D- BC 00 OD 5110 LDY DIRBUF+256,X
1210- F0 21 5120 BEQ .4 ...END OF LIST
1212- 84 01 5130 STY BPNTR+1
1214- BD 00 OC 5140 LDA DIRBUF,X
1217- 85 00 5150 STA BPNTR
1219- 20 51 12 5160 JSR CHECK.FOR.SEL.LINE
-----
5170 *
121C- A0 10 5180 .2 LDY #10
121E- B1 00 5190 LDA (BPNTR),Y
1220- 30 03 5200 BMI .3 ...SYS FILE
1222- A0 5C 5210 LDY #Q.DIR
1224- 2C 5220 .HS 2C
1225- A0 54 5230 .3 LDY #Q.SYS
1227- 20 68 12 5240 JSR MSG
122A- 20 79 11 5250 JSR PRINT.BPNTR.NAME

```

```



122D- E6 06 5270 *-----
122F- C6 0C 5280 INC DIR.INDEX
1231- D0 D8 5290 DEC LENGTH
1233- A5 06 5300 BNE .1
1235- C5 08 5310 LDA DIR.INDEX
5320 CMP MAX.DIRPNT
5330 *-----
1237- FO 04 5340 CLEAR.LINE.OR.PRINT.MORE.MSG
1239- AO 64 5350 BEQ .1 CLEAR LINE
123B- D0 2B 5360 LDY #Q.MORE
123D- 20 9C FC 5370 BNE MSG ... ALWAYS
1240- 4C 8E FD 5380 JSR CLREOL
5390 JMP CROUT
5400 *-----
1243- A9 10 5410 SETUP.DISPLAY.LOOP
1245- 85 0C 5420 LDA #16 MAX 16 LINES IN LIST
1247- AO 00 5430 STA LENGTH
1249- 84 0A 5440 LDY #0
124B- C8 5450 STY MAX.LINE
124C- 84 25 5460 INY SAME AS VTAB 3, HTAB 1
124E- 4C 8E FD 5470 STY CV
5480 JMP CROUT
5490 *-----
1251- A5 0A 5500 CHECK.FOR.SEL.LINE
1253- C5 09 5510 LDA MAX.LINE SEE IF CURRENT LINE SHOULD
1255- D0 0C 5520 CMP SEL.LINE BE INVERSE MODE
1257- A5 00 5530 BNE .1 ... NO
1259- 85 02 5540 LDA BPNTN ... YES, SO SETUP POINTER
125B- A5 01 5550 STA SPNTR
125D- 85 03 5560 LDA BPNTN+1
125F- A9 3F 5570 STA SPNTR+1 & SET INVERSE MODE
1261- 85 32 5580 LDA #3F
1263- 60 5590 STA INVFLG
5600 RTS
1264- 20 ED FD 5610 MSG1 JSR COUT
1267- C8 5620 INY
1268- B9 6E 12 5630 MSG LDA QTS Y
126B- D0 F7 5640 BNE MSG1
126D- 60 5650 RTS
5660 *-----
126E- 00- 5670 QTS .EQ *
126E- 5680 Q.SDV .EQ *-QTS
127E- 00 5690 >ASC "S/D Volume Name"
11- 5700 .HS 00
127F- 8D 5710 Q.VHELP .EQ *-QTS
1280- 5720 .HS 8D
129F- 8D 5730 >ASC "Select with ARROWS and <RETURN>"
12A0- 5740 .HS 8D
12C0- 8D 00 5750 >ASC "See Volumes with <ESCAPE> or 0-8"
54- 5760 .HS 8D00
12C2- 5770 Q.SYS .EQ *-QTS
12C9- 00 5780 >ASC "SYS -- "
5C- 5790 .HS 00
12CA- 5800 Q.DIR .EQ *-QTS
12D1- 00 5810 >ASC "DIR -- "
64- 5820 .HS 00
12D2- 5830 Q.MORE .EQ *-QTS
12DC- 8D 00 5840 >ASC " <<<MORE>>>"
5850 .HS 8D00
5860 *-----
12DE- 01 00 5870 CLOSE .DA #1,#0
12E0- 02 00 00
12E3- 20 5880 ONLINE .DA #2,#0,BUFFER
12E4- 03 80 02
12E7- 00 08 5890 OPEN .DA #3,PATHNAME,OPNBUF
12E9- 5900 O.REF .BS 1
12EA- 04 5910 READ .DA #4
12EB- 5920 R.REF .BS 1
12EC- 00 20 00
12EF- 9F 5930 .DA BUFFER,$9F00
12F0- 5940 ACTLEN .BS 2
12F2- 01 80 02 5950 PATH .DA #1,PATHNAME
5960 *-----
12F5- 5970 .BS $1300-*7
5980 QPATCH.EP
5990 .EP
13EA- 8D E1 12 6000 QPATCH STA ONLINE+1
13ED- 8D ED 13 6010 STA *
13FO- 60 6020 RTS
6030 *-----

```

Sometimes there just is not enough memory to hold the entire symbol table of a large assembly, especially in the ProDOS version of the S-C Macro Assembler. In that version, the symbol table normally begins at \$1000 and grows upward, while the source program snugs up against \$7400, hanging downward from there. Even with the use of the .INB directive to bring in segments of the source code one disk block at a time, extra-large programs can run out of memory during assembly. It can be especially frustrating in an Apple //e or later machine, when you know there is a lot of free memory just across the Soft-Switch River, over there in Aux-land.

Until now I have resisted using this memory, trying to remain fully compatible with older 64K machines and trying to keep the speed advantages of all-main-memory. But finally, the need became personal enough. I created a special version which puts the entire symbol table in Aux RAM. It will run on a 128K or larger //e, //c, or IIGs; I haven't tried it, but it should also run in an Apple II Plus with an Applied Engineering Transwarp card plugged and turned on. The symbol table still begins at \$1000, but in AuxRAM; and it can rise as high as \$BFFF without challenge. That is \$B000 total bytes, or about twice as many as were available between \$1000 and the bottom of the source program in MainRAM. The space below \$1000, down as far as \$800, is occupied by macro private labels, if you use any. Obviously, there is also now more room in Main RAM for your source code and/or object code. A Minus: if you have a /RAM disk installed in AuxRAM, the symbol table walks all over it. However, if you are using a RamWorks card or the like, with their PRODRIVE software, bank 0 of AuxRAM is left available for just these type uses.

If you need something like this, it's finally here. I'm calling it Version 2.1, and as a registered owner of the ProDOS version of the S-C Macro Assembler you can have a copy for only \$10.

PROGRAMMER



Applied Engineering is seeking an experienced 6502 and 65816 machine language programmer. 2 years minimum programming experience is required. We offer an exciting opportunity for the experienced programmer to take his skills to the limit. Applied Engineering offers an excellent compensation package including paid vacations, 11 paid holidays per year, health insurance program and more.

Applied Engineering's location in the suburbs of north Dallas offers a "buyer's market" for housing, as well as excellent schools, shopping and entertainment.

Successful applicant should have heavy machine language experience on the Apple IIe and IIGs as well as familiarity with AppleWorks. We're the best at what we do; if you are too, please send your resume to:

Applied Engineering
P.O. Box 5100
Carrollton, TX 75011
Attn: Personnel

WANTED

At long last, Addison-Wesley assures me they have printed the "IIgs Toolbox Reference" manuals, Volumes 1 and 2. These are probably indispensable tools for any serious IIgs programmers. Until now, you could only get them in beta versions through APDA. They are over 750 pages each, and cost \$26.95 each (\$24 plus shipping charge if you order from us). Each of the standard tools is described in great detail, with examples in both assembly language and C.

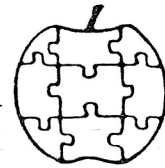
If they kept the same arrangement as my pre-beta copy (dated Nov 1986), there are a total of 23 chapters in the set. In my edition, Volume 1 covers toolsets in general, Desktop Bus, Control Mngr, Desk Mngr, Dialog Mngr, Event Mngr, Font Mngr, Integer Math, Line Edit, Memory Mngr, Menu Mngr, Misc.Tools, and Print Mngr; Volume 2 covers Quickdraw, SANE, Scheduler, Scrap Mngr, Sound Mngr, Std. File Operations, Text.tools, Tool Locator, and Window Manager. Volume 2 also includes an appendix on writing your own tool set.

There remains one more book in the IIgs series, the "IIgs Programmer's Introduction". A-W is now predicting April '88 for the publishing date, and a price of \$32.95. I guess I am old-fashioned, but to my mind this book should have been published FIRST, and included FREE with every IIgs purchase. Oh, well.... We will accept orders on this one now, and hold them until the book comes, at a price of \$30 plus shipping (see note on page 3 for details on shipping charges).

DON LANCASTER STUFF			
INTRODUCTION TO POSTSCRIPT	ASK THE GURU	APPLE IIc/IIe ABSOLUTE RESET	POSTSCRIPT SHOW & TELL
A 65 min user group VHS video with Don Lancaster sharing many of his laser publishing and Postscript programming secrets. Includes curve tracing, \$5 toner refilling, the full Kroy Kolor details, page layouts, plus bunches more.	An entire set of reprints to Don Lancaster's ASK THE GURU columns, all the way back to column one. Edited and updated. Both Apple and desktop publishing resources are included that are not to be found elsewhere.	Now gain absolute control over your Apple! You stop any program at any time. Eliminates all dropouts on your HIRES screen dumps. Gets rid of all hole blasting. For any IIc or IIe.	Unique graphics and text routines the others don't even dream of. For most any Postscript printer. Fully open, unlocked, and easily adaptable to your own needs. Available for Apple, PC, Mac, ST, many others.
\$39.50	\$24.50	\$19.50	\$39.50
FREE VOICE HELPLINE			VISA/MC
SYNERGETICS			
Box 809-SC	Thatcher, AZ 85552		(602) 428-4073

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$24 per year in the USA, Canada, and Mexico, or \$36 in other countries. Back issues are \$1.80 each for Volumes 1-7, \$2.40 each from later Volumes (plus postage). A subscription to the newsletter with a Monthly Disk containing all program source code and article text is \$64 per year in the USA, Canada and Mexico, and \$90 to other countries.

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)



Peeking Inside AppleWorks 1.3, Part 3:

Keyboard Input Subroutines	2
Percentage Printer	20
Another Quick Two-Digit Decimal Printer	24
New Versions of S-C Word Processor	25
Printing the ProDOS Date and Time	27

Another Way to Assemble into SYS Files

In the August 1987 issue of AAL I told how to patch the ProDOS version of S-C Macro Assembler so that target files (.TF directive) could be type SYS rather than BIN. Unfortunately the patches only work if the file is not already on the disk. In order to avoid an error message you can simply delete the target file before each assembly, and the easiest way to do this is by making an EXEC file. For example, I was working on a large program. SC.ACF is the file with a lot of include (.IN) directives. SCWP.SYSTEM is the target file. With the following three lines of text in an EXEC file named ASM, I can do an assembly by merely typing "--ASM":

```
DELETE SCWP.SYSTEM
LOAD SC.ACF
ASM
```

Unless the assembler is patched, SCWP.SYSTEM will be a type BIN file after assembly. With the following three lines in an EXEC file named SYS, I can change the filetype of SCWP.SYSTEM from BIN to SYS by typing "--SYS":

```
VERIFY SCWP.SYSTEM
$800:A9 07 8D B4 BE A9 FF 8D B8 BE 20 00 BF C3 B4 BE 4C DA FD
MGO$800
```

The VERIFY command reads the file info into a buffer at \$BEB4. The code being poked into \$800 changes the buffer and does an MLI call to SET FILE INFO. It then prints out the error code returned, which we hope is 00. If not 00, there was an error.

Peeking Inside AppleWorks 1.3, Part 3:
 Keyboard Input Subroutines...Bob Sander-Cederlof

In this, the third of the series delving into the AppleWorks code, we will look into the low-level keyboard input code and a few associated routines. It is probably not extremely exciting, but it is fundamental, and just the right amount of code for this issue of AAL.

This time I decided to give an exact disassembly, rather than rewriting routines to my own liking as I worked through them. Nevertheless, I could not resist showing you my revisions of two subroutines, as you will see below. Until now, I have been working from a "raw" disassembly listing: that is, one produced with the Apple monitor "L" command, and my pencil. After figuring out what I wanted to reproduce, I sat down to the keyboard and typed in an equivalent program. This time I used the S-C DisAssembler to produce a set of source files, and then worked them over with the editing facilities of the S-C Macro Assembler. The result will re-assemble to an exact copy of APLWORKS.SYSTEM (version 1.3). Finally, I lifted out the subroutines I wanted to discuss this month and put them together in one source file. The result is on the following pages.

The S-C DisAssembler made my work a lot easier, but it did not do everything. It does work from a script which can detail which address ranges are code, and which are data. The version I was using also could differentiate between hex strings, ASCII strings, and address tables. Furthermore, SCDA lets me give label names in advance when I know what I want to call them. However, SCDA does not handle all the parameters following JSR XXXX lines which are part of the AppleWorks code. It does handle JSR \$BF00, which is the ProDOS MLI call, but not the multitude of AppleWorks calls which I discussed in the December 1987 issue of AAL. SCDA also does not automatically generate local labels within subroutines. All labels not already given names are generated as "I.xxxx", where xxxx is the hexadecimal address. I have gone through the source code and replaced these with either meaningful names or local labels. I did leave one "I." label, because I did not know what to call it, in line 1490. SCDA also generates "X." labels for variables or subroutines outside the range being disassembled, and "Z." labels for page-zero references. I have left some of these in that form, pending more information about what they ought to be called.

If you look at the listing that follows, you may notice some strange comment lines that are filled with hexadecimal numbers. For example, see line 1640:

```
1640 * (1D35) 1066 1192 137F 193F 19C5 1BFD
```

These lines were produced by the S-C DisAssembler, and are embedded cross-reference lines. The number in parentheses is the address of the label which follows on the next line, in this case "AW.KEYIN". The list of numbers after the parentheses are addresses of instructions which refer to

S-C Macro Assembler Version 2.0	DOS \$100, ProDOS \$100, both for \$120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners	\$20
ProDOS Upgrade Kit for Version 2.0 DOS owners	\$30
Cross Assemblers for owners of S-C Macro Assembler	\$32.50 to \$50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 80A8, 8051, 8085, 1802/4/5, PDP-11, GI1650/70, Mitsubishi 50740, others)	
Source Code of any S-C Macro Assembler or Cross Assembler	each, additional \$100
S-C DisAssembler (ProDOS only)	without source code \$30, with source \$50
RAK-Ware DISASM (DOS only)	without source code \$30, with source \$50
ProVIEW (ProDOS-based disk utility program)	\$20
Full Screen Editor for S-C Macro (with complete source code)	\$49
S-C Cross Reference Utility	without source code \$20, with source \$50
S-C Word Processor, both DOS & ProDOS, both 40- & 80-columns, with complete source code	\$50
DP18 and DPPP, double precision math for Applesoft, including complete source code	\$50
ES-CAPE (Extended S-C Applesoft Program Editor), including Version 2.0 With Source Code	\$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)	\$30
Bag of Tricks 2 (Quality Software)	(\$49.95) \$45
S-C Documentor (complete commented source code of Applesoft ROMs)	\$50
Copy II Plus (Central Point Software)	(\$39.95) \$30
AAL Quarterly Disks	each \$15, or any four for \$45

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Vinyl disk pages, 6"x8.5", hold two disks each	10 for \$6
Diskette Mailing Protectors (hold 1 or 2 disks)	40 cents each
(Cardboard folders designed to fit 6"x9" Envelopes.)	or \$25 per 100
Envelopes for Diskette Mailers	6 cents each
Sider 20 Meg Hard Disk, includes controller & software	(\$695) \$550 +
Sider 40 Meg Hard Disk, includes controller & software	(\$995) \$860 +
Minuteman 250 Uninterruptible Power Supply	(\$359) \$320 +
Minuteman 300 Uninterruptible Power Supply	(\$549) \$490 +
65802 Microprocessor, 4 MHz (Western Design Center)	\$25
quikLoader EPROM System (SCRG)	(\$179) \$170
PROMGRAMMER (SCRG)	(\$149.50) \$140

"Exploring the Apple IIgs"	Gary B. Little	(\$22.95) \$21
"Apple IIgs Technical Reference"	Michael Fischer	(\$19.95) \$19
"65816/65802 Assembly Language Programming"	Michael Fischer	(\$21.95) \$20
"Programming the 65816"	David Eyes & Ron Lichty	(\$22.95) \$21
"Programming the Apple IIgs in C and Assembly Language"	Mark Andrews	(\$18.95) \$18
"Apple //e Reference Manual"	Apple Computer	(\$24.95) \$23
"Apple //c Reference Manual"	Apple Computer	(\$24.95) \$23
"ProDOS-8 Technical Reference Manual"	Apple Computer	(\$29.95) \$27
"ProDOS-16 Technical Reference Manual"	Apple Computer	(\$29.95) \$27
"Apple IIgs Firmware Reference"	Apple Computer	(\$24.95) \$23
"Apple IIgs Hardware Reference"	Apple Computer	(\$24.95) \$23
"Apple IIgs Toolbox Reference, Volume 1"	Apple Computer	(\$26.95) \$24
"Apple IIgs Toolbox Reference, Volume 2"	Apple Computer	(\$26.95) \$24
"Apple IIgs Programmer's Introduction"	Apple Computer	(\$32.95) \$30
"ProDOS Inside and Out"	Dennis Doms & Tom Weishaar	(\$16.95) \$16
"Beneath AppleProDOS"	Don Worth & Pieter Lechner	(\$19.95) \$18
"Beneath Apple DOS"	Don Worth & Pieter Lechner	(\$19.95) \$18
"Inside the Apple //c"	Gary B. Little	(\$19.95) \$18
"Inside the Apple //e"	Gary B. Little	(\$19.95) \$18
"Understanding the Apple //e"	Jim Sather	(\$24.95) \$23
"Understanding the Apple II"	Jim Sather	(\$22.95) \$21
"Apple II+/IIe Troubleshooting & Repair Guide"	Brenner	(\$19.95) \$18
"Assembly Language for Applesoft Programmers"	Finley & Myers	(\$18.95) \$18
"Now That You Know Apple Assembly Language"	Jules Gilder	(\$19.95) \$18
"Enhancing Your Apple II, vol. 1"	Don Lancaster	(\$15.95) \$15
"Enhancing Your Apple II, vol. 2"	Don Lancaster	(\$17.95) \$17
"Assembly Cookbook for the Apple II/IIe"	Don Lancaster	(\$21.95) \$20
"Microcomputer Graphics"	Roy E. Myers	(\$14.95) \$14
"Assembly Lines -- the Book"	Roger Wagner	(\$19.95) \$12

* These items add \$2 for first item, \$.75 for each additional item for US shipping.
 + Inquire for shipping cost.
 Customers outside USA inquire for postage needed.
 Texas residents please add 8% sales tax to all orders.
 << Master Card, VISA, Discover and American Express >>

S-C Software Corporation
 2331 Gus Thomasson #125
 DALLAS, TX 75228
 Phone 214-324-2050



AW.KEYIN. In this program, they just happen to all be JSR or JMP instructions. The disassembler produces lines like this for every label, and I left in the important ones.

Another thing you will notice in the listing is the heavy use of .PH and .EP directives. Each subroutine is surrounded by a pair of these. I wanted the listing here to show the same addresses as I found inside AppleWorks, and the .PH directive lets me do so. The object code that is stored in RAM during assembly of this source code will not be useful, because it will not be located in the addresses shown; rather, it will be all packed together starting at \$0800, the default object code address. But I do not intend to use the code in this form, just display it. If you want to use one or more of these subroutines, include them in your own code WITHOUT the .PH and .EP directives.

In order to assemble these subroutines without the rest of the body of APLWORKS.SYSTEM, I had to define some subroutines not included here. Lines 1450-1470 show these names. DISPLAY.STRING and BASE.CALC.A were included in the January 1988 issue of AAL. I intend to include PRINTER.DRIVER in a future issue.

The main subroutine I want to talk about this issue begins at line 1510, and I have called it AW.KEYIN. As line 1640 shows, this is called from a lot of places; its main purpose is to get the next character from the keyboard. It also does a lot of other stuff, as we will see.

I broke the KEYIN subroutine into four parts. The first part is lines 1640-1720, and is the only entry point used by AppleWorks. The second part (lines 1730-2570) handles the actual character input, getting a character either from the type-ahead buffer or from the keyboard. The third part (lines 2580-3560) checks for certain special characters and acts on them. The last part (lines 3570-3630) stores the character in \$84 and returns.

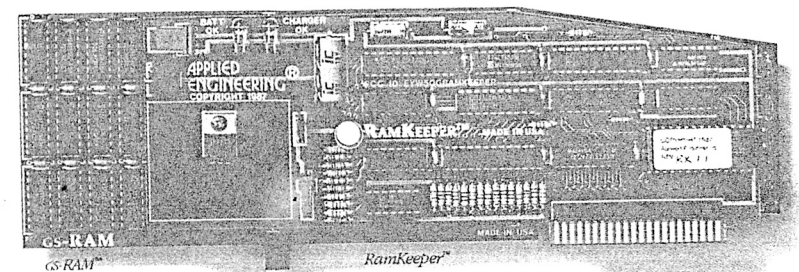
Looking at the first section first, notice lines 1660-1680. For reasons I have not yet determined, a flag is kept in pagezero location \$A4 which can cause all keyboard input to be bypassed. You may remember, this same location also controls screen display (see the Jan 88 article). In both cases, if \$A4 is non-zero, the operation is bypassed. You will get no display on the screen, and no keystrokes will be waited for. KEYIN will tell the world you typed an ESCAPE key, no matter what you may have really done. I don't know why all this is here yet.

Assuming \$A4 is zero, Lines 1690-1720 will save the current cursor position. Apparently the cursor position may be changed in some circumstances within the KEYIN routine, and so we save them for later restoration.

The second part of KEYIN gets the next character. The "next character" may have already been typed a while ago, and tucked away in the type-ahead buffer. If so, KEYIN will get it from there. If not, it will wait until you type one.

RamKeeper™

For the "Instant On" Apple IIGs.



Permanent Storage with an "Electronic Hard Disk"

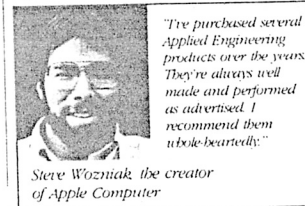
Now when you turn on your IIGs your favorite program can appear on screen in just a few seconds! With RamKeeper, your IIGs memory card will retain stored programs and stored data while your IIGs is turned off. RamKeeper allows you to divide your IIGs memory into part "electronic hard disk" and part RAM for your programs workspace—in almost any way you want and at anytime you want. GS-RAM, GS-RAM Plus, Apple IIGs memory card and most other IIGs memory cards are compatible with RamKeeper.

Supports Up to Two IIGs Memory Cards at the Same Time

If you bought your IIGs with Apple's memory card and later wished you had the GS-RAM, no problem. RamKeeper will support both cards plugged into RamKeeper simultaneously!

How it Works

Just unplug your IIGs memory card



"I've purchased several Applied Engineering products over the years. They're always well made and performed as advertised. I recommend them wholeheartedly."

Steve Wozniak, the creator of Apple Computer

from your computer, plug your IIGs memory card into RamKeeper, plug RamKeeper into the IIGs memory slot. If you have another IIGs memory card, an additional card socket on RamKeeper will accommodate your second card. That's all there is to it!

Reliability from the Most Experienced

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple so you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, reliability is everything! That's why Applied Engineering uses the more dependable Gel-Cell's instead of Ni-Cad batteries (if Ni-Cad's aren't discharged periodically, they lose much of their capacity). RamKeeper has close to 6 times (about 6 hours) the "total power failure" back-up time of other systems. When power returns, RamKeeper automatically recharges the battery to a full charge. With power from your wall outlet, RamKeeper will back-up your IIGs memory cards RAM indefinitely.

RamKeeper Has and Does It All!

- Allows instant access to your programs without slow disk delays
- Configure Kilobytes or Megabytes of instant ROM storage for your favorite programs

- Reduces power strain to your internal IIGs power supply
- Contains back-up status LED's
- Can support up to two IIGs memory cards simultaneously
- Supports both 256K installed memory chip boards like GS-RAM and the Apple IIGs Memory Expansion Card as well as 1 MEG installed memory chip boards like GS-RAM Plus
- 5-year hassle-free warranty
- 15 day money back guarantee
- Proudly made in the U.S.A.
- RamKeeper comes complete with battery, software and documentation

Only \$179.00!

(GS-RAM card shown in photo not included)

Order Your RamKeeper Today!

See your dealer or call (214) 241-6060, 9:00-11:00 CST, 7 days a week, or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 if outside U.S.A.

AE APPLIED ENGINEERING™

The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

Prices subject to change without notice.

You will recall that I talked about the POLL.KEYBOARD subroutine and the type-ahead buffer in the December 1987 issue. Lines 3970-4300 of this month's code show two more subroutines associated with the type-ahead buffer. CLEAR.KEYBUF will empty the type-ahead buffer, and CHECK.KEYBUF will test whether there are any characters in the buffer or not. For some reason there does not appear to be any subroutine to get a character out of the type-ahead buffer; instead, that code is included in-line wherever it is needed. This is not very efficient, but it "is what is". Lines 1760-1850 are one example of this. If it were up to me, I think I would have written one subroutine which returned Carry Clear if there were no characters in the buffer, or Carry Set if there was a least one character. In the latter case, I would also return the next character from the buffer in the A-register. Something like this:

```

GET.CHAR.FROM.KEYBUF.IF.ANY
    LDA KEYBUF.OUT
    CMP KEYBUF.IN
    BEQ .2          ...buffer is empty
    TAX            Use the "out" index
    LDA KEYBUF,X   Get character from buffer
    INX            Advance buffer index
    CPX #10        Compare to buffer size
    BCC .1         ...not off the end yet
    LDX #0         Wrap-around to beginning
.1   STX KEYBUF.OUT New "out" index
    SEC            Indicate we got a character
    RTS
.2   CLC            Indicate buffer was empty
    RTS

```

If that subroutine existed, I could replace lines 1760-1850 with

```

1760 JSR GET.CHAR.FROM.KEYBUF.IF.ANY
1770 BCS .11      ...got a char

```

and eliminate the CHECK.KEYBUF subroutine. Cleaner code, in my estimation, and shorter too.

If there is no character waiting in the type-ahead buffer, lines 1860 and following will get one from the keyboard. This is not as simple as it sounds, because there are a lot of options. First, there are three possible choices for the kind of cursor display during keyboard input. As the comments in lines 1860-1940 say, you may choose a blinking underline cursor, a flashing block cursor, or no cursor at all. If you are an AppleWorks user, you will recognize the first two options as the insert and overwrite cursors. The third, no cursor at all, used when a message like "Press Any Key to Continue" is displayed. The options are selected by values in two variables I have named KEYIN.CURSOR.FLAG and KEYIN.CURSOR.TYPE.

Just in case we are going to display a cursor, lines 1960-2050 retrieve the current screen character at the cursor position and save it. If the current cursor position is in AUX RAM, then screen memory is left switched in the AUX position. Line 2540 will later switch it back to MAIN RAM, after a keystroke has been read.

Lines 2070-2080 test whether we have chosen to display a cursor or not. If not, lines 2090-2140 will call on the READ.KEYBOARD subroutine to wait for you to type something. READ.KEYBOARD (lines 3670-3950) is a little smarter than the average keyboard reader. First, it does not wait forever. You call it with a timeout value in the Y-register. The loop which polls the keyboard counts down the timeout value. If it reaches zero, READ.KEYBOARD returns with a FALSE status; if you type a key before timeout, READ.KEYBOARD returns with the character in the A-register and a TRUE status. In addition, notice at line 3750 that READ.KEYBOARD stores whatever character was in the A-register on the screen. This character is determined by your choice of a cursor display. Finally, this subroutine reads the Open- and Solid-Apple keys. If either one of them is depressed, bit 7 of the character value is made 1; if neither is depressed, bit 7 is made 0.

If you do want a cursor display, lines 2150-2170 decide which type you want. Lines 2180-2310 handle the blinking underline, and lines 2320-2460 handle the flashing block. The blinking underline is in reality a repeating series of three characters: an underline, a blank, and the original screen character. The timeout values are chosen so that the underline and blank each are on the screen for about 17% of the time, and the original character the remaining 66%. If you like changing such things, this is the place to do it. You can change the overall speed of the blink, or change the ratio, or change the characters. You might choose one of the mousetext characters here, just for fun, instead of the underline. For example, \$5C instead of \$DF would display an under- and over-line character.

The flashing block cursor display involves change the character retrieved from the screen to the inverse equivalent character, and alternating between it and the original character. Lines 2330-2420 put up the inverse character for 79% of the time, and lines 2430-2460 put up the normal character for 21% of the time.

Regardless of which of the two displays you choose, when you type a key control will go to ".10" at line 2480. Here the original screen character will be restored.

And, regardless of which of three cursor-types, you eventually wind up at ".11", line 2540. Here we turn screen RAM back to MAIN, and test whether you have opted for further analysis of the input character or not. The variable at \$FC4, which I did not give a meaningful name yet, controls that option. If \$FC4 is non-zero, the game is all over; if zero, the KEYIN.ANALYSIS section gets to do some work.

I have detailed what KEYIN.ANALYSIS does in the comments in lines 2580-2790. Any normal printing character will simply be passed to you without further action. Any control character will clear the type-ahead buffer and then be passed to you. Any Apple-character will also clear the type-ahead buffer, and if it is not one of the special ones it will be passed to you.

The "special" Apple-characters are what KEYIN.ANALYSIS is here for. Lines 2990-3040 map lower-case letters to the upper-case range. Apple-E is used to change the cursor-type. If you have a blinking underline cursor on the screen, typing Apple-E makes it change to a flashing block cursor. And vice versa. The cursor-toggling code is in lines 3500-3560. Apple-H is used to print the screen contents on your printer. The code to do the screen dump is located right here, in lines 3170-3400. Apple-Q, -S, and -Y are also looked for. If you type and Apple-/, lines 2900-2980 change it to an Apple-? and pass the new code on to you.

By the way, that code for the Apple-/ is sure strange. It tests for the "/" character THREE TIMES! Why? I can think of two reasonable answers. First, maybe there used to be two other keys besides "/" which were changed to "?"; if so, maybe the author simply patched the code the way it is now rather than revising and re-assembling it. Naw.... More likely is that this is a "hook" for catching knockoffs of the code. I have found other hooks, such as deliberate use of a 3-byte reference to a pagezero variable when a 2-byte instruction could have been used. Anyone copying the code to create an illegal copy of AppleWorks for sale under their own name might miss these hooks, and find themselves waking up in court.

The Screen Print code is rather interesting. It calls on a subroutine I call PRINTER.DRIVER for printing each line, but that subroutine is not listed here. There are four different types of calls to PRINTER.DRIVER, controlled by the value in the A-register. If (A) is zero, the printer is initialized. On an Apple //e this means sending out a control-I code, and "80N", which turns off the screen echo many printer interfaces would otherwise perform. On an Apple //c it sends out the controls to set up the baud rate and LF after CR for a serial ImageWriter printer. More on this in a future issue.

Lines 3230-3290 copy a line from the screen to a buffer starting at \$90J, and then print it on the printer. At lines 3250-3270 PRINTER.DRIVER is called with (A)=79, the line length, which means to print a string whose address follows the JSR instruction. Then at lines 3280-3290 PRINTER.DRIVER is called with (A)=\$FE, which means to print a carriage return. Finally, Lines 3340-3350 call PRINTER.DRIVER with (A)=\$FF, meaning to "close" the driver.

After the screen print is completed, lines 3370-3390 make sure the cursor is back where it started. However, I did not notice anywhere it ever got moved, so this may be redundant code. The subroutine called here, MOVE.CURSOR.TO.XY, is shown in lines 4320-4480. It uses a round-about method, building a string for last month's DISPLAY.STRING subroutine. The setup for



SPECIAL !!! EXPANDED RAM/ROM BOARD: \$39.00

Similar to our \$30 RAM/ROM dev board described below. Except this board has two sockets to hold your choice of 2-2K RAM, 2-2K ROM or even 2-4K ROM for a total of 8K. Mix RAM and ROM too. Although Apple limits access to only 2K at a time, soft switches provide convenient socket selection. Hard switches control defaults.

IMPROVED III][IN A MAC (ver 2.0): \$75.00

Now includes faster graphics, UniDisk support and more! Bi-directional data transfers are a snap! This Apple II emulator runs DOS 3.3/PRODOS (including 6502 machine language routines) on a 512K MAC or MACPLUS. All Apple II features are supported such as HI/LO-RES graphics, 40/80 column text, language card and joystick. Also included: clock, RAM disk, keyboard buffer, on-screen HELP, access to the desk accessories and support for 4 logical disk drives. Includes 2 MAC diskettes (with emulation, communications and utility software, plus DOS 3.3 and PRODOS system masters, including Applesoft and Integer BASIC) and 1 Apple II diskette.

SCREEN.GEN: \$35.00

Develop HI-RES screens for the Apple II on a Macintosh. Use MACPAINT (or any other application) on the MAC to create your Apple II screen. Then use SCREEN.GEN to transfer directly from the MAC to an Apple II (with SuperSerial card) or IIC. Includes Apple II diskette with transfer software plus fully commented SOURCE code.

MIDI-MAGIC for Apple //c: \$49.00

Compatible with any MIDI equipped music keyboard, synthesizer, organ or piano. Package includes a MIDI-out cable (plugs directly into modem port - no modifications required!) and 6-song demo diskette. Large selection of digitized QRS player-piano music available for 19.00 per diskette (write for catalog). MIDI-MAGIC compatible with Apple II family using Passport MIDI card (or our own input/output card w/drum sync for only \$99.00).

FONT DOWNLOADER & EDITOR: \$39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. Special functions (like expanded, compressed etc.) supported. Includes HIRES screen editor to create custom fonts and special graphics symbols. For Apple II, II+, //e. Specify printer: Apple Imagewriter, Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/85, or Okidata 92/192.

* FONT LIBRARY DISKETTE #1: \$19.00 contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

DISASM 2.2e : \$30.00 (\$50.00 with SOURCE Code)

Use this intelligent disassembler to investigate the inner workings of Apple II machine language programs. DISASM converts machine code into meaningful, symbolic source compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even provides label substitution. Address-based triple cross reference generator included. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his ASSEMBLY COOKBOOK.

The 'PERFORMER' CARD: \$39.00 (\$59.00 with SOURCE Code)

Converts a 'dumb' parallel printer I/F card into a 'smart' one. Simple command menu. Features include perforation skip, auto page numbering with date & title, large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), Okidata 82A/83A with Okigraph & Okidata 92/93.

'MIRROR' ROM: \$25.00 (\$45.00 with SOURCE Code)

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support.

RAM/ROM DEVELOPMENT BOARD: \$30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

C-PRINT For The APPLE //c: \$69.00

Connect standard parallel printers to an Apple //c serial port. Separate P/S included. Just plug in and print!

Unless otherwise specified, all Apple II diskettes are standard (not copy protected!) 3.3 DOS.

Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders OK.

RAK-WARE 41 Ralph Road W. Orange NJ 07052 (201) 325-1885



DISPLAY.STRING is also handled in a round-about way, using the code I called POINT.PSTR.AT.OA00, lines 4500-4600. I haven't yet found any reason why this code is not simpler. Why go through the back bedroom to get from the kitchen to the dining room? I think I would have written MOVE.CURSOR.TO.XY like this:

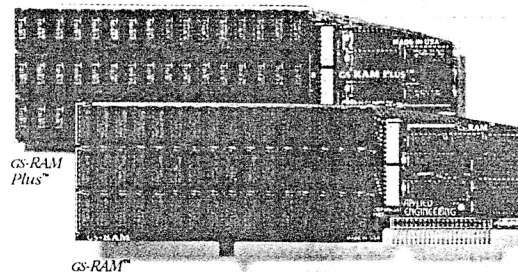
```
SC.MOVE.CURSOR.TO.XY
TXA
CLC
ADC AW.LEFT
STA AW.CH
TYA
ADC AW.TOP
STA AW.CV
RTS
```

The screen print code also calls on the subroutine I show in lines 4620-4940, COPY.SCRN.LINE.TO.0900. This subroutine picks up one line of characters, translating them from display codes to printer codes, and places the result in a buffer at \$0900. Another subroutine, MAP.SCRN.CHARS.TO.INTERNAL, does the translation. That MAP... subroutine is never called from anywhere else, so it really should be considered part of the COPY... subroutine. Together they take 72 bytes. I rewrote COPY..., and my simpler, shorter version is shown in lines 5240-5600. Mine only takes 50 bytes. If someone had time to go over the whole program the same way, there might be room for a lot of new features.

```
1000 #SAVE AW.SUBS.3
1010 #-----
14-- 1020 AW.CH .EQ $14
15-- 1030 AW.CV .EQ $15
16-- 1040 AW.BASE .EQ $16,17
80-- 1050 PSTR .EQ $80,81
98-- 1060 Z.B4 .EQ $84
98-- 1070 PNTR .EQ $98,99
A4-- 1080 Z.A4 .EQ $A4
1090 #-----
0900- 1100 X.0900 .EQ $0900 Used during Screen Print (Apple-H)
0901- 1110 X.0901 .EQ $0901
1120 #-----
0A00- 1130 X.OA00 .EQ $0A00 Used for building little strings
0A01- 1140 X.OA01 .EQ $0A01
0A02- 1150 X.OA02 .EQ $0A02
1160 #-----
0EA7- 1170 X.OEA7 .EQ $0EA7
1180 #-----
0F3B- 1190 X.OF3B .EQ $0F3B
0F3D- 1200 X.OF3D .EQ $0F3D
0FC4- 1210 X.OFC4 .EQ $0FC4
1220 #-----
1099- 00 OA 1230 .PH $1099
1240 HANDLE.OA00 .DA X.OA00
1250 .EP
1260 #-----
1270 #-----
1280 .PH $1176
1176- 01 1290 KEYIN.CURSOR.TYPE .HS 01 00=underline, 01=flashing
1177- 01 1300 KEYIN.CURSOR.FLAG .HS 01 00=no cursor, 01=cursor
1178- 01 1310 .BS 2 other variables
117A- 00 1320 KEYBUF .BS 10 type-ahead buffer
1184- 00 1330 KEYBUF.IN .HS 00
1185- 00 1340 KEYBUF.OUT .HS 00
1350 .EP
1360
```

For Those Who Want the Most. From Those Who Make the Best. GS-RAM™

Now expand the IIGS™ RAM and ROM with up to 8 MEG of "Instant On" memory with the all new GS-RAM!



GS-RAM has an all new design. A design that delivers higher performance including increased speed, greater expandability, and improved software.

More Sophisticated, Yet Easier to Use

GS-RAM works with all IIGs software. In fact any program that runs on Apple's smaller memory card runs on the GS-RAM. But with GS-RAM you can have more memory, improved performance, and almost unlimited expansion capabilities. We've designed the new GS-RAM to be easier to use too—you don't have to adjust the size of your RAM disk every time you use a DMA device. No other RAM card with more than 4 banks of memory installed can make the same claim.

More than Just Hardware

Each GS-RAM and GS-RAM Plus includes the most powerful set of IIGs software enhancements available anywhere. In fact, our nearest competitor offers only a fraction of the invaluable programs that we include with each GS-RAM card. This software includes the most powerful disk-caching program available, the GS-RAM Cache. The Cache will make most of your applications run up to 7 times faster. Also included is a diagnostic utility that lets you test your GS-RAM by graphically showing the location of any bad or improperly installed RAM chips. And for AppleWorks users, we give you our exclusive Expander program that dramatically enhances both the capabilities and speed of AppleWorks.

Making AppleWorks Even Better

Applied Engineering's Expander program eliminates AppleWorks internal memory limits allowing it to recognize up to 8 megabytes of desktop workspace. You can increase the limits from only 7,250 lines to 22,600 lines in the word processor and from 6,350 records to 22,600 records in the database. The Expander allows all of AppleWorks, including print functions, to automatically load into RAM. The clipboard size will increase from 255 to 2,042 lines maximum. GS-RAM will automatically segment larger files so you can save them onto multiple floppies. And

GS-RAM provides a built-in print buffer that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

Grow by Kilobytes or Megabytes

We offer GS-RAM in two configurations so you can increase your memory 256K at a time (GS-RAM) or a megabyte at a time (GS-RAM Plus). Both are IIGs compatible and both come with our powerful enhancement software. GS-RAM can hold up to 1.5 MEG of 256K chips and GS-RAM Plus can hold up to 6 MEG using 1 MEG chips. And since both use standard RAM chips (not high-priced SIMMs), you'll find expanding your GS-RAM or GS-RAM Plus easy, convenient, and very economical. For further expansion, you can plug a 2 MEG "piggyback" card into the GS-RAM's expansion port for up to 3.5 MEG of total capacity. Or up to a whopping 8 MEG on GS-RAM Plus. If a GS-RAM owner outgrows 3.5 MEG, he can easily upgrade to GS-RAM Plus for a nominal charge.

Permanent Storage for an "Instant On" Apple

With our RamKeeper™ back-up option, your GS-RAM or GS-RAM Plus will retain both programs and data while your IIGs is turned off! Now when you turn your IIGs back on, your favorite software is on your screen in under 4 seconds! With RamKeeper you can divide your IIGs memory into part "electronic hard disk" and part extended RAM. Even change the memory boundaries at any time—and in any way—you want. Because



Steve Wozniak, the creator of Apple Computer

"In quality, performance, compatibility, expandability and support, Applied Engineering's GS-RAM and GS-RAM Plus are number one."

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple, you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, Reliability is everything. That's why Applied Engineering uses state-of-the-art "GEL-CELLS" instead of Ni-Cad batteries (if Ni-Cads aren't discharged periodically, they lose much of their capacity). RamKeeper has about 6 hours of "total power failure" back-up time. That's 6 times the amount of other systems. But with power from your wall outlet, RamKeeper will back-up GS-RAM, GS-RAM Plus, or most other IIGs memory cards indefinitely. Should you ever have a "total power failure," RamKeeper switches to its 6 hour battery. When power returns, RamKeeper will automatically recharge the battery to full power. RamKeeper incorporates a dual-rate charger, status L.E.D.'s, and advanced power reducing circuitry. RamKeeper comes complete with battery, software, and documentation.

GS-RAM's Got it ALL!

- 5-year warranty — parts & labor
- 6 RAM banks (most cards have 4)
- Memory expansion port
- ROM expansion port
- Ultra-fast disk caching on ProDOS 8 AND ProDOS 16.
- Expands AppleWorks internal limits
- Includes hi-res self test
- No soldered-in RAM chips
- Expandable to 8 MEG
- No configuration blocks to set
- RamKeeper back-up option allows permanent storage of programs & data
- 15-day money-back guarantee
- Proudly made in the U.S.A.

GS-RAM with 256K	\$189
GS-RAM with 512K	\$259
GS-RAM with 1 MEG	\$399
GS-RAM with 1.5 MEG	\$539
GS-RAM with 2.5 to 3.5 MEG	CALL
GS-RAM Plus with 1-8 MEG	CALL
RamKeeper Option	\$179

Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside U.S.A.

AE APPLIED ENGINEERING™
The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006
Prices subject to change without notice

GS-RAM, GS-RAM Plus and RamKeeper are trademarks of Applied Engineering. Other brands and product names are registered trademarks of their respective holders.

```

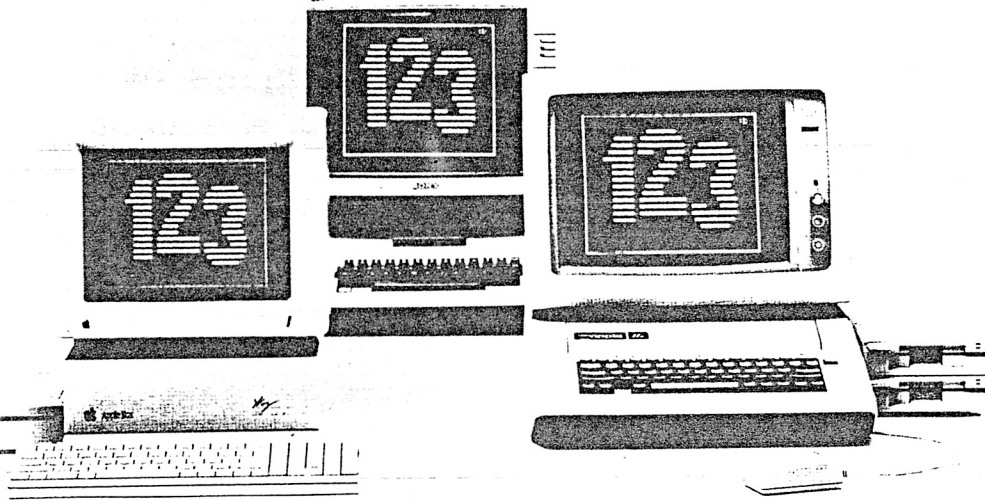
1370 *-----
1380 KEYBOARD .EQ $C000
1390 STROBE .EQ $C010
1400 SCR.N.MAIN .EQ $C054
1410 SCR.N.AUX .EQ $C055
1420 APPLE.OPEN .EQ $C061
1430 APPLE.SOLID .EQ $C062
1440 *-----
1450 DISPLAY.STRING .EQ $14D1 subroutine in AAL Jan 88
1460 BASE.CALC.A .EQ $1717 subroutine in AAL Jan 88
1470 PRINTER-DRIVER .EQ $1C21 subroutine in future AAL
1480 *-----
1490 I.1DOE .EQ $1DOE
1500 *-----
1510 .PH $1D30
1520 *-----
1530 * (1D30) 1D71 1D7B 1D8C 1D9A 1DA6 1DBC 1DCA
1540 KEYIN.CHAR.UNDER.CURSOR .BS 1
1550 * (1D31) 1D6C 1DC7 1FOF
1560 KEYIN.COLUMN.INDEX .BS 1
1570 * (1D32) 1E27 1E2A 1E3C 1E3F
1580 KEYIN.APPLE.H.LINE.NUMBER .BS 1
1590 * (1D33) 1D3E 1E4E
1600 KEYIN.CURSOR.CH .BS 1
1610 * (1D34) 1D43 1E51
1620 KEYIN.CURSOR.CV .BS 1
1630 *-----
1640 * (1D35) 1066 1192 137F 193F 19C5 1BFD
1650 AW.KEYIN
1660 LDA Z.A4 If non-zero, exit now as if
1670 BEQ .1 ...you typed <ESC>
1680 JMP KEYIN.EXIT.ESCAPE
1690 .1 LDA AW.CH Save current cursor position
1700 STA KEYIN.CURSOR.CH
1710 LDA AW.CV
1720 STA KEYIN.CURSOR.CV
1730 *-----
1740 * (1D46) 1E1A 1E57 1E78
1750 KEYIN.ANOTHER.CHAR
1760 JSR CHECK.KEYBUF Any characters in key-buffer?
1770 BEQ .2 ...no
1780 LDX KEYBUF.OUT ...yes, get char from keybuf
1790 LDA KEYBUF,X
1800 INX Bump keybuf out-index
1810 CPX #10 At end of buffer?
1820 BCC .1 ...no
1830 LDX #400 ...yes, wrap around to beginning
1840 .1 STX KEYBUF.OUT Save new keybuf out-index
1850 JMP .11 Use the character
1860 *-----
1870 * Key-buffer is empty, so we need to get a character
1880 * directly from the keyboard. Therefore, we must:
1890 * 1. Save character now on screen under cursor
1900 * 2. Put up an appropriate cursor
1910 * a. blinking underline
1920 * b. flashing screen char
1930 * c. no cursor at all
1940 * 3. Get a keystroke.
1950 *-----
1960 .2 LDA AW.CV Point to the character at the cursor
1970 JSR BASE.CALC.A
1980 LDA AW.CH
1990 LSR
2000 BCS .3 ...Odd column, main RAM
2010 STA SCR.N.AUX ...Even column, aux RAM
2020 TAY
2030 STY KEYIN.COLUMN.INDEX
2040 LDA (AW.BASE),Y
2050 STA KEYIN.CHAR.UNDER.CURSOR
2060 *-----Select type of cursor-----
2070 LDX KEYIN.CURSOR.FLAG
2080 BNE .5 ...we do want a cursor display
2090 *-----No cursor at all-----
2100 .4 LDY #3FF Long time-out count
2110 LDA KEYIN.CHAR.UNDER.CURSOR
2120 JSR READ.KEYBOARD
2130 BEQ .4 ...No key yet
2140 BNE .11 ...got a keystroke!

```

```

2150 *---Some type of cursor-----
2160 .5 LDX KEYIN.CURSOR.TYPE
2170 BNE .8 ...Use flashing character
2180 *---Use blinking underline-----
2190 Repeat loop of underline, char, blank, char
2200 .6 LDA #DF Underline character
2210 CMP KEYIN.CHAR.UNDER.CURSOR
2220 BNE .7 ...screen not now an underline
2230 LDA #* ...now underline, change to blank
2240 .7 LDY #28 Short time-out
2250 JSR READ.KEYBOARD
2260 BNE .10 ...got a key!
2270 LDA KEYIN.CHAR.UNDER.CURSOR
2280 LDY #108 Long time-out
2290 JSR READ.KEYBOARD
2300 BNE .10 ...got a key!
2310 BEQ .6 ...always (no keystroke yet)
2320 *---Use flashing character-----
2330 .8 LDA KEYIN.CHAR.UNDER.CURSOR
2340 AND #7F Change character to inverse
2350 CMP #40
2360 BCC .9
2370 CMP #60
2380 BCS .9
2390 AND #BF
2400 .9 LDY #6C
2410 JSR READ.KEYBOARD
2420 BNE .10 ...got a key!
2430 LDA KEYIN.CHAR.UNDER.CURSOR
2440 LDY #28 Short time-out
2450 JSR READ.KEYBOARD
2460 BEQ .8 ...no keystroke yet
2470 *---Got a key, restore scrnchar---
2480 .10 PHA
2490 LDY KEYIN.COLUMN.INDEX
2500 LDA KEYIN.CHAR.UNDER.CURSOR
2510 STA (AW.BASE),Y
2520 PLA
2530 *-----
2540 .11 STA SCR.N.MAIN Be sure in main RAM
2550 LDX X.OFC4 Should we analyze the char?
2560 BEQ KEYIN.ANALYSIS ...yes
2570 JMP KEYIN.EXIT ...no, just store and return
2580 *-----
2590 * Analyze the Character
2600 *
2610 * 1. If character is $20-7F, just store it and return.
2620 * 2. Otherwise, start by clearing the key-buffer.
2630 * 3. If character is $00-1F, it will be fall through all
2640 * other tests and return after being stored.
2650 * 4. If character is $80-FF, an Apple key was down.
2660 * 5. If character is $E1-FA, it is lower-case and will
2670 * be changed to upper-case ($C1-DA).
2680 * 6. If character is not one of the following, just
2690 * store it and return.
2700 *
2710 * Apple-/ Change to Apple-?, and return.
2720 * Apple-H Print the screen, get another char.
2730 * Apple-Q Clobber Z.A4, substitute <ESC>, return.
2740 * Apple-Y Change to Control-Y and return.
2750 * Apple-S If $EA7 non-zero, just store and return.
2760 * Else, clobber Z.A4, substitute <ESC>,
2770 * and return.
2780 * Apple-E Toggle Cursor Mode, get another char.
2790 *-----
2800 KEYIN.ANALYSIS
2810 CMP #20
2820 BCC .1 ...Control Character, analyze it.
2830 CMP #7F
2840 BCS .1 ...Apple Character, analyze it.
2850 JMP KEYIN.EXIT
2860 *---Analyze the keychar-----
2870 .1 PHA Save character temporarily...
2880 JSR CLEAR.KEYBUF
2890 PLA ...and get the character back.

```



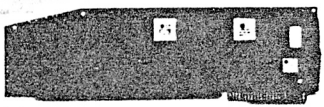
Now Apple speaks IBM. Three times faster than IBM.

Introducing PC Transporter.™
The Apple® II expansion board that lets you run MS®-DOS programs.

Now your Apple II can run over 10,000 programs you could never use before. Like Lotus® 1-2-3® MultiMate® dBASE III PLUS® Even Flight Simulator®

With PC Transporter, MS-DOS programs run on your Apple II like they do on IBM® PC's or compatibles. With one important difference. PC Transporter runs most of those programs *three times faster* than an IBM PC/XT®. Plus, to speed through number-crunching tasks, you can use our optional 8087-2 math coprocessor chip. It plugs into a socket on the PC Transporter.

Less expensive than an IBM clone.
Sure, a stripped-down IBM



clone costs about the same as the PC Transporter. But the peripherals it takes to get the clone up and running make the clone cost about three times what our American-made card costs.

You don't have to buy new hardware to use PC Transporter. **Works with the hardware you already own.**

With PC Transporter, MS-DOS programs see your Apple hardware as IBM hardware. You can use the same hardware you have now.

With IBM software, your Apple hardware works just like IBM hardware. Including your drives, monitors, printers, printer cards, clock cards and serial clocks.

You can use your IIe® or IIgs™ keyboard with IBM software. Or use our optional IBM-style keyboard (required for the II Plus). You can use your Apple mouse. Or an IBM compatible serial mouse.

Plenty of power.
PC Transporter gives you as much as 640K of user RAM and 128K of system RAM in the IBM mode.

PC Transporter also is an Apple expansion card, adding up to 768K of extra RAM in the Apple mode. The Apple expansion alone is a \$300 value.

Easy to install.
You can install PC Transporter in about 15 minutes, even if you've never added an expansion board. You don't need special tools. Simply plug it into an Apple expansion slot (1 through 7 except 3), connect a few cables and a disk drive, and go!



PC Transporter taps into the world's largest software library. Now your Apple can run most of the IBM software you use at work. And it opens a new world of communications programs, games and bulletin boards.

A universal disk drive controller.

PC Transporter supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes. You'll shift instantly between Apple ProDOS and IBM MS-DOS.

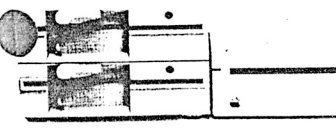
You'll need our versatile 5.25" 360K drive system to run IBM applications from 5.25" floppy disks. Use your Apple 5.25" drive for Apple 5.25" disks.

An Apple Disk 3.5 Drive will support the new 3.5" disks whether they're IBM MS-DOS formatted or Apple ProDOS formatted. The PC Transporter acts like an Apple Disk 3.5 Drive disk controller for IIgs, IIe, and II Plus users.

PC Transporter supports up to 5 drives in a number of combinations.

For example, you can connect a 5.25 Applied Engineering 360K dual-drive system directly to the card. Then plug two daisy-chained Apple 3.5 Drives (not the Apple UniDisk 3.5) to the dual-drive system. For a fifth drive, use a ProDOS file as an IBM hard disk.

PC Transporter controls Apple and IBM compatible disk drives. It supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes.



Versatile data storage.
PC Transporter reads MS-DOS and translates it into Apple native ProDOS. You can store IBM programs and data on any ProDOS storage device including the Apple 3.5 Drive, Apple UniDisk™ 3.5, Apple 5.25" drive, SCSI or ProDOS compatible hard drives. (You can use the Apple UniDisk 3.5 with its own controller card but not for directly booting an IBM formatted disk.)

You can even use our 360K PC compatible drive for ProDOS

Make your Apple speak IBM.

PC Transporter memory choices.

RAM in Apple mode:	RAM in IBM mode:	Price:
384K	256K	\$489.00
512K	384K	529.00
640K	512K	569.00
768K	640K	609.00

Note: The IBM mode is 128K less because the PC Transporter uses 128K for system memory.

IIgs Installation Kit 49.00

IIe/II Plus Installation Kit 39.00

PC Transporter Accessories

5.25" IBM Format 360K Drive Systems

Single-Drive System 269.00

Dual-Drive System 399.00

Half-Height Drive 135.00
(add to Single-Drive system to make Dual-Drive)

IBM-Style Keyboard 139.00
(required for Apple II Plus. Requires IBM Keyboard Cable.)

IBM Keyboard Cable 34.00

Sony RGB Monitor 499.00

Analog RGB Cable 39.00
(for use with Sony monitor)

Digital RGB Cable 39.00
(for use with Sony monitor)

Digital RGB Adapter 24.00

ColorSwitch 44.00
(included with IIgs Installation Kit)

128K ZIP 40.00

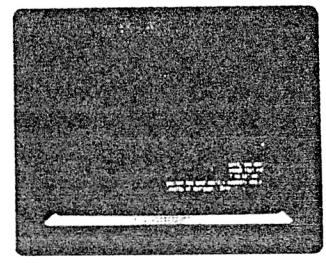
PC Transporter per set

Memory Expansion Chip Set

8087-2 Math Co-processor Chip 229.00

Heavy Duty Power Supply 69.00
(IIe and II Plus only)

See your dealer or call or send check or money order to Applied Engineering, MasterCard, VISA and COD welcome. Texas residents add 6 1/4% sales tax.



PC Transporter produces better IBM graphics than IBM. Analog is sharper than digital. So with an analog RGB monitor, PC Transporter's CGA graphics and text are superior to IBM's digital display — even while running IBM software! And, you can also use an Apple composite monitor in IBM text or graphics mode.

storage and a 143K Apple 5.25" drive for MS-DOS storage.

Created by Apple's original designers.

The brains behind PC Transporter were also behind your Apple II.

The PC Transporter design team includes the former project managers for the creation of the Apple IIe and IIc. The co-designer of the Apple II disk controller. And the first full-time Apple programmer and author of the ProDOS operating system.

So you know the PC Transporter and your Apple were made for each other.

Support and service from the leader in Apple add-ons.

Applied Engineering sells more Apple peripheral boards than anyone else — including Apple Computer. So you know we'll be around after the sale.

PC Transporter comes with a 15-day money back guarantee. If you're not fully satisfied after using it, return it for a full refund. PC Transporter also comes with a 1-year warranty.

How to get your PC Transporter today.

See your dealer. Or call Applied Engineering any day between 9 a.m. and 11 p.m. CST at 214-241-6060.

AE Applied Engineering
The Apple enhancement experts.
P.O. Box 798, Carrollton, TX 75006
214-241-6060
A Division of AE Research Corporation


```

2900 *---Check for Apple-Slash-----
1DFB- C9 AF 2910 CMP #"/"
1DFD- F0 08 2920 BEQ .2 ...Apple-Slash
1DEF- C9 AF 2930 CMP #"/" <<<I don't know why they do it 3 times>>>
1DF1- F0 04 2940 BEQ .2 <<<Maybe just for the fun of it.....>>>
1DF3- C9 AF 2950 CMP #"/"
1DF5- D0 05 2960 BNE .3 ...not Apple-/
1DF7- A9 BF 2970 .2 LDA #?" ...Substitute Apple-?
1DF9- 4C 7D 1E 2980 JMP KEYIN.EXIT
*---Map Lower-Case to Upper-----
1DFC- C9 E1 3000 .3 CMP #"a"
1DFE- 90 06 3010 BCC .4 ...not lower-case letter
1E00- C9 FB 3020 CMP #"z"+1
1E02- B0 02 3030 BCS .4 ...not lower-case letter
1E04- 29 DF 3040 AND #DF
*---If Apple-Y, make Ctrl-Y-----
1E05- C9 D9 3060 .4 CMP #*Y* Check for Apple-Y
1E08- D0 02 3070 BNE .5 ...no
1E0A- 29 1F 3080 AND #F1F Changes $D9 to $19, control-Y
*---Check for Apple-H-----
1E0C- C9 C8 3100 .5 CMP #*H* Check for Apple-H
1E0E- D0 4A 3110 BNE .9 ...not Apple-H
1E10- AD 0E 1D 3120 LDA I.1DOE ???
1E13- D0 05 3130 BNE .6 ...ignore the Apple-H
1E15- AD 3B 0F 3140 LDA X.OF3B
1E18- D0 03 3150 BNE .7 ...go ahead and print
1E1A- 4C 46 1D 3160 .6 JMP KEYIN.ANOTHER.CHAR
*---Print the screen-----
1E1D- 8D 3D 0F 3180 .7 STA X.OF3D
1E20- A9 00 3190 LDA #00 "OPEN" Printer
1E22- 20 21 1C 3200 JSR PRINTER.DRIVER
1E25- A9 00 3210 LDA #0 For LINE = 0 to 23
1E27- 8D 32 1D 3220 STA KEYIN.APPLE.H.LINE.NUMBER
1E2A- AD 32 1D 3230 .8 LDA KEYIN.APPLE.H.LINE.NUMBER
1E2D- 20 7A 18 3240 JSR COPY.SCRN.LINE.TO.0900
1E30- A9 4F 3250 LDA #79 Print 79 characters from 0900
1E32- 20 21 1C 3260 JSR PRINTER.DRIVER
1E35- 00 09 3270 .DA X.0900
1E37- A9 FE 3280 LDA #FE Print CRLF
1E39- 20 21 1C 3290 JSR PRINTER.DRIVER
1E3C- EE 32 1D 3300 INC KEYIN.APPLE.H.LINE.NUMBER Next Line
1E3F- AD 32 1D 3310 LDA KEYIN.APPLE.H.LINE.NUMBER
1E42- C9 18 3320 CMP #24 Last line yet?
1E44- 90 E4 3330 BCC .8 ...no, keep printing
1E46- A9 FF 3340 LDA #FF "CLOSE" Printer
1E48- 20 21 1C 3350 JSR PRINTER.DRIVER
1E4B- 20 E0 1F 3360 JSR CLEAR.KEYBUF
1E4E- AC 33 1D 3370 LDX KEYIN.CURSOR.CH Put cursor back...
1E51- AE 34 1D 3380 LDY KEYIN.CURSOR.CV ...but I don't know how
1E54- 20 23 18 3390 JSR MOVE.CURSOR.TO.XY it could have moved.
1E57- 4C 46 1D 3400 JMP KEYIN.ANOTHER.CHAR Get another character.
*-----
1E5A- C9 D1 3420 .9 CMP #*Q* Check for Apple-Q
1E5C- F0 09 3430 BEQ .10
1E5E- C9 D3 3440 CMP #*S* Check for Apple-S
1E60- D0 0A 3450 BNE .11
1E62- AE A7 0E 3460 LDX X.OEA7
1E65- D0 05 3470 BNE .11
1E67- 85 A4 3480 .10 STA Z.A4 Apple-Q or -S clobbers Z.A4
1E69- 4C 7B 1E 3490 JMP KEYIN.EXIT.ESCAPE
*---If Apple-E, change cursor---
1E6C- C9 C5 3500 .11 CMP #*E* Check for Apple-E
1E6E- D0 0D 3520 BNE KEYIN.EXIT
1E70- AD 76 11 3530 LDA KEYIN.CURSOR.TYPE
1E73- 49 01 3540 EOR #01 Toggle btwn $00 and $01
1E75- 8D 76 11 3550 STA KEYIN.CURSOR.TYPE
1E78- 4C 46 1D 3560 JMP KEYIN.ANOTHER.CHAR
*-----
3570 * (1E7B) 1D39 1E69
3580 KEYIN.EXIT.ESCAPE
1E7B- A9 1B 3600 LDA #1B Say you typed <ESC>
3610 KEYIN.EXIT
1E7D- 85 84 3620 STA Z.84 Store character here too
1E7F- 60 3630 RTS
*-----
3640 .EP
3650

```

```

FOA- *
3680 .PH $1FOA
3690 KEYBOARD.TIMEOUT .BS 2
3700 *
3710 * (1FOC) 1D7E 1D95 1D9F 1DB7 1DC1
3720 READ.KEYBOARD
1FOC- 8C 0B 1F 3730 STY KEYBOARD.TIMEOUT+1
1FOF- AC 31 1D 3740 LDY KEYIN.COLUMN.INDEX
1F12- 91 16 3750 STA (AW.BASE),Y
1F14- AD 00 CO 3760 .1 LDA KEYBOARD
1F17- 30 13 3770 BMI .2 ...got a keystroke
1F19- AD 77 11 3780 LDA KEYIN.CURSOR.FLAG
1F1C- F0 F6 3790 BEQ .1 If no cursor, then no time-out either
1F1E- CE 0A 1F 3800 DEC KEYBOARD.TIMEOUT
1F21- D0 F1 3810 BNE .1 ...more time left
1F23- CE 0B 1F 3820 DEC KEYBOARD.TIMEOUT+1
1F26- D0 EC 3830 BNE .1 ...more time left
1F28- A2 00 3840 LDX #00 timed out, return false
1F2A- F0 11 3850 BEQ .4 ...always
1F2C- 8D 10 CO 3860 .2 STA STROBE Clear the strobe
1F2F- AE 61 CO 3870 LDX APPLE.OPEN
1F32- 30 07 3880 BMI .3 Apple, leave bit 7 = 1
1F34- AE 62 CO 3890 LDX APPLE.SOLID
1F37- 30 02 3900 BMI .3 Apple, leave bit 7 = 1
1F39- 29 7F 3910 AND #7F No Apple, make bit 7 = 0
1F3B- A2 01 3920 .3 LDX #01 Return TRUE
1F3D- 60 3930 .4 RTS
3940 *-----
3950 .EP
3960
3970 .PH $1FEO
3980 *-----
3990 * (1FEO) 1084 181D 1939 1BF1 1DE7 1E4B
4000 * Clear type-ahead buffer
*-----
4010
4020 CLEAR.KEYBUF
1FE0- A9 00 4030 LDA #00
1FE2- 8D 84 11 4040 STA KEYBUF.IN
1FE5- 8D 85 11 4050 STA KEYBUF.OUT
1FE8- 60 4060 RTS
*-----
4070 .EP
4080
4090
4100 .PH $13B2
*-----
4110
4120 * (13B2) 137A 1D46
4130 * Check whether any characters are queued up in the
4140 * keyboard buffer. If so, return TRUE (status .NE.).
4150 * If not, return FALSE (status .EQ.).
4160 *-----
4170 CHECK.KEYBUF
13B2- AD 84 11 4180 LDA KEYBUF.IN If pointers are same, the buffer
13B5- CD 85 11 4190 CMP KEYBUF.OUT is empty.
13B8- F0 04 4200 BEQ .1 ...it is empty
13BA- A9 01 4210 LDA #01 ...not empty, return TRUE (.NE.)
13BC- D0 02 4220 BNE .2
13BE- A9 00 4230 .1 LDA #00
13C0- 60 4240 .2 RTS
*---Alternate code to do same---
4250 *** LDA KEYBUF.IN If pointers are same, the buffer
4260 *** CMP KEYBUF.OUT is empty.
4270 *** RTS .NE. if not empty, .EQ. if empty
*-----
4280
4290 .EP
4300
4310
4320 .PH $1823
*-----
4330
4340 * (1823) 1024 13A2 1A58 1A89 1B1B 1E54 1E86 1E90 20B6 2B66 2B82
4350 * Move cursor to column (X), line (Y)
4360 * Works by building a string for STRING.DISPLAY
*-----
4370
4380 MOVE.CURSOR.TO.XY
1823- 8E 01 0A 4390 STX X.OA01 Build string "05.XX.YY"
1826- 8C 02 0A 4400 STY X.OA02
1829- A9 05 4410 LDA #05 "GoToXY" code
182B- 8D 00 0A 4420 STA X.OA00
182E- 20 A9 1E 4430 JSR POINT.PSTR.AT.OA00
1831- A9 03 4440 LDA #3 String has 3 characters
1833- 20 D1 14 4450 JSR DISPLAY.STRING
1836- 60 4460 RTS

```

```

4470 *-----*
4480 .EP
4490
4500 .PH $1EA9
4510 *-----*
4520 * (1EA9) 182E 1F85 1FEC 208B 20CC
4530 POINT.PSTR.AT.CA00
1EA9- AD 99 10 4540 LDA HANDLE.OA00
1EAC- 85 80 4550 STA PSTR
1EAE- AD 9A 10 4560 LDA HANDLE.OA00+1
1EB1- 85 81 4570 STA PSTR+1
1EB3- 60 4580 RTS
4590 *-----*
4600 .EP
4610
4620 .PH $187A
4630 *-----*
4640 * (187A) 1036 1A81 1E2D
4650 * Used by Apple-H Screen Print function
4660 * (A) = line to be copied
4670 * Copies 80 characters to buffer at $0900
4680 *-----*
4690 COPY.SCRN.LINE.TO.0900
187A- 20 17 17 4700 JSR BASE.CALC.A
187D- A2 00 4710 LDX #0
187F- A0 00 4720 LDY #0
1881- B1 16 4730 .1 LDA (AW.BASE),Y
1883- 30 06 4740 BMI .2 80-FF
1885- 20 94 1E 4750 JSR MAP.SCRN.CHARS.TO.INTERNAL
1888- 4C 8D 18 4760 JMP .3
188B- 29 7F 4770 AND #$7F
188D- 9D 01 09 4780 .3 STA X.0901,X
1890- 8D 55 CO 4790 STA SCR.N.AUX
1893- B1 16 4800 LDA (AW.BASE),Y
1895- 30 06 4810 BMI .4
1897- 20 94 1E 4820 JSR MAP.SCRN.CHARS.TO.INTERNAL
189A- 4C 9F 18 4830 JMP .5
189D- 29 7F 4840 AND #$7F
189F- 9D 00 09 4850 .5 STA X.0900,X
18A2- 8D 54 CO 4860 STA SCR.N.MAIN
18A5- E8 4870 INX
18A6- E8 4880 INX
18A7- C8 4890 INY
18A8- C0 28 4900 CPY #40
18AA- 90 D5 4910 BCC .1 ...more on this line
18AC- 60 4920 RTS
4930 *-----*
4940 .EP
4950
4960 .PH $1E94
4970 *-----*
4980 * (1E94) 1885 1897
4990 *
5000 * Only called from subroutine which copies
5010 * a screen line to $0900, and only for
5020 * character values $00-7F.
5030 *
5040 * 00-1F to 40-5F
5050 * 20-3F no change
5060 * 40-5F to 80-9F (Mouse Graphics)
5070 * 60-7F no change
5080 *-----*
5090 MAP.SCRN.CHARS.TO.INTERNAL
1E94- C9 20 5100 CMP #$20
1E96- B0 04 5110 BCS .1 not 00-1F
1E98- 09 40 5120 ORA #$40 Change 00-1F to 40-5F
1E9A- D0 0C 5130 BNE .2 ...always
1E9C- C9 40 5140 .1 CMP #$40
1E9E- 90 08 5150 BCC .2 ...20-3F
1EA0- C9 60 5160 CMP #$60
1EA2- B0 04 5170 BCS .2 ...60-7F
1EA4- 29 BF 5180 AND #$BF Change 40-5F to 00-1F
1EA6- 09 80 5190 ORA #$80 and then to 80-9F
1EA8- 60 5200 .2 RTS
5210 *-----*
5220 .EP
5230

```

```

5240 *-----*
5250 * A Shorter Version of COPY.SCRN.LINE.TO.0900
5260 *
5270 SC.COPY.SCRN.LINE.TO.0900
0A15- 20 17 17 5280 JSR BASE.CALC.A
0A18- A2 00 5290 LDX #0
0A1A- A0 00 5300 LDY #0
0A1C- 8D 55 CO 5310 .1 STA SCR.N.AUX Even char first
0A1F- 20 2E 0A 5320 JSR GET.MAP.PUT.SCRN.CHAR
0A22- 8D 54 CO 5330 STA SCR.N.MAIN Then odd char
0A25- 20 2E 0A 5340 JSR GET.MAP.PUT.SCRN.CHAR
0A28- C8 5350 INY
0A29- C0 28 5360 CPY #40
0A2B- 90 EF 5370 BCC .1 ...more on this line
0A2D- 60 5380 RTS
5390 *-----*
5400 * 00-1F to 40-5F
5410 * 20-3F no change
5420 * 40-5F to 80-9F (Mouse Graphics)
5430 * 60-7F no change
5440 * 80-FF to 00-7F
5450 *-----*
5460 GET.MAP.PUT.SCRN.CHAR
0A2E- B1 16 5470 LDA (AW.BASE),Y
0A30- 30 0E 5480 BMI .1 Change 80-FF to 00-7F
0A32- C9 60 5490 CMP #$60 now have 00-7F
0A34- B0 0C 5500 BCS .2 ...60-7F no change
0A36- 69 40 5510 ADC #$40 Change 00-5F to 40-9F
0A38- 30 08 5520 BMI .2 ...40-5F became 80-9F
0A3A- C9 60 5530 CMP #$60
0A3C- 90 04 5540 BCC .2 ...00-1F became 40-5F
0A3E- 29 3F 5550 AND #$3F Change 60-7F back to 20-3F
0A40- 29 7F 5560 .1 AND #$7F Change 80-FF to 00-7F
0A42- E8 5570 .2 INX
0A43- 9D 00 09 5580 STA X.0900,X
0A46- 60 5590 RTS
5600 *-----*

```

EnterSoft:

Basic-like macros which make the complex simple. Don't re-write that multiplication routine for the hundredth time! Get EnterSoft instead! Do 8/16/32/64 bit Math/Input-Output/Graphics simply without all of the hassles. These routines are a must for the serious programmer who doesn't want to spend all of his/her time trying to re-invent the wheel. DOS 3.3 Version=\$30.00, ProDos Version=\$30.00, BOTH for only \$50.00. GET YOURS TODAY.

A Shape Table Program:

For once! A shape table program which is logically organized into its component parts. Each section resides in its own program. The editor, disk access, Hi-Res section; each section is separate. Written almost entirely in Basic, it is easily modified. Not copyprotected! Put them on a Hard Disk, Ram Drive, anywhere! DOS 3.3 Version=\$20.00, ProDos Version=\$20.00, BOTH for \$30.00!

Send Check or Money Order To:	ProDos Upgrade for DOS 3.3 EnterSoft Owners = \$20.00
Mark Manning Simulacron I/Baggy Game P.O. Box 58598 Webster, TX 77598	Thanks for the letters - Keep Writing!

The following program will print the ratio M/N as a percentage, assuming M and N are 24-bit integers. The percentage will be rounded to the nearest integer value, and print as one, two, or three digits followed by the "%" symbol. If M is more than 999% of N, the value printed will be garbled.

The straightforward way to compute this percentage would be to compute P = 100M/N, convert the quotient to decimal, and print it. To accomplish rounding, you could check the remainder after the division: if twice the remainder is greater than or equal to N, increment the quotient.

After a little head-scratching, I discovered another method. I accomplish the division and the conversion to decimal at the same time, and never multiply by 100. For rounding I cheated a little, and added N/256 to M before dividing.

N/256 is a breeze to compute, because it merely means offsetting the addition loop by one byte. However, the CORRECT rounding term would be N/200. The difference between N/256 and N/200 is 56N/51200, or about .0011N. This is "negligible", at least to me. It in effect means that I am rounding up percentages with fractional parts above .4989, instead of just those with fractions .5 or higher. Who will ever notice? Since my main use for this routine is to print what fraction of my hard disk is in use, it will be plenty close enough. (In fact, maybe I don't even need to round at all!)

The division/conversion loop works by using partial division. Each time I call the division loop, I divide M by N; however, I only loop enough times to get the next decimal digit of the quotient. Then I multiply the remainder by ten, so that the next division will generate the next digit. Trust me, it really works!

I added the digit printout to the tail end of the same subroutine which generates the next digit, and put in some logic to suppress leading zeroes. It looks funny if 10% prints out as 010%, so I ignore that leading zero. On the other hand, the logic makes sure 0% does not print as just "%".

To use the subroutine, you first have to store the 24-bit value for N. I wrote a subroutine for this, but you don't necessarily have to use it. If you do use it, load the most significant byte in the A-register, the middle byte in X, and the low byte in Y. I call this loading a 24-bit value into AX Y. Then do a JSR STORE.N. In contrast to the usual way you see multi-byte values stored in most 6502 code, I store M and N in High-to-Low order. This made the various loops inside the GET.DIGIT subroutine shorter. Lines 1100-1160 are the STORE.N subroutine. Nothing fancy here!

After storing the N-value, load up the M-value in AX Y and do a JSR PERCENT.CALC. The percentage that M is of N will be printed, and the subroutine will return. I wrote a demonstration program, shown in lines 2000-2260. This program

sets N=255 and then prints out all percentages for M = 0 to 255. You can vary the value of N and see different effects.

When you call PERCENT.CALC, lines 1180-1200 store your M-value. Lines 1210-1230 initialize the leading zero flag so that those zeroes will be suppressed. Lines 1240-1350 accomplish the pseudo-rounding I described above. I append another byte to the M-value, which is in effect after the radix point. [Radix point? What's a radix point? In decimal numbers, we call it a decimal point. In binary numbers, we could call it a binary point. In general, it is the demarcation between the integral and fractional parts of a number.] Finally, lines 1370-1410 generate and print three digits of the answer followed by the %-sign.

GET.DIGIT, lines 1430-1800, does all the real work. Lines 1440-1530 subtract N from M until M goes negative. The Y-register counts how many times this takes, less one. Unless M was greater than or equal to 10N, the number in Y will be a value from 0-9, and will be the first or next digit of the percentage. But before printing that digit, I need to modify the remainder.

Lines 1540-1610 add N back one time, so that the remainder is positive. We subtracted once too often, so this fixes things. Then lines 1620-1710 multiply the remainder by 10. Next time GET.DIGIT is called, we will generate the next digit of the percentage.

Lines 1720-1790 print the digit, unless it is a leading zero. If the digit is not zero, it is obviously not a leading zero, so it is printed. Any time a digit gets printed, I store a negative value in my leading zero flag, indicating that any future zeroes cannot be called "leading" (see line 1780). The leading zero flag started out at 2, and each time I test it gets decremented in line 1750. If the first two digits are both zero, it will go negative forcing the third digit to print even it is also zero.

If you are interested, it is very simple to modify this program so that it prints out a rounded percentage in the format xxx.x%, to the nearest tenth of a percent. All we have to do is change the rounding term from N/256 to N/2048, which affects the code in lines 1240-1350, and then add the follow lines:

1392	LDA #". "	Print a decimal point
1394	JSR MON.COUT	
1396	JSR GET.DIGIT	Print the tenths digit

Other simple modifications could change the variable size from 24-bits to 16-bits, 32-bits, or whatever you need.

```

1000 *SAVE S.PERCENT.CALC
1010
1020 * Subroutine for printing M/N as a percentage
1030 * where M and N are 24-bit integers.
1040 * 1. With (AXY)=N, do JSR STORE.N
1050 * 2. With (AXY)=M, do JSR PERCENT.CALC
1060
FD8E- 1070 MON.CROUT .EQ $FD8E
FD8A- 1080 MON.PRBYTE .EQ $FD8A
FD8D- 1090 MON.COUT .EQ $FD8D
1100
1110 STORE.N
1120 STA N MOST SIGNIFICANT
1130 STX N+1
1140 STY N+2 LEAST "
1150 RTS
1160
1170 PERCENT.CALC
1180 STA M MOST SIGNIFICANT
1190 STX M+1
1200 STY M+2 LEAST "
1210 *---Init Leading Zero Flag---
1220 LDA #2 Start with LZ-flag = 2
1230 STA Z
1240 *---Add N/256 to Round Result---
1250 LDA N+2 Accuracy would demand adding
1260 STA M+3 N/200, but N/256 is close enough.
1270 CLC N N 56N
1280 LDX #1 --- = --- + ---
1290 LDA M+1,X .1 200 256 200*256
1300 ADC N,X
1310 STA M+1,X And 56/51200 = .0010937 (very small)
1320 DEX
1330 BPL .1
1340 BCC .2
1350 INC M
1360 *---Compute & Print Digits---
1370 .2 JSR GET.DIGIT Hundreds digit
1380 JSR GET.DIGIT Tens digit
1390 JSR GET.DIGIT Units digit
1400 LDA #""
1410 JMP MON.COUT
1420
1430 GET.DIGIT
1440 LDY #-1 Y will be the quotient
1450 SEC
1460 .1 INY Increment Quotient
1470 LDX #2
1480 .2 LDA M,X Subtract Denominator
1490 SBC N,X
1500 STA M,X
1510 DEX
1520 BPL .2
1530 BCS .1 This goes around once to often...
1540 *---Add N back in once---
1550 LDX #2
1560 .3 LDA M,X So we need to add it back once.
1570 ADC N,X
1580 STA M,X
1590 STA T,X Save copy of M in T, to make it
1600 DEX easier to multiply by 10.
1610 BPL .3
1620 *---Multiply M by 10---
1630 JSR M.TIMES.2 M = 2 (4M + T)
1640 JSR M.TIMES.2 ...now we have 4M
1650 LDX #3 ...add T (copy of original M)
1660 .4 LDA M,X
1670 ADC T,X
1680 STA M,X
1690 DEX
1700 BPL .4
1710 JSR M.TIMES.2 5M times 2 is 10M
1720 *---Print digit if not leading zero---
1730 TYA digit is quotient from above
1740 BNE .5 ...digit not zero, so print it
1750 DEC Z Is it a leading zero?
1760 BPL .6 ...yes, don't print it.
1770 .5 ORA #0" Make it ASCII
1780 STA Z Kill LZ-flag by setting bit 7 = 1
1790 JSR MON.COUT Print the digit
1800 .6 RTS

```

```

1810 *---
1820 M.TIMES.2
1830 LDX #3 Double 4-byte value in M
1840 CLC
1850 .1 ROL M,X
1860 DEX
1870 BPL .1
1880 RTS
1890 *---
1900 M .BS 4
1910 N .BS 3
1920 T .BS 4
1930 Z .BS 1 LEADING ZERO FLAG
1940 *---
1950 * Test Routine for PERCENT.CALC
1960 * For M = 0 to 255
1970 * Print M,PERCENT(M/255)
1980 * Next M
1990 *---
2000 TT
2010 LDA #0
2020 STA TM Start TM=0
2030 TAX
2040 LDY #255 Set N = 255
2050 JSR STORE.N
2060 .1 LDY TM Set M to current TM
2070 TYA
2080 AND #7 If TM Mod 8 = 0, start new line
2090 BNE .2 ...same line
2100 JSR MON.CROUT
2110 .2 TYA Get M again
2120 JSR MON.PRBYTE Print M-value too
2130 LDA #"" followed by one blank
2140 JSR MON.COUT
2150 LDA #0
2160 TAX Leading bytes of M = 00 00
2170 JSR PERCENT.CALC print M/N percent
2180 LDA #""
2190 JSR MON.COUT followed by two blanks
2200 JSR MON.COUT
2210 INC TM Next TM
2220 BNE .1 ...until wraps around to 00
2230 RTS Finished
2240 *---
2250 TM .BS 1
2260 *---
088D- A2 03
088F- 18
0890- 3E 97 08
0893- CA
0894- 10 FA
0896- 60
08A3- A9 00
08A5- 8D D6 08
08A8- AA
08A9- A0 FF
08AB- 20 00 08
08AE- AC D6 08
08B1- 98
08B2- 29 07
08B4- D0 03
08B6- 20 8E FD
08B9- 98
08BA- 20 DA FD
08BD- A9 A0
08BF- 20 ED FD
08C2- A9 00
08C4- AA
08C5- 20 0A 08
08C8- A9 A0
08CA- 20 ED FD
08CD- 20 ED FD
08D0- EE D6 08
08D3- D0 D9
08D5- 60
08D6-

```

DON LANCASTER STUFF

<p>INTRODUCTION TO POSTSCRIPT</p> <p>A 65 min user group VHS video with Don Lancaster sharing many of his laser publishing and Postscript programming secrets.</p> <p>Includes curve tracing, \$5 toner refilling, the full Kroy Kolor details, page layouts, plus bunches more.</p> <p style="text-align: center;">\$39.50</p> <p style="text-align: center;">FREE VOICE HELPLINE</p>	<p>ASK THE GURU</p> <p>An entire set of reprints to Don Lancaster's ASK THE GURU columns, all the way back to column one. Edited and updated.</p> <p>Both Apple and desktop publishing resources are included that are not to be found elsewhere.</p> <p style="text-align: center;">\$24.50</p>	<p>APPLE IIc/IIe ABSOLUTE RESET</p> <p>Now gain absolute control over your Apple! You stop any program at any time.</p> <p>Eliminates all dropouts on your HIRES screen dumps. Gets rid of all hole blasting. For any IIc or IIe.</p> <p style="text-align: center;">\$19.50</p>	<p>POSTSCRIPT SHOW & TELL</p> <p>Unique graphics and text routines the others don't even dream of. For most any Postscript printer.</p> <p>Fully open, unlocked, and easily adaptable to your own needs. Available for Apple, PC, Mac, ST, many others.</p> <p style="text-align: center;">\$39.50</p> <p style="text-align: right;">VISA/MC</p>
--	--	--	--

SYNERGETICS

Box 809-SC Thatcher, AZ 85552 (602) 428-4073

I have written a number of articles in the past about converting values from binary to decimal and printing or displaying them. Usually such routines need to handle large numbers, but sometimes they are limited to single-byte values. And once in a while, you know in advance the value will be between 0 and 99 decimal.

For example, when you are printing the date you know in advance that the day, month, and year numbers are only two digits each. The following program is actually used in one such date printer, to print the day number and year number. For simplicity, it always prints two digits, even if the first digit is a zero. This is the way I want it to be when printing the year, but the day would probably look better without a leading zero.

Lines 1000-1140 in the listing which follows are a test routine which call on my PD subroutine to print every possible value from 00 to 99. Lines 1150 to the end are the PD subroutine. Lines 1240-1290 are not actually assembled, because the variable "blank.fill" is zero. If you change line 1010 to make "blank.fill" equal to 1, lines 1240-1290 will be assembled. Then values less than 10 will print with a leading blank rather than a leading zero.

```

1000 *SAVE Q2D.DECIMAL
00-      1010 blank.fill .eq 0
FDDED-  1020 *-----
1030 COUT .EQ $FDED
1040 *-----
0800- A0 00 1050 T      LDY #0      For Y = 0 to 99
0802- 98 1060 .1     TYA        A = Y
0803- 20 14 08 1070 JSR PD      Print two digits decimal
0806- A9 A0 1080 LDA # " "    Print two spaces
0808- 20 ED FD 1090 JSR COUT
080B- 20 ED FD 1100 JSR COUT
080E- C8 1110 INY        Next Y
080F- C0 64 1120 CPY #100
0811- 90 EF 1130 BCC .1
0813- 60 1140 RTS        Finished!
1150 *-----
0814- A2 AF 1160 PD      LDX # "0"-1  Start with ASCII zero-1
0816- 38 1170 SEC        Set up subtraction
0817- E8 1180 .1     INX        Increment ten's digit
0818- E9 0A 1190 SEC #10    Take out ten
081A- B0 FB 1200 BCS .1     Still more tens
081C- 69 BA 1210 ADC # "0"+10  Add back one ten, and make ASCII
081E- 48 1220 PHA        Save unit's digit
081F- 8A 1230 TXA        Get ten's digit
1240 .do blank.fill
1250 cmp # "0"
1260 bne .2     If these are assembled, print
1270 lda # " "   00-09 as " 0" through " 9"
1280 .2
1290 .fin
0820- 20 ED FD 1300 JSR COUT    Print ten's digit
0823- 68 1310 PLA        Get unit's digit
0824- 4C ED FD 1320 JMP COUT    and print it
1330 *-----
    
```

A long time ago I wished there were some sort of word processor for the Apple II. Paul Lutus wrote AppleWriter, and I bought a copy. It was limited to a 40-column display, and only showed upper-case on the screen, due to limitations in the old Apple II and II Plus machines. It cost \$50, and that seemed like a pretty good price.

After a while I found out about the Paymar Lower-Case Adapter, and added lower-case display to the screen. Some patches inside AppleWriter made it work with the adapter. Then the shift-key mod was invented, and I installed that also. I started hoping for even more....

So, I wrote my own word processor, based on AppleWriter's features, and using some of the internal techniques Paul Lutus developed which made AppleWriter faster than any other word processors available. I simplified the editing commands, speeded up and enhanced a lot of the features, and significantly shortened the code. I gave it the ability to use standard text files, with blazingly fast disk load and save. It gradually grew into a product, which we sold with all the source code for \$50. More than a pretty good price.

However, the S-C Word Processor was still limited to a 40-column display. Bob Deen, a high school student at the time, was doing some programming work for us. He did a lot of work on our Cross Assemblers, for example. He was also using the S-C Word Processor a lot, so I asked him to make an 80-column version for the Apple //e. He succeeded, and also added "widow" and "orphan" protection. (See the article in AAL, July, 1984.) By the way, Bob Deen is now a computer scientist at Jet Propulsion Laboratories in Pasadena, California, doing image enhancement software.

Meanwhile, Apple made a widow out of DOS 3.3 by bringing out the ProDOS system. The S-C Word Processor was still tied to DOS, partly because of my fancy disk I/O and the catalog-menu system. Then last December Bob Gardner sent us a ProDOS version! (Bob, who lives in Washington state, has been a loyal customer and friend since at least 1983.) He did a terrific job, making the ProDOS version even better than the DOS one. The only drawback was that it only worked in 40-columns. Well, in March he sent an 80-column version.

Now, still for only \$50, you get both DOS and ProDOS versions which work in both 40- and 80-columns. To use the 80-column versions you have to have an Apple //e, //c, or IIgs. The 40-column versions will work on those or older Apples, but the older Apples do need lower-case display and shift-key mods.

I wouldn't want you to think the S-C Word Processor has all the features of the Word Perfect, or other such major products. No, it is not that sophisticated. But it does have all the basic features we need for everyday work, it is very fast, and you get all the source code so you can personalize it.

Some of you have done some extensive personalizing already. Horst Schneider, a retired businessman in Denver, modified it to become a part of his business management package, adding mail merge and other features along the way. Larry Skutchan, who works at American Printing House for the Blind, made a talking version which works with Street Electronics' "Echo" speech synthesizers. If you are interested in either of these, I could put you in touch with Horst or Larry.

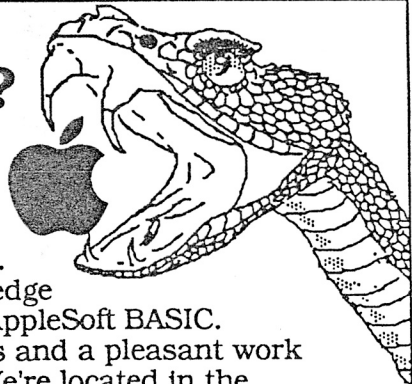
A curious bit of history: all three of the programmers who have made the major contributions to SCWP are named Bob! I guess we could give it the nickname of the Three-Bob Word Processor! No, it is inexpensive, but we do charge more than three shillings.

If you already have an earlier version of the S-C Word Processor and would like to get an update to the latest, send \$7.50 (or only \$5 if you only want the ProDOS disk).

Do You Have Apple Knowledge?

If you do, Applied Engineering would like to put your knowledge to work. We're looking for someone to fill a position in our Technical Support group. You must have a strong working knowledge of AppleWorks, ProDOS, DOS 3.3, and AppleSoft BASIC. Applied Engineering offers good benefits and a pleasant work environment. Office is non-smoking. We're located in the Carrollton area.

So if you've got the knowledge and want to sink your teeth into a position with an ever-expanding company, give us a call at (214) 241-8060.



ProDOS-8 stores the date and time information from in four bytes in the Global Page starting at \$BF90:

```

$BF90: MMMDDDDD  Low-order bits of Month, Day (1-31)
$BF91: YYYYYYMM  Year (0-99), high bit of Month
$BF92: 00mmmmmm  Minute (0-59)
$BF93: 00hhhhhh  Hour (0-23)

```

The following subroutine, lines 1000-1590, will print out the date in the form DD-MMM-YY. Lines 1600-1800 are an alternative method for printing out the 3-letter month name abbreviation. Lines 1810-1910 print the time in the form hh:mm.

```

1020 *-----
1030 * Subroutine to print date from ProDOS Global Page
1040 * in form DD-MMM-YY.
1050 * Two different methods for printing the 3-letter month
1060 * name are shown, with month-name table in normal and
1070 * transposed order.
1080 *-----
BF90- 1090 DATE .EQ $BF90,BF91  Date in form: MMMDDDDD, YYYYYYMM
1100 * Time in form: 00mmmmmm, 000hhhhh
FDED- 1110 COUT .EQ $FDED
1120 *-----
1130 PRINT.DATE
0800- AD 90 BF 1140 LDA DATE Get MMMDDDDD
0803- 29 1F 1150 AND #$1F Isolate Day of Month
0805- 20 33 08 1160 JSR PD Print the day number
0808- A9 AD 1170 LDA #- Print a dash
080A- 20 ED FD 1180 JSR COUT
1190 *-----PRINT MONTH FROM TABLE-----
080D- AD 91 BF 1200 LDA DATE+1 Get YYYYYYMM
0810- 4A 1210 LSR High bit of Month-number into Carry
0811- 48 1220 PHA Save OYYYYYYY on stack
0812- AD 90 BF 1230 LDA DATE Get MMMDDDDD
0815- 6A 1240 ROR MMMDDDDD
0816- 4A 1250 LSR OMMMMDDD
0817- 4A 1260 LSR 00MMMMDD
0818- 4A 1270 LSR 000MMMMD
0819- 4A 1280 LSR 0000MMMM Month number (1-12)
081A- AA 1290 TAX
081B- BD 45 08 1300 LDA MONTH.TBL.1-1,X 1st letter
081E- 20 ED FD 1310 JSR COUT
0821- BD 51 08 1320 LDA MONTH.TBL.2-1,X 2nd letter
0824- 20 ED FD 1330 JSR COUT
0827- BD 5D 08 1340 LDA MONTH.TBL.3-1,X 3rd letter
082A- 20 ED FD 1350 JSR COUT
082D- A9 AD 1360 LDA #- Print dash
082F- 20 ED FD 1370 JSR COUT
1380 *-----PRINT YEAR-----
0832- 68 1390 PLA GET OYYYYYYY FROM STACK
1400 *---Fall into PD subroutine---
0833- A2 AF 1410 PD LDX #-0*-1 Start with ASCII zero-1
0835- 38 1420 SEC Set up subtraction
0836- E8 1430 .1 INX Increment ten's digit
0837- E9 0A 1440 SBC #10 Take out ten
0839- B0 FB 1450 BCS .1 Still more tens
083B- 69 BA 1460 ADC #-0*+10 Add back one ten, and make ASCII
083D- 48 1470 PHA Save unit's digit
083E- 8A 1480 TXA Get ten's digit
083F- 20 ED FD 1490 JSR COUT Print ten's digit
0842- 68 1500 PLA Get unit's digit
0843- 4C ED FD 1510 JMP COUT and print it
1520 *-----
1530 .MA AS
1540 .AS -/]1/
1550 .EM
1560 *-----
0846- 1570 MONTH.TBL.1 >AS "JFMAMJJASOND"
0852- 1580 MONTH.TBL.2 >AS "AEAPAUUECOE"
085E- 1590 MONTH.TBL.3 >AS "NBRRYNLGPTVC"

```

```

1600 *-----
1610 ALTERNATIVE.MONTH.PRINTER
086A- AD 91 BF 1620 LDA DATE+1 GET YYYYYYM
086D- 4A 1630 LSR M INTO CARRY
086E- AD 90 BF 1640 LDA DATE GET MMMDDDDD
0871- 6A 1650 ROR MMMDDDDD
0872- 4A 1660 LSR OMMMMDD
0873- 4A 1670 LSR OMMMMDD
0874- 4A 1680 LSR OQMMMMD
0875- 4A 1690 LSR OQOQMMM
0876- 8D 8B 08 1700 STA TEMP Multiply month number by 3
0879- 0A 1710 ASL
087A- 6D 8B 08 1720 ADC TEMP
087D- AA 1730 TAX Index is 3,6,9,...
087E- A0 03 1740 LDY #3 Print 3 consecutive letters
0880- BD 89 08 1750 .1 LDA MONTH.TABLE-3,X
0883- 20 ED FD 1760 JSR COUT
0886- E8 1770 INX Next letter
0887- 88 1780 DEY
0888- D0 F6 1790 BNE .1
088A- 60 1800 RTS Finished
1810 *-----
088B- 1820 TEMP .BS 1
088C- 1830 MONTH.TABLE >AS "JANFEBMARAPR MAYJUNJUL AUGSEP OCTNOVDEC"
1840 *-----
1850 PRINT.TIME
08B0- AD 93 BF 1860 LDA DATE+3 Get 00hhhhh
08B3- 20 33 08 1870 JSR PD
08B6- A9 BA 1880 LDA #": "
08B8- 20 ED FD 1890 JSR COUT
08BB- AD 92 BF 1900 LDA DATE+2 Get 00mmmmm
08BE- 4C 33 08 1910 JMP PD

```

The value stored in the Global Page may not be current. It is automatically updated every time you close or flush a file, or you can force it to be updated by using the MLI call shown in lines 1920-end. If you have looked into the Global Page description in the manuals, you may have noticed that \$BF06 is a vector to the date/time update code. Don't try to use it directly unless you are sure the Language Card is properly switched before and after the call. The best way is to use the MLI call, as I did.

```

1920 *-----
1930 UPDATE.DATE.AND.TIME
08C1- 20 00 BF 1940 JSR $BF00 MLI ENTRY POINT
08C4- 82 00 00 1950 .DA #82,0000 GET DATE/TIME, NO PARMS
08C7- 60 1960 RTS
1970 *-----

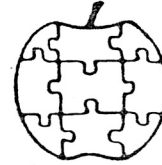
```

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$24 per year in the USA, Canada, and Mexico, or \$36 in other countries. Back issues are \$1.80 each for Volumes 1-7, \$2.40 each from later Volumes (plus postage). A subscription to the newsletter with a Monthly Disk containing all program source code and article text is \$64 per year in the USA, Canada and Mexico, and \$90 to other countries.

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)

Apple

\$2.40



Assembly

Line

Volume 8 -- Issue 6

March, 1988

Peeking Inside AppleWorks 1.3, Part 4:

Applications Overlay Manager	2
Dissecting AppleWorks SEG.M0 and SEG.M1 Files	19
Backup/Restore for RamFactor DOS Partition	24

Something Personal

I had another birthday last weekend. Now my age is a perfect square again, for the seventh time. Another 15 years and it will be square again. At most I can expect two or three more square ages. Numbers like these are interesting to me. Moses said, after recalling the brevity and frailty of human life, "So teach us to number our days, that we may apply our hearts unto wisdom."

Certainly we all should echo that prayer. We want our years to count for something. Solomon said, "The fear of the LORD is the beginning of knowledge: but fools despise wisdom and instruction." David said, "The fool has said in his heart, 'There is no God'." James said, "If any of you lack wisdom, let him ask of God, who gives to all men liberally, and upbraids not; and it shall be given him." Paul said, "In Jesus Christ are hidden all the treasures of wisdom and knowledge."

And I say, the longer I live, the more I find that God is there, and faithful to his Word. The better part of wisdom is trusting the truly trustworthy. Solomon says it best: "Trust in the LORD with all your heart, and lean not unto your own understanding. In all your ways acknowledge him, and he shall direct your paths."

As I enter my jubilee year, I re-commit myself to doing just that.

AppleWorks is a lot bigger than a normal Apple. The 6502 microprocessor is restricted to 64K of memory address space, and the fraction of this left to a ProDOS-based program is normally less than 48K. The AppleWorks applications use large areas of memory for the active data, so the amount left for programs is very small. So how does it work? By using overlays.

The file on an AppleWorks Program disk named APLWORKS.SYSTEM (which we have been analyzing over the last four months) contains subroutines which stay in RAM regardless of what activity you are performing. After starting up AppleWorks, this code resides in RAM from \$1000 up to \$2Exx. RAM from \$800 to \$FFF is used for various variables and buffers. RAM from \$3000 up to \$BEFF is used for application programs and data.

All of the rest of the code for the three applications is on the two files named SEG.M0 and SEG.M1. These two files together contain 43 small segments of code, designed to be loaded into RAM only when needed. The first 15 segments appear to me to be the data base application; the next 8, the word processor; and the rest, the spreadsheet. At the beginning of each file is an "index" which tells where in the file each program segment begins. The article "Dissecting AppleWords SEG.M0 and SEG.M1 Files" later in this same issue goes into detail about the structure of the index.

This month we will look in detail at the code inside AppleWorks which loads in the various overlays. Since it happens to be very near the beginning of APLWORKS.SYSTEM, I am also listing the JMP vector that starts at \$1000.

Since each of the 43 overlays will need to use various subroutines from APLWORKS.SYSTEM, the author of AppleWorks needed a straightforward way for them to know their addresses. Woz's monitor is a good example of what happens when you do not provide such a method.

Back in 1977 we found hundreds of neat entry points in that tiny little ROM, all at very specific addresses. We used them flagrantly, to save RAM for better uses. Then Apple started taking routines out, moving others, until they finally printed a list of the 110 or so they will continue to support. There is no easy system to using these entry points, because Woz originally coded them with the idea of squeezing the most function into the smallest amount of memory.

The Apple IIgs monitor, on the other hand, is a good example of what happens when you go overboard in trying to provide a calling system. After acquiring over ten pounds of documentation, I still am only dimly understanding all the ins and outs of the toolboxes. I know it all starts by loading a 16-bit coded number into the X-register, and doing a JSL \$E10000 command. Parameters are passed on the stack, and an error code is returned in the A-register. All is done very systematically, very cleanly, very macintoshly, but not very efficiently. The toolbox calls must be done in full 16-bit mode, cannot use the registers to pass data, eat up many machine cycles getting to the actual tool and back again, and do require the use of the A- and X-registers. Still, it may be the best way to create, organize, and control an open-ended set of tools in a machine like the IIgs.

ProDOS-8 MLI gives an example of another method, in which a single entry point processes all calls. The byte following the JSR to that entry point

S-C Macro Assembler Version 2.0	DOS \$100, ProDOS \$100, both for \$120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners	\$20
ProDOS Upgrade Kit for Version 2.0 DOS owners	\$30
Cross Assemblers for owners of S-C Macro Assembler	\$32.50 to \$50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051, 8085, 1802/4/5, PDP-11, GI1650/70, Mitsubishi 50740, others)	
Source Code of any S-C Macro Assembler or Cross Assembler	each, additional \$100
S-C DisAssembler (ProDOS only)	without source code \$30, with source \$50
RAK-Ware DISASM (DOS only)	without source code \$30, with source \$50
ProVIEW (ProDOS-based disk utility program)	\$20
Full Screen Editor for S-C Macro (with complete source code)	\$49
S-C Cross Reference Utility	without source code \$20, with source \$50
S-C Word Processor, both DOS & ProDOS, both 40- & 80-columns, with complete source code	\$50
DP18 and DPPP, double precision math for Applesoft, including complete source code	\$50
ES-CAPE (Extended S-C Applesoft Program Editor), including Version 2.0 With Source Code	\$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)	\$30
Bag of Tricks 2 (Quality Software)	(\$49.95) \$45
S-C Documentor (complete commented source code of Applesoft ROMs)	\$50
Copy II Plus (Central Point Software)	(\$39.95) \$30
AAL Quarterly Disks	each \$15, or any four for \$45

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Vinyl disk pages, 6"x8.5", hold two disks each	10 for \$6
Diskette Mailing Protectors (hold 1 or 2 disks)	40 cents each
(Cardboard folders designed to fit 6"x9" Envelopes.)	or \$25 per 100
Envelopes for Diskette Mailers	6 cents each

Sider 20 Meg Hard Disk, includes controller & software	(\$695) \$550 +
Sider 40 Meg Hard Disk, includes controller & software	(\$995) \$860 +

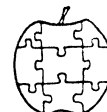
Minuteman 250 Uninterruptible Power Supply	(\$359) \$320 +
Minuteman 300 Uninterruptible Power Supply	(\$549) \$490 +

65802 Microprocessor, 4 MHz (Western Design Center)	\$25
quikLoader EPROM System (SCRG)	(\$179) \$170
PROMGRAMMER (SCRG)	(\$149.50) \$140

"Exploring the Apple IIgs"	Gary B. Little	(\$22.95)	\$21
"Apple IIgs Technical Reference"	Michael Fischer	(\$19.95)	\$19
"65816/65802 Assembly Language Programming"	Michael Fischer	(\$21.95)	\$20
"Programming the 65816"	David Eyes & Ron Lichty	(\$22.95)	\$21
"Programming the Apple IIgs in C and Assembly Language"	Mark Andrews	(\$18.95)	\$18
"Apple //e Reference Manual"	Apple Computer	(\$24.95)	\$23
"Apple //c Reference Manual"	Apple Computer	(\$24.95)	\$23
"ProDOS-8 Technical Reference Manual"	Apple Computer	(\$29.95)	\$27
"ProDOS-16 Technical Reference Manual"	Apple Computer	(\$29.95)	\$27
"Apple IIgs Firmware Reference"	Apple Computer	(\$24.95)	\$23
"Apple IIgs Hardware Reference"	Apple Computer	(\$24.95)	\$23
"Apple IIgs Toolbox Reference, Volume 1"	Apple Computer	(\$26.95)	\$24
"Apple IIgs Toolbox Reference, Volume 2"	Apple Computer	(\$26.95)	\$24
"Apple IIgs Programmer's Introduction"	Apple Computer	(\$32.95)	\$30
"ProDOS Inside and Out"	Dennis Doms & Tom Weishaar	(\$16.95)	\$16
"Beneath AppleProDOS"	Don Worth & Pieter Lechner	(\$19.95)	\$18
"Beneath Apple DOS"	Don Worth & Pieter Lechner	(\$19.95)	\$18
"Inside the Apple //c"	Gary B. Little	(\$19.95)	\$18
"Inside the Apple //e"	Gary B. Little	(\$19.95)	\$18
"Understanding the Apple //e"	Jim Sather	(\$24.95)	\$23
"Understanding the Apple II"	Jim Sather	(\$22.95)	\$21
"Apple II+/IIe Troubleshooting & Repair Guide"	Brenner	(\$19.95)	\$18
"Assembly Language for Applesoft Programmers"	Finley & Myers	(\$18.95)	\$18
"Now That You Know Apple Assembly Language"	Jules Gilder	(\$19.95)	\$18
"Enhancing Your Apple II, vol. 1"	Don Lancaster	(\$15.95)	\$15
"Enhancing Your Apple II, vol. 2"	Don Lancaster	(\$17.95)	\$17
"Assembly Cookbook for the Apple II/IIe"	Don Lancaster	(\$21.95)	\$20
"Microcomputer Graphics"	Roy E. Myers	(\$14.95)	\$14
"Assembly Lines -- the Book"	Roger Wagner	(\$19.95)	\$12

* These items add \$2 for first item, \$.75 for each additional item for US shipping.
 + Inquire for shipping cost.
 Customers outside USA inquire for postage needed.
 Texas residents please add 8% sales tax to all orders.
 << Master Card, VISA, Discover and American Express >>

S-C Software Corporation
 2331 Gus Thomasson #125
 DALLAS, TX 75228
 Phone 214-324-2050



contains a command code, and the next two bytes point to a parameter block. ProDOS-16 uses two bytes for the command code and four bytes for the parameter block address.

Robert Lissner uses a simple system of vectoring all subroutine calls through several JMP vectors throughout AppleWorks. Some of his subroutines pass all their data in the three 6502 registers, some use fixed locations in page zero or in the \$800-\$FFF area, and some use a standard calling sequence with parameters after the JSR. One set of JMP vectors starts at \$1000, and is used for calling all of the APLWORKS.SYSTEM subroutines. Another set begins at \$D002 in the alternate \$D000 bank of RAM, where either SEG.00 or SEG.XM has been loaded. Each overlay segment also begins with a JMP vector.

I have shown the JMP vector for APLWORKS.SYSTEM in lines 1520-2030 of the listing. To save space in the newsletter, I plugged in actual hexadecimal addresses for all those subroutines which are not listed in this issue. Where I have given them names, I included them as comments. The rest of them I will name later, when I get to them and figure out what they do and think of a unique meaningful moniker.

When AppleWorks is first fired up by executing APLWORKS.SYSTEM, one of the tasks is to look for either a 64K (or larger) memory card in the auxiliary slot or a card like RamFactor in one of the other slots. If it finds a RamFactor-like card with enough free memory, it loads SEG.XM into the 4K area at \$D000. (RamFactor is Applied Engineering's version of the Apple Memory Card, and of course there are other companies also making these kinds of cards. In the rest of this article I will call this kind of memory SlotRAM memory.) If there is not enough SlotRAM memory available but there is 64K in the AuxRAM, AppleWorks loads SEG.00 instead.

How can it do that? ProDOS supposedly has that \$D000 space all tied up!

Well, ProDOS claims it all, but only really USES from \$D100 through \$D3FF. This is where the standard QUIT code is kept. During initialization AppleWorks copies that \$300-byte area to the SEG.PR file, and then loads the appropriate SEG.00 or SEG.XM file. When you QUIT out of AppleWorks, it copies those \$300 bytes from SEG.PR back into \$D100 before doing the ProDOS Quit call. Note: AppleWorks only saves and restores \$300 bytes! If you are using a non-standard Quit processor which takes over \$300 bytes, running AppleWorks will clobber it. You will then have to reboot after quitting AppleWorks.

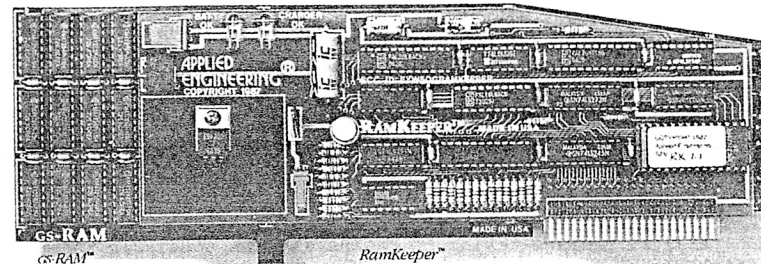
There are 24 subroutines inside the \$D000 area which are accessed through a JMP vector starting at \$D002. Depending on which SEG.xx is loaded there, they either talk to AuxRAM or SlotRAM. The routines that are of interest this month are equated in lines 1200-1220; listing them will have to wait for a future issue.

I defined some useful macros in lines 1320-1510. Macros are gradually growing on me. When I first added them to the S-C Assembler II, creating the S-C Macro Assembler version 1.0, I really couldn't think of many uses for them beyond sales appeal. Then I started using them for generating various types of data lists, and often-used code sequences like MLI calls. Now I am finding more and more uses.

The MLI.SL macro is a slight modification of my standard MLI-call macro. I added lines 1360-1400 to generate the error-tracking code which Lissner

RamKeeper™

For the "Instant On" Apple IIGs.



Permanent Storage with an "Electronic Hard Disk"

Now when you turn on your IIGs your favorite program can appear on screen in just a few seconds! With RamKeeper, your IIGs memory card will retain stored programs and stored data while your IIGs is turned off. RamKeeper allows you to divide your IIGs memory into part "electronic hard disk" and part RAM for your programs workspace—in almost any way you want and at anytime you want. GS-RAM, GS-RAM Plus, Apple IIGs memory card and most other IIGs memory cards are compatible with RamKeeper.

Supports Up to Two IIGs Memory Cards at the Same Time

If you bought your IIGs with Apple's memory card and later wished you had the GS-RAM, no problem. RamKeeper will support both cards plugged into RamKeeper simultaneously!

How it Works

Just unplug your IIGs memory card



Steve Wozniak, the creator of Apple Computer

"I've purchased several Applied Engineering products over the years. They're always well made and performed as advertised. I recommend them wholeheartedly."

from your computer, plug your IIGs memory card into RamKeeper, plug RamKeeper into the IIGs memory slot. If you have another IIGs memory card, an additional card socket on RamKeeper will accommodate your second card. That's all there is to it!

Reliability from the Most Experienced

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple so you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, reliability is everything! That's why Applied Engineering uses the more dependable Gel-Cells instead of Ni-Cad batteries (if Ni-Cad's aren't discharged periodically, they lose much of their capacity). RamKeeper has close to 6 times (about 6 hours) the "total power failure" back-up time of other systems. When power returns, RamKeeper automatically recharges the battery to a full charge. With power from your wall outlet, RamKeeper will back-up your IIGs memory cards RAM indefinitely.

RamKeeper Has and Does It All!

- Allows instant access to your programs without slow disk delays
- Configure Kilobytes or Megabytes of instant ROM storage for your favorite programs

- Reduces power strain to your internal IIGs' power supply
- Contains back-up status LED's
- Can support up to two IIGs memory cards simultaneously
- Supports both 256K installed memory chip boards like GS-RAM and the Apple IIGs Memory Expansion Card as well as 1 MEG installed memory chip boards like GS-RAM Plus
- 5-year hassle-free warranty
- 15 day money back guarantee
- Proudly made in the USA
- RamKeeper comes complete with battery, software and documentation

Only \$179.00!

(GS-RAM card shown in photo not included)

Order Your RamKeeper Today!

See your dealer or call (214) 241-6060, 9:00-11:00 CST, 7 days a week or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 if outside USA.

AE APPLIED ENGINEERING™

The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

Prices subject to change without notice.

uses. After nearly every call to MLI he uses a BEQ to branch around an INC of the error flag. ProDOS returns with status EQ and carry clear if there was no error, or NE and carry set if there was an error. The various ProDOS manuals make it clear that the carry status is supposed to be the preferred error flag, and I always got the impression that future versions might not support the EQ/NE method. Well, now they will HAVE to continue that support, because the world's most popular Apple program says so. Most other software I have looked at or written uses the CC/CS method, including such basic software as BASIC.SYSTEM.

If I use the SLI.ML macro with only two parameters, it generates only the two-line MLI call. If I add a third parameter, it generates the two extra lines. The third parameter in this macro is never actually used, just counted. The .DO]#>2 line says "only assemble the following lines (up to the .FIN line) if the number of parameters is greater than two."

By using the private label :1 in the macro definition I avoid having to clutter up the code with lots of extra local or normal labels. This makes the listing easier to read with understanding. For example, look at lines 4340-4360. Those three lines actually generate 12 lines of code with three labels. You do have to remember when you are reading the code what these macros generate, if you are trying to understand the code. In these three lines, remember that any errors cause SEGLOAD.ERROR.FLAG to be incremented. If you want to, you can leave out the .LIST MOFF line that I used at the beginning (the unlisted line 1010). Then all of the code generated by each macro will be listed during assembly. I wanted to suppress the macro expansion listing to make the code easier to understand and to make it fit in the newsletter.

The HS macro is entirely for the purpose of shortening the listing. I use it in the SEGMENT.TABLE definition in lines 2330-2590. This table would take up two or three times as much paper if I did not use the macro.

The MSG macro is useful for defining strings that begin with a character count and include only printing ASCII characters. Since almost all of the text messages included in AppleWorks are like that, MSG is quite useful. I used it here in line 5230.

The SEGMENT.TABLE in lines 2230-2600 keeps track of vital information about the segments. There are four bytes for each segment. The first byte is the page number the segment should be loaded at. For example, Segment 01 has 35 in this byte, so it should be loaded at \$3500.

The second byte of each group of four is used to keep track of how long it has been since the segment was loaded. Each time a segment is loaded the second byte for its entry in the SEGMENT.TABLE is zeroed while the second byte for all other entries is incremented, by the code in lines 3220-3360. I call this byte the "age" of a segment. I have not yet found any code which takes advantage of the information in the age-byte, but at least it is there. It may be, or could be, used to determine which segments to keep in AuxRAM or SlotRAM if there is not room for all of them.

The third and fourth bytes of each four-byte entry are 0000 if the segment must be loaded from disk. Naturally, this is what they are initially. After a segment is loaded from disk, if there is room in AuxRAM or SlotRAM it is also copied there. Then these two bytes in the segment table are set to a coded address so that the segment can be downloaded from RAM the next time it is needed.

When you select one of the three applications from the main menu in AppleWorks, LOAD.PROGRAM.SEGMENT.A will be called. I put a lot of information about the calling sequence of this program in the comments in lines 3000-3140. Basically, the segment number will be in the A-register. Lines 3170-3210 save this number and multiply it by four to make an index into the segment table. As already mentioned, lines 3220-3360 then increment the second byte of all entries and zero the second byte for the entry for the segment we want to load.

Lines 3370-3430 check to see if this segment is the same as the last one loaded. If it is, there is nothing left to do so it exits. I say "exits" rather than "returns" because it may or may not return. Bit 7 in the A-register at the time of call controls whether it returns after loading a segment or jumps into the loaded segment. If it does the latter, it jumps to \$xx02, where xx is the page number the segment was loaded into. In this case, the X-register is a function number to be interpreted by the segment. When the segment is finished, it may return with an RTS to the program which called LOAD.PROGRAM.SEGMENT.A. The exit options are processed in lines 4810-4880.

If the segment is not the same as the most recently loaded one, line 3430 stores the new segment number in the CURRENTLY.LOADED.SEGMENT variable.

Lines 3440-3520 pick up the load address byte out of the segment table and install it in the various places it will be needed later. If that byte is 00, then the segment does not exist and the program returns with an RTS. This should never happen, of course, and I presume if it did it would be a bug. There are three null segments (0F, 14, and 15), but I would be surprised if any useful purpose is served by trying to load them. On the other hand, if there is some code somewhere which attempts to load all the segments in a range so that they will be copied in to AuxRAM or SlotRAM memory, the null segments might be included in the range without any disastrous effects.

Lines 3530-3570 treat segment \$20 in a special way. I am not sure what that segment is, or why it is special. If you are trying to load segment \$20 and the flag at \$EA7 is non-zero it will not be loaded. Instead the loader will exit, either with an RTS or by using the function call (JMP to \$4502 with a code in the X-register). I presume that \$EA7 is set non-zero when function \$20 is loaded, and stays that way until it is covered up by something else. This would let functions within segment \$20 be called without reloading it unnecessarily even when other segments have been loaded after it was. I don't know, it sounds sort of kludgy. I'll just have to wait until I have looked into and disassembled a lot more of the AppleWorks code.

Lines 3580-3690 make sure that the variable CURRENT.APPLICATION is changed whenever you load segments \$01, \$10, or \$18. I haven't found any use for this yet, and I am just hoping I have correctly guessed its significance. I have assumed that those segment numbers are the initially segments for each of the three applications, and have so indicated in the comments in lines 2130-2210.

Lines 3700-3750 look at the third and fourth bytes in the segment table entry to see if they are 0000. If so, the segment must be loaded from the AppleWorks Program disk. If not, lines 3760-3810 download the segment from AuxRAM or SlotRAM memory. The downloading is handled by a subroutine from

either SEG.00 or SEG.XM, which I will probably describe in detail in a future issue of AAL. They handle AuxRAM or SlotRAM in interesting ways.

If a segment must be loaded from disk, the subroutine beginning at line 3830 takes over. Lines 3840-3920 modify the general SEG.xx pathname buffer to point to either SEG.M0 or SEG.M1. For some reason the first nine segments are stored on SEG.M0 and the rest on SEG.M1. Once upon a time it probably made sense....

Lines 3930-3970 will then try to open the selected SEG.Mx file. If the file will not open, AppleWorks assumes the only possible reason could be that the disk is not mounted and calls CALL.FOR.AWPROGRAM.DISK (lines 2780-2950) to tell you to do so. The message says "Place the AppleWorks PROGRAM disk in Drive 1 and press Return." Actually you SHOULD be able to place the disk in ANY drive, if you have specified a complete pathname for the program disk. And, actually, you can press any key, not just RETURN.

Once the file has been opened successfully, lines 3990-4050 store the reference number ProDOS assigns the file in all the other IOBs. Line 4060 calls CLR.PRODOS.BITMAP to make sure ProDOS will allow reading into the range \$800-9FFF. Lines 4070-4080 clear a byte used as an error flag for all the subsequent MLI calls. After going through all the motions of loading the segment this flag will still contain a zero if no errors were reported by ProDOS.

Lines 4090-4120 read in the first 140 bytes of the SEG.Mx file. The front of the file contains a segment offset table with 3-byte values for each segment. These three bytes tell what offset (or MARK) value to send via the MLI "Set Mark" call before reading in the segment. Subtracting a segment's offset from that of the next segment gives the length of the segment we want to load. Since there are 43 segments, will need 43 3-byte values for starting addresses, plus one more for the zero-th entry, plus still one more for computing the length of the 43rd segment. That is a total of 45*3 bytes, or 135. Appleworks reads 140, which allows for one more segment to be added without changing this constant.

Note that the program goes right on reading the SEG.Mx file whether there is a reading error or not. SEGLOAD.ERROR.FLAG gets incremented if there is an error, but nothing else is done about it until we have gone through all the motions of loading the segment and closing the file. If there was any error at all, lines 4380-4390 find it out and return with an RTS to whoever called the segment loader. This seems a little dangerous, because the user will never know what happened. A glitched AppleWorks Program disk could cause very erratic behavior without any explanation, yet AppleWorks COULD have reported what was wrong and exactly which segment could not be loaded.

Lines 4130-4180 multiply the segment number by 3 to get a pointer into the SEG.Mx segment offset table. Lines 4190-4320 pick up the offset and save it for the Set Mark MLI call, and also compute the segment length for the Read MLI call. Lines 4340-4360 set the mark, read the segment, and close the file. After the segment has been read, line 4370 calls SET.PRODOS.BITMAP to re-protect all RAM from \$800 through \$9FFF.

The first two bytes of every segment on both SEG.Mx files contains the end address plus 1 of that segment. This value may be different from the number of bytes loaded plus the starting address, if the segment needs some private RAM at the end of itself. However, I haven't found any cases where this is so. We already knew the length in order to read the segment from



SPECIAL !!! EXPANDED RAM/ROM BOARD: \$39.00

Similar to our \$30 RAM/ROM dev board described below. Except this board has two sockets to hold your choice of 2-2K RAM, 2-2K ROM or even 2-4K ROM for a total of 8K. Mix RAM and ROM too. Although Apple limits access to only 2K at a time, soft switches provide convenient socket selection. Hard switches control defaults.

IMPROVED III II IN A MAC (ver 2.0): \$75.00

Now includes faster graphics, UniDisk support and more! Bi-directional data transfers are a snap! This Apple II emulator runs DOS 3.3/PRODOS (including 6502 machine language routines) on a 512K MAC or MACPLUS. All Apple II features are supported such as HI/LO-RES graphics, 40/80 column text, language card and joystick. Also included: clock, RAM disk, keyboard buffer, on-screen HELP, access to the desk accessories and support for 4 logical disk drives. Includes 2 MAC diskettes (with emulation, communications and utility software, plus DOS 3.3 and PRODOS system masters, including Applesoft and Integer BASIC) and 1 Apple II diskette.

SCREEN.GEN: \$35.00

Develop HI-RES screens for the Apple II on a Macintosh. Use MACPAINT (or any other application) on the MAC to create your Apple II screen. Then use SCREEN.GEN to transfer directly from the MAC to an Apple II (with SuperSerial card) or IIC. Includes Apple II diskette with transfer software plus fully commented SOURCE code.

MIDI-MAGIC for Apple //c: \$49.00

Compatible with any MIDI equipped music keyboard, synthesizer, organ or piano. Package includes a MIDI-out cable (plugs directly into modem port - no modifications required!) and 6-song demo diskette. Large selection of digitized QRS player-piano music available for 19.00 per diskette (write for catalog). MIDI-MAGIC compatible with Apple II family using Passport MIDI card (or our own input/output card w/drum sync for only \$99.00).

FONT DOWNLOADER & EDITOR: \$39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. Special functions (like expanded, compressed etc.) supported. Includes HIRES screen editor to create custom fonts and special graphics symbols. For Apple II, II+, //e. Specify printer: Apple Imagewriter, Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/85, or Okidata 92/192. * FONT LIBRARY DISKETTE *1: \$19.00 contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

DISASM 2.2e : \$30.00 (\$50.00 with SOURCE Code)

Use this intelligent disassembler to investigate the inner workings of Apple II machine language programs. DISASM converts machine code into meaningful, symbolic source compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even provides label substitution. Address-based triple cross reference generator included. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his ASSEMBLY COOKBOOK.

The 'PERFORMER' CARD: \$39.00 (\$59.00 with SOURCE Code)

Converts a 'dumb' parallel printer I/F card into a 'smart' one. Simple command menu. Features include perforation skip, auto page numbering with date & title, large HIRES graphics & text screen dumps. Specify printer: MX-80 with Grafrax-80, MX-100, MX-80/100 with Grafraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93.

'MIRROR' ROM: \$25.00 (\$45.00 with SOURCE Code)

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support.

RAM/ROM DEVELOPMENT BOARD: \$30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

C-PRINT For The APPLE //c: \$69.00

Connect standard parallel printers to an Apple //c serial port. Separate P/S included. Just plug in and print!

Unless otherwise specified, all Apple II diskettes are standard (not copy protected!) 3.3 DOS.

Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders OK.

RAK-WARE 41 Ralph Road W. Orange NJ 07052 (201) 325-1885



disk, so recomputing the same value seems like make-work. But who knows? Lines 4400-4520 pick up the address from the beginning of the segment, subtract the loading address, and store the result in SL.LEN. That gives the "uploader" the number of bytes to copy into AuxRAM or SlotRAM.

Lines 4530-4610 compute the actual address of the current segment's entry in the segment table, and save this address at SL.SEG. We are gradually building up the calling sequence for the "uploader". Lines 4620-4710 determine whether there is enough RAM left in either AuxRAM or SlotRAM, whichever is being used, for uploading the segment. If so, lines 4720-4770 call on the uploader to do so. The uploader will set the 3rd and 4th bytes for the segment in the segment table so that future calls to load this segment can download them from RAM instead of reading the AppleWorks Program disk.

Lines 4810-4880 have already been discussed above. They decide whether to return to the caller with an RTS, or to JMP into the segment just loaded with a function code in the X-register. Lines 4930-5200 are the parameter blocks, or IOBs, used by the various MLI calls in the segment loader.

Lines 5220-5240 define the message used by the program which prompts you to mount the AppleWorks program disk. I used the .PH and .EP because this message does not immediately follow the IOBs in the real code. The .PH address shows where it really is assembled in version 1.3. Lines 5250-5400 are the two subroutines for de-protecting and protecting RAM from \$800 through \$9FFF.

Assembly Language Programmers Wanted

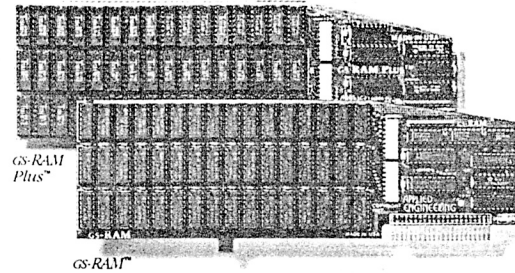
The American Printing House for the Blind (APH) is looking for assembly language programmers to write and modify educational and applications software for the Apple II series. People interested should be familiar with 6502, 65C02, and 65C816 processors. They should also know both DOS 3.3 and ProDOS. Knowledge of the Z-80 is also highly desirable. The position requires some travel, speaking, and writing. Interested candidates should send a resume and some example code to:

American Printing House for the Blind
Larry Skutchan
P. O. Box 6085
Louisville, KY 40206

An Equal Opportunity Employer

For Those Who Want the Most. From Those Who Make the Best. GS-RAM™

Now expand the IIcs™ RAM and ROM with up to 8 MEG of "Instant On" memory with the all new GS-RAM!



GS-RAM has an all new design. A design that delivers higher performance including increased speed, greater expandability, and improved software.

More Sophisticated, Yet Easier to Use

GS-RAM works with all IIcs software. In fact any program that runs on Apple's smaller memory card runs on the GS-RAM. But with GS-RAM you can have more memory, improved performance, and almost unlimited expansion capabilities. We've designed the new GS-RAM to be easier to use too—you don't have to adjust the size of your RAM disk every time you use a DMA device. No other RAM card with more than 4 banks of memory installed can make the same claim.

More than Just Hardware

Each GS-RAM and GS-RAM Plus includes the most powerful set of IIcs software enhancements available anywhere. In fact, our nearest competitor offers only a fraction of the invaluable programs that we include with each GS-RAM card. This software includes the most powerful disk-caching program available, the GS-RAM Cache. The Cache will make most of your applications run up to 7 times faster. Also included is a diagnostic utility that lets you test your GS-RAM by graphically showing the location of any bad or improperly installed RAM chips. And for AppleWorks users, we give you our exclusive Expander program that dramatically enhances both the capabilities and speed of AppleWorks.

Making AppleWorks Even Better

Applied Engineering's Expander program eliminates AppleWorks internal memory limits allowing it to recognize up to 8 megabytes of desktop workspace. You can increase the limits from only 7,250 lines to 22,600 lines in the word processor and from 6,350 records to 22,600 records in the database. The Expander allows all of AppleWorks, including print functions, to automatically load into RAM. The clipboard size will increase from 255 to 2,042 lines maximum. GS-RAM will automatically segment larger files so you can save them onto multiple floppies. And

GS-RAM provides a built-in print buffer that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

Grow by Kilobytes or Megabytes

We offer GS-RAM in two configurations so you can increase your memory 256K at a time (GS-RAM) or a megabyte at a time (GS-RAM Plus). Both are IIcs compatible and both come with our powerful enhancement software. GS-RAM can hold up to 1.5 MEG of 256K chips and GS-RAM Plus can hold up to 6 MEG using 1 MEG chips. And since both use standard RAM chips (not high-priced SIMMs), you'll find expanding your GS-RAM or GS-RAM Plus easy, convenient, and very economical. For further expansion, you can plug a 2 MEG "piggyback" card into the GS-RAM's expansion port for up to 3.5 MEG of total capacity. Or up to a whopping 8 MEG on GS-RAM Plus. If a GS-RAM owner outgrows 3.5 MEG, he can easily upgrade to GS-RAM Plus for a nominal charge.

Permanent Storage for an "Instant On" Apple

With our RamKeeper™ back-up option, your GS-RAM or GS-RAM Plus will retain both programs and data while your IIcs is turned off! Now when you turn your IIcs back on, your favorite software is on your screen in under 4 seconds! With RamKeeper you can divide your IIcs memory into part "electronic hard disk" and part extended RAM. Even change the memory boundaries at any time—and in any way—you want. Because



Steve Wozniak, the creator of Apple Computer

"In quality, performance, compatibility, expandability and support, Applied Engineering's GS-RAM and GS-RAM Plus are number one."

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple, you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, Reliability is everything. That's why Applied Engineering uses state-of-the-art "GEL-CELLS" instead of Ni-Cad batteries (if Ni-Cads aren't discharged periodically, they lose much of their capacity). RamKeeper has about 6 hours of "total power failure" back-up time. That's 6 times the amount of other systems. But with power from your wall outlet, RamKeeper will back-up GS-RAM, GS-RAM Plus, or most other IIcs memory cards indefinitely. Should you ever have a "total power failure," RamKeeper switches to its 6-hour battery. When power returns, RamKeeper will automatically recharge the battery to full power. RamKeeper incorporates a dual-rate charger, status L.E.D.'s, and advanced power reducing circuitry. RamKeeper comes complete with battery, software, and documentation.

GS-RAM's Got it ALL!

- 5-year warranty — parts & labor
- 6 RAM banks (most cards have 4)
- Memory expansion port
- ROM expansion port
- Ultra-fast disk caching on ProDOS 8 AND ProDOS 16.
- Expands AppleWorks internal limits
- Includes hi-res self test
- No soldered-in RAM chips
- Expandable to 8 MEG
- No configuration blocks to set
- RamKeeper back-up option allows permanent storage of programs & data
- 15-day money-back guarantee
- Proudly made in the USA

GS-RAM with 256K	\$189
GS-RAM with 512K	\$259
GS-RAM with 1 MEG	\$399
GS-RAM with 1.5 MEG	\$539
GS-RAM with 2.5 to 3.5 MEG	CALL
GS-RAM Plus with 1-8 MEG	CALL
RamKeeper Option	\$179

Order today!

See your dealer or call Applied Engineering today, 9 am to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside USA.

AE APPLIED ENGINEERING™
The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006
Prices subject to change without notice

GS-RAM, GS-RAM Plus and RamKeeper are trademarks of Applied Engineering. Other brands and product names are registered trademarks of their respective holders.

```

1000 *SAVE AW.SRC.V8N6

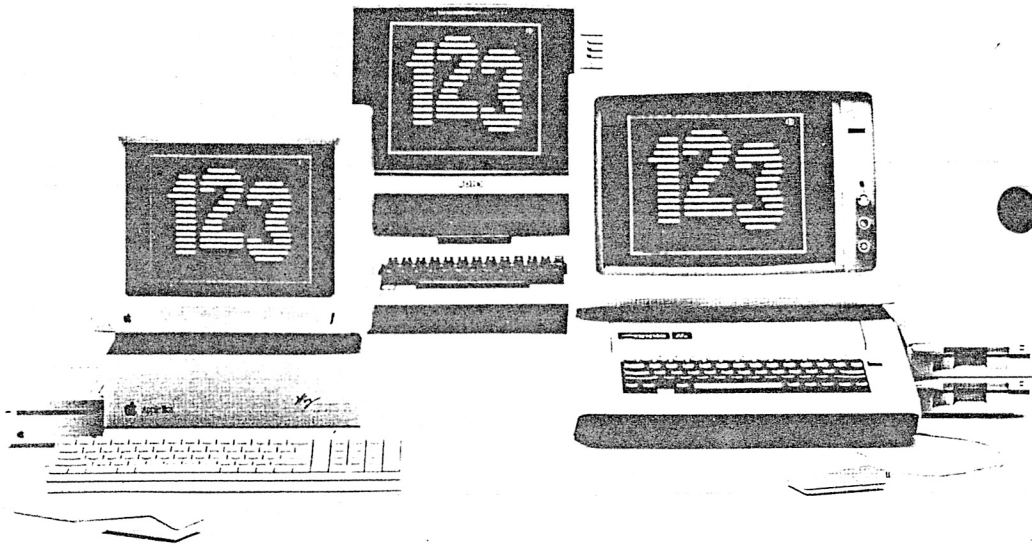
1020 ***missing line above was ".LIST MOFF", which
1030 ***shrinks the listing by not listing macro expansions
1040 *---AppleWorks Variables-----
0900- 1050 BUF.900 .EQ $0900
0A00- 1060 STR.A00 .EQ $0A00
0EA7- 1070 X.OEA7 .EQ $0EA7
0F14- 1080 SEGMENT.ADDRESS .EQ $0F14
0FCE- 1090 X.OFCE .EQ $0FCE
0FF2- 1100 X.OFF2 .EQ $0FF2
0FF5- 1110 FRER.MEMORY.xxxxK .EQ $0FF5
1120 *-----
1130 DISPLAY.ON.LINE.23 .EQ $1FF5
1D35- 1140 AW.KEYIN .EQ $1D35
B700- 1150 X.B700 .EQ $B700
1160 *---ProDOS Global Page-----
BF00- 1170 MLI .EQ $BF00
BF58- 1180 PRODOS.BITMAP .EQ $BF58
1190 *---Subroutines from SEG.00 or SEG.XM---
D002- 1200 Measure.free.Memory .EQ $D002
D005- 1210 Download.from.AuxRAM.or.Memory.Card .EQ $D005
D011- 1220 Upload.to.AuxRam.or.Memory.Card .EQ $D011
1230 *---Page Zero Variables-----
85- 1240 Z.85 .EQ $85
8D- 1250 SEGLOAD.ERROR.FLAG .EQ $8D
9A- 1260 P0 .EQ $9A
9E- 1270 P4 .EQ $9E
9F- 1280 P5 .EQ $9F
A4- 1290 DISPLAY.ACTIVE.FLAG .EQ $A4
1300 *-----
1310 .PH $1000
1320 *-----
1330 .MA MLI.SL Macro for MLI calls
1340 JSR MLI
1350 .DA #1,SLIOB.12
1360 .DO |#>2 If 3rd parameter, include these lines
1370 BEQ :1 ...no error reported by MLI
1380 INC SEGLOAD.ERROR.FLAG error, so set flag
1390 :1
1400 .FIN
1410 .EM
1420 *-----
1430 .MA HS Macro to shrink listing only
1440 .HS 11
1450 .EM
1460 *-----
1470 .MA MSG Macro to make a string with first byte
1480 .DA #1;-1 giving the length, rest is ASCII.
1490 .AS "1"
1500 :1
1510 .EM
1520 *-----
1000- 4C 25 3E 1530 JMP $3E25 RELOCATE.AND.START+$1000
1003- 4C 86 11 1540 JMP CALL.FOR.AWPROGRAM.DISK
1006- 4C A1 11 1550 JMP $11A1 LOAD.PROGRAM.SEGMENT.A
1009- 4C 41 13 1560 JMP $1341 APPEND.STRINGS
100C- 4C 66 13 1570 JMP $1366 CLEAR.MAIN.WINDOW
100F- 4C 6E 13 1580 JMP $136E
1012- 4C 9D 13 1590 JMP $139D CLR.LINE.X.TO.LINE.Y
1015- 4C D1 14 1600 JMP $14D1 DISPLAY.STRING (A) bytes, starting at $80,81
1018- 4C 9D 17 1610 JMP $179D CONVERT.A.TO.RJBF.STRING
101B- 4C 15 18 1620 JMP $1815 HANG
101E- 4C D1 17 1630 JMP $17D1 DIVIDE.PO.BY.P2
1021- 4C 18 18 1640 JMP $1818 BEEP.AND.CLEAR.KEYBUF
1024- 4C 23 18 1650 JMP $1823 MOVE.CURSOR.TO.XY
1027- 4C 37 18 1660 JMP $1837
102A- 4C 42 18 1670 JMP $1842
102D- 4C 50 18 1680 JMP $1850
1030- 4C 6C 18 1690 JMP $186C
1033- 4C 72 18 1700 JMP $1872
1036- 4C 7A 18 1710 JMP $187A COPY.SCRN.LINE.TO.0900
1039- 4C B4 18 1720 JMP $18B4 GET.A.PARMS
103C- 4C 1D 19 1730 JMP $191D
103F- 4C 77 1A 1740 JMP $1A77
1042- 4C FC 1A 1750 JMP SET.PRODOS.BITMAP
1045- 4C 00 1B 1760 JMP CLR.PRODOS.BITMAP
1048- 4C 0B 1B 1770 JMP $1B0B DRAW.TOP.AND.BOTTOM.LINES
104B- 4C 2B 1B 1780 JMP $1B2B
104E- 4C 3A 1B 1790 JMP $1B3A MULTIPLY.X.BY.Y
1051- 4C 4E 1B 1800 JMP $1B4E MULTIPLY.PO.BY.P2
1054- 4C 84 1B 1810 JMP $1B84 MOVE.BLOCK.DOWN
1057- 4C AC 1B 1820 JMP $1BAC MOVE.BLOCK.UP
105A- 4C DF 1B 1830 JMP $1BDF
105D- 4C F1 1B 1840 JMP $1BF1 WAIT.FOR.SPACEReturn.OR.ESCAPE
1060- 4C 21 1C 1850 JMP $1C21 PRINTER.DRIVER
1063- 4C 0F 1D 1860 JMP $1C0F
1066- 4C 35 1D 1870 JMP $1D35 AW.KEYIN
1069- 4C 80 1E 1880 JMP $1E80 MOVE.CURSOR.TO.TCGL.THROW

```

```

106C- 4C 8A 1E 1890 JMP $1E8A
106F- 4C B4 1E 1900 JMP $1EB4 MAP.LOWER.TO.UPPER char in A-reg
1072- 4C BF 1E 1910 JMP $1EBF FILTER.LC.TO.UC string
1075- 4C D9 1E 1920 JMP $1ED9 COMPARE.STRINGS
1078- 4C F8 1E 1930 JMP $1EF8 MOVE.STRING
107B- 4C 3E 1F 1940 JMP $1F3E DISPLAY.AT
107E- 4C 29 20 1950 JMP $2029
1081- 4C D1 1F 1960 JMP $1FD1 DELAY.A.TENTHS
1084- 4C E0 1F 1970 JMP $1FE0 CLEAR.KEYBUF
1087- 4C 93 20 1980 JMP $2093 DISPLAY.STRING.PO
108A- 4C E9 1F 1990 JMP $1FE9 DISPLAY.TOKEN.X
108D- 4C AE 20 2000 JMP $20AE
1090- 4C BE 20 2010 JMP $20BE
1093- 4C F5 1F 2020 JMP $1FF5 DISPLAY.ON.LINE.23
1096- 4C D6 20 2030 JMP $20D6
2040 *-----
2050 *---CONSTANTS & VARIABLES-----
2060 *-----
1099- 00 0A 2070 HANDLE.STR.A00 .DA STR.A00
109B- A7 10 2080 HANDLE.SEGMENT.TABLE .DA SEGMENT.TABLE
2090 *-----
2100 * Holds result after calling CONVERT.A.TO.RJBF.STRING
2110 * but this result is apparently never referenced.
109D- 2120 RJBF.STRING >HS 03.20.20.20
2130 *-----
2140 * 10A1 holds 00, 01, 02, or 03, indicating which of segments
2150 * 1, 16, or 24 was the most recently loaded.
2160 * 00 means none of these have yet been loaded.
2170 * 01 means segment 1 (Data Base)
2180 * 02 means segment 16 (Word Processor)
2190 * 03 means segment 24 (Spread Sheet)
10A1- 00 2200 CURRENT.APPLICATION .HS 00
10A2- 2210 APPLICATION.SEGMENT.LIST >HS FF.01.10.18
2220 *-----
2230 * Segment Table
2240 * There are 43 program segments in files SEG.M0 and SEG.M1
2250 * Four bytes in this table for each segment.
2260 * Byte 1: Starting page to load segment into.
2270 * Byte 2: Age of segment (00=just loaded, FF=oldest)
2280 * Bytes 3,4: 0000 if must be loaded from disk
2290 * xxxx if in RAM or Aux RAM
2300 *-----
10A6- FF 2310 CURRENTLY.LOADED.SEGMENT .HS FF
2320 SEGMENT.TABLE
2330 >HS 30.000000 Dummy Entry, seg 00
10A7- 2340 >HS 35.000000 Seg 01 (Data Base)
10AB- 2350 >HS 45.000000.45.000000 Segs 02,03
10AF- 2360 >HS 45.000000.45.000000 Segs 04,05
10BF- 2370 >HS 45.000000.4E.000000 Segs 06,07
10C7- 2380 >HS 4E.000000.4A.000000 Segs 08,09
10CF- 2390 >HS 4E.000000.4E.000000 Segs 0A,0B
10D7- 2400 >HS 4E.000000.53.000000 Segs 0C,0D
10DF- 2410 >HS 4E.000000.00.000000 Segs 0E,0F
2420 *-----
10E7- 2430 >HS 35.000000 Seg 10 (Word Processor)
10EB- 2440 >HS 3D.000000.3D.000000 Segs 11,12
10F3- 2450 >HS 40.000000.00.000000 Segs 13,14
10FB- 2460 >HS 00.000000.67.000000 Segs 15,16
1103- 2470 >HS 77.000000 Seg 17
2480 *-----
1107- 2490 >HS 33.000000 Seg 18 (Spread Sheet)
110B- 2500 >HS 3B.000000.53.000000 Segs 19,1A
1113- 2510 >HS 53.000000.53.000000 Segs 1B,1C
111B- 2520 >HS 53.000000.53.000000 Segs 1D,1E
1123- 2530 >HS 53.000000.63.000000 Segs 1F,20
112B- 2540 >HS 64.000000.64.000000 Segs 21,22
1133- 2550 >HS 64.000000.64.000000 Segs 23,24
113B- 2560 >HS 64.000000.64.000000 Segs 25,26
1143- 2570 >HS 64.000000.64.000000 Segs 27,28
114B- 2580 >HS 64.000000.64.000000 Segs 29,2A
1153- 2590 >HS 64.000000 Seg 2B
2600 *-----
BO- 2610 SEGMENT.TABLE.SIZE .EQ *SEGMENT.TABLE
2620 *-----
1157- 00 2620 POSITION.IN.STRING .HS 00 Used by DISPLAY.STRING
2630 *-----
2640 * Note **SECRET** limitation: the pathname
2650 * to the SEG.M0 and SEG.M1 files is limited
2660 * to a total of 29 bytes, including the "/".
1158- 01 2F 2670 SEGMENT.PATHNAME .HS 01.2F
115A- 2680 .BS 28
2690 *-----
1176- 01 2700 KEYIN.CURSOR.TYPE .HS 01 00=underline, 01=flashing
1177- 01 2710 KEYIN.CURSOR.FLAG .HS 01 00=no cursor, 01=cursor
1178- 00 2720 SCROLL.DIRECTION .HS 00 Used by DISPLAY.STRING
1179- 00 2730 BYTES.IN.STRING .HS 00 Used by DISPLAY.STRING
2740 *-----
117A- 2750 KEYBUF .BS 10
1184- 00 2760 KEYBUF.IN .HS 00
1185- 00 2770 KEYBUF.OUT .HS 00

```



Now Apple speaks IBM. Three times faster than IBM.

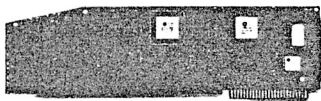
**Introducing
PC Transporter™
The Apple® II expansion
board that lets you
run MS®-DOS programs.**

Now your Apple II can run over 10,000 programs you could never use before. Like Lotus® 1-2-3®, MultiMate®, dBASE III PLUS®, Even Flight Simulator®.

With PC Transporter, MS-DOS programs run on your Apple II like they do on IBM® PCs or compatibles. With one important difference. PC Transporter runs most of those programs *three times faster* than an IBM PC/XT®.

Plus, to speed through number-crunching tasks, you can use our optional 8087-2 math co-processor chip. It plugs into a socket on the PC Transporter.

Less expensive than an IBM clone.
Sure, a stripped-down IBM



clone costs about the same as the PC Transporter. But the peripherals it takes to get the clone up and running make the clone cost about three times what our American-made card costs.

You don't have to buy new hardware to use PC Transporter. **Works with the hardware you already own.**

With PC Transporter, MS-DOS programs see your Apple hardware as IBM hardware. You can use the same hardware you have now.

With IBM software, your Apple hardware works just like IBM hardware. Including your drives, monitors, printers, printer cards, clock cards and serial clocks.

You can use your Iie® or IIGs™ keyboard with IBM software. Or use our optional IBM-style keyboard (required for the II Plus).

You can use your Apple mouse. Or an IBM compatible serial mouse.

Plenty of power.

PC Transporter gives you as much as 640K of user RAM and 128K of system RAM in the IBM mode.

PC Transporter also is an Apple expansion card, adding up to 768K of extra RAM in the Apple mode. The Apple expansion alone is a \$300 value.

Easy to install.

You can install PC Transporter in about 15 minutes, even if you've never added an expansion board. You don't need special tools. Simply plug it into an Apple expansion slot (1 through 7 except 3), connect a few cables and a disk drive, and go!



PC Transporter taps into the world's largest software library. Now your Apple can run most of the IBM software you use at work. And it opens a new world of communications programs, games and bulletin boards.

A universal disk drive controller.

PC Transporter supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes. You'll shift instantly between Apple ProDOS and IBM MS-DOS.

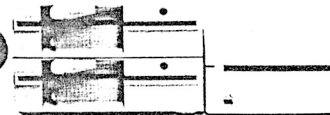
You'll need our versatile 5.25" 360K drive system to run IBM applications from 5.25" floppy disks. Use your Apple 5.25" drive for Apple 5.25" disks.

An Apple Disk 3.5 Drive will support the new 3.5" disks whether they're IBM MS-DOS formatted or Apple ProDOS formatted. The PC Transporter acts like an Apple Disk 3.5 Drive disk controller for IIGs, Iie, and II Plus users.

PC Transporter supports up to 5 drives in a number of combinations.

For example, you can connect a 5.25 Applied Engineering 360K dual-drive system directly to the card. Then plug two daisy-chained Apple 3.5 Drives (not the Apple UniDisk 3.5) to the dual-drive system. For a fifth drive, use a ProDOS file as an IBM hard disk.

PC Transporter controls Apple and IBM compatible disk drives. It supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes.



Versatile data storage.

PC Transporter reads MS-DOS and translates it into Apple native ProDOS. You can store IBM programs and data on any ProDOS storage device including the Apple 3.5 Drive, Apple UniDisk™ 3.5, Apple 5.25" drive, SCSI or ProDOS compatible hard drives. (You can use the Apple UniDisk 3.5 with its own controller card for storing programs and data, but not for directly booting an IBM formatted disk.)

You can even use our 360K PC compatible drive for ProDOS

Make your Apple speak IBM.

PC Transporter memory choices.

RAM in Apple mode:	RAM in IBM mode:	Price:
384K	256K	\$489.00
512K	384K	529.00
640K	512K	569.00
768K	640K	609.00

Note: The IBM mode is 128K less because the PC Transporter uses 128K for system memory.

IIGs Installation Kit 49.00

Iie/II Plus Installation Kit 39.00

PC Transporter Accessories

5.25" IBM Format 360K Drive Systems

Single-Drive System 269.00

Dual-Drive System 399.00

Half-Height Drive 135.00
(add to Single-Drive system to make Dual-Drive)

IBM-Style Keyboard 139.00
(required for Apple II Plus. Requires IBM Keyboard Cable.)

IBM Keyboard Cable 34.00

Sony RGB Monitor 499.00

Analog RGB Cable 39.00
(for use with Sony monitor)

Digital RGB Cable 39.00
(for use with Sony monitor)

Digital RGB Adapter 24.00

ColorSwitch 44.00
(included with IIGs Installation Kit)

128K ZIP 40.00

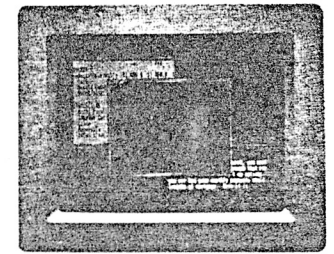
PC Transporter per set

Memory Expansion Chip Set

8087-2 Math Co-processor Chip 229.00

Heavy Duty Power Supply 69.00
(Iie and II Plus only)

See your dealer or call or send check or money order to Applied Engineering, MasterCard, VISA and COD welcome. Texas residents add 6 1/4% sales tax.



PC Transporter produces better IBM graphics than IBM Analog is sharper than digital. So with an analog RGB monitor PC Transporter's CGA graphics and text are superior to IBM's digital display — even while running IBM software!

And, you can also use an Apple composite monitor in IBM text or graphics mode.

storage and a 143K Apple 5.25" drive for MS-DOS storage.

Created by Apple's original designers.

The brains behind PC Transporter were also behind your Apple II.

The PC Transporter design team includes the former project managers for the creation of the Apple Iie and Iic. The co-designer of the Apple II disk controller. And the first full-time Apple programmer and author of the ProDOS operating system.

So you know the PC Transporter and your Apple were made for each other.

Support and service from the leader in Apple add-ons.

Applied Engineering sells more Apple peripheral boards than anyone else — including Apple Computer. So you know we'll be around after the sale.

PC Transporter comes with a 15-day money back guarantee. If you're not fully satisfied after using it, return it for a full refund. PC Transporter also comes with a 1-year warranty.

How to get your PC Transporter today.

See your dealer. Or call Applied Engineering any day between 9 a.m. and 11 p.m. CST at 214-241-6060.

AE Applied Engineering

The Apple enhancement experts.

P.O. Box 798, Carrollton, TX 75006
214-241-6060

A Division of AE Research Corporation

```

2780 *-----
2790 * Subroutine to request mounting of the AppleWorks
2800 * Program Disk, so a SEG.xx file can be opened.
2810 *-----
2820 * (1186) 1003 124B 275F
2830 CALL.FOR.AWPROGRAM.DISK
2840 LDA DISPLAY.ACTIVE.FLAG Save Display lockout flag
2850 PHA
2860 LDA #0 Allow display
2870 STA DISPLAY.ACTIVE.FLAG
2880 JSR DISPLAY.ON.LINE.23 *Place the AppleWorks
2890 .DA MSG..1 PROGRAM disk in Drive 1 and press Return.*
2900 JSR AW.KEYIN Wait for Any Key
2910 LDA #0 Indicate Program Disk mounted
2920 PLA X.OFCE
2930 STA X.OFCE Restore Display lockout flag
2940 PLA
2950 STA DISPLAY.ACTIVE.FLAG
2960 RTS
2970 *-----
2980 SEGMENT.INDEX .BS 1
2990 SEGMENT.SAVEX .BS 1
3000 SEGMENT.NUMBER .BS 1
3010 * (A)=Segment Number or Segment Number + $80.
3020 * There are 43 segments, numbered 1 to 43.
3030 * Segments 1-9 are in file SEG.M0, and
3040 * segments 10-43 are in file SEG.M1.
3050 *
3060 * The Segment is loaded at $xx00, where xx
3070 * is the first byte for the entry in the
3080 * Segment Table.
3090 *
3100 * Bit 7 of A controls the execution option.
3110 * If 0, then exit with JMP $xx02.
3120 * If 1, then exit with an RTS.
3130 * (X)=Function Number in segment
3140 *-----
3150 * (11A1) 1006 1877
3160 LOAD.PROGRAM.SEGMENT.A
3170 STA SEGMENT.NUMBER
3180 STX SEGMENT.SAVEX
3190 ASL #4 to get index into Segment Table
3200 ASL
3210 STA SEGMENT.INDEX
3220 *-----Increment all entries-----
3230 LDX #SEGMENT.TABLE.SIZE
3240 DEX For X = $AC to $04 step -4
3250 DEX
3260 DEX
3270 DEX
3280 INC SEGMENT.TABLE+1,X
3290 BNE .2
3300 DEC SEGMENT.TABLE+1,X max value is $FF
3310 CFX #0
3320 BNE .1
3330 *-----Clear indexed entry-----
3340 LDX SEGMENT.INDEX
3350 LDA #0
3360 STA SEGMENT.TABLE+1,X
3370 *-----Keep track of loaded segment-----
3380 LDA SEGMENT.NUMBER
3390 AND #$7F
3400 CMP CURRENTLY.LOADED.SEGMENT
3410 BNE .3
3420 JMP LOAD.EXIT.BY.OPTION Already loaded, so we're done!
3430 STA CURRENTLY.LOADED.SEGMENT
3440 *-----Get Load Address-----
3450 LDX SEGMENT.INDEX
3460 LDY SEGMENT.TABLE,X
3470 BNE .4
3480 JMP LOAD.EXIT.RTS ...no such segment, quit now
3490 STY SLOB.READ.SEGMENT+3
3500 STY 10+1 In call to Downloader Subroutine
3510 STY SL.ADR+1
3520 STY SEGMENT.EXECUTION.VECTOR+2
3530 *-----Is this segment $20?-----
3540 CMP #$20
3550 BNE .5
3560 LDX X.OEA7
3570 BNE .11 ...Exit
3580 *-----Keep track of application-----
3590 LDX CURRENT.APPLICATION
3600 BEQ .6
3610 CMP APPLICATION.SEGMENT.LIST,X
3620 BEQ .11 ...Exit
3630 LDX #3
3640 CMP APPLICATION.SEGMENT.LIST,X
3650 BEQ .8
3660 DEX
3670 BNE .7
3680 BEQ .9 ...not in the list
3690 STX CURRENT.APPLICATION

```

```

3700 *-----Decide how to load it-----
3710 .9 LDX SEGMENT.INDEX If address in SEGMENT.TABLE is 0000,
3720 LDA SEGMENT.TABLE+2,X then load from Program Disk;
3730 STA SEGMENT.ADDRESS otherwise, download from Ram
3740 ORA SEGMENT.TABLE+3,X
3750 BEQ LOAD.SEGMENT.FROM.DISK
3760 LDA SEGMENT.TABLE+3,X
3770 STA SEGMENT.ADDRESS+1
3780 JSR Download.from.AuxRAM.or.Memory.Card
3790 .DA SEGMENT.ADDRESS
3800 .10 .HS 00.00 H1-byte filled in by program
3810 .11 JMP LOAD.EXIT.BY.OPTION ...Exit
3820 *-----
3830 LOAD.SEGMENT.FROM.DISK
3840 LDY SEGMENT.PATHNAME Change name end to 'M0' or 'M1'
3850 LDA #'M'
3860 STA SEGMENT.PATHNAME-1,Y
3870 LDA #'0'
3880 LDX CURRENTLY.LOADED.SEGMENT
3890 CPX #10
3900 BCC .1 Segments 1-9 from SEG.M0
3910 LDA #'1' Segments 10-43 from SEG.M1
3920 .1 STA SEGMENT.PATHNAME,Y
3930 *-----Attempt to open SEG.Mx-----
3940 .2 >MLI.SL C8,OPEN Open the file
3950 BEQ .3 ...no problems
3960 JSR CALL.FOR.AWPROGRAM.DISK
3970 JMP .2
3980 *-----
3990 .3 LDA #0 Indicate Program Disk mounted
4000 STA X.OFCE
4010 LDA SLIOB.OPEN+5 Copy File RefNum to IOBs
4020 STA SLIOB.READ.SEGMENT+1
4030 STA SLIOB.CLOSE+1
4040 STA SLIOB.SETMARK+1
4050 STA SLIOB.READ.INDEX+1
4060 JSR CLR.PRODOS.BITMAP
4070 LDA #0 Clear Error Flag
4080 STA SEGLOAD.ERROR.FLAG
4090 *-----Read Segment Index-----
4100 * $8C bytes at beginning of SEG.Mx file
4110 * into buffer at $0900.
4120 >MLI.SL CA,READ.INDEX,S
4130 *-----Form segnum*3 for index-----
4140 LDA CURRENTLY.LOADED.SEGMENT
4150 ASL
4160 CLC
4170 ADC CURRENTLY.LOADED.SEGMENT
4180 TAX
4190 *-----Get Mark and Length-----
4200 LDA #3 Do for 3 loops
4210 STA P0
4220 LDY #0 Start with lsb
4230 SEC
4240 .4 LDA BUF.900,X Byte of Mark for this segment
4250 STA SLIOB.SETMARK+2,Y
4260 LDA BUF.900+3,X Byte of Mark for next segment
4270 SBC BUF.900,X Byte of Mark for this segment
4280 STA SLIOB.READ.SEGMENT+4,Y Byte of Length
4290 INX
4300 DEX
4310 DEC P0
4320 BNE .4 More bytes
4330 *-----Read the Segment-----
4340 >MLI.SL CE,SETMARK,S Set Mark
4350 >MLI.SL CA,READ.SEGMENT,S Read Segment
4360 >MLI.SL CC,CLOSE,S Close File
4370 JSR SET.PRODOS.BITMAP
4380 LDA SEGLOAD.ERROR.FLAG
4390 BNE LOAD.EXIT.RTS ...Error(s), give it up.
4400 *-----Save length of segment-----
4410 LDA #0 Build pointer to segment in P4,P5
4420 STA P4
4430 LDA SEGMENT.EXECUTION.VECTOR+2
4440 STA P5
4450 LDY #0 Get first two bytes
4460 LDA (P4),Y which are end address + 1
4470 STA SL.LEN and subtract load address
4480 INY to get length
4490 LDA (P4),Y (I thought we already knew
4500 SEC the length!)
4510 SBC P5
4520 STA SL.LEN+1

```



```

12D9- AD 9E 11 4530 *---See if room to save segment in RAM---
12DC- 18 4540 LDA SEGMENT.INDEX
12DD- 69 02 4550 CLC Build pointer to vector in SEGMENT.TABLE
12DF- 6D 9B 10 4560 ADC #2
12E2- 8D 0B 13 4570 ADC HANDLE.SEGMENT.TABLE
12E5- AD 9C 10 4580 STA SL_SEG
12E8- 69 00 4590 LDA HANDLE.SEGMENT.TABLE+1
12EA- 8D 0C 13 4600 ADC #0
12ED- 20 02 D0 4610 STA SL_SEG+1
12F0- AD F6 0F 4620 JSR Measure.free.Memory
12F3- D0 0E 4630 LDA FREE.MEMORY.xxxxK+1 If non-zero, at least 256K
12F5- AD F5 0F 4640 BNE .5 ...which is plenty!
12F8- C9 10 4650 LDA FREE.MEMORY.xxxxK
12FA- B0 07 4660 CMP #16 Is there at least 16K?
12FC- 0A 4670 BCS .5 ...Yes, plenty
12FD- 0A 4680 ASL Convert to # pages free
12FE- CD 10 13 4690 ASL
1301- 90 15 4700 CMP SL.LEN+1 Compare to what is needed
1303- A9 01 4710 BCC LOAD.EXIT.BY.OPTION ...not enough room
1305- 8D F3 0F 4720 .5 LDA #1
1308- 20 11 D0 4730 STA X.OFF3
130B- 00 00 4740 JSR Upload.to.AuxRam.or.Memory.Card
130D- 00 00 4750 SL_SEG .HS 00.00 Address in SEGMENT.TABLE of vector
130F- 00 00 4760 SL.ADR .HS 00.00 Load Address of Segment
1311- A9 00 4770 SL.LEN .HS 00.00 Length of Segment
1313- 8D F3 0F 4780 LDA #0
1316- 85 85 4790 STA X.OFF3
4800 STA Z.85 ...fall into LOAD.EXIT...
4810 *-----
4820 LOAD.EXIT.BY.OPTION
1318- AE 9F 11 4830 LDX SEGMENT.SAVEX
131B- AD A0 11 4840 LDA SEGMENT.NUMBER
131E- 30 03 4850 BMI LOAD.EXIT.RTS
4860 SEGMENT.EXECUTION.VECTOR
1320- 4C 02 FF 4870 JMP $FF02 Page value filled in by program
4880 so JMPs to $xx02 in segment.
4890 *-----
1323- 60 4900 LOAD.EXIT.RTS RTS
4910 *-----
4920 *---OPEN IOB---
4930 SLIOB.OPEN
1324- 03 58 11 4940 .DA #3,SEGMENT.PATHNAME,X.B700
1327- 00 B7 4950 .HS 00 Open RefNum
1329- 00 4960 *
4970 *---READ IOB---
4980 SLIOB.READ.SEGMENT
132A- 04 4990 .DA #4
132B- 00 5000 .HS 00 RefNum
132C- 00 00 5010 .DA $0000 Load Address
132E- 00 38 5020 .DA $3800 Load Length
1330- 00 00 5030 .DA $0000 Actual Length
5040 *
5050 *---CLOSE IOB---
5060 SLIOB.CLOSE
1332- 01 5070 .DA #1
1333- 00 5080 .HS 00 RefNum
5090 *
5100 *---SETMARK IOB---
5110 SLIOB.SETMARK
1334- 02 5120 .DA #2
1335- 00 5130 .HS 00 RefNum
1336- 00 00 00 5140 .HS 00.00.00
5150 *
5160 *---READ IOB---
5170 SLIOB.READ.INDEX
1339- 04 5180 .DA #4
133A- 00 5190 .HS 00 RefNum
133B- 00 09 8C 5200 .DA BUF.900,$008C,$0000
133E- 00 00 00 5210 *
5220 .PH $13C1
13C1- 5230 MSG..1 >MSG "Place the AppleWorks PROGRAM disk in Drive 1 and press Return. "
5240 .EP
5250 *-----
5260 .PH $1AFC
5270 * (1AFC) 1042 12B9 24C6 270D 272E 278F 2B2E 2CF3
5280 * (1B00) 1345 1265 23F1 249F 26E4 2771 2960 2CEA
5290 SET.PRODOS.BITMAP
1AFC- A9 FF 5300 LDA #AFC Protect all main RAM from ProDOS
1AFE- D0 02 5310 BNE SC.BM ...always
5320 CLR.PRODOS.BITMAP
1B00- A9 00 5330 LDA #A00 De-protect all main RAM
1B02- A2 13 5340 SC.BM LDX #13 Affects RAM from $0800 thru $9FFF
1B04- 9D 58 BF 5350 .1 STA PRODOS.BITMAP,X
1B07- CA 5360 DEX
1B08- D0 FA 5370 BNE .1
1B0A- 60 5380 RTS
5390 .EP
5400 *-----

```

Dissecting AppleWorks SEG.M0 and SEG.M1 Files...

...Bob Sander-Cederlof

The AppleWorks Program disk contains two files of type \$00, called SEG.M0 and SEG.M1. These contain the actual code for the three applications (data base, word processing, and spreadsheet) as a series of overlay segments.

These two files could really be just one, and I don't understand why they are not. It takes extra logic inside AppleWorks to manage them as two files, and the code would be a little simpler if there were only one. As it is, the first nine of 43 overlays are on SEG.M0, and the remaining 34 are on SEG.M1.

The overlay loader first decides which file to open, and then reads in the first 140 bytes of that file. There is a "directory" of sorts at the beginning of each file: one 3-byte value for each segment in the file. The 3-byte value is the offset within the file where the overlay begins. For example, looking at the beginning of the SEG.M0 file for AppleWorks version 1.3, in 3-byte groups, I see:

```

00 00 00
21 00 00
4C 30 00
9A 35 00
and so on

```

This means that overlay segment 0, which does not really exist, begins at offset \$000000 in the file. Overlay segment 1 begins at \$000021, or with the 33rd byte of the file. Segment 2 begins at \$034C in the file, and so on. By subtracting the beginning address of the segment I want to load from the address of the beginning of the next segment I get the number of bytes in the desired segment.

I decided to write a program to analyze these file headers for me, and print out a list of file offsets and lengths for each segment. The program follows, but before describing it I need to mention a table inside APLWORKS.SYSTEM. A table I call "SEGMENT.TABLE" begins at \$10A7 (in version 1.3). There are four bytes for each segment in this table. The first of each group of four is the page number where the corresponding segment should be loaded. Entries for segments \$0F, \$14, and \$15 are zero, meaning these segments do not exist. Segment \$00 does not exist either, but it is eliminated by other means. The other three bytes of each 4-byte group are used by the overlay loader to keep track of which overlay segments are already in RAM, in AuxRAM, or in a RamFactor type card.

I decided to copy the loading page numbers from this table into my little analysis program, so that it could also print out the loading address for each segment. You can see my copy at lines 1860-1930. I have added three labels to indicate which overlay segments belong to which of the three applications. The Data Base code is in segments \$01 through \$0F, the Word Processor in segments \$10-17, and the Spreadsheet in segments \$18-2B. At

least that is what I think is true, someone correct me if you know better.

My analysis program has several interesting wrinkles, of interest beyond the overall function of the program. I have defined two useful macros in lines 1070-1180. The first one, PRINT, generates a JSR PRINT followed by a zero-terminated string. As long as the string is purely printable ASCII characters which will fit in a .AS directive, the macro works nicely. The PRINT subroutine in lines 1680-1840 picks up the string which follows the JSR PRINT and prints it out, modifies the return address, and returns to the next instruction following the string.

The second macro calls on the Apple monitor to print a byte in hexadecimal. If there is no parameter in the >HEX macro call line, the macro will assume the byte to be printed is already in the A-register and generate only a JSR PRBYTE line. However, if you include one parameter, the macro will generate a LDA instruction to load the value into the A-register first. For example:

```
>HEX          generates JSR PRBYTE
>HEX #$24     generates LDA #$24
               JSR PRBYTE
>HEX BUFFER   generates LDA BUFFER
               JSR PRBYTE
>HEX "BUF+2,Y" generates LDA BUF+2,Y
               JSR PRBYTE
```

Note that in the last example I had to put the BUF+2,Y in quotation marks, so that the assembler would include the ",Y" as part of the 11 parameter.

When you use >HEX with no parameter, you also must not put a comment on the line. The first word of any comment would be considered as a parameter, generating bad results.

I assembled the program with the .LIST MOFF option, so that macro expansions are not shown.

The program assumes that you have BLOADED the SEG.Mx file header into a buffer starting at \$0A00. On my disk I have those files in a subdirectory called AW, so after assembling the program I typed:

```
BLOAD AW/SEG.M0,T$00,A$A00,L200
MGO T
BLOAD AW/SEG.M1,T$00,A$A00,L200
MGO T
```

Lines 1200-1280 search through the index beginning at \$0A00 for the first 3-byte entry which is not all zero. There is one 000000 entry in the SEG.M0 file, and there are ten of them in the SEG.M1 file. The first non-zero entry actually also points

Just past the end of the index header, so I save that value for a loop termination count in lines 1290-1310.

Lines 1330-1350 print "SEGMENT." and the two digit segment number in hexadecimal. Lines 1360-1390 decide whether the segment exists or not, and prints ": null" if it does not. If the segment does exist, lines 1410-1560 print out the "A\$xx00" load address, using the value from my LOAD.ADDRESS.TABLE; the "B\$xxxxxx" file offset; and the "L\$xxxx" segment length.

Lines 1570-1630 advance to the next three byte entry, and loop back if not at the end of the index header.

Using the information that prints out I could easily load any individual segment from within one of the SEG.Mx files and save it on its own private BIN file. For example, to load and save segment \$20 I would type:

```
BLOAD SEG.M1,T$00,A$1000,B$0144BB,L$1E4E
BSAVE SEGMENT.20,A$1000,L$1E4E
```

I could modify the analysis program to generate an EXEC file which would include two lines like these for each existing segment. Then EXECing the file would automatically produce 40 separate binary files, one for each overlay segment (not 43, because there are three "null" segments). This would make it easier to disassemble each one. I probably will end up modifying it this way eventually.

Another interesting program would create a new SEG.Mx file from a set of separate binary files within a subdirectory. What do you bet Robert Lissner has just such a program?

DON LANCASTER STUFF

INTRODUCTION TO POSTSCRIPT

A 65 min user group VHS video with Don Lancaster sharing many of his laser publishing and Postscript programming secrets.

Includes curve tracing, \$5 toner refilling, the full Kroy Kolor details, page layouts, plus bunches more.

\$39.50

FREE VOICE HELPLINE

ASK THE GURU

An entire set of reprints to Don Lancaster's ASK THE GURU columns, all the way back to column one. Edited and updated.

Both Apple and desktop publishing resources are included that are not to be found elsewhere.

\$24.50

APPLE IIc/IIe ABSOLUTE RESET

Now gain absolute control over your Apple! You stop any program at any time.

Eliminates all dropouts on your HIRES screen dumps. Gets rid of all hole blasting. For any IIc or IIe.

\$19.50

POSTSCRIPT SHOW & TELL

Unique graphics and text routines the others don't even dream of. For most any Postscript printer.

Fully open, unlocked, and easily adaptable to your own needs. Available for Apple, PC, Mac, ST, many others.

\$39.50

VISA/MC

SYNERGETICS

Box 809-SC

Thatcher, AZ 85552

(602) 428-4073

```

1010 *SAVE S.SHOW.INDEX
1020 -----
FD8E- 1030 CROUT .EQ $FD8E
FDDA- 1040 PRBYTE .EQ $FDDA
FDED- 1050 COUT .EQ $FDED
1060 -----
1070 .MA PRINT
1080 JSR PRINT Print 00-term'd string after
1090 .AS -"1" here is the string
1100 .HS 00 here is the 00-terminator
1110 .EM
1120 -----
1130 .MA HEX
1140 .DO ]# If any parameter, LDA it
1150 LDA ]1 here is the LDA
1160 .FIN
1170 JSR PRBYTE Print <A> in hexadecimal
1180 .EM
1190 -----
1200 T
0800- A2 FF 1210 LDX #-1 X will be the segment number
0802- A0 FD 1220 LDY #-3 FIND FIRST NON-ZERO ENTRY
0804- C8 1230 .1 INY
0805- C8 1240 INY
0806- C8 1250 INY
0807- E8 1260 INX
0808- B9 00 OA 1270 LDA BUF,Y
080B- F0 F7 1280 BEQ .1 ...this entry was empty
080D- 38 1290 SEC
080E- E9 03 1300 SBC #3
0810- 8D 8B 08 1310 STA Y.LIMIT
1320 -----
0813- 1330 .2 >PRINT "SEGMENT."
081F- 8A 1340 TXA PRINT SEGMENT NUMBER
0820- 1350 >HEX
0823- BD B0 08 1360 LDA LOAD.ADDRESS.TABLE,X
0826- D0 0E 1370 BNE .3 not a null segment
0828- 1380 >PRINT ": null"
0833- 4C 7E 08 1390 JMP .4
1400 -----
0836- 1410 .3 >PRINT ": A$" Print load address
083F- 1420 >HEX "LOAD.ADDRESS.TABLE,X
0845- 1430 >PRINT "00, B$" Print file offset
084F- 1440 >HEX "BUF+2,Y
0855- 1450 >HEX "BUF+1,Y
085B- 1460 >HEX "BUF Y
0861- 1470 >PRINT ", L$" Print segment length
0869- 38 1480 SEC
086A- B9 03 OA 1490 LDA BUF+3,Y
086D- F9 00 OA 1500 SBC BUF,Y
0870- 48 1510 PHA
0871- B9 04 OA 1520 LDA BUF+4,Y
0874- F9 01 OA 1530 SBC BUF+1,Y
0877- 1540 >HEX
087A- 68 1550 PLA
087B- 1560 >HEX
087E- 20 8E FD 1570 .4 JSR CROUT Next line
0881- E8 1580 INX Next segment number
0882- C8 1590 INY Next header pointer
0883- C8 1600 INY
0884- C8 1610 INY
0885- CC 8B 08 1620 CPY Y.LIMIT Into first segment?
0888- 90 89 1630 BCC .2 ...no, more lines
088A- 60 1640 RTS ...done
1650 -----
088B- 1660 Y.LIMIT .BS 1
1670 -----
088C- 68 1690 PRINT
088D- 8D 9D 08 1700 PLA POP RETURN ADDRESS
0890- 68 1710 PLA BECAUSE IT POINTS TO STRING
0891- 8D 9E 08 1720 STA .2+2
0894- EE 9D 08 1730 .1 INC .2+1 BUMP POINTER TO NEXT CHAR
0897- D0 03 1740 BNE .2
0899- EE 9D 08 1750 INC .2+1
089C- AD 33 33 1760 .2 LDA $3333 GET NEXT CHAR OF STRING
089F- FO 06 1770 BEQ .3 00 = END OF STRING
08A1- 20 ED FD 1780 JSR COUT PRINT CHAR
08A4- 4C 94 08 1790 JMP .1 ...NEXT

```

```

08A7- AD 9E 08 1800 .3 LDA .2+2 PUT RETURN ADDRESS ON STACK
08AA- 48 1810 PHA
08AB- AD 9D 08 1820 LDA .2+1
08AE- 48 1830 PHA
08AF- 60 1840 RTS
1850 -----
1860 LOAD.ADDRESS.TABLE
1870 .HS 30 dummy entry for segment 00
08B0- 30
08B1- 35 45 45
08B4- 45 45 45
08B7- 4E 4E 4A 1880 DATA.BASE .HS 35.45.45.45.45.4E.4E.4A
08BA- 4E 4E 4E
08BD- 53 4E 00 1890 .HS 4E.4E.4E.53.4E.00
08C0- 35 3D 3D
08C3- 40 00 00
08C6- 67 77 1900 WORD.PROC .HS 35.3D.3D.40.00.00.67.77
08C8- 33 3B 53
08CB- 53 53 53
08CE- 53 53 45 1910 SPRD.SHEE .HS 33.3B.53.53.53.53.53.45
08D1- 64 64 64
08D4- 64 64 64
08D7- 64 64 64 1920 .HS 64.64.64.64.64.64.64.64
08DA- 64 64 1930 .HS 64.64
1940 -----
0A00- 1950 BUF .EQ $A00
1960 -----

```

EnterSoft:

Basic-like macros which make the complex simple. Don't re-write that multiplication routine for the hundredth time! Get EnterSoft instead! Do 8/16/32/64 bit Math/Input-Output/Graphics simply without all of the hassles. These routines are a must for the serious programmer who doesn't want to spend all of his/her time trying to re-invent the wheel. DOS 3.3 Version=\$30.00, ProDos Version=\$30.00, BOTH for only \$50.00. GET YOURS TODAY.

A Shape Table Program:

For once! A shape table program which is logically organized into its componet parts. Each section resides in its own program. The editor, disk access, Hi-Res section; each section is separate. Written almost entirely in Basic, it is easily modified. Not copyprotected! Put them on a Hard Disk, Ram Drive, anywhere! DOS 3.3 Version=\$20.00, ProDos Version=\$20.00, BOTH for \$30.00!

```

Send Check or Money Order To:
Mark Manning
c/o Simulacron I/Baggy Game
P.O. Box 58598
Webster, TX 77598

ProDos Upgrade for DOS 3.3
EnterSoft Owners = $20.00

Thanks for the letters -
Keep Writing!

```

The Applied Engineering RamFactor memory card is now widely distributed, and with good reason. It is among my very favorite cards, and I use it heavily every day. I use mine with the RamCharger battery backup system, so that the memory stays loaded and ready all the time.

I have mine partitioned into two parts: a DOS 3.3 partition of 140K (floppy size), and a ProDOS partition with all the rest. I have been using Copy II Plus to backup the ProDOS partition on a 3.5 inch disk, but I haven't been keeping an up-to-date backup of the DOS partition. (Copy II Plus cannot access the DOS partition, or at least not when I have just loaded Copy II Plus out of the ProDOS partition on the same card.)

I have sometimes used FID to copy every file from the DOS partition to a floppy, but that takes a long time. In fact, when I tried it today, it took 160 seconds to save 31 files. And that only backs up the files. If the RamFactor is somehow clobbered, I will also need to restore the DOS image. My DOS is significantly patched, so I would really like to have it on the floppy too. Let's see.... I could boot from the RamFactor, go into Applesoft, load the HELLO program, directly type in the INIT command to initialize a floppy, then go into FID and copy all the rest of the files. Too much! And anyway, how do I get the floppy's contents copied back into the RamFactor?

Looking at the whole problem another way, what if I did not have the RamCharger? Then I would need to copy all the files and a DOS image into the card at least once a day. That could be really tiresome.

I decided to write a program to simplify things. My program has three parts: Format, Backup, and Restore. FORMAT.FLOPPY will do a raw format of a floppy disk. That is, it only writes the sector headers for 16 sectors on each of 35 tracks; it does not write a DOS image or an empty catalog. If the floppy I want to use has already been formatted, I can skip using FORMAT.FLOPPY.

BACKUP copies all the sectors of all 35 tracks from the RamFactor to the Floppy, and RESTORE does the reverse. BACKUP takes 46 seconds to copy and verify all 560 sectors, and RESTORE takes 25 seconds to read them back. Not as fast as possible, considering how fast Locksmith can copy one un-protected floppy to another, but my program is considerably shorter than Locksmith. If I leave out the Verify phase, BACKUP takes only 25 seconds.

Since all sector I/O is done by calls to RWTS, this same program could be used to backup and restore floppy-sized volumes on the Sider Hard Disk, with only minor modifications. Sider comes with a modified version of FID which already can do an "image" copy, as they call it, but it is too slow for me.

Another set of modifications would make my program work with 400K DOS partitions on the RamFactor and 3.5 inch disks formatted for use with DOS.

I decided to keep the program simple, for now. The slot numbers for the floppy drive and the RamFactor are assembled in at lines 1020-1030. A fancy program would probably do a slot search to find them, and give you a choice if there were more than one of either in the computer. A fancy program would also give you a little menu for selecting FORMAT, BACKUP, or RESTORE; I didn't do that either, but you can easily add one. One more improvement would be to automatically format the floppy if it isn't already formatted.

Well, let's look at what I DID do! Lines 1300-1370 show the two entry points for BACKUP and RESTORE. They load the slot numbers (shifted into the high nybble, so we say it is slot*16) into the A- and X-registers and go to COPY. COPY copies 35 tracks of 16 sectors each from the slot in the A-register to the slot in the X-register.

Lines 1390-1790 are the COPY subroutine. As each track is copied, I display the track number in hex, and the letters R, W, and V on the screen. After the letter R is displayed, I read the entire track from the source slot/drive into a buffer which starts at \$2000. After the letter W is displayed, I write out the track to the destination slot/drive. After the letter V is displayed, I read the entire track again, this time from the destination slot/drive. If RWTS does not report any error, I assume the track is good. A more excellent way might be to read the destination track into a different buffer, and compare all 4096 bytes with the original.

After the track is verified I print two more spaces, making the total number of characters displayed for each track, eight. This means I display either 5 or 10 tracks on a screen line, depending on whether the screen is set to 40- or 80-columns.

If you want to skip the verify step altogether, you can delete lines 1620-1660 and add one more JSR COUT after line 1700.

Reading or writing a track is handled by lines 1840-2070, RW.TRACK. This finishes setting up the IOB and calls RWTS to do the I/O. If RWTS reports any error during the copy process, copying stops and I print all the interesting information about the error. Lines 2090-2280 do the printing. You can also stop a copy by typing any key. Lines 1710-1720 look for a keypress after finishing each track, and abort if one is found.

RW.TRACK reads the sectors in the order 15 down to 0. Of course, RWTS translates the logical sector numbers into "real" sector numbers, but we don't need to worry about that. There is an optimum order to read or write sectors, and it depends on several factors. First, it depends on the interleaving order on the disk, as viewed through the RWTS logical sector numbers. Second, it depends on how much time is wasted between reading or writing each sector. Programs like Locksmith read an entire track in one revolution of the disk, once the beginning of any

sector is found. Locksmith also writes an entire track in one revolution. Using RWTS that is not possible, but you can probably do it in an average of 2.5 revolutions if you are smart enough. The drive spins at 5 revolutions per second, by the way.

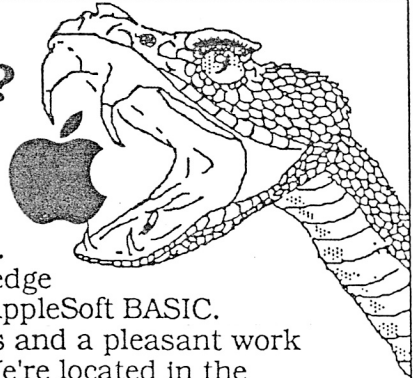
I tried reading the sectors in the order 0 to 15, and it took 100 seconds just to READ 35 tracks. In the order 15 to 0, this took only 24 seconds. The disk turns 120 times in 24 seconds, so I am averaging less than 3.5 revolutions per track including the time it takes to step from track to track. (Theodore Roosevelt used to warn national leaders around the world about the hazards and long-term negative results of a habit of revolution, but I think he was on a different track.)

If you decide to type in this program, with or without any modifications, be very careful about using it. You can easily wipe out the contents of the RamFactor with only a tiny bug. I carefully made a backup with FID before testing RESTORE. It turned out I didn't need it, but I am still glad I did.

Do You Have Apple Knowledge?

If you do, Applied Engineering would like to put your knowledge to work. We're looking for someone to fill a position in our Technical Support group. You must have a strong working knowledge of AppleWorks, ProDOS, DOS 3.3, and AppleSoft BASIC. Applied Engineering offers good benefits and a pleasant work environment. Office is non-smoking. We're located in the Carrollton area.

So if you've got the knowledge and want to sink your teeth into a position with an ever-expanding company, give us a call at (214) 241-6060.



```

1000 *SAVE S.FAST RAMFACTOR BACKUP
1010 *-----
04- 1020 RFSLOT .EQ 4 RAMFACTOR SLOT
06- 1030 FLSLOT .EQ 6 FLOPPY SLOT
1040 *-----
03E3- 1050 GETIOB .EQ $3E3
03D9- 1060 RWTS .EQ $3D9
1070 *-----
C000- 1080 KEYBOARD .EQ $C000
C010- 1090 STROBE .EQ $C010
1100 *-----
FD8E- 1110 CROUT .EQ $FD8E
FD8E- 1120 PRBYTE .EQ $FD8E
FD8E- 1130 COUT .EQ $FD8E
1140 *-----
1150 .DUMMY
1160 .OR $B7E8
B7E8- 1170 JOB .BS 1 Reference "Beneath Apple DOS"
B7E9- 1180 SLOT16 .BS 1
B7EA- 1190 DRIVE .BS 1
B7EB- 1200 VOLUME .BS 1
B7EC- 1210 TRACK .BS 1
B7ED- 1220 SECTOR .BS 1
B7EE- 1230 .BS 2 ADDR OF DCT
B7F0- 1240 BUFADR .BS 2
B7F2- 1250 .BS 2
B7F4- 1260 CMD .BS 1
B7F5- 1270 ERRCOD .BS 1
1280 .ED
1290 *-----
1300 BACKUP
0800- A9 40 1310 LDA #RFSLOT*16 From RamFactor to Floppy
0802- A2 60 1320 LDX #FLSLOT*16
0804- 4C 0B 08 1330 JMP COPY
1340 *-----
1350 RESTORE
0807- A9 60 1360 LDA #FLSLOT*16 From Floppy to RamFactor
0809- A2 40 1370 LDX #RFSLOT*16
1380 *-----
080B- 8D 61 08 1390 COPY STA SRC.SLOT16 Save Source Slot*16
080E- 8E 62 08 1400 STX DES.SLOT16 Save Destination Slot*16
0811- A9 01 1410 LDA #1 Both are drive 1
0813- 8D EA B7 1420 STA DRIVE
0816- A0 00 1430 LDY #0 For Track = 0 to 34
0818- 8C EC B7 1440 .1 STY TRACK
081B- 98 1450 TYA
081C- 20 DA FD 1460 JSR PRBYTE Print track number in hex
081F- A9 AD 1470 LDA #-" and a dash...
0821- 20 ED FD 1480 JSR COUT
1490 *---READ TRACK---
0824- A9 D2 1500 LDA #"R" Print "R"
0826- AE 61 08 1510 LDX SRC.SLOT16 Read track from Source Slot
0829- A0 01 1520 LDY #1 READ COMMAND
082B- 20 63 08 1530 JSR RW.TRACK
082E- B0 60 1540 BCS RWTS.ERROR ...Error
1550 *---WRITE TRACK ON FLOPPY---
0830- A9 D7 1560 LDA #"W" Print "W"
0832- AE 62 08 1570 LDX DES.SLOT16 Write track to Dest. Slot
0835- A0 02 1580 LDY #2 WRITE COMMAND
0837- 20 63 08 1590 JSR RW.TRACK
083A- B0 54 1600 BCS RWTS.ERROR ...Error
1610 *---VERIFY TRACK ON FLOPPY---
083C- A9 D6 1620 LDA #"V" Print "V"
083E- AE 62 08 1630 LDX DES.SLOT16 Read track for Dest. Slot
0841- A0 01 1640 LDY #1 READ COMMAND
0843- 20 63 08 1650 JSR RW.TRACK
0846- B0 48 1660 BCS RWTS.ERROR ...Error
1670 *---CHECK FOR ABORT---
0848- A9 A0 1680 LDA #" " Print 2 blanks
084A- 20 ED FD 1690 JSR COUT allowing 8 screen columns per track
084D- 20 ED FD 1700 JSR COUT
0850- AD 00 C0 1710 LDA KEYBOARD Press any key to abort
0853- 30 08 1720 BMI 2 ...ABORT
1730 *---NEXT TRACK---
0855- AC EC B7 1740 LDY TRACK
0858- C8 1750 INY
0859- C0 23 1760 CPY #35 limit to 35 tracks
085B- 90 BB 1770 BCC .1 ...more to do
085D- 8D 10 C0 1780 .2 STA STROBE ...done
0860- 60 1790 RTS

```

```

1800 *-----*
0861- 1810 SRC.SLOT16 .BS 1
0862- 1820 DES.SLOT16 .BS 1
2000- 1830 TRKBUF .EQ $2000 ... 2FFF
1840 *-----*
1850 * (A)=CHAR TO BE PRINTED
1860 * (X)=SLOT#16
1870 * (Y)=1 for READ, 2 for WRITE
1880 *-----*
1890 RW.TRACK
0863- 8E E9 B7 1900 STX SLOT16 Save slot#16 in IOB
0865- 8C F4 B7 1910 STY CMD Save command in IOB
0866- 20 ED FD 1920 JSR COUT Print R, W, or V
086C- A9 00 1930 LDA #0
086E- 8D EB B7 1940 STA VOLUME Accept any volume number
0871- 8D FO B7 1950 STA BUFADR Lo-byte of buffer address=00
0874- A9 20 1960 LDA /TRKBUF Hi-byte of buffer address
0876- 8D F1 B7 1970 STA BUFADR+1
0879- A0 0F 1980 LDY #15 For Sector = 15 to 0 step -1
087B- 8C ED B7 1990 .1 STY SECTOR
087E- 20 E3 03 2000 JSR GETIOB Setup for RWTS Call
0881- 20 D9 03 2010 JSR RWTS
0884- B0 09 2020 BCS .2 ...ERROR
0886- EE F1 B7 2030 INC BUFADR+1 Next Buffer Address
0889- AC ED B7 2040 LDY SECTOR
088C- 88 2050 DEY Next Sector
088D- 10 EC 2060 BPL .1 ...more to do
088F- 60 2070 .2 RTS ...done
2080 *-----*
2090 RWTS.ERROR
0890- 20 8E FD 2100 JSR CROUT Print all that's of interest
0893- AD E9 B7 2110 LDA SLOT16 Slot # 16
0896- 20 BA 08 2120 JSR PRBYTE.SPAC
0899- AD EA B7 2130 LDA DRIVE Drive number
089C- 20 BA 08 2140 JSR PRBYTE.SPAC
089F- AD EC B7 2150 LDA TRACK Track number
08A2- 20 BA 08 2160 JSR PRBYTE.SPAC
08A5- AD ED B7 2170 LDA SECTOR Sector number
08A8- 20 BA 08 2180 JSR PRBYTE.SPAC
08AB- AD F4 B7 2190 LDA CMD Command Code
08AE- 20 BA 08 2200 JSR PRBYTE.SPAC
08B1- AD F5 B7 2210 LDA ERRCOD Error Code
08B4- 20 BA 08 2220 JSR PRBYTE.SPAC
08B7- 4C 8E FD 2230 JMP CROUT
2240 *-----*
2250 PRBYTE.SPAC
08BA- 20 DA FD 2260 JSR PRBYTE Print value in hex
08BD- A9 A0 2270 LDA #" " and a space
08BF- 4C ED FD 2280 JMP COUT
2290 *-----*
2300 FORMAT.FLOPPY
08C2- A9 01 2310 LDA #1
08C4- 8D EB B7 2320 STA VOLUME Make it volume 1 (why not?)
08C7- 8D EA B7 2330 STA DRIVE on Drive 1
08CA- A9 60 2340 LDA #FLSLOT#16 Do it to the floppy
08CC- 8D E9 B7 2350 STA SLOT16
08CF- A9 04 2360 LDA #4 FORMAT COMMAND Code
08D1- 8D F4 B7 2370 STA CMD
08D4- 20 E3 03 2380 JSR GETIOB Set up RWTS call
08D7- 20 D9 03 2390 JSR RWTS
08DA- B0 B4 2400 BCS RWTS.ERROR
08DC- 60 2410 RTS Done
2420 *-----*

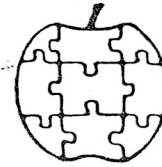
```

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$24 per year in the USA, Canada, and Mexico, or \$36 in other countries. Back issues are \$1.80 each for Volumes 1-7, \$2.40 each from later Volumes (plus postage). A subscription to the newsletter with a Monthly Disk containing all program source code and article text is \$64 per year in the USA, Canada and Mexico, and \$90 to other countries.

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)

Apple

\$2.40



Assembly

Line

Volume 8 -- Issue 7

April, 1988

Peeking Inside AppleWorks 1.3, Part 5:

Menu Display and Selection	2
ProDOS File Transformer (S-C into TXT)	18
Modify CATALOG to Show All AuxTypes	29
BLOADing ProDOS Directories	31
More on trip to Phoenix	32

68HC11 Cross Assembler Now Here!

At long last, I have written a cross assembler for the Motorola 68HC11. A lot of you have been asking about it, and thanks to the persistence of one customer it is now available. See our ad on page 3 for pricing and a list of other cross assemblers.

A Visit to Phoenix

The weekend of June 10-12 I attended the AZ Apple Fiesta in Phoenix, Arizona. (How can this be reported in the "April" issue? Sorry, we are still woefully behind in the publishing schedule, it is now late June.) The main attraction for me was a chance to spend a day at Western Design Center with Bill Mensch, the chief designer of the 65816 and its predecessors.

Secondly, I wanted to attend Randy Hyde's mini-conference on a new standard for 6502 family assembly language. Randy was there, with Brian Fitzgerald who now supports and markets the Lisa assembler. Roger Wagner represented the Merlin interests, and Mike Westerfield the ORCA/M and APW. Paul Santa-Maria and Chuck Kelly from ProDev Inc. in Michigan came to see about standardizing a symbol table format for use with symbolic debuggers. I also met P. J. Curran from ProData 21 in Florida. Since one cannot fly the 1000 miles to Phoenix directly from Dallas for less than \$250 each way, I drove 200 miles south to Austin to get a round trip ticket for only \$138. Bill Morgan lives in Austin now, and he decided to come also. Jeff Creamer and Don Lancaster were also involved in this conference. As far as I could tell, we did not make any real decisions about any changes to existing assemblers, or even future ones. Neither did we gain any hard data about the new opcodes and addressing modes of the 65832. But it was a very positive meeting.

(continued on page 32.)

Part 5 already? This is rapidly turning into a major series, and a popular one at that. A lot of you have called or written telling how much you like it (thank you!), so I will do my best to keep it interesting and useful.

This month I am going to document some subroutines which constitute a major portion of the "AppleWorks Human Interface". Namely, the menu display and selection subroutines.

We hear so much these days about the MacIntosh human interface, and indeed it has become the standard for the IIgs, like it or not. Personally, I can live with it, but I prefer plain old text menus. Why? They are faster, consume less of the computer's resources (leaving more of them for end users), and are quicker (for me) to use. Well-designed text menus with a consistent screen layout and selection method are easier for me than icons and mice.

How can that be? I thought Apple had proven the icon menu to be easier? I suppose they decided that after asking "the rest of us", leaving out the experienced computer people. But isn't text what we are trained from childhood to read? Some languages are written with icons (Chinese, for example), but MOST languages are written with text. And isn't the text-menu drilled into us from childhood also? Even the word "menu" reminds us of a list of textual lines for selecting the components of a meal. And what about multiple-choice tests?

And even some of the Apple folk believe in text, as is evidenced by the Control Panel and other Classic Desk Accessories on the IIgs.

All the above to justify adding my voice to that of Tom Weishaar. Tom is the author of the Open-Apple newsletter, which all Apple II lovers ought to read. We both think the AppleWorks human interface is a great starting point for the Apple II general operating environment. With just a little improvement, mostly in the area of pathname entry, and just a little generalization, we could use it for running all sorts of applications and compilers for all sorts of languages.

AppleWorks has an amazingly consistent human interface throughout. At almost every level you will see a simple list of numbered choices, with the same control scheme. One of the lines will be displayed in inverse mode, and you can either move the "bar" with the up- and down-arrow keys, or by typing the item number. When the bar is on your choice, the RETURN key will select it. ESCAPE will pop out of the menu to the previous level. You can usually get a help screen by typing Apple-?. The directions for getting help are shown at the right end of the bottom line, and the action ESCAPE will cause are shown at the right end of the top line.

If there happen to be more than nine numbered options, so that some of them have two-digit options, AppleWorks even allows you

S-C Macro Assembler Version 2.0	DOS \$100, ProDOS \$100, both for \$120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners	\$20
ProDOS Upgrade Kit for Version 2.0 DOS owners	\$30
Cross Assemblers for owners of S-C Macro Assembler	\$32.50 to \$50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68HC11, 68000, Z-80, Z-8, 8048, 8051, 8085, RCA 1802/4/5, PDP-11, GI1650/70, Mitsubishi 50740, others)	
Source Code of any S-C Macro Assembler or Cross Assembler	each, additional \$100
S-C DisAssembler (ProDOS only)	without source code \$30, with source \$50
RAK-Ware DISASM (DOS only)	without source code \$30, with source \$50
ProVIEW (ProDOS-based disk utility program)	\$20
Full Screen Editor for S-C Macro (with complete source code)	\$49
S-C Cross Reference Utility	without source code \$20, with source \$50
S-C Word Processor, both DOS & ProDOS, both 40- & 80-columns, with complete source code	\$50
DP18 and DPPP, double precision math for Applesoft, including complete source code	\$50
ES-CAPE (Extended S-C Applesoft Program Editor), including Version 2.0 With Source Code	\$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)	\$30
Bag of Tricks 2 (Quality Software)	(\$49.95) \$45 *
S-C Documentor (complete commented source code of Applesoft ROMs)	\$50
Copy II Plus (Central Point Software)	(\$39.95) \$30 *
AAL Quarterly Disks	each \$15, or any four for \$45

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Vinyl disk pages, 6"x8.5", hold two disks each	10 for \$6 *
Diskette Mailing Protectors (hold 1 or 2 disks)	40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.)	or \$25 per 100 *
Envelopes for Diskette Mailers	6 cents each

Sider 20 Meg Hard Disk, includes controller & software	(\$695) \$550 +
Sider 40 Meg Hard Disk, includes controller & software	(\$995) \$860 +

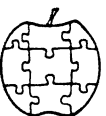
Minuteman 250 Uninterruptible Power Supply	(\$359) \$320 +
Minuteman 300 Uninterruptible Power Supply	(\$549) \$490 +

65802 Microprocessor, 4 MHz (Western Design Center)	\$25 *
quikLoader EPROM System (SCRG)	(\$179) \$170 *
PROMGRAMMER (SCRG)	(\$149.50) \$140 *

"Exploring the Apple IIgs"	Gary B. Little	(\$22.95)	\$21 *
"Apple IIgs Technical Reference"	Michael Fischer	(\$19.95)	\$19 *
"65816/65802 Assembly Language Programming"	Michael Fischer	(\$21.95)	\$20 *
"Programming the 65816"	David Eyes & Ron Lichty	(\$22.95)	\$21 *
"Programming the Apple IIgs in C and Assembly Language"	Mark Andrews	(\$18.95)	\$18 *
"Apple //e Reference Manual"	Apple Computer	(\$24.95)	\$23 *
"Apple //c Reference Manual"	Apple Computer	(\$24.95)	\$23 *
"ProDOS-8 Technical Reference Manual"	Apple Computer	(\$29.95)	\$27 *
"ProDOS-16 Technical Reference Manual"	Apple Computer	(\$29.95)	\$27 *
"Apple IIgs Firmware Reference"	Apple Computer	(\$24.95)	\$23 *
"Apple IIgs Hardware Reference"	Apple Computer	(\$24.95)	\$23 *
"Apple IIgs Toolbox Reference, Volume 1"	Apple Computer	(\$26.95)	\$24 *
"Apple IIgs Toolbox Reference, Volume 2"	Apple Computer	(\$26.95)	\$24 *
"Apple IIgs Programmer's Introduction"	Apple Computer	(\$32.95)	\$30 *
"ProDOS Inside and Out"	Dennis Doms & Tom Weishaar	(\$16.95)	\$16 *
"Beneath Apple ProDOS"	Don Worth & Pieter Lechner	(\$19.95)	\$18 *
"Beneath Apple DOS"	Don Worth & Pieter Lechner	(\$19.95)	\$18 *
"Inside the Apple //c"	Gary B. Little	(\$19.95)	\$18 *
"Inside the Apple //e"	Gary B. Little	(\$19.95)	\$18 *
"Understanding the Apple //e"	Jim Sather	(\$24.95)	\$23 *
"Understanding the Apple II"	Jim Sather	(\$22.95)	\$21 *
"Apple II+/IIE Troubleshooting & Repair Guide"	Brenner	(\$19.95)	\$18 *
"Assembly Language for Applesoft Programmers"	Finley & Myers	(\$18.95)	\$18 *
"Now That You Know Apple Assembly Language"	Jules Gilder	(\$19.95)	\$18 *
"Enhancing Your Apple II, vol. 1"	Don Lancaster	(\$15.95)	\$15 *
"Enhancing Your Apple II, vol. 2"	Don Lancaster	(\$17.95)	\$17 *
"Assembly Cookbook for the Apple II/IIE"	Don Lancaster	(\$21.95)	\$20 *
"Microcomputer Graphics"	Roy E. Myers	(\$14.95)	\$14 *

* These items add \$2 for first item, \$.75 for each additional item for US shipping.
+ Inquire for shipping cost.
Customers outside USA inquire for postage needed.
Texas residents please add 8% sales tax to all orders.
<< Master Card, VISA, Discover and American Express >>

S-C Software Corporation
2331 Gus Thomasson #125
DALLAS, TX 75228
Phone 214-324-2050



move the bar to the two-digit lines by typing their numbers. When you type a digit, the bar moves to one of the first nine lines. Then if you type a second digit, AppleWorks tries to move to the line with that two-digit number. If there is not one, you will hear a beep. If there is, the bar will move to that line. You can observe this action even on a menu with fewer than ten lines. When you select a line by typing a digit, you will not only see the bar move, but also the help and escape messages will change. On the top line you will see "Escape: Erase entry", and on the bottom line instead of an offer of help you will see "xxxK Avail." (the number of available memory bytes). When you type a second digit, you hear the beep and the help and escape info changes back to what it was before you typed the first digit.

Pretty slick. Until now I always assumed that numeric selection would either be limited to single-digit items, or that I would have to require a terminating character from the user so I would know when he had typed all the digits. Robert Lissner wins again! (Some of you have pointed out his name used to be "Rupert", but I read somewhere that he now prefers "Robert", or Bob.)

AppleWorks also allows the menu items to be anywhere on the screen. They do not even all have to be in one column, or vertically aligned. You can use single, double, or even variable vertical spacing.

Look ahead to the DISPLAY.MENU.LINE subroutine, lines 4580-5190. When you want to put a menu on the screen, you call this subroutine. The variable LAST.LINE.NUMBER.OF.MENU must be initially zeroed, and this subroutine will increment it each time you call. You call this subroutine once for each numbered line of your menu. Call it like this:

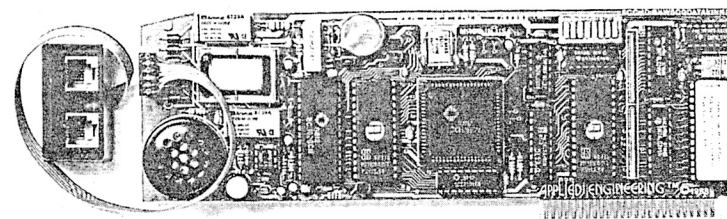
```
JSR DISPLAY.MENU.LINE
.DA #column,#line
.DA string
```

The string parameter is the address of a string which begins with a length byte. Column and line numbers are either absolute or relative. If you give values like 0-23 for line and 0-79 for column, they are absolute. If you add \$80 to line and/or column, it/they become relative to the coordinates stored in MENU.CORNER.LEFT and MENU.CORNER.TOP. Lines 4850-5000 compute the actual coordinates of the beginning of your menu line.

A block of 93 bytes called MENU.TABLE allows for keeping track of 30 menu lines. Each entry in this table is three bytes, and the zero'th entry is not used. Each time you call DISPLAY.MENU.LINE one entry is added to this table. Each three-byte entry consists of the column, line, and length of the menu item.

DISPLAY.MENU.LINE also takes care of prefixing the menu line number. You do not include it in your string. (This allows you to use the same strings in different menus.) Lines

The new DataLink™ 2400 modem from Applied Engineering, it's a lot more than just twice as fast.



Applied Engineering's new DataLink™ 2400. Simply put, the finest modem on the market for your Apple IIs, IIe or II+.

Bring home a world of information... from up to the minute flight information to whole libraries of resource materials. Even download free software and games.

Twice the speed.

At transmission speeds up to 2400 bps (bits-per-second), Applied Engineering's new DataLink 2400 is capable of putting text on the screen faster than the human eye can follow. That means you can capture a great deal more material in less time than with 1200 bps modems. And unlike other modems, the DataLink 2400 comes complete with powerful, easy-to-use communications software.



Complete communications software included.

Both our new DataLink 2400 and our DataLink 1200 modems feature AE's exclusive communications software—on disk and in ROM—everything needed to get you immediately up and running. Our powerful DataTerm software for the IIs and IIe supports VT-52 screen emulation, macros, file transfers, on-line time display, recording buffer and more. It even stores hundreds of phone numbers for auto-dialing and log on. And for II+ and 64K IIe owners, our OnLine 64 software has many of the same powerful features.

Worldwide compatibility.

The DataLink 2400 is fully compatible with Bell 103 and 212 protocols, as well as European protocol CCITT V.22 BIS, V.22 and V.21. It operates at varying transmission speeds from 0-300, 1200 and 2400 bps.

The new 2400, like our best-selling DataLink™ 1200, carries a full five year warranty and comes complete with two modular phone jacks for data and voice calls, a thoughtful feature that means fewer wires to connect. We also include an extra long telephone cable, in case your computer is across the room from your telephone jack. You can track the progress of calls either electronically or via on-board speaker. And built-in diagnostics reliably check transmission accuracy.

Prices subject to change without notice. Brands and product names are registered trademarks of their respective holders.

Packed with important features:

- Non-volatile memory for modem configuration
- Full Hayes AT compatibility
- Point-to-Point, ASCII Express, Access II compatibility; in addition to AE's included DataTerm and OnLine 64 software.
- Super Serial Card "Front End" for highest software compatibility (unlike others)
- Adaptive equalization and descrambling
- Hardware configuration for DSR and DCD
- PC Transporter (MS-DOS) compatibility
- FCC certified design

\$204.90 in freebies.

We also throw in a nice collection of goodies—a free subscription to the GENie network worth \$29.95, \$60 of free on-line time from NewsNet, a free \$50 subscription to the Official Airline Guide and a fee-waived membership to The Source worth \$49.95 plus \$15 of free on-line time.

That's \$204.90 worth of free memberships, discounts and on-line time when you purchase the powerful DataLink 2400 at \$239.

on-line time when you purchase the powerful DataLink 2400 at \$239.

DataLink 1200 reduced.

Loaded with all the features of the new 2400, (except CCITT, DSR/DCD and non-volatile ROM configurations) our 1200 bps DataLink modem, complete with software and freebies, is an affordable alternative at only \$179.

DataLink 1200..... \$179
DataLink 2400..... \$239

Order today!

To order or for more information, see your dealer or call (214) 241-6060 today, 9 am to 11 pm, 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10 outside USA.

AE APPLIED ENGINEERING™
The Apple enhancement experts.

(214) 241-6060

P.O. Box 5100, Carrollton, TX 75011

4690-4800 start building a display string in what I call the STR.A00 buffer. It begins with token \$05, which means "position cursor". Then it inserts the line number in the format " d. " or "dd. ". Lines 4850-5000 insert the chosen cursor position after the \$05 token. Lines 5010-5140 append your string to this position and line number string. Finally, lines 5150-5190 display the result.

So you can see that to put a complete menu on the screen we would first zero LAST.LINE.NUMBER.OF.MENU; then clear the window we are using; then call DISPLAY.MENU.LINE once for each menu item; and finally, add any titles and explanations.

Once a menu is on the screen, the next logical step is to call SELECT.MENU.LINE so the user can tell you his choice. This subroutine is shown beginning at line 2000. You call it with a line number in the A-register. The subroutine even expects that you have just loaded that number into the A-register, because it starts with a BEQ instruction. If the EQ status is set, probably meaning you just did a LDA #0, the menu will begin with the bar on the first menu line. You will also get the first item if the value in A is larger than the last menu item number, or if A=1. You can begin with any other line by putting that line number in A. You could keep track of the item selected that last time this menu was used, like Copy II Plus does.

The subroutine finally returns when you type either ESCAPE, Apple-?, or RETURN. In the first two cases MENU.LINE.SELECTED and the A-register will be zero; in the third case they will be, you guessed it, the number of the menu line selected.

Lines 2000-2110 figure out which line you want to begin with, and inverse it. Unless, that is, the value in Z.A4 is non-zero. I think that variable must be an "escape flag" of some sort. If it is non-zero, SELECT.MENU.LINE exits the same way it would if you typed the ESCAPE key.

Lines 2120-2130 display the generic instructions for making a menu selection on the screen. The text of this message is shown in lines 5200-5230.

The rest of SELECT.MENU.LINE divides nicely into two main sections. Lines 2150-2770 handle the selection until you type a digit; lines 2780-3430 take over when you type a digit, and don't let go until you either type an arrow (up-, down, or left-), DELETE, RETURN, or ESCAPE. I noticed a significant chunk of duplicate code in the two sections: see lines 2420-2470 and lines 3330-3380. I feel certain that the code could be re-arranged a little and save space. We don't really need it, but a little here and a little there could make room for many new features.

Lines 2150-2190 invert the current line, move the cursor down to the bottom line at the end of the "Type..." message, and wait for you to type something. If you remember AW.KEYIN from the Feb 88 issue, it will also take its own action if you type certain keys. For example, Apple-/ gets changed into Apple-?;

Apple-Q and Apple-S store a non-zero value in Z.A4 and change the character to ESCAPE; and several other interesting things. Look at page 13 in that issue, lines 2580-2790, for a summary.

Look ahead to lines 2900-3000 for an interesting use of the Z.A4 flag. If you typed Apple-Q or Apple-S, it will be changed into ESCAPE by AW.KEYIN and Z.A4 will be set non-zero. Then the test at 2970-2980 will make the jump all the way to SML.ESCAPED. If you typed ESCAPE, that test will merely send you back to SML.WAITING. Interesting.... So if you are at this level, where the escape info line says "Escape: Erase entry": typing one ESCAPE takes you back to SML.WAITING; typing two ESCAPES would get you to SML.ESCAPED, or typing one Apple-Q would do the same thing. Whoever called SELECT.MENU.LINE can tell the difference, though, because Z.A4 will still be non-zero for Apple-Q or zero for two ESCAPES. (For some reason I never can remember what Apple-Q and -S are really used for.)

Lines 2240-2310 branch appropriately for up- or down-arrow or RETURN. Lines 2590-2770 handle the up- and down-arrows: simply a matter of changing the display of the current line back to normal mode, and raising or lowering the selected line number. The line number changes circularly, so that up-arrow off the top wraps around to the bottom, and down-arrow at the bottom goes to the top.

Lines 2340-2500 check for a digit and operate on it. Notice that only digits 1-9 are accepted here, while digits 0-9 are accepted at lines 3170-3200. This will be the first digit of a menu line number, so it cannot be zero. The other place it will be the second digit, so it could be zero. If this first digit is acceptable (between 1 and the last menu line number), lines 2420-2470 will echo the character at the cursor on line 23, restore the current line to normal mode, and store the new menu line number.

Lines 2480-2490 call a subroutine in the \$D000 area to change the escape and help info. This subroutine decides which pair of messages to display by the letter in the caller's A-register. Letter "E" makes escape say "Escape: Erase entry" and help say how much memory is left. Letter "S", used in lines 1910-1920, makes help say "Apple-? for help" and escape say whatever string was saved in a buffer at \$0CFD. The subroutine has several other options.

There appears to be either a bug or a leftover at lines 2520-2530. If you type Apple-? you get to this line. It loads up Z.89, and then acts like you typed ESCAPE. However, the contents of Z.89 are not needed here, because the first instruction after the JMP is another LDA!

I have probably already said enough about lines 2800-3430, especially since it is so similar to the section above. I would, however, like to look at a few lines. The code in lines 3210-3280 multiplies the previous menu line number by ten and adds in the next digit. At first glance it looks reasonably efficient for space, since it calls on a multiplication

subroutine. Nevertheless, I can recode it in one less byte and many fewer cycles without calling the subroutine. Change lines 3210-3280 to the following:

```

3210 AND #0F isolate new digit
3220 STA Z.91 save for later
3230 LDA MENU.LINE.SELECTED
3240 ASL old * 2
3250 ASL old * 4
3260 ADC MENU.LINE.SELECTED old*5
3270 ASL old * 10
3280 ADC Z.91 plus new digit

```

Let's move ahead. When you type RETURN, lines 3600-3720 will clear line 23, restore the selected line to normal mode, and draw an arrow over the line number ("-->"). Either ESCAPE or RETURN brings you to lines 3740-3810, which erase the MENU.TABLE and return with the selected line number in the A-register. It makes no sense to me to erase MENU.TABLE, since it will not be used again until it is re-loaded anyway, but maybe there is some reason it is needed.

Lines 3860-4540 will copy the selected line off the screen into a buffer at \$0900, and then re-display it in one of three ways: the entire line NORMAL, the entire line INVERSE, or just the portion (A) characters long starting in column (X) in INVERSE.

Now if only AppleWorks could handle pathname entry as easily! The human interface presented by such Quit Code replacements as ProSel, Squirt, or the S-C Program Selector would be an improvement.

```

80- 1030 PSTR .EQ $80,81
84- 1040 CHAR.FROM.KEYIN .EQ $84
8C- 1050 MENU.LINE.SELECTED .EQ $8C
89- 1060 Z.89 .EQ $89
91- 1070 Z.91 .EQ $91 result of multiply here
9A- 1080 P0 .EQ $9A parameters from GET.4.PARMS
9B- 1090 P1 .EQ $9B "
9C- 1100 P2 .EQ $9C "
A4- 1110 Z.A4 .EQ $A4 escape flag???
1120 #-----
0A00- 1130 STR.A00 .EQ $A00
0FBE- 1140 MENU.CORNER.LEFT .EQ $0FBE
0FBF- 1150 MENU.CORNER.TOP .EQ $0FBF
1160 #-----
1170 .DUMMY
1180 .OR $0EB6
0EB6- 1190 MENU.TABLE .BS 3*31 Room for 30 3-byte entries
0F13- 1200 LAST.LINE.NUMBER.OF.MENU .BS 1
1210 .ED
1220 #-----Covered in AAL-----
14D1- 1230 DISPLAY.STRING .EQ $14D1 Jan 88, page 10
179D- 1240 CONVERT.A.TO.RJBF.STRING .EQ $179D
1818- 1250 BEEP.AND.CLEAR.KEYBUF .EQ $1818
1823- 1260 MOVE.CURSOR.TO.XY .EQ $1823 Feb 88, page 17
1B3A- 1270 MULTIPLY.X.BY.Y .EQ $1B3A
187A- 1280 COPY.SCRN.LINE.TO.0900 .EQ $187A Feb 88, page 18
18AE- 1290 GET.4.PARMS .EQ $18AE Dec 87, page 7
1D35- 1300 AW.KEYIN .EQ $1D35 Feb 88, page 12
1E80- 1310 MOVE.CURSOR.TO.TCOL.TROW .EQ $1E80
1EA9- 1320 POINT.PSTR.AT.STR.A00 .EQ $1EA9 Feb 88, page 18
1EF8- 1330 MOVE.STRING .EQ $1EF8 Dec 87, page 8
1F3E- 1340 DISPLAY.AT .EQ $1F3E
1FE0- 1350 CLEAR.KEYBUF .EQ $1FE0 Feb 88, page 17
2093- 1360 DISPLAY.STRING.PO .EQ $2093
1FE9- 1370 DISPLAY.TOKEN.X .EQ $1FE9
1FF5- 1380 DISPLAY.ON.LINE.23 .EQ $1FF5
1390 #-----
D023- 1400 SHOW.ESCAPE.AND.HELP.MSGS .EQ $D023
1410 #-----

```



SPECIAL !!! EXPANDED RAM/ROM BOARD: \$39.00

Similar to our \$30 RAM/ROM dev board described below. Except this board has two sockets to hold your choice of 2-2K RAM, 2-2K ROM or even 2-4K ROM for a total of 8K. Mix RAM and ROM too. Although Apple limits access to only 2K at a time, soft switches provide convenient socket selection. Hard switches control defaults.

IMPROVED !!! II IN A MAC (ver 2.0): \$75.00

Now includes faster graphics, UniDisk support and more! Bi-directional data transfers are a snap! This Apple II emulator runs DOS 3.3/PRODOS (including 6502 machine language routines) on a 512K MAC or MACPLUS. All Apple II features are supported such as HI/LO-RES graphics, 40/80 column text, language card and joystick. Also included: clock, RAM disk, keyboard buffer, on-screen HELP, access to the desk accessories and support for 4 logical disk drives. Includes 2 MAC diskettes (with emulation, communications and utility software, plus DOS 3.3 and PRODOS system masters, including Applesoft and Integer BASIC) and 1 Apple II diskette.

SCREEN.GEN: \$35.00

Develop HI-RES screens for the Apple II on a Macintosh. Use MACPAINT (or any other application) on the MAC to create your Apple II screen. Then use SCREEN.GEN to transfer directly from the MAC to an Apple II (with SuperSerial card) or IIC. Includes Apple II diskette with transfer software plus fully commented SOURCE code.

MIDI-MAGIC for Apple //c: \$49.00

Compatible with any MIDI equipped music keyboard, synthesizer, organ or piano. Package includes a MIDI-out cable (plugs directly into modem port - no modifications required!) and 6-song demo diskette. Large selection of digitized QRS player-piano music available for 19.00 per diskette (write for catalog). MIDI-MAGIC compatible with Apple II family using Passport MIDI card for our own input/output card w/drum sync for only \$99.00.

FONT DOWNLOADER & EDITOR: \$39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. Special functions (like expanded, compressed etc.) supported. Includes HIRES screen editor to create custom fonts and special graphics symbols. For Apple II, II+, IIc. Specify printer: Apple Imagewriter, Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/85, or Okidata 92/192.

* FONT LIBRARY DISKETTE #1: \$19.00 contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

DISASM 2.2e : \$30.00 (\$50.00 with SOURCE Code)

Use this intelligent disassembler to investigate the inner workings of Apple II machine language programs. DISASM converts machine code into meaningful, symbolic source compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even provides label substitution. Address-based triple cross reference generator included. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his ASSEMBLY COOKBOOK.

The 'PERFORMER' CARD: \$39.00 (\$59.00 with SOURCE Code)

Converts a 'dumb' parallel printer I/F card into a 'smart' one. Simple command menu. Features include perforation skip, auto page numbering with date & title, large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), Okidata 82A/83A with Okigraph & Okidata 92/93.

'MIRROR' ROM: \$25.00 (\$45.00 with SOURCE Code)

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support.

RAM/ROM DEVELOPMENT BOARD: \$30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

C-PRINT For The APPLE //c: \$69.00

Connect standard parallel printers to an Apple //c serial port. Separate P/S included. Just plug in and print!

Unless otherwise specified, all Apple II diskettes are standard (not copy protected!) 3.3 DOS. Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders OK.

RAK-WARE 41 Ralph Road W. Orange N.J. 07052 (201) 325-1885



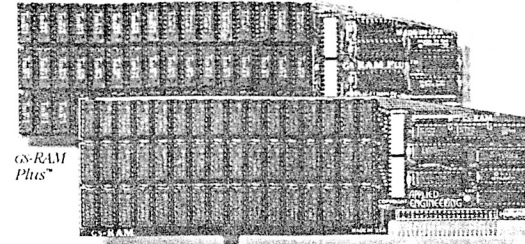
```

1420 .MA MSG Macro to make a string with
1430 .DA #1, *-1 first byte is the length,
1440 .AS #1# the rest is ASCII.
1450 :1
1460 .EM
1470 -----
1480 .PH $18DD
1490 -----
1500 * Load X-reg with pointer into MENU.TABLE
1510 * which is menu line number times 3.
1520 * (18DD) 18E4 18F2 1A4D
1530 GET.MENU.TABLE.INDEX
18DD- A5 8C 1540 LDA MENU.LINE.SELECTED
18DF- 0A 1550 ASL
18E0- 65 8C 1560 ADC MENU.LINE.SELECTED
18E2- AA 1570 TAX
18E3- 60 1580 RTS
1590 -----
1600 * Re-display entire line in NORMAL mode.
1610 * (18E4) 1973 198B 199A 1A21 1A39 1A4A
1620 MAKE.MENU.LINE.NORMAL
18E4- 20 DD 18 1630 JSR GET.MENU.TABLE.INDEX
18E7- BD B7 0E 1640 LDA MENU.TABLE+1,X Get screen line #
18EA- A8 1650 TAY
18EB- A9 00 1660 LDA #00 Display entire line NORMAL
18ED- AA 1670 TAX
18EE- 20 77 1A 1680 JSR REVERSE.A.SCREEN.LINE
18F1- 60 1690 RTS
1700 -----
1710 * Re-Display the menu line in INVERSE mode.
1720 * (18F2) 1936 19AF 1A27
1730 MAKE.MENU.LINE.INVERSE
18F2- 20 DD 18 1740 JSR GET.MENU.TABLE.INDEX
18F5- BD B8 0E 1750 LDA MENU.TABLE+2,X get length of menu msg
18F8- 48 1760 PHA
18F9- BD B7 0E 1770 LDA MENU.TABLE+1,X get screen line #
18FC- A8 1780 TAY
18FD- BD B6 0E 1790 LDA MENU.TABLE,X get starting column of msg
1900- 18 1800 CLC
1901- 69 04 1810 ADC #4 skip over "##."
1903- AA 1820 TAX start at blank before msg itself
1904- 68 1830 PLA get length again
1905- 18 1840 CLC
1906- 69 02 1850 ADC #2 include leading and trailing blank
1908- 20 77 1A 1860 JSR REVERSE.A.SCREEN.LINE
190B- 60 1870 RTS
1880 -----
1890 * (190C) 19D8 19E2 19EC 19F6
1900 RESTORE.ESCAPE.AND.HELP.MSGS
190C- A9 53 1910 LDA #53
190E- 20 23 DO 1920 JSR SHOW.ESCAPE.AND.HELP.MSGS
1911- 20 80 1E 1930 JSR MOVE.CURSOR.TO.TCOL.TROW
1914- 20 93 20 1940 JSR DISPLAY.STRING.PO
1917- 1A 19 1950 .DA TWO.BLANKS
1919- 60 1960 RTS
1970 -----
1970 TWO.BLANKS .HS 02.20.20
1990 -----
2000 * (A) = menu line number; if =0, or >lastline, use 1
2010 -----
2020 SELECT.MENU.LINE
191D- F0 07 2030 BEQ .1 Use top menu line
191F- CD 13 OF 2040 CMP LAST.LINE.NUMBER.OF.MENU
1922- 90 04 2050 BCC .2 Use selected line
1924- F0 02 2060 BEQ .2 Use selected line (bottom)
1926- A9 01 2070 .1 LDA #1 Use top menu line
1928- 85 8C 2080 .2 STA MENU.LINE.SELECTED
192A- A5 A4 2090 LDA Z.A4
192C- F0 03 2100 BEQ .3
192E- 4C 3F 1A 2110 JMP SML.ESCAPED.WITHOUT.INVERSE
1931- 20 F5 1F 2120 .3 JSR DISPLAY.ON.LINE.23
1934- 6A 14 2130 .DA MENU.MSG "Type number, or use arrows, then press Return"
2140 * (1936) 1997 19AC 19DB
2150 SML.WAITING
1936- 20 F2 18 2160 JSR MAKE.MENU.LINE.INVERSE
1939- 20 E0 1F 2170 .1 JSR CLEAR.KEYBUF
193C- 20 80 1E 2180 JSR MOVE.CURSOR.TO.TCOL.TROW
193F- 20 35 1D 2190 JSR AW.KEYIN
1942- C9 1B 2200 CMP #1B did he type ESCAPE?
1944- D0 03 2210 BNE .2 ...not ESCAPE
1946- 4C 32 1A 2220 JMP SML.ESCAPED

```

For Those Who Want the Most. From Those Who Make the Best. GS-RAM™

Now expand the IIcs™ RAM and ROM with up to 8 MEG of "Instant On" memory with the all new GS-RAM!



GS-RAM has an all new design. A design that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

More Sophisticated, Yet Easier to Use

GS-RAM works with all IIcs software. In fact any program that runs on Apple's smaller memory card runs on the GS-RAM. But with GS-RAM you can have more memory, improved performance, and almost unlimited expansion capabilities. We've designed the new GS-RAM to be easier to use too—you don't have to adjust the size of your RAM disk every time you use a DMA device. No other RAM card with more than 4 banks of memory installed can make the same claim.

More than Just Hardware

Each GS-RAM and GS-RAM Plus includes the most powerful set of IIcs software enhancements available anywhere. In fact, our nearest competitor offers only a fraction of the invaluable programs that we include with each GS-RAM card. This software includes the most powerful disk caching program available, the GS-RAM Cache. The Cache will make most of your applications run up to 7 times faster. Also included is a diagnostic utility that lets you test your GS-RAM by graphically showing the location of any bad or improperly installed RAM chips. And for AppleWorks users, we give you our exclusive Expander program that dramatically enhances both the capabilities and speed of AppleWorks.

Making AppleWorks Even Better

Applied Engineering's Expander program eliminates AppleWorks internal memory limits allowing it to recognize up to 8 megabytes of desktop workspace. You can increase the limits from only 7,250 lines to 22,600 lines in the word processor and from 6,350 records to 22,600 records in the database. The Expander allows all of AppleWorks, including print functions, to automatically load into RAM. The clipboard size will increase from 255 to 2,042 lines maximum. GS-RAM will automatically segment larger files so you can save them onto multiple floppies. And

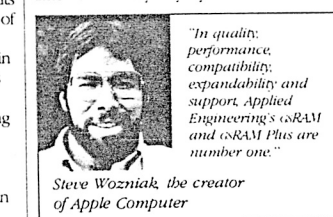
GS-RAM provides a built-in print buffer that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

Grow by Kilobytes or Megabytes

We offer GS-RAM in two configurations so you can increase your memory 256K at a time (GS-RAM) or a megabyte at a time (GS-RAM Plus). Both are IIcs compatible and both come with our powerful enhancement software. GS-RAM can hold up to 1.5 MEG of 256K chips and GS-RAM Plus can hold up to 6 MEG using 1 MEG chips. And since both use standard RAM chips (not high-priced SIMMs), you'll find expanding your GS-RAM or GS-RAM Plus easy, convenient, and very economical. For further expansion, you can plug a 2 MEG "piggyback" card into the GS-RAM's expansion port for up to 3.5 MEG of total capacity. Or up to a whopping 8 MEG on GS-RAM Plus. If a GS-RAM owner outgrows 3.5 MEG, he can easily upgrade to GS-RAM Plus for a nominal charge.

Permanent Storage for an "Instant On" Apple

With our RamKeeper™ back-up option, your GS-RAM or GS-RAM Plus will retain both programs and data while your IIcs is turned off. Now when you turn your IIcs back on, your favorite software is on your screen in under 4 seconds! With RamKeeper you can divide your IIcs memory into part "electronic hard disk" and part extended RAM. Even change the memory boundaries at any time—and in any way—you want. Because



"In quality, performance, compatibility, expandability and support, Applied Engineering's GS-RAM and GS-RAM Plus are number one."

Steve Wozniak, the creator of Apple Computer

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple, you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, Reliability is everything. That's why Applied Engineering uses state-of-the-art "GEL-CELLS" instead of Ni-Cad batteries (if Ni-Cads aren't discharged periodically, they lose much of their capacity). RamKeeper has about 6 hours of "total power failure" back-up time. That's 6 times the amount of other systems. But with power from your wall outlet, RamKeeper will back-up GS-RAM, GS-RAM Plus, or most other IIcs memory cards indefinitely. Should you ever have a "total power failure," RamKeeper switches to its 6-hour battery. When power returns, RamKeeper will automatically recharge the battery to full power. RamKeeper incorporates a dual-rate charger, status LED's, and advanced power reducing circuitry. RamKeeper comes complete with battery, software, and documentation.

GS-RAM's Got it ALL!

- 5-year warranty — parts & labor
- 6 RAM banks (most cards have 4)
- Memory expansion port
- ROM expansion port
- Ultra-fast disk caching on ProDOS 8 AND ProDOS 16.
- Expands AppleWorks internal limits
- Includes hi-res self test
- No soldered-in RAM chips
- Expandable to 8 MEG
- No configuration blocks to set
- RamKeeper back-up option allows permanent storage of programs & data
- 15-day money-back guarantee
- Proudly made in the U.S.A.

GS-RAM with 256K	\$189
GS-RAM with 512K	\$259
GS-RAM with 1 MEG	\$399
GS-RAM with 1.5 MEG	\$539
GS-RAM with 2.5 to 3.5 MEG	CALL
GS-RAM Plus with 1-8 MEG	CALL
RamKeeper Option	\$179

Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside U.S.A.

AE APPLIED ENGINEERING™
The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006
Prices subject to change without notice

GS-RAM, GS-RAM Plus and RamKeeper are trademarks of Applied Engineering. Other brands and product names are registered trademarks of their respective holders.

1949- C9 OD	2230	.2	CMP # \$OD	is it RETURN?
194B- DO 03	2240		BNE .3	...not RETURN
194D- 4C 45	2250		JMP SML.RETURN	
1950- C9 OB	2270			
1952- FO 37	2280	.3	CMP # \$OB	up arrow?
1954- C9 OA	2290		BEQ SML.UP.ARROW	
1956- FO 42	2300		CMP # \$OA	down arrow?
1958- C9 BF	2310		BEQ SML.DOWN.ARROW	
195A- FO 25	2320		CMP # "?"	Apple and "?"
195C- C9 31	2330		BEQ .5	...question mark
195E- 90 26	2340		CMP # \$31	
1960- C9 3A	2350		BCC .6	...not digit, beep!
1962- BO 22	2360		CMP # \$3A	
1964- 29 OF	2370		BCS .6	...not digit, beep!
1966- CD 13	2380		AND # \$OF	
1969- FO 02	2390		CMP LAST.LINE.NUMBER.OF.MENU	
196B- BO 19	2400		BEQ .4	...bottom line
196D- 48	2410		BCS .6	...beyond the bottom, beep!
196E- A6 84	2420	.4	PHA	save the digit typed
1970- 20 E9	2430		LDX CHAR.FROM.KEYIN	
1973- 20 E4	2440	1F	JSR DISPLAY.TOKEN.X	
1976- 68	2450	18	JSR MAKE.MENU.LINE.NORMAL	
1977- 85 8C	2460		PLA	get digit typed again
1979- A9 45	2470		STA MENU.LINE.SELECTED	select line 1-9
197B- 20 23	2480	DO	LDA # \$45	
197E- 4C AF	2490	19	JSR SHOW.ESCAPE.AND.HELP.MSGS	
	2500		JMP SML.DIGIT	
	2510		*---Apple-? typed---	
1981- A5 89	2520	.5	LDA Z.89	...why? opcode JMPed to is LDA also!
1983- 4C 3F	2530	1A	JMP SML.ESCAPED.WITHOUT.INVERSE	
	2540			
1986- 20 18	2550	18	JSR BEEP.AND.CLEAR.KEYBUF	
1989- DO AE	2560		BNE .1	...always
	2570			
	2580		* (198B) 1952 19EF	
	2590		SML.UP.ARROW	
198B- 20 E4	2600	18	JSR MAKE.MENU.LINE.NORMAL	
198E- C6 8C	2610		DEC MENU.LINE.SELECTED	
1990- DO 05	2620		BNE .1	...still in range
1992- AD 13	2630	OF	LDA LAST.LINE.NUMBER.OF.MENU	
1995- 85 8C	2640		STA MENU.LINE.SELECTED	
1997- 4C 36	2650	19	JMP SML.WAITING	
	2660			
	2670		* (199A) 1956 19F9	
	2680		SML.DOWN.ARROW	
199A- 20 E4	2690	18	JSR MAKE.MENU.LINE.NORMAL	
199D- A5 8C	2700		LDA MENU.LINE.SELECTED	
199F- CD 13	2710	OF	CMP LAST.LINE.NUMBER.OF.MENU	
19A2- BO 04	2720		BCS .1	...already on bottom, wrap to top
19A4- E6 8C	2730		INC MENU.LINE.SELECTED	next line down
19A6- DO 04	2740		BNE .2	...always
19A8- A9 01	2750	.1	LDA #1	top line now
19AA- 85 8C	2760		STA MENU.LINE.SELECTED	
19AC- 4C 36	2770	19	JMP SML.WAITING	
	2780			
	2790		* (19AF) 197E	
	2800		SML.DIGIT	
19AF- 20 F2	2810	18	JSR MAKE.MENU.LINE.INVERSE	
19B2- 20 80	2820	1E	JSR MOVE.CURSOR.TO.TCOL.TROW	
19B5- A2 07	2830		LDX # \$07	CURSOR RIGHT TOKEN
19B7- 20 E9	2840	1F	JSR DISPLAY.TOKEN.X	
19BA- A5 8C	2850		LDA MENU.LINE.SELECTED	
19BC- C9 OA	2860		CMP #10	on line 1-9?
19BE- 90 05	2870		BCC .2	...yes
19CO- A2 07	2880		LDX # \$07	CURSOR RIGHT TOKEN
19C2- 20 E9	2890	1F	JSR DISPLAY.TOKEN.X	
19C5- 20 35	2900	1D	JSR AW.KEYIN	get another char, maybe 2nd digit
19C8- C9 1B	2910		CMP # \$1B	is it ESCAPE?
19CA- FO 08	2920		BEQ .3	...yes
19CC- C9 08	2930		CMP # \$08	
19CE- FO 04	2940		BEQ .3	...backspace
19DO- C9 7F	2950		CMP # \$7F	...is it DELETE?
19D2- DO OA	2960		BNE .4	...yes
19D4- A5 A4	2970	.3	LDA Z.A4	
19D6- DO 5A	2980		BNE SML.ESCAPED	
19D8- 20 OC	2990	19	JSR RESTORE.ESCAPE.AND.HELP.MSGS	
19DB- 4C 36	3000	19	JMP SML.WAITING	

```

1A6D- 3820 *-----
1A72- 01 02 3830 ARRCW >MSG " -->"
3840 I.1A72 .HS 01.02 1 BYTE STRING, CLEAR LINE TOKEN
3850 *
3860 * Re-display a screen line in INVERSE or NORMAL mode
3870 * (Y) = screen line to be re-displayed
3880 * (X) = column number to begin with
3890 * (A) = 0 then re-display entire line NORMAL
3900 * (A) > 78 then redisplay entire line INVERSE
3910 * (A) = length of middle section of line to be
3920 * displayed in INVERSE.
3930 *-----
1A74- 3940 RASL.X .BS 1
1A75- 3950 RASL.Y .BS 1
1A76- 3960 RASL.A .BS 1
3970 *-----
3980 * (1A77) 103F 18EE 1908
3990 REVERSE.A.SCREEN.LINE
1A77- 8E 74 1A 4000 STX RASL.X Save the column
1A7A- 8C 75 1A 4010 STY RASL.Y Save the line
1A7D- 8D 76 1A 4020 STA RASL.A Save the length
1A80- 98 4030 TYA get line number
1A81- 20 7A 18 4040 JSR COPY.SCRN.LINE.TO.0900
1A84- A2 00 4050 LDX #0 start in column 0
1A86- AC 75 1A 4060 LDY RASL.Y on line Y
1A89- 20 23 18 4070 JSR MOVE.CURSOR.TO.XY
1A8C- AD 76 1A 4080 LDA RASL.A get specified length
1A8F- F0 09 4090 BEQ .1 ...0 means display it all NORMAL
1A91- C9 4F 4100 CMP #79
1A93- 90 1C 4110 BCC .2 ...it really is a length
1A95- A2 0A 4120 LDX #0A ...>78, INVERSE the line
1A97- 20 E9 1F 4130 JSR DISPLAY.TOKEN.X
1A9A- AD FA 1A 4140 LDA HANDLE.0900
1A9D- 85 80 4150 STA PSTR
1A9F- AD FB 1A 4160 LDA HANDLE.0900+1
1AA2- 85 81 4170 STA PSTR+1
1AA4- A9 4F 4180 LDA #79 display 79 characters
1AA6- 20 D1 14 4190 JSR DISPLAY.STRING
1AA9- A2 0B 4200 LDX #0B NORMAL TOKEN
1AAB- 20 E9 1F 4210 JSR DISPLAY.TOKEN.X
1AAE- 4C F9 1A 4220 JMP .3 ...ONLY AN RTS
4230 *-----
1AB1- AD FA 1A 4240 LDA HANDLE.0900 Point to line image
1AB4- 85 80 4250 STA PSTR
1AB6- AD FB 1A 4260 LDA HANDLE.0900+1
1AB9- 85 81 4270 STA PSTR+1
1ABB- AD 74 1A 4280 LDA RASL.X Display up to X in NORMAL
1ABE- 20 D1 14 4290 JSR DISPLAY.STRING
1AC1- A2 0A 4300 LDX #0A INVERSE TOKEN
1AC3- 20 E9 1F 4310 JSR DISPLAY.TOKEN.X
1AC6- AD 74 1A 4320 LDA RASL.X Display middle section INVERSE
1AC9- 85 80 4330 STA PSTR
1ACB- AD FB 1A 4340 LDA HANDLE.0900+1
1ACE- 85 81 4350 STA PSTR+1
1AD0- AD 76 1A 4360 LDA RASL.A # chars in middle section
1AD3- 20 D1 14 4370 JSR DISPLAY.STRING
1AD6- A2 0B 4380 LDX #0B NORMAL TOKEN
1AD8- 20 E9 1F 4390 JSR DISPLAY.TOKEN.X
1ADB- AD 74 1A 4400 LDA RASL.X Display rest of line NORMAL
1ADE- 18 4410 CLC
1ADF- 6D 76 1A 4420 ADC RASL.A
1AE2- B0 15 4430 BCS .3 ...line too long, forget the rest
1AE4- C9 4F 4440 CMP #79
1AE6- B0 11 4450 BCS .3 ...line too long, forget the rest
1AE8- 85 80 4460 STA PSTR
1AEA- AD FB 1A 4470 LDA HANDLE.0900+1
1AED- 85 81 4480 STA PSTR+1
1AEF- A9 4F 4490 LDA #79 79 chars
1AF1- 38 4500 SEC
1AF2- E5 80 4510 SBC PSTR
1AF4- F0 03 4520 BEQ .3 ...no space left on the line
1AF6- 20 D1 14 4530 JSR DISPLAY.STRING
1AF9- 60 4540 RTS
4550 *-----
1AFA- 00 09 4560 HANDLE.0900 .DA $0900

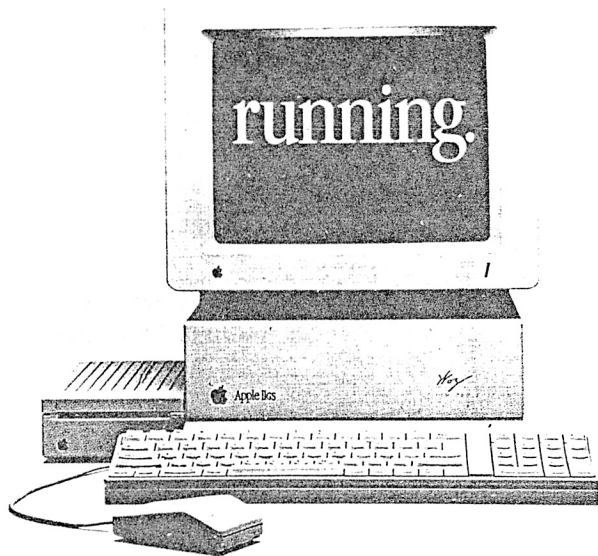
```

```

4570 *-----
4580 .PH $2029
4590 *-----
4600 * Display string P2 at P0,P1 after "xx. "
4610 * where xx is decimal form of (LAST.LINE.NUMBER.OF.MENU)
4620 * LAST.LINE.NUMBER.OF.MENU is incremented by this routine
4630 * Stores 3 bytes in MENU.TABLE:
4640 * column, line, and string length
4650 * MENU.TABLE has room for 31 such entries.
4660 *-----
4670 DISPLAY.MENU.LINE
2029- 20 AE 18 4680 JSR GET.4.PARMS Get P0-P3
202C- EE 13 0F 4690 INC LAST.LINE.NUMBER.OF.MENU
202F- A9 05 4700 LDA #05 "GO TO XY" TOKEN
2031- 8D 00 0A 4710 STA STR.A00
2034- AD 13 0F 4720 LDA LAST.LINE.NUMBER.OF.MENU
2037- 20 9D 17 4730 JSR CONVERT.A.TO.RJBF.STRING
203A- 8E 03 0A 4740 STX STR.A00+3 blank or first digit of ##
203D- 8C 04 0A 4750 STY STR.A00+4 units digit of ##
2040- A9 2E 4760 LDA #'
2042- 8D 05 0A 4770 STA STR.A00+5 string is "##. "
2045- A9 20 4780 LDA #'
2047- 8D 06 0A 4790 STA STR.A00+6
204A- 8D 07 0A 4800 STA STR.A00+7
204D- AD 13 0F 4810 LDA LAST.LINE.NUMBER.OF.MENU
2050- 0A 4820 ASL multiply by 3
2051- 6D 13 0F 4830 ADC LAST.LINE.NUMBER.OF.MENU
2054- AA 4840 TAX save for index into MENU.TABLE
4850 *---Column number-----
2055- A5 9A 4860 LDA P0 Get specified column
2057- 10 06 4870 BPL .1 ...it is absolute
2059- 29 7F 4880 AND #$7F ...it is relative
205B- 18 4890 CLC
205C- 6D BE 0F 4900 ADC MENU.CORNER.LEFT
205F- 8D 01 0A 4910 STA STR.A00+1 Put column # into string
2062- 9D B6 0E 4920 STA MENU.TABLE,X and save in table
4930 *---Line number-----
2065- A5 9B 4940 LDA P1 Get specified line number
2067- 10 06 4950 BPL .2 ...it is absolute
2069- 29 7F 4960 AND #$7F ...it is relative
206B- 18 4970 CLC
206C- 6D BF 0F 4980 ADC MENU.CORNER.TOP
206F- 8D 02 0A 4990 STA STR.A00+2 Put line # into string
2072- 9D B7 0E 5000 STA MENU.TABLE+1,X and save in table
5010 *---Append text of menu line-----
2075- A0 00 5020 LDY #0
2077- B1 9C 5030 LDA (P2),Y Save length of string in table
2079- 9D B8 0E 5040 STA MENU.TABLE+2,X
207C- A8 5050 TAY Point at last char in string
207D- 18 5060 CLC Compute length of combined strings
207E- 69 08 5070 ADC #8 05.xx.yy "##. " = 8 chars
2080- AA 5080 TAX Point at last char of combination
2081- 48 5090 PHA Save total length for display subr.
2082- B1 9C 5100 LDA (P2),Y get next char of caller's string
2084- 9D FF 09 5110 STA STR.A00-1,X append to ours
2087- CA 5120 DEX
2088- 88 5130 DEY
2089- D0 F7 5140 BNE .3 ...more to copy
5150 *---Display the string-----
208B- 20 A9 1E 5160 JSR POINT.PSTR.AT.STR.A00
208E- 68 5170 PLA Get length off stack
208F- 20 D1 14 5180 JSR DISPLAY.STRING
2092- 60 5190 RTS
5200 *-----
5210 .PH $146A
5220 MENU.MSG >MSG "Type number, or use arrows, then press Return
5230 *-----
146A-

```

In about the time it takes to read this headline, you can have the Finder up and



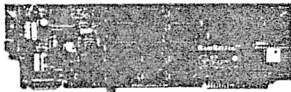
Now your favorite program can be ready to go seconds after you flip your Apple IIgs on.

With Applied Engineering's RamKeeper™ card, your IIgs retains stored programs and data when you turn your computer off.

RamKeeper powers up to two memory cards simultaneously when your Apple IIgs is off. And battery backup keeps power to the boards even during power failures. Your programs and data are stored in a permanent, "electronic hard disk," always ready to run.

Superior power backup.

Applied Engineering has the most experience in battery-



RamKeeper lets you keep programs and data in permanent, "electronic hard disk" memory. Turn your Apple IIgs on and you're ready to work.

backed memory for Apple computers. We were the first to offer battery-backed memory with our RamFactor™/RamCharger™ combination. Now RamKeeper sets the standard for IIgs memory backup.

Our experience shows in the way we designed and built RamKeeper. We used sealed Gel/

Cell batteries — far more reliable than Ni-Cads in this application. Ni-Cads lose much of their capacity if they're not discharged periodically. Just when you need them most, Ni-Cads could run out of power.

Our Gel/Cell pack, which is included in our price, gives you up to six hours of total power failure backup. That's about 6 times longer than other systems.

RamKeeper uses a Switching Power Supply — the same technology used by Apple for the IIgs power supply. This design uses energy much more efficiently to keep your Apple running cooler.

Our sealed Gel/Cell battery

stays outside your computer case. With other systems, the batteries are installed under the IIgs power supply where a leak could ruin computer circuitry.

Put two memory boards in the same slot.

You might have bought your IIgs with Apple's memory card. But now you want the features of Applied's gs-RAM™ card. RamKeeper efficiently resolves the dilemma.

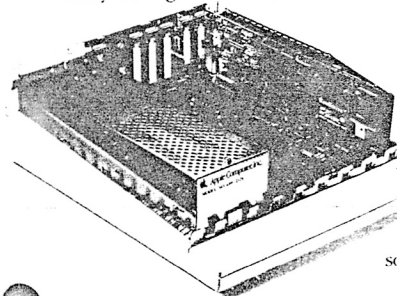
You can use RamKeeper with your current IIgs memory card and add another memory card — all in the same slot. Just attach your current memory card to one side of your new RamKeeper card, connect the second card to the other side and plug RamKeeper into the slot.

Of course RamKeeper works fine with just one memory card. But you can use two and still keep Slot 7 open with our optional Slot-Mover.

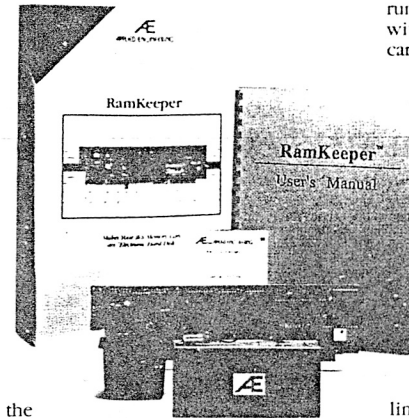
Makes all of your memory usable memory.

RamKeeper can power up to 16 Meg of memory. Other systems are limited to only 8 Meg. In addition, RamKeeper lets you mix and match different types of cards. For example, you can have a gs-RAM Plus™ using 1 Meg RAM chips and an Apple card using 256K RAM chips. Other systems are limited in the combinations they allow.

RamKeeper firmware automatically configures for two cards



RamKeeper is easy to install. Just plug it in. Even when you use two memory boards, you don't have jumpers. You can have two memory boards but use only one slot.



It all comes with RamKeeper ... board, Gel/Cell battery pack, easy-to-understand instructions, and Applied's powerful AppleWorks Expander software.

when the second card is installed. Other systems make you manually move jumpers.

RamKeeper configures memory linearly. Other systems don't, so they create memory gaps that can cause program crashes or keep some programs from using as much as half of your memory.

You easily decide how much memory you'll devote to ROM and to RAM from the IIgs Desk Accessories menu. You can configure Kilobytes or Megabytes of instant ROM storage for your favorite programs. And you can change ROM size any time without affecting stored files.

Protected from program crashes.

RamKeeper controlling firmware is in an EPROM. A program crash can't take out the operating software. With other systems, operating software is installed in RAM from a floppy. If the program crashes, it can take the operating software with it; and reinstalling the disk-based operating software destroys data in memory.

Verifies data security.

RamKeeper firmware uses optional startup checksums to verify that no data has been lost while power was off. The firmware also

runs ROM and RAM memory tests without disturbing data on the card.

Free AppleWorks Enhancement software.

Applied's powerful AppleWorks Enhancement software is free with RamKeeper. It makes AppleWorks faster and far more powerful by eliminating AppleWorks internal memory limits. Word processor limits go from only 7,250 lines to 22,600 lines. Database limits go from 6,350 records to 22,600 records. The clipboard size

limit is increased from 255 to 2,042 lines. It even automatically segments large files so you can save them on multiple floppies. No other company expands your IIgs' AppleWorks internal limits.

In addition, the most powerful disk-caching program available comes with the RamKeeper. The cache significantly increases access time to the Apple 3.5 Drive. Most applications will run up to 7 times faster.

The largest maker of Apple expansion boards.

Applied Engineering has sold more expansion boards than anyone else. And we've been in business 8 years, long enough to see the vast majority of our competitors come and go.

All of our products are crafted in the U.S.A. We back RamKeeper with a five year parts and labor warranty. And a 15-day, no questions asked, money back guarantee.

Only \$189.00.

See your dealer. Or call 214-241-6060, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA, C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside U.S.A. Prices subject to change without notice.

AE APPLIED ENGINEERING™
The Apple enhancement experts.
P.O. Box 5100 • Carrollton, Texas 75011.
(214) 241-6060

RamKeeper, RamFactor, gs-RAM, and gs-RAM Plus are trademarks of Applied Engineering.

ProDOS File Transformer.....Bob Sander-Cederlof

Have you ever wanted to convert a file from one filetype to another? I have, and it seems that there should be a command in BASIC.SYSTEM to allow it. Some other command shells, such as Davex by DAL Systems, do have such a command.

Of course, the flexibility of the BLOAD and BSAVE commands does allow you to get the job done. You can BLOAD the source file, specifying the filetype and load address; CREATE an empty file of the new type; and BSAVE into that new file, specifying the filetype, beginning RAM address, and length.

But what if you want to go just a little further, and also make some slight changes to the file's contents? Then you need an intelligent file transformer. You need a program that will ask for an input and an output pathname, read the input file and transform the data appropriately, and write the transformed data on the output file. This article presents a specific file transformer, and you may either use it as is or modify it to your own needs.

I have often been asked, by programmers who did not own the S-C Macro Assembler, for a program which would convert S-C source code files into ordinary text files. Back in the December, 1983, issue of Apple Assembly Line I published such a program, but it was written in Applesoft for running under DOS 3.3. I don't believe that program would operate properly under ProDOS, but even if it did it would be terribly slow.

Under DOS 3.3, S-C source code is stored in type "I" files. DOS does all the actual LOADING and SAVEing, being fooled into believing that the source code is Integer BASIC. Under ProDOS, S-C source code is stored in type \$FA files, which Apple has set aside for Integer BASIC source programs. Since Integer BASIC has never been supported under ProDOS, and probably never will be, I chose to use this file type for S-C source code. In fact, when you are using the S-C Macro Assembler this filetype is called "S-C".

The source code text is stored in a slightly compressed format in these files. The same format is used for both the DOS and ProDOS versions, so that you can use Apple's CONVERT program or a utility like COPY II PLUS to move files from one operating system to the other. If you were to read a file byte-by-byte under both operating systems, you would notice one difference: under DOS the file's length is stored in the first two bytes; under ProDOS the length is kept in the directory.

Each line of S-C source code begins with a line number. This number may be any value from 0 to 65535. The transformer should write the source lines on the output text file without this line number, because most assemblers using text files for source do not use explicit line numbers.

Each line of S-C source code is stored in the file in a form first defined by Steve Wozniak's Integer BASIC:

byte 1: number of bytes in entire line (n)
byte 2: lo-byte of line number in binary
byte 3: hi-byte of line number in binary
bytes 4...n-1: tokenized form of source line
byte n: 00

For example, an empty line with line number 1000 would be stored as:

04 E8 03 00

The tokenized form of S-C source is almost plain ASCII. Each byte is stored in ASCII with bit 7 zero, with two exceptions. First, blanks are stored in a compressed form. One blank is stored with the code \$81; two blanks are stored with the single code \$82; in general, a string of n consecutive blanks is stored with the single code \$80+n. The largest compressed blank code is \$BF, which stands for 63 consecutive blanks. If there are more than 63 blanks in a row, they will be represented by two or more compressed-blank codes. For example, the line

1010 STAR LDA #0 Zero

would be stored as:

14 number of bytes in line = 20
F2 03 line number is \$03F2 = 1010
53 54 41 52 "STAR"
83 three blanks
4C 44 41 81 "LDA "
23 30 85 "#0 "
5A 65 72 6F "Zero"
00 end of line token

While blank is the most frequently repeated character in a source program, there are others. The token \$C0, followed by a repetition count and an ASCII character, represents any string of the same character. For example, the line

1020 *-----

would be stored as:

08 8 bytes in line
FC 03 line number 1020
2A "*"
C0 20 2D means 32 consecutive "--"
00 end of line

With the above information, you can see that the file transformer will have to read in the first byte of each line, which tells how many bytes are in the rest of that line. Then it should read in the rest of that line. Once the entire line is in RAM, the transformer should scan through the bytes of the

line, copying ASCII characters to the new line, and expanding any compressed characters into the new line. Finally, the new line should be written on the output file. When the transformer reaches the end of the input file it is finished.

Let me insert here a reminder that even though this transformer is very specific, you should be able to easily modify it to handle other types of transformations. For example, the subscriber data base I use to produce mailing labels every month is kept on a series of plain text files, understandable only to the mailing list program I wrote about eight years ago. Even though it is plain text, AppleWorks cannot read it into a structured database unless I make some "transformations" first. At a minimum, I need a transformer that will split the city, state, and zip code line into three separate lines. I could easily write such a transformer by modifying the following program a little. Another transformer could read a binary file and write out a text file in the Intel or Motorola hex format, for later transmission through a serial port to an EPROM programmer. The possibilities are endless.

Now back to the program ALREADY written. I wrote two versions, and you can assemble either one of them by changing line 1060. As shown here, the fancier version is selected. The simpler version assumes specific filenames assembled into the code in lines 4340-4480. The fancier version prompts for filenames or pathnames to be typed in when you run the program. The code for both versions is listed, but only the fancy version was actually assembled.

Lines 1200-1400 define four macros I used in the program. The first one is for calling the ProDOS Machine Language Interface, or MLI. The second one creates a call to a subroutine which prints 00-terminated strings, followed by the string to be printed. The third generates a call to a subroutine which prints strings which begin with a byte count. The fourth generates calls to a subroutine which reads a line from the keyboard into a specified buffer, and then stores the length in the first byte of the buffer.

It turns out I do not use both the PRSTR and RDSTR macros, just one or the other. In the simple version, PRSTR is used to display the pre-assembled filenames; in the fancy version, RDSTR is used to let you type in the filenames.

Lines 1450-1550 call on two subroutines to open the input and output files. If they are successful, the file reference numbers returned by MLI will be stored into parameter blocks for reading and writing those files. If not, the transformer program will just close all files and end. I'll discuss the two opening subroutines later.

I decided after a few tests that I wanted some sort of progress indication on the screen while the program was busy. I decided to list the line number on the screen. Line 1570 uses the PRINT macro to display "LINE NUMBER: ". Inside the main loop, at line 1680, I call DISPLAY.LINE.NUMBER to display the current line number in five-digit decimal, and then backspace 5 times.

This makes a very attractive (to me) indicator. In a more general transformer, you might change this to display a hexadecimal file position, or some other meaningful parameter.

The main loop runs from line 1590 to line 1790. First lines 1590-1620 try to read three bytes. If there is any error, I assume it is due to reaching the end of the source file, and that ends the loop. If no error, then the input buffer contains the byte count for the whole line, and the line number. Since we already read the first three bytes, lines 1630-1670 compute the number of bytes left in the line and set up the parameter block to read that many. Then I print out the line number as described above. The DISPLAY.LINE.NUMBER subroutine also checks to see if any key has been pressed on the keyboard, which is interpreted to mean you want to abort the transformer. Line 1690 branches in that case, and the files are closed. Lines 1700-1710 read the rest of the source line.

Line 1730 calls the CONVERT.ONE.LINE subroutine, which scans the source line and builds an expanded output line. Lines 1740-1780 write the expanded line on the output file. Any error returned by MLI from this write ends the main loop. Probably I should have printed out an error message for this condition, because it is abnormal. It could happen if you were trying to write into a write-protected file or on a write-protected disk. Looking at it now, I think I would change line 1780 to "BCS .4", and insert the following lines:

```
1862 .4 JSR OPEN.ERROR
1864 >PRINT "\UNABLE TO WRITE ON OUTPUT FILE"
1866 RTS
```

Once the main loop ends, lines 1800-1820 truncate the output file. It may be that your output file already existed, and was longer than necessary to contain the new information. These lines chop off any extra data. Line 1810 reads the current file position, and line 1820 sets that value as the new end-of-file. Finally, line 1840 closes both files.

I ended the program with a simple RTS. You might want a fancier ending. For example, you might want a message on the screen saying "I AM FINISHED" or the like. Then you might want a short menu on the screen allowing a choice of transforming another file, ending with an RTS, or doing a ProDOS QUIT call. See, I left something for you to do!

If you do start adding features, you might also like to add the ability to transform a range of lines from the source file, by specifying a beginning and ending line number. You might want to add tests for the correct file types on the input and output files. You might want to add a menu-style pathname entry, so that you could work with complicated directory structures without a photographic memory.

That last suggestion leads me to warn that the program given here requires that you enter complete pathnames, or else have a legitimate prefix set. It does not allow slot and drive specification using ",Sx" or ",Dx" either. Now don't let me

discourage you by listing all the things I left out. As is, the program is quite useful.

Lines 1870-1970 are the subroutine to open the input file. It first prints the prompt message "INPUT: ", and then waits for you type in a pathname. (If you selected the "simple" version, it instead prints out the pre-assembled pathname.) Line 1950 calls on ProDOS to open the file. If there is any error, lines 1990-2050 print out the error number and the subroutine returns with carry set. If there is no error, the subroutine returns with carry clear. You might want to add some code to this subroutine to read the filetype and make sure it is type \$FA.

Lines 2070-2270 open the output file. This is a little more complicated, because the output file may or may not already exist. If the file does not yet exist, the open call will return with error number \$46. Line 2160 tests for this error number. Any other error will be printed out, and the subroutine will return with carry set. Lines 2180-2260 attempt to CREATE a file. First the current date and time are read and stored in the parameter block, and then ProDOS is called to create the file. The parameter block used here assumes you want the output file to be type TXT (\$04). If you modify the program for a different transformation you may want to change the file type. You might also want to add some code to check an existing file for the correct file type.

Lines 2290-2510 are the parameter blocks for the various MLI calls I used.

Lines 2560-2830 display the current line number and test for a keypress. The subroutine converts the binary line number to decimal one digit at a time. Nothing spectacular, and in fact it is very similar to the subroutine Woz used inside Integer BASIC over ten years ago. Lines 2740-2780 send out five backspaces to put the cursor back over the first digit. I originally tried to do this by just storing a cursor position back in location \$24, but that only works in 40-column mode. Using backspaces it also works in 80-column mode, even with the 80-column cards used in Apple II+ machines. Lines 2790-2830 check for a keypress and return carry set if there was one.

Finally to the heart of this transformer: lines 2870-3330 convert one input line into one output line. Lines 2950-2970 zero two pointers, one for the input line scan and the other for the output line fill. These are used by the GET.CHAR and PUT.CHAR subroutines in lines 3180-3330. GET.CHAR picks up the character pointed to from the input line, and advances the pointer. PUT.CHAR stores the character in the A-register into the output line where the pointer points, and then advances the pointer. PUT.CHAR.MULTIPLE uses the X-register to store multiple copies of the character in the A-register.

Lines 2980-3000 pick up the next character from the input line, and branch according to its type. If the character is 00, this is the end of the line: then lines 3150 tack an ASCII <RETURN> code on the end of the output line, and the subroutine is finished. If the character is positive, it is simple ASCII and

lines 3010 simply copy it to the output line. If the character is negative, it is either a compressed blank string or a compressed string of some other character. Lines 3030-3090 handle compressed blanks by putting the blank count into the X-register, a blank in the A-register, and calling PUT.CHAR.MULTIPLE. Lines 3100-3130 handle the other kind of repeated character by reading the repeat count and character code from the input line and then calling PUT.CHAR.MULTIPLE.

Lines 3340-3550 are very similar in function to the PRINT subroutine I published last month in my SHOW INDEX program. A difference is the use of the "\" character in the string. Last month's program would have merely printed the "\". The program this month will print a <RETURN> when it sees a "\". This enabled me to use the PRINT macro to generate calls to the PRINT subroutine.

Lines 3560-3830 are a subroutine for printing strings which begin with a length byte. This subroutine is not ever called in the fancy version of my program given here. In the simple version it would be called to print the pre-assembled pathnames. I went ahead and assembled the subroutine anyway, because it is an interesting one. The macro PRSTR will call it, putting the address of the string to be printed as data immediately after the JSR PRSTR.

The subroutine used in the fancy version to read in a pathname is given in lines 3870-4250. The address of the buffer for receiving the pathname is given as data following the JSR RDSTR. Lines 3880-4010 accomplish the task of copying this address into the code below, into lines 4150 and 4240. Where the listing shows \$3333, the address of the buffer will be stored. The loop in lines 4030-4210 keeps reading characters, storing them into that buffer, and displaying them on the screen.

If a backspace is read, the character at the end of the buffer is backed out and the string backspace-space-backspace is displayed on the screen. When a <RETURN> is read, the loop terminates and the number of bytes before the <RETURN> is stored in the first byte of the buffer by lines 4200-4250.

The subroutine GET.VIA.PNTR in lines 4270-4320 is called by both PRSTR and RDSTR to pickup the address which follows the JSR calling those subroutines.

Lines 4340-4480 either assemble two 65-byte buffers (fancy version) or two pathnames. Lines 4490 to the end assemble the other buffers used. Since the buffers used by ProDOS for the open files must begin on a page boundary, line 4530 skips ahead to the next page boundary.

For those of you who do not own the S-C Macro Assembler, I will start including this file transformer in executable form on future issues of the AAL Monthly Disk. Remember, you can save yourself a lot of typing by subscribing to the Monthly Disk. These disks include all of the source code in S-C format and also the text of all of the articles.

1000 *SAVE SC.2.TEXT

```

1020 ***line missing above was ".LIST MOFF,CON"
1030 *-----
1040 .OR $2000
1050 *-----
1060 FANCY .EQ 1 =0 to use pre-assembled file names
1070 *-----
1080 RDKEY .EQ $FDOC A few handy Monitor subroutines
1090 CROUT .EQ $FD8E
1100 PRBYTE .EQ $FD8E
1110 COUT .EQ $FDED
1120 *-----
1130 MLI.DATE .EQ $BF90 thru BF93 System DATE and TIME
1140 *-----
1150 KEYBOARD .EQ $C000 For aborting a conversion run
1160 STROBE .EQ $C010
1170 *-----
1180 PIN .EQ $00 Scanning pointer for input line
1190 POUT .EQ $01 Stuffing pointer for output line
1200 *-----
1210 .MA MLI
1220 JSR $BF00
1230 .DA ##1,1,2
1240 .EM
1250 *-----
1260 .MA PRINT
1270 JSR PRINT Print 00-term'd string after
1280 .AS -"1" here is the string
1290 .HS 00 here is the 00-terminator
1300 .EM
1310 *-----
1320 .MA PRSTR
1330 JSR PRSTR Print string beginning with byte count
1340 .DA 1 address of string
1350 .EM
1360 *-----
1370 .MA RDSTR Read a string from keyboard or EXEC
1380 JSR READ.STRING
1390 .DA 1 address of string
1400 .EM
1410 *-----
1420 * Program to read an S-C Macro source file
1430 * and write it as a text file.
1440 *-----
1450 FILE.TRANSFORMER
1460 JSR OPEN.INPUT.FILE Open Source File
2000- 20 78 20 1470 BCS .3 ...unable to open it
2003- B0 69 1480 LDA IREF Get RefNum for READ
2005- AD F4 20 1490 STA IOB.READ+1
2008- 8D FC 20 1500 *---Open the text file-----
200B- 20 A8 20 1510 JSR OPEN.OUTPUT.FILE
200E- B0 5E 1520 BCS .3 ...unable to open it
2010- AD FA 20 1530 LDA OREF Get RefNum for WRITE
2013- 8D 0C 21 1540 STA IOB.EOF+1 and for Truncation
2016- 8D 04 21 1550 STA IOB.WRITE+1
1560 *---Print "LINE NUMBER"-----
2019- 1570 >PRINT "\LINE NUMBER: " for progress indicator
1580 *---read byte count from source file
202C- A9 03 1590 .1 LDA #3 read 3 bytes: byte count + line #
202E- 8D FF 20 1600 STA IOB.READ+4
2031- 1610 >MLI CA,IOB.READ
2037- B0 29 1620 BCS .2 END OF FILE
1630 *---read rest of line from source file
2039- 38 1640 SEC rest of line is 3 less
203A- AD D6 22 1650 LDA LINE.SC than byte count
203D- E9 03 1660 SBC #3
203F- 8D FF 20 1670 STA IOB.READ+4
2042- 20 12 21 1680 JSR DISPLAY.LINE.NUMBER to indicate progress
2045- B0 1B 1690 BCS .2 ...wants to abort
2047- 1700 >MLI CA,IOB.READ
204D- B0 13 1710 BCS .2 END OF FILE
1720 *---Build Output Line-----
204F- 20 5A 21 1730 JSR CONVERT.ONE.LINE
1740 *---write line on text file-----
2052- A5 01 1750 LDA POUT
2054- 8D 07 21 1760 STA IOB.WRITE+4 # BYTES TO WRITE
2057- 1770 >MLI CB,IOB.WRITE
205D- B0 03 1780 BCS .2
205F- 4C 2C 20 1790 JMP .1

```

```

1800 *---truncate text file-----
1810 .2 >MLI CF,IOB.EOF GET MARK
2062- 1820 >MLI DO,IOB.EOF SET EOF
2068- 1830 *---Close both files-----
206E- 1840 .3 >MLI CC,IOB.CLOSE
2074- 20 8E FD 1850 JSR CROUT
2077- 60 1860 RTS
1870 *-----
1880 OPEN.INPUT.FILE
2078- 1890 >PRINT "\ INPUT: "
1900 .DO FANCY
2086- 1910 >RDSTR PATHI
1920 .ELSE
1930 >PRSTR PATHI
1940 .FIN
1950 >MLI C8,IOB.OPENI
208B- 1960 BCS OPEN.ERROR
2091- B0 01 1970 RTS
2093- 60
1980 *-----
1990 OPEN.ERROR
2094- 48 2000 PHA
2095- 68 2010 >PRINT "\ERROR: "
20A2- 20 DA FD 2020 PLA
20A3- 20 DA FD 2030 JSR PRBYTE
20A6- 38 2040 SEC
20A7- 60 2050 RTS
2060 *-----
2070 OPEN.OUTPUT.FILE
20A8- 2080 >PRINT "\OUTPUT: "
2090 .DO FANCY
20B6- 2100 >RDSTR PATHO
2110 .ELSE
2120 >PRSTR PATHO
2130 .FIN
2140 .1 >MLI C8,IOB.OPENO
20BB- 2150 BCC .3
20C1- 90 1F 2160 CMP #46 was it FILE NOT FOUND?
20C3- C9 46 2170 BNE OPEN.ERROR
20C5- D0 CD 2180 >MLI 82,0
20C7- 2190 LDY #3
20CD- A0 03 2200 .2 LDA MLI.DATE,Y
20CF- B9 90 BF 2210 STA IOB.CREATE+8,Y
20D2- 99 EB 20 2220 DEY
20D5- 88 2230 BPL .2
20D6- 10 F7 2240 >MLI C0,IOB.CREATE
20D8- 2250 BCC .1
20DE- 90 DB 2260 BCS OPEN.ERROR
20E0- B0 B2 2270 .3 RTS
20E2- 60 2280 *-----
20E3- 07 2290 IOB.CREATE .HS 07
20E4- 95 22 2300 .DA PATHO
20E6- C3 04 00
20E9- 00 01 00
20EC- 00 00 00 2310 .HS C3.04.0000.01.0000.0000
2320 *-----
20EF- 03 2330 IOB.OPENI .HS 03
20F0- 54 22 2340 .DA PATHI
20F2- 00 25 2350 .DA BUF.I
20F4- 2360 IREF .BS 1
2370 *-----
20F5- 03 2380 IOB.OPENO .HS 03
20F6- 95 22 2390 .DA PATHO
20F8- 00 29 2400 .DA BUF.O
20FA- 2410 OREF .BS 1
2420 *-----
20FB- 04 00 2430 IOB.READ .HS 04.00
20FD- D6 22 2440 .DA LINE.SC
20FF- 00 00 2450 .DA 0
2101- 00 00 2460 .DA 0
2470 *-----
2103- 04 00 2480 IOB.WRITE .HS 04.00
2105- D6 23 2490 .DA LINE.TEXT
2107- 00 00 2500 .DA 0
2109- 00 00 2510 .DA 0
2520 *-----
210B- 02 00 00
210E- 00 00 2530 IOB.EOF .HS 02.00.00.00.00
2110- 01 00 2540 IOB.CLOSE .HS 01.00

```

```

2550 -----
2560 DISPLAY.LINE.NUMBER
2570 LDY #4
2114- A2 B0 2580 .1 LDX #10
2116- AD D7 22 2590 .2 LDA LINE.SC+1
2119- D9 50 21 2600 CMP DECTEL,Y
2110- AD D8 22 2610 LDA LINE.SC+2
211F- F9 55 21 2620 SBC DECTBH,Y
2122- 90 10 2630 BCC .3
2124- E8 2640 INX
2125- 8D D8 22 2650 STA LINE.SC+2
2128- AD D7 22 2660 LDA LINE.SC+1
212B- F9 50 21 2670 SBC DECTEL,Y
212E- 8D D7 22 2680 STA LINE.SC+1
2131- 4C 16 21 2690 JMP .2
2134- 8A 2700 .3 TXA
2135- 20 ED FD 2710 JSR COUT
2138- 88 2720 DEY
2139- 10 D9 2730 BPL .1
213B- A0 05 2740 LDY #5
213D- A9 88 2750 LDA #88
213F- 20 ED FD 2760 .4 JSR COUT
2142- 88 2770 DEY
2143- D0 FA 2780 BNE .4
2145- AD 00 CO 2790 LDA KEYBOARD
2148- C9 80 2800 CMP #80 SET CARRY IF ANY KEY
214A- 90 03 2810 BCC .5
214C- 8D 10 CO 2820 STA STROBE
214F- 60 2830 .5 RTS
2840 -----
2150- 01 0A 64 2850 DECTEL .DA #1,#10,#100,#1000,#10000
2153- E8 10
2155- 00 00 00
2158- 03 27 2860 DECTBH .DA /1,/10,/100,/1000,/10000
2870 -----
2880 #---build output line
2890 #--- ignore line numbers
2900 #--- expand blanks and multiples
2910 #--- use low ascii
2920 #--- end with OD (return)
2930 -----
2940 CONVERT.ONE.LINE
2950 LDY #0
215C- 84 01 2960 STY POUT
215E- 84 00 2970 STY PIN
2160- 20 8E 21 2980 .1 JSR GET.CHAR ...End of Line
2163- F0 24 2990 BEQ .4 ...blanks or other multiple
2165- 30 06 3000 BMI .2 ...simple ASCII char
2167- 20 96 21 3010 JSR PUT.CHAR
216A- 4C 60 21 3020 JMP .1
216D- C9 CO 3030 .2 CMP #CO ...other multiple
216F- B0 0B 3040 BCS .3
2171- 29 7F 3050 AND #7F
2173- AA 3060 TAX
2174- A9 20 3070 LDA #20
2176- 20 98 21 3080 JSR PUT.CHAR.MULTIPLE
2179- 4C 60 21 3090 JMP .1
217C- 20 8E 21 3100 .3 JSR GET.CHAR get count
217F- AA 3110 TAX
2180- 20 8E 21 3120 JSR GET.CHAR get repeated char
2183- 20 98 21 3130 JSR PUT.CHAR.MULTIPLE
2186- 4C 60 21 3140 JMP .1
2189- A9 OD 3150 .4 LDA #OD
218B- 4C 96 21 3160 JMP PUT.CHAR
3170 -----
3180 GET.CHAR
3190 LDY PIN
218E- A4 00 3200 INC PIN
2190- E6 00 3210 LDA LINE.SC,Y
2192- B9 D6 22 3220 RTS
3230 -----
3240 PUT.CHAR
3250 LDX #1
2196- A2 01 3260 PUT.CHAR.MULTIPLE
3270 LDY POUT
2198- A4 01 3280 .1 STA LINE.TEXT,Y
219A- 99 D6 23 3290 INY
219D- C8 3300 DEX
219E- CA 3310 BNE .1
219F- D0 F9 3320 STY POUT
21A1- 84 01 3330 RTS
21A3- 60 3340

```

```

3350 PRINT
3360 PLA * POP RETURN ADDRESS
21A4- 68 3370 STA PNTR+1 BECAUSE IT POINTS TO STRING
21A5- 8D 51 22 3380 PLA
21A8- 68 3390 STA PNTR+2
21A9- 8D 52 22 3390 JSR PRINT.VIA.PNTR
21AC- 20 C3 21 3400 LDA PNTR+2 PUT RETURN ADDRESS ON STACK
21AF- AD 52 22 3410 PHA
21B2- 48 3420 LDA PNTR+1
21B3- AD 51 22 3430 PHA
21B6- 48 3440 RTS
21B7- 60 3450
3460 -----
21B8- 09 80 3470 PVP ORA #80
21BA- C9 DC 3480 CMP #1
21BC- D0 02 3490 BNE .1
21BE- A9 8D 3500 LDA #8D
21C0- 20 ED FD 3510 JSR COUT PRINT CHAR
3520 PRINT.VIA.PNTR
21C3- 20 48 22 3530 JSR GET.VIA.PNTR GET NEXT CHAR OF STRING
21C6- D0 F0 3540 BNE PVP 00 = END OF STRING
21C8- 60 3550 RTS
3560 -----
3570 # Print a string which begins with a byte count
3580 # JSR PRSTR
3590 # .DA address of byte count
3600 -----
3610 PRSTR
3620 PLA
21C9- 68 3630 STA PNTR+1
21CA- 8D 51 22 3640 PLA
21CD- 68 3650 STA PNTR+2
21CE- 8D 52 22 3660 JSR GET.VIA.PNTR
21D1- 20 48 22 3670 TAX
21D4- AA 3680 JSR GET.VIA.PNTR
21D5- 20 48 22 3690 TAY
21D8- A8 3700 LDA PNTR+2 PUT RETURN ADDRESS ON STACK
21D9- AD 52 22 3710 PHA
21DD- AD 51 22 3720 LDA PNTR+1
21E0- 48 3730 PHA
21E1- 8E 51 22 3740 STY PNTR+1
21E4- 8C 52 22 3750 STY PNTR+2 POINT AT STRING NOW
21E7- 20 50 22 3760 JSR PNTR GET BYTE COUNT
21EA- AA 3770 TAX
21EB- 20 48 22 3780 .1 JSR GET.VIA.PNTR GET NEXT CHAR OF STRING
21EE- 09 80 3790 ORA #80
21F0- 20 ED FD 3800 JSR COUT
21F3- CA 3810 DEX
21F4- D0 F5 3820 BNE .1
21F6- 60 3830 RTS

```

DON LANCASTER STUFF

<p style="text-align: center; font-weight: bold;">INTRODUCTION TO POSTSCRIPT</p> <p>A 65 min user group VHS video with Don Lancaster sharing many of his laser publishing and Postscript programming secrets.</p> <p>Includes curve tracing, \$5 toner refilling, the full Kroy Kolor details, page layouts, plus bunches more.</p> <p style="text-align: center; font-weight: bold;">\$39.50</p> <p style="text-align: center; font-size: small;">FREE VOICE HELPLINE</p>	<p style="text-align: center; font-weight: bold;">ASK THE GURU</p> <p>An entire set of reprints to Don Lancaster's ASK THE GURU columns, all the way back to column one. Edited and updated.</p> <p>Both Apple and desktop publishing resources are included that are not to be found elsewhere.</p> <p style="text-align: center; font-weight: bold;">\$24.50</p>	<p style="text-align: center; font-weight: bold;">APPLE IIc/IIe ABSOLUTE RESET</p> <p>Now gain absolute control over your Apple! You stop any program at any time.</p> <p>Eliminates all dropouts on your HIRES screen dumps. Gets rid of all hole blasting. For any IIc or IIe.</p> <p style="text-align: center; font-weight: bold;">\$19.50</p>	<p style="text-align: center; font-weight: bold;">POSTSCRIPT SHOW & TELL</p> <p>Unique graphics and text routines the others don't even dream of. For most any Postscript printer.</p> <p>Fully open, unlocked, and easily adaptable to your own needs. Available for Apple, PC, Mac, ST, many others.</p> <p style="text-align: center; font-weight: bold;">\$39.50</p> <p style="text-align: right; font-size: small;">VISA/MC</p>
--	--	--	--

SYNERGETICS

Box 809-SC Thatcher, AZ 85552 (602) 428-4073

```

3840 *-----*
3850 * Input a string from the keyboard (or EXEC file)
3860 *-----*
3870 READ.STRING
21F7- 68          PLA
21F8- 8D 51 22 3890 STA PNTR+1
21FB- 68          PLA
21FC- 8D 52 22 3900 STA PNTR+2
21FF- 20 48 22 3920 JSR GET.VIA.PNTR
2202- 8D 34 22 3930 STA .2+1
2205- 8D 45 22 3940 STA .3+1
2208- 20 48 22 3950 JSR GET.VIA.PNTR
220B- 8D 35 22 3960 STA .2+2
220E- 8D 46 22 3970 STA .3+2
2211- AD 52 22 3980 LDA PNTR+2      PUT RETURN ADDRESS ON STACK
2214- 48          PHA
2215- AD 51 22 4000 LDA PNTR+1
2218- 48          PHA
4020 *-----*
2219- A2 01      4030 .0 LDX #1
221B- 20 0C FD  4040 .1 JSR RDKEY
221E- 09 80      4050 ORA #80
2220- C9 88      4060 CMP #88
2222- D0 0F      4070 BNE .2
2224- CA         4080 DEX
2225- F0 F2      4090 BEQ .0
2227- 20 ED FD  4100 JSR COUT
222A- A9 A0      4110 LDA # " "
222C- 20 ED FD  4120 JSR COUT
222F- A9 88      4130 LDA #88
2231- D0 08      4140 BNE .25
2233- 9D 33 33  4150 .2 STA $3333,X
2236- E0 40      4160 CPY #64
2238- B0 E1      4170 BCS .1
223A- E8         4180 INX
223B- 20 ED FD  4190 .25 JSR COUT
223E- C9 8D      4200 CMP #8D
2240- D0 D9      4210 BNE .1
2242- CA         4220 DEX
2243- CA         4230 DEX
2244- 8E 33 33  4240 .3 STX $3333
2247- 60         4250 RTS
4260 *-----*
2248- EE 51 22  4270 GET.VIA.PNTR
224B- D0 03      4280 INC PNTR+1      BUMP POINTER TO NEXT CHAR
224D- EE 51 22  4290 BNE PNTR
2250- AD 33 33  4300 INC PNTR+1
2253- 60         4310 PNTR LDA $3333  GET NEXT CHAR OF STRING
4320 RTS
4330 *-----*
2254-          4340 .DO FANCY
          4350 PATHI .BS 65
          4360 .ELSE
          4370 PATHI .DA #PATHI.LEN
          4380 .AS "SC.2.TEXT"
          4390 PATHI.LEN .EQ *-PATHI-1
          4400 .FIN
4410 *-----*
2295-          4420 .DO FANCY
          4430 PATHO .BS 65
          4440 .ELSE
          4450 PATHO .DA #PATHO.LEN
          4460 .AS "TTT"
          4470 PATHO.LEN .EQ *-PATHO-1
          4480 .FIN
4490 *-----*
22D6-          4500 LINE.SC .BS 256
23D6-          4510 LINE.TEXT .BS 256
4520 *-----*
24D6-          4530 .BS #+255/256*256-* Pad to next page
2500-          4540 BUF.I .BS $400
2900-          4550 BUF.O .BS $400
          4560 *-----*

```

Modify CATALOG to Show All AuxTypes*.....Bob Sander-Cederlof

Apple Computer has started spreading the word that they are going to make further use of the AuxType field in the ProDOS file directory. Until now it has had three primary purposes, depending on the file type.

Binary files (type BIN, or \$06) use the AuxType field to hold the loading address. System files (type SYS, or \$FF) files do too, but since they always load at \$2000 it is never used. Applesoft files (type BAS, or \$FC) also use it for the loading address, but I don't believe it is ever used. I haven't tested it, though. If you use the S-C Macro Assembler under ProDOS, it uses a filetype called "S-C" for source code files. This is type \$FA, which ProDOS reserved for Integer BASIC, just in case. S-C used type I files under DOS, so I decided to use type INT files under ProDOS. Anyway, I put the loading address in AuxType for these files too. It is never used.

Text files use AuxType to specify the record length for random-access files; it is 0000 for sequential files.

AppleWorks files (types ADB or \$19, AWP or \$1A, and ASP or \$1B) are the most creative, using 15 of the 16 AuxType bits to modify the file name characters. Each bit corresponds to one of the file name characters, allowing AppleWorks to show and recognize the file name with lower-case letters and spaces in it. Regular ProDOS only displays upper-case letters in the names, and does not allow spaces. AppleWorks could have just stored the real ASCII codes in the filename field, but then the rest of ProDOS would not be able to access the files. By using the AuxType field, all is compatible.

Probably there are some other uses for AuxType that I do not know about. And now Apple has decided to use it to allow more than 256 different file types. It seems that 256 is not going to be enough now that we have ProDOS-16. If you are writing software for the Apple market and you need some private filetypes, you are supposed to write Apple Developer Tech Support and tell them of your needs. They in turn will assign you the appropriate file types. Since there are more requests than 256, they are starting to categorize them. For example, there may be a future file type for all word processors, with different AuxType values to distinguish various kinds of word processor files.

Apple has suggested that we start displaying the AuxType information in CATALOGs for more types than just TXT and BIN. I looked into BASIC.SYSTEM and found a patch that makes it display for all file types. I also looked into the S-C Macro Assembler's SCASM.SYSTEM for the same kind of patch.

In BASIC.SYSTEM, you need to change one byte from \$27 to \$11. The byte is at \$A513 after BASIC.SYSTEM is operating, so you could do a POKE 42259,17 to turn the feature on, and POKE 42259,39 to turn it off. If you want to make a permanent change, you can do it this way:

```

]BLOAD BASIC.SYSTEM,TSYS,A$2000
]POKE 12051,17
]BSAVE BASIC.SYSTEM,TSYS,A$2000

```

Location 12051 is \$2F13, which is where the code resides when the file is BLOADED as above.

In SCASM.SYSTEM, you need to change one byte from \$16 to \$00. The byte is near \$B000 when the S-C Macro Assembler is operating, but the exact location depends on the particular edition you have. I suggest dis-assembling starting at \$AFF0 and looking for the following code:

```

C9 04  CMP  #$04
D0 04  BNE  ...
A9 D2  LDA  #$D2
D0 06  BNE  ...
C9 06  CMP  #$06
D0 16  BNE  ...      change this "16" to "00"
A9 C1  LDA  #$C1

```

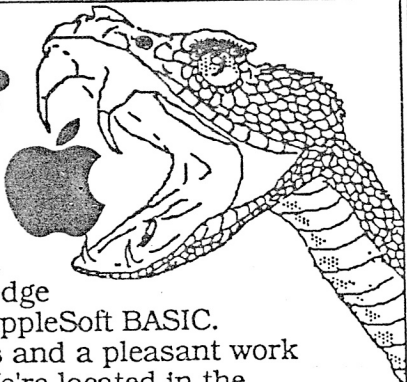
In the version I was using that "16" byte was at \$B00E. From within the assembler I typed "\$B00E:00" and that did the trick. If the "\$" commands don't work in your computer, you can go to the monitor by typing "MNT" and then make the patch.

If you want to make a permanent change, you need to BLOAD SCASM.SYSTEM, patch the byte, and BSAVE it again just like I did with BASIC.SYSTEM above. In my version, I found the byte at \$510E when the file was loaded.

Do You Have Apple Knowledge?

If you do, Applied Engineering would like to put your knowledge to work. We're looking for someone to fill a position in our Technical Support group. You must have a strong working knowledge of AppleWorks, ProDOS, DOS 3.3, and AppleSoft BASIC. Applied Engineering offers good benefits and a pleasant work environment. Office is non-smoking. We're located in the Carrollton area.

So if you've got the knowledge and want to sink your teeth into a position with an ever-expanding company, give us a call at (214) 241-6060.



BLOADing Directories.....Bob Sander-Cederlof

Did you know that ProDOS will let you BLOAD a directory just like any other kind of file? I did not until today.

For example, if I want to load the directory of my Sider hard disk into memory, I can type BLOAD /HARD1,TDIR,A\$1000. I can load in any subdirectory the same way. This works under both BASIC.SYSTEM (Applesoft) and SCASM.SYSTEM (the S-C Macro Assembler) shells.

In both cases, if you care to, you can find the length of the directory in bytes in locations \$BEDB and \$BEDC. \$BEDB will always contain 00, and \$BEDC will be the number of pages in the directory, or twice the number of blocks.

I tried BSAVEing... but it is prohibited. You get a FILE LOCKED message for your efforts.

EnterSoft:

Basic-like macros which make the complex simple. Don't re-write that multiplication routine for the hundredth time! Get EnterSoft instead! Do 8/16/32/64 bit Math/Input-Output/Graphics simply without all of the hassles. These routines are a must for the serious programmer who doesn't want to spend all of his/her time trying to re-invent the wheel. DOS 3.3 Version=\$30.00, ProDos Version=\$30.00, BOTH for only \$50.00. GET YOURS TODAY.

A Shape Table Program:

For oncel! A shape table program which is logically organized into its componet parts. Each section resides in its own program. The editor, disk access, Hi-Res section; each section is separate. Written almost entirely in Basic, it is easily modified. Not copyprotected! Put them on a Hard Disk, Ram Drive, anywhere! DOS 3.3 Version=\$20.00, ProDos Version=\$20.00, BOTH for \$30.00!

Send Check or Money Order To:	ProDos Upgrade for DOS 3.3 EnterSoft Owners = \$20.00
Mark Manning c/o Simulacron I/Baggy Game P.O. Box 58598 Webster, TX 77598	Thanks for the letters - Keep Writing!

Visiting Phoenix, continued from front page.

Bill Mensch gave several interesting talks regarding the history of the 6502 family and its future. The 65832 is on the horizon, but not fully defined. We know it will have 32-bit registers, multiply/divide instructions, and probably even floating point arithmetic. We can also be certain it will be plug compatible with the 65816, so we will be able to plop them directly into Apple IIgs sockets. We still have opportunity to send ideas for its features to Bill. Suggestion: do it in writing, be brief, keep any such letter to one or two ideas and no more than two pages. If you have more ideas, send them in separate letters, giving Bill time to digest them. Send your ideas directly to Bill at Western Design Center, 2166 East Brown Road, Mesa, AZ 85213. If you are VERY serious, give him a call at (602)962-4545.

You might also want to write for information about their new line of microCOMPUTERS: the W65C134 includes a 65C02, RAM, ROM, and oodles of I/O goodies all in one 68-pin package; the W65C265 will be even better, and built around a 65C816; and the W65C365 will include a 65C832.

A real highlight of the trip was meeting Don Lancaster, and his wife and daughter. Don was there to show and sell his many books and disks full of Apple and Postscript knowledge, and both he and wife Bea gave several seminars during the conference. If you are doing ANYTHING with laser printers, you need to look at what Don has. His "Ask the Guru" column in Computer Shopper is a gold mine, and you can get a neatly bound set of reprints for only \$24.50. Speaking of gold mines reminds me of caves, and the fact that the Lancasters are avid spelunkers.

Bill and I also enjoyed our lengthy discussions with Jeff Creamer, Steve Stephenson, and John & Ron Wrenholt. Jeff is a teacher in Prescott, AZ, and has published in these pages. Steve is chief programmer for Checkmate Technology, and gave me some wonderful disk files about AppleWorks. Expect to see some of his stuff here in the near future. John is the founder of Big Red Apple Club, now called Big Red Computer Club. He and brother Ron have quite a lot to offer any Print Shop aficionados, including a new program which lets you print labels with color graphics.

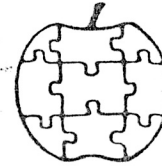
"Do not be conformed to this world, but be transformed by the renewing of your mind, that you may discover the good, and acceptable, and perfect will of God."

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$24 per year in the USA, Canada, and Mexico, or \$36 in other countries. Back issues are \$1.80 each for Volumes 1-7, \$2.40 each from later Volumes (plus postage). A subscription to the newsletter with a Monthly Disk containing all program source code and article text is \$64 per year in the USA, Canada and Mexico, and \$90 to other countries.

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)

Apple

\$2.40



Assembly

Line

Volume 8 -- Issue 8

May, 1988

About S-C Macro Cross Assemblers	3
Peeking Inside AppleWorks 1.3, Part 6	4
Klaxon Sound Effect	20
AppleWorks Segment Functions	21
More on AuxTypes in ProDOS CATALOGs	24
New Version 1.2 of BASIC.SYSTEM	25
The Apple IIx Wish-O-Gram	28

The Last Issue

With deep regret I have to inform you that this is the last issue of Apple Assembly Line. The income at S-C Software has suddenly and finally decreased beyond the point at which AAL can continue to be published. I have found employment elsewhere, and for the time being have put S-C Software to sleep. I will be working full time now as a programmer for a certain major manufacturer of Apple peripheral boards, for whom I have written a considerable amount of firmware over the last five years: Applied Engineering.

If your subscription expiration date is still in the future, then I owe you something in place of those future issues of the newsletter. I would like to repay you with materials I have on hand, such as back issues of the newsletter, copies of my software products, or perhaps books.

In order to determine how many months remain on your subscription, look at the mailing label. At the right end of the top line you will see a four-digit number, such as 8812. The first two digits are the year, the last two digits are the month of what should have been your last issue. The issue in your hands is issue 8805. Subtract 88 from the year of your expiration date, and multiply the remainder by 12; add the product to the month of your expiration date, and subtract 5; the result should be the number of issues I owe you. For example, if your expiration code is 8903, I owe you 10 more issues.

If possible, I would like to give you back issues in place of future ones. Please write me and let me know which issues you would like. Include some second choices in case I run out of some of the issues. If you had a subscription which included monthly disks, I will also include the disks with the back issues.

Of course, if you already have all of the back issues you will want to work out some other arrangement. Alternatively, you may want to select some item(s) from among the following software and hardware products and apply your remaining subscription toward the purchase price. The actual amount of your credit depends on how much you paid for your subscription; I'll trust you to work that out.

By the time you are reading this, I will be working during the day for my new employer. Please write with your request, or call and leave complete details on my answering machine at (214) 324-2050, by September 30, 1988.

S-C Macro Assembler Version 2.0 DOS \$100, ProDOS \$100
 both for \$120
 Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners \$20
 ProDOS Upgrade Kit for Version 2.0 DOS owners \$30

Cross Assemblers for owners of S-C Macro Assembler . . \$32.50 to \$50 each
 (More info on these below)

S-C DisAssembler (ProDOS only) without source code \$30
 with source code \$50

ProVIEW (ProDOS-based disk utility program) \$20

Full Screen Editor for S-C Macro (with complete source code) \$49

S-C Cross Reference Utility without source code \$20
 with source code \$50

S-C Word Processor, both DOS & ProDOS, both 40- & 80-columns,
 with complete source code \$50

DP18 and DPFP, double precision math for Applesoft,
 including complete source code. . . \$50

S-C Documentor (complete commented source code of Applesoft ROMs)
 \$50

AAL Quarterly Disks each \$15, or any four for \$45

Vinyl disk pages, 6"x8.5", hold two disks each 10 for \$6 *

Sider 20 Meg Hard Disk, includes controller & software . . . (\$695) \$550 +

65802 Microprocessor, 4 MHz (Western Design Center) \$25 *

quikLoader EPROM System (SCRG) (\$179) \$170 *
 PROMGRAMER (SCRG) (\$149.50) \$140 *

* These items add \$2 for first item, \$.75 for each additional item for shipping in USA.
 + Add \$10 for shipping in USA.
 Customers outside USA inquire for postage needed.
 Texas residents please add 8% sales tax to all orders.
 << Master Card, VISA, Discover and American Express >>

About S-C Macro Cross Assemblers.....Bob Sander-Cederlof

Combining the versatile Apple II with the S-C Macro Cross Assemblers provides a cost effective and powerful development system for many different microprocessors.

All of the S-C Macro Assemblers are all identical in operation: only the language assembled is different. Each S-C Macro Assembler is a complete macro assembler with an integrated, co-resident program editor, and operates in any member of the Apple II family having at least 48K RAM and one disk drive (ProDOS versions require 64K RAM). Each is written in 6502 assembly language for execution in Apple II series computers, but assembles standard mnemonics for the target processor into binary object code for that processor. The standard version assembles code for either 6502, normal 65C02, the Rockwell special version of the 65C02, or 65C816 microprocessors. Cross Assemblers are available for a wide variety of microprocessors.

S-C Cross Assemblers are sold as supplements to the standard S-C Macro Assembler. The S-C Macro Assembler, complete with 120-page reference manual, costs \$100 for either the DOS 3.3 or ProDOS version, or \$120 for both; once you have it, you may add as many Cross Assemblers as you wish at a much lower price. The following S-C Macro Cross Assembler versions are now available:

Microprocessor	DOS 3.3	ProDOS	Both
Motorola:			
6800,1,2,8/6301	\$50	\$50	\$70
6805	\$50	\$50	\$70
6809	\$32.50	n/a	
68HC11	\$50	\$50	\$70
68000	\$50	n/a	
Mitsubishi:			
50740 series	\$50	\$50	\$70
Intel:			
8048 family	\$32.50	n/a	
8051 family	\$32.50	n/a	
8080/8085	\$32.50	n/a	
Zilog:			
Z-80	\$32.50	n/a	
Z-8	\$32.50	n/a	
RCA:			
1802/1805	\$32.50	n/a	
DEC:			
LSI-11	\$50	n/a	
General Instruments:			
GI-1650	\$50	\$50	\$70
GI-1670	\$50	\$50	\$70
Sharp:			
LH5801	\$50	\$50	\$70

The assembled object code may be directed either to Apple memory or to a DOS 3.3 or ProDOS binary file. If you have an EPROM burner, the object can be burned into EPROMs. (We recommend and sell the SCRG PromGramer, which burns any 24- or 28-pin EPROM from 2716 through 27512. It fits in an Apple slot, and is only \$140.) Other options are to download your object code to a target system via a serial port, to download to a ROM-emulator, or to execute directly out of Apple memory with a co-processor card. Co-processor cards for the Z-80, 6809, and 68000 are available from various manufacturers.

This seems like a good time to give some sort of index to all of the pieces of AppleWorks I have discussed in the past five issues of AAL. The following table is in order by address within the code.

1000-1185	JMP vector, etc.	Mar 88, Feb 88
1186-119D	CALL.FOR.AWPROGRAM.DISK	Mar 88
119E-122C	LOAD.PROGRAM.SEGMENT.A	Mar 88
122D-1340	LOAD.SEGMENT.FROM.DISK	Mar 88
1341-1365	APPEND.STRINGS	Dec 87
1366-136D	CLEAR.MAIN.WINDOW	this issue
136E-139C	FLUSH.KEYBUF.CHECKING.ESC	this issue
139D-13B1	CLR.LINE.X.TO.LINE.Y	this issue
13B2-13C0	CHECK.KEYBUF	Feb 88
13C1-14CF	various messages	Mar 88, this issue
14D0	INVERSE.FLAG	Jan 88
14D1-153F	DISPLAY.STRING	Jan 88
1540-1543	FUN.CLR.LINE	Jan 88
1544-1549	FUN.CLR.CH.TO.EOL	Jan 88
154A-1551	FUN.HOME	Jan 88
1552-1575	FUN.CLR.CH.TO.EOS	Jan 88
1576-1592	FUN.GOTO.XY	Jan 88
1593-1598	FUN.CURSOR.LEFT	Jan 88
1599-159B	FUN.HANG.UP	Jan 88
159C-15A0	more of CURSOR.LEFT	Jan 88
15A1-15AA	FUN.CURSOR.RIGHT	Jan 88
15AB-15BB	FUN.CURSOR.UP	Jan 88
15BC-15C9	FUN.CURSOR.DOWN	Jan 88
15CA-160A	SCROLL	Jan 88
160B-160E	FUN.INVERSE	Jan 88
160F-1616	FUN.NORMAL	Jan 88
1617-1621	FUN.CORNER.BR	Jan 88
1622-1628	FUN.CURSOR.BOL	Jan 88
1629-1633	FUN.CORNER.TL	Jan 88
1634-1644	FUN.FULL.SCREEN	Jan 88
1645-165D	FUN.BEEP	Jan 88
165E-1715	FUN.SHUFFLE	Jan 88
1716-1737	BASE.CALC	Jan 88
1738-1778	CLR.CH.TO.EOL	Jan 88
1779-179C	FUN.TBL	Jan 88
179D-17D0	CONVERT.A.TO.RJBF.STRING	this issue
17D1-1814	DIVIDE.P0.BY.P2	this issue
1815-1817	Another HANG.UP	Jan 88
1818-1822	BEEP.AND.CLEAR.KEYBUF	this issue
1823-1836	MOVE.CURSOR.TO.XY	Feb 88
1837-1841	SHOW.HELP.STRING	this issue
1842-184F	WARN.IF.FULLDESK	this issue
1850-186B	SHOW.FULLDESK.WARNING	this issue
186C-1871	SHOW.COMMAND.ENTRY	this issue
1872-1879	CALL.ORGANIZER	this issue
187A-18AC	COPY.SCRN.LINE.TO.0900	Feb 88
18AD-18DC	GET.x.PARMS	Dec 87
18DD-18E3	GET.MENU.TABLE.INDEX	Apr 88
18E4-18F1	MAKE.MENU.LINE.NORMAL	Apr 88
18F2-190B	MAKE.MENU.LINE.INVERSE	Apr 88
190C-191C	RESTORE.ESCAPE.AND.HELP	Apr 88
191D-1A73	SELECT.MENU.LINE	Apr 88
1A74-1AFB	REVERSE.A.SCREEN.LINE	Apr 88

1AFC-1AFF	SET.PRODOS.BITMAP	Mar 88
1B00-1B0A	CLR.PRODOS.BITMAP	Mar 88
1B0B-1B2A	DRAW.TOP.AND.BOTTOM.LINES	this issue
1B2B-1B33	POST.CHANGE.FLAG	this issue
1B34-1B4D	MULTIPLY.X.BY.Y	this issue
1B4E-1B83	MULTIPLY.P0.BY.P2	this issue
1B84-1BAB	MOVE.BLOCK.DOWN	Dec 87
1BAC-1BDE	MOVE.BLOCK.UP	this issue
1BDF-1BF0	POP.ESCAPE.ROAD.MAP	this issue
1BF1-1C13	WAIT.FOR.SPACE.RETURN.OR.ESCAPE	this issue
1C14-1C20	variables for PRINTER	this issue
1C21-1D0D	PRINTER.DRIVER	this issue
1D0E	???	
1D0F-1D2F	PUSH.ESCAPE.ROAD.MAP	this issue
1D30-1D34	various variables	Feb 88
1D35-1D45	AW.KEYIN	Feb 88
1D46-1DDA	KEYIN.ANOTHER.CHAR	Feb 88
1DDB-1E7F	KEYIN.ANALYSIS	Feb 88
1E80-1E89	MOVE.CURSOR.TO.TCOL.TROW	this issue
1E8A-1E93	SAVE.GOTO.XY	this issue
1E94-1EA8	MAP.SCRN.CHARS.TO.INTERNAL	Feb 88
1EA9-1EB3	POINT.PSTR.AT.OA00	Feb 88
1EB4-1EBE	MAP.LOWER.TO.UPPER	this issue
1EBF-1ED8	FILTER.LC.TO.UC	Dec 87
1ED9-1EF7	COMPARE.STRINGS	Dec 87
1EF8-1F09	MOVE.STRING	Dec 87
1F0A-1F3D	READ.KEYBOARD	Feb 88
1F3E-1F9F	DISPLAY.AT	this issue
1FA0-1FA6	TRUNCATE.TO.79.IF.OVER.80	this issue
1FA7-1FD0	POLL.KEYBOARD	Dec 87
1FD1-1FDF	DELAY.TENTHS	Jan 88
1FE0-1FE8	CLEAR.KEYBUF	Feb 88
1FE9-1FF4	DISPLAY.TOKEN.X	this issue
1FF5-2028	DISPLAY.ON.LINE.23	this issue
2029-2092	DISPLAY.MENU.LINE	Apr 88

As you can see, I am trying to fill in all the gaps this month. There is still more code beyond \$2092 in the main section of AppleWorks, running all the way up to \$2E84, but I have not finished disassembling all of it yet.

DON LANCASTER STUFF

INTRODUCTION TO POSTSCRIPT

A 65 min user group VHS video with Don Lancaster sharing many of his laser publishing and Postscript programming secrets.

Includes curve tracing, \$5 toner refilling, the full Kroy Kolor details, page layouts, plus bunches more.

\$39.50

FREE VOICE HELPLINE

ASK THE GURU

An entire set of reprints to Don Lancaster's ASK THE GURU columns, all the way back to column one. Edited and updated.

Both Apple and desktop publishing resources are included that are not to be found elsewhere.

\$24.50

APPLE IIc/IIe ABSOLUTE RESET

Now gain absolute control over your Apple! You stop any program at any time.

Eliminates all dropouts on your HIRES screen dumps. Gets rid of all hole blasting. For any IIc or IIe.

\$19.50

POSTSCRIPT SHOW & TELL

Unique graphics and text routines the others don't even dream of. For most any Postscript printer.

Fully open, unlocked, and easily adaptable to your own needs. Available for Apple, PC, Mac, ST, many others.

\$39.50

VISA/MC

SYNERGETICS

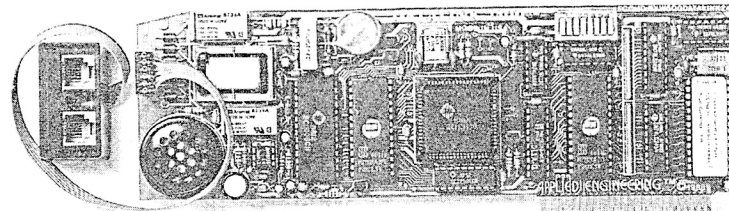
Box 809-SC

Thatcher, AZ 85552

(602) 428-4073

109D-	1030	RJBF.STRING	.EQ \$109D	Mar 88 issue
11A1-	1040	LOAD.PROGRAM.SEGMENT.A	.EQ \$11A1	Mar 88 issue
13B2-	1050	CHECK.KEYBUF	.EQ \$13B2	Feb 88 issue
14D1-	1060	DISPLAY.STRING	.EQ \$14D1	Jan 88 issue
1823-	1070	MOVE.CURSOR.TO.XY	.EQ \$1823	Feb 88 issue
18AE-	1080	GET.4.PARMS	.EQ \$18AE	Dec 87 issue
18B2-	1090	GET.2.PARMS	.EQ \$18B2	Dec 87 issue
18B4-	1100	GET.A.PARMS	.EQ \$18B4	Dec 87 issue
1B84-	1110	MOVE.BLOCK.DOWN	.EQ \$1B84	Dec 87 issue
1D35-	1120	AW.KEYIN	.EQ \$1D35	Feb 88 issue
1EA9-	1130	POINT.PSTR.AT.OA00	.EQ \$1EA9	Feb 88 issue
1EF8-	1140	MOVE.STRING	.EQ \$1EF8	Dec 87 issue
1FE0-	1150	CLEAR.KEYBUF	.EQ \$1FE0	Feb 88 issue
2093-	1160	DISPLAY.STRING.PO	.EQ \$2093	
20AE-	1170	DRAW.BOTTOM.LINE	.EQ \$20AE	
20BE-	1180	REPEAT.CHAR.Y.X.TIMES	.EQ \$20BE	
C082-	1190	RCM.D000	.EQ \$C082	
C083-	1200	RAM.D000	.EQ \$C083	
D002-	1210	MEASURE.FREE.MEMORY	.EQ \$D002	
D026-	1220	X.D026	.EQ \$D026	
D044-	1230	DISPLAY.K.AVAIL	.EQ \$D044	
D029-	1240	DISPLAY.FUNCTION.AND.ESCAPE.MAP	.EQ \$D029	
FD2D-	1250	MON.COUT	.EQ \$FD2D	
	1260	-----		
24-	1270	MON.CH	.EQ \$24	
25-	1280	MON.CV	.EQ \$25	
36-	1290	CSWL	.EQ \$36	
37-	1300	CSWH	.EQ \$37	
80-	1310	PSTR	.EQ \$80,81	
88-	1320	FLAG.FOUND.ESCAPE	.EQ \$88	
89-	1330	Z.89	.EQ \$89	
	1340	*---used by Multiply & Divide---		
91-	1350	M.REG	.EQ \$91,92	
93-	1360	MUL.X.FACTOR	.EQ \$93	
94-	1370	MUL.Y.FACTOR	.EQ \$94	
	1380	-----		
95-	1390	WINDOW.TOP	.EQ \$95	initially 2 (3rd from top)
96-	1400	WINDOW.BOTTOM	.EQ \$96	initially 21 (3rd from bottom)
98-	1410	PNTR	.EQ \$98,99	
9A-	1420	P0	.EQ \$9A	
9B-	1430	P1	.EQ \$9B	
9C-	1440	P2	.EQ \$9C	
9D-	1450	P3	.EQ \$9D	
9E-	1460	P4	.EQ \$9E	
9F-	1470	P5	.EQ \$9F	
A0-	1480	COUNT	.EQ \$A0,A1	
A1-	1490	Z.A1	.EQ \$A1	
F0-	1500	DIVISOR	.EQ \$F0,F1	
F2-	1510	REMAINDER	.EQ \$F2,F3	
	1520	-----		
0A00-	1530	STR.A	.EQ \$0A00	
0C6C-	1540	FLAG.CURRENT.FILE	.EQ \$0C6C	
0CC6-	1550	FLAG.FOUND.SPACE	.EQ \$0CC6	
0CE8-	1560	ESCAPE.ROAD.MAP	.EQ \$0CE8	nine 21-byte entries
0EA8-	1570	OPEN.PRINTER.FLAG	.EQ \$0EA8	
0EAD-	1580	TCOL	.EQ \$0EAD	
0EAE-	1590	TROW	.EQ \$0EAE	
0EB2-	1600	X.OEB2	.EQ \$0EB2	
0EB3-	1610	X.OEB3	.EQ \$0EB3	
0EB4-	1620	X.OEB4	.EQ \$0EB4	
	1630	-----		
	1640	* Following 164 bytes are copy of first 164 bytes		
	1650	* in SEG.PR file (OF19...OFBD):		
	1660	-----		
0F3D-	1670	CURRENT.PRINTER.NO	.EQ \$0F3D	Currently Active Printer # (1-3)
	1680	-----		
	1690	* Three 36-byte data areas, one for each printer		
	1700	1:	\$F51-F74	
	1710	2:	\$F75-F98	
	1720	3:	\$F99-FBC	
	1730	-----		
0F61-	1740	X.OF61	.EQ \$0F61	
0F70-	1750	X.OF70	.EQ \$0F70	
0F71-	1760	X.OF71	.EQ \$0F71	
0F73-	1770	X.OF73	.EQ \$0F73	
	1780	-----		
0FC4-	1790	FLAG.DONT.ANALYZE.KEY	.EQ \$0FC4	
0FC6-	1800	SLOTROM.MAP	.EQ \$0FC6	...FCD (8 BYTES)
0FEF-	1810	FLAG.APPLE.2C	.EQ \$0FEF	01=Apple //c, else 00.
0FF3-	1820	HIDE.FULLDESK.WARNING	.EQ \$0FF3	
0FF4-	1830	CHAR.FOR.BOTTOM.LINE	.EQ \$0FF4	
0FF5-	1840	KBYTES.DESKTOP.LEFT	.EQ \$0FF5,FF6	
	1850	-----		

The new DataLink™ 2400 modem from Applied Engineering, it's a lot more than just twice as fast.



Applied Engineering's new DataLink™ 2400. Simply put, the finest modem on the market for your Apple II, IIe or II+.

Bring home a world of information... from up to the minute flight information to whole libraries of resource materials. Even download free software and games.

Twice the speed.

At transmission speeds up to 2400 bps (bits-per-second), Applied Engineering's new DataLink 2400 is capable of putting text on the screen faster than the human eye can follow. That means you can capture a great deal more material in less time than with 1200 bps modems. And unlike other modems, the DataLink 2400 comes complete with powerful, easy-to-use communications software.

Complete communications software included.

Both our new DataLink 2400 and our DataLink 1200 modems feature AE's exclusive communications software—on disk and in ROM—everything needed to get you immediately up and running. Our powerful DataTerm software for the IIe and IIc supports VT-52 screen emulation, macros, file transfers, on-line time display, recording buffer and more. It even stores hundreds of phone numbers for auto-dialing and log on. And for II+ and 64k IIe owners, our OnLine 64 software has many of the same powerful features.

Worldwide compatibility.

The DataLink 2400 is fully compatible with Bell 103 and 212 protocols, as well as European protocol CCITT V.22 BIS, V.22 and V.21. It operates at varying transmission speeds from 0-300, 1200 and 2400 bps.

The new 2400, like our best-selling DataLink™ 1200, carries a full five year warranty and comes complete with two modular phone jacks for data and voice calls, a thoughtful feature that means fewer wires to connect. We also include an extra long telephone cable, in case your computer is across the room from your telephone jack. You can track the progress of calls either electronically or via on-board speaker. And built-in diagnostics reliably check transmission accuracy.

Prices subject to change without notice. Brands and product names are registered trademarks of their respective holders.

Packed with important features:

- Non-volatile memory for modem configuration
- Full Hayes AT compatibility
- Point-to-Point, ASCII Express, Access II compatibility, in addition to AE's included DataTerm and OnLine 64 software.
- Super Serial Card "Front End" for highest software compatibility (unlike others)
- Adaptive equalization and descrambling
- Hardware configuration for DSR and DCD
- PC Transporter (MS-DOS) compatibility
- FCC certified design

\$204.90 in freebies.

We also throw in a nice collection of goodies—a free subscription to the Genie network worth \$29.95, \$60 of free on-line time from NewsNet, a free \$50 subscription to the Official Airline Guide and a fee-waived membership to The Source worth \$49.95 plus \$15 of free on-line time. That's \$204.90 worth of free memberships, discounts and on-line time when you purchase the powerful DataLink 2400 at \$239.

DataLink 1200 reduced.

Loaded with all the features of the new 2400, (except CCITT, DSR/DCD and non-volatile ROM configurations) our 1200 bps DataLink modem, complete with software and freebies, is an affordable alternative at only \$179.

DataLink 1200..... \$179
DataLink 2400..... \$239

Order today!

To order or for more information, see your dealer or call (214) 241-6060 today, 9 am to 11 pm, 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10 outside U.S.A.

AE APPLIED ENGINEERING™
The Apple enhancement experts.

(214) 241-6060

P.O. Box 5100, Carrollton, TX 75011

```

1860 .ph $1366
1870 *-----
1880 * CLEAR WINDOW, THEN SET FULL SCREEN
1890 *-----
1900 * (1366) 100C 25E2 2710 2ACA 2D34
1910 CLEAR.MAIN.WINDOW
1920 LDX WINDOW.TOP
1930 LDY WINDOW.BOTTOM
1940 JSR CLR.LINE.X.TO.LINE.Y
1950 RTS
1960 *-----
1970 * Flush buffer, but set flags if <SPC> or <ESC> found
1980 *-----
1990 * (136E) 100F
2000 FLUSH.KEYBUF.CHECKING.ESCAPE.AND.SPACE
2010 LDA #0
2020 STA FLAG.FOUND.ESCAPE
2030 STA FLAG.FOUND.SPACE
2040 LDA #1 Signal not to analyze char in KEYIN
2050 STA FLAG.DONT.ANALYZE.KEY
2060 JSR CHECK.KEYBUF
2070 BEQ .3 ...no chars in buffer
2080 JSR AW.KEYIN
2090 CMP #$1B <ESC>?
2100 BNE .2 ...no
2110 LDA #1 ...yes, set flag and get another
2120 STA FLAG.FOUND.ESCAPE
2130 BNE .1 ...always
2140 CMP #$20 <SPACE>?
2150 BNE .1 ...no, get another
2160 LDA #1 ...yes, set flag and get another
2170 STA FLAG.FOUND.SPACE
2180 BNE .1 ...always
2190 LDA #0 now buffer is empty
2200 STA FLAG.DONT.ANALYZE.KEY Restore char-analysis in KEYIN
2210 RTS return
2220 *-----
2230 * Clear screen from line (X) to line (Y), and
2240 * set window to full screen.
2250 * (X)=top line to clear
2260 * (Y)=bottom line to clear
2270 *-----
2280 * (139D) 1012 136A
2290 CLR.LINE.X.TO.LINE.Y
2300 STX CLRSTR+4
2310 LDX #79
2320 JSR MOVE.CURSOR.TO.XY
2330 JSR DISPLAY.STRING.PO
2340 .DA CLRSTR
2350 RTS
2360 *-----
2370 CLRSTR .DA #6 6 bytes in string
2380 .HS 0C Set bottom-right corner of window
2390 .HS 05,00.00 Go to xx,yy
2400 .HS 04 Clear inside window
2410 .HS 0F Back to a full-scrn window
2420 *-----
2430 .ph $13C1
2440 *-----
2450 .MA MSG MACRO TO SHORTEN LISTING
2460 .DA #:-*-1
2470 .AS "]1"
2480 :1
2490 .EM
2500 .MA AS MACRO TO SHORTEN LISTING
2510 .AS /]1/
2520 .EM
2530 *-----
2540 MSG..1 >MSG "Place the AppleWorks PROGRAM disk in Drive 1 and
2550 MSG..2 .DA #MSG..3*-1
2560 .HS 81 Open-Apple picture press Return.
2570 >AS "-? for Help"
2580 MSG..3 .DA #MSG..4*-1
2590 >AS "Type entry or use "
2600 .HS 81 Open-Apple picture
2610 >AS " commands "
2620 MSG..4 >MSG "Press Space Bar to continue "
2630 MSG..5 >MSG "Do you really want to do this"
2640 MSG..6 >MSG "Type number, or use arrows, then press Return "
2650 MSG..7 .DA #MSG..X*-1
2660 .HS 0A inverse
2670 >AS " WARNING. Desktop is full. Action not completed. "
2680 .HS 0B normal
2690 MSG..X

```



SPECIAL !!! EXPANDED RAM/ROM BOARD: \$39.00

Similar to our \$30 RAM/ROM dev board described below. Except this board has two sockets to hold your choice of 2-2K RAM, 2-2K ROM or even 2-4K ROM for a total of 8K. Mix RAM and ROM too. Although Apple limits access to only 2K at a time, soft switches provide convenient socket selection. Hard switches control defaults.

IMPROVED !!!]I IN A MAC (ver 2.0): \$75.00

Now includes faster graphics, UniDisk support and more! Bi-directional data transfers are a snap! This Apple II emulator runs DOS 3.3/PRODOS (including 6502 machine language routines) on a 512K MAC or MACPLUS. All Apple II features are supported such as HI/LO-RES graphics, 40/80 column text, language card and joystick. Also included: clock, RAM disk, keyboard buffer, on-screen HELP, access to the desk accessories and support for 4 logical disk drives. Includes 2 MAC diskettes (with emulation, communications and utility software, plus DOS 3.3 and PRODOS system masters, including Applesoft and Integer BASIC) and 1 Apple II diskette.

SCREEN.GEN: \$35.00

Develop HI-RES screens for the Apple II on a Macintosh. Use MACPAINT (or any other application) on the MAC to create your Apple II screen. Then use SCREEN.GEN to transfer directly from the MAC to an Apple II (with SuperSerial card) or IIC. Includes Apple II diskette with transfer software plus fully commented SOURCE code.

MIDI-MAGIC for Apple //c: \$49.00

Compatible with any MIDI equipped music keyboard, synthesizer, organ or piano. Package includes a MIDI-out cable (plugs directly into modem port - no modifications required!) and 6-song demo diskette. Large selection of digitized QRS player-piano music available for 19.00 per diskette (write for catalog). MIDI-MAGIC compatible with Apple II family using Passport MIDI card (or our own input/output card w/drum sync for only \$99.00).

FONT DOWNLOADER & EDITOR: \$39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. Special functions (like expanded, compressed etc.) supported. Includes HIRES screen editor to create custom fonts and special graphics symbols. For Apple II, II+, IIe. Specify printer: Apple Imagewriter, Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/85, or Okidata 92/192.

* FONT LIBRARY DISKETTE #1: \$19.00 contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

DISASM 2.2e: \$30.00 (\$50.00 with SOURCE Code)

Use this intelligent disassembler to investigate the inner workings of Apple II machine language programs. DISASM converts machine code into meaningful, symbolic source compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even provides label substitution. Address-based triple cross reference generator included. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his ASSEMBLY COOKBOOK.

The 'PERFORMER' CARD: \$39.00 (\$59.00 with SOURCE Code)

Converts a 'dumb' parallel printer I/F card into a 'smart' one. Simple command menu. Features include perforation skip, auto page numbering with date & title, large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93.

'MIRROR' ROM: \$25.00 (\$45.00 with SOURCE Code)

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support.

RAM/ROM DEVELOPMENT BOARD: \$30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

C-PRINT For The APPLE //c: \$69.00

Connect standard parallel printers to an Apple //c serial port. Separate P/S included. Just plug in and print!

Unless otherwise specified, all Apple II diskettes are standard (not copy protected!) 3.3 DOS.

Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders OK.

RAK-WARE 41 Ralph Road W. Orange NJ 07052 (201) 325-1885



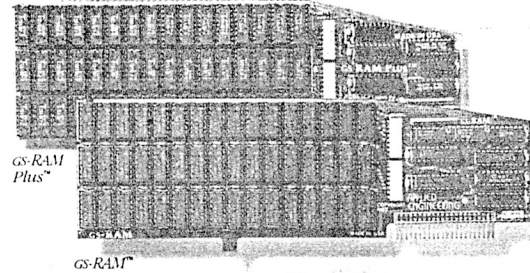
```

2700 -----
2710 .ph $179D
2720 -----
2730 * (179D) 1018 2037 25A0 25C5 25D6
2740 * Convert (A) to right-justified, blank-filled
2750 * decimal string in RJBF.STRING. Also return
2760 * the three digits in A,X,Y. In both places,
2770 * the three digits are in ASCII, with leading
2780 * zeroes converted to blanks.
2790 -----
2800 CONVERT.A.TO.RJBF.STRING
2810 LDY #10' Start with ASCII zero
2820 .1 CMP #100 Count the hundreds
2830 BCC .2 ...none left
2840 SEC take 100 out
2850 SBC #100
2860 INY and count it
2870 BNE .1 ...always
2880 .2 LDX #10' Start with ASCII zero for 10's
2890 .3 CMP #10 Count the tens
2900 BCC .4 ...none left
2910 SEC take 10 out
2920 SBC #10
2930 INX and count it
2940 BNE .3 ...always
2950 .4 ORA #10' Change units to ASCII
2960 CPY #10' Change leading zero to blank
2970 BNE .5 ...not a leading zero
2980 LDY #1' ...lz, change to blank
2990 CPX #10' Might be another lz, tens digit
3000 BNE .5 ...not a leading zero
3010 LDX #1' ...lz, change tens to blank
3020 .5 STY RJBF.STRING+1 store the 3 characters
3030 TAY
3040 STX RJBF.STRING+2
3050 STY RJBF.STRING+3
3060 LDA RJBF.STRING+1 also return in AX
3070 RTS
3080 -----
3090 * (17D1) 101E
3100 DIVIDE.PO.BY.P2
3110 JSR GET.4.PARMS
3120 .1 LDA (P0),Y Get two arguments via P0-P3
3130 STA M.REG,Y Dividend
3140 LDA (P2),Y
3150 STA DIVISOR,Y Divisor
3160 INY
3170 CPY #1
3180 BEQ .1
3190 LDA DIVISOR
3200 ORA DIVISOR+1
3210 BNE .2
3220 SEC Division by zero, return .CS.
3230 BCS .5 ...always
3240 .2 LDA #400
3250 STA REMAINDER
3260 STA REMAINDER+1
3270 LDX #10
3280 .3 ROL M.REG
3290 ROL M.REG+1
3300 ROL REMAINDER
3310 ROL REMAINDER+1
3320 SEC
3330 LDA REMAINDER
3340 SBC DIVISOR
3350 TAY
3360 LDA REMAINDER+1
3370 SBC DIVISOR+1
3380 BCC .4
3390 STY REMAINDER
3400 STA REMAINDER+1
3410 .4 DEX
3420 BNE .3
3430 ROL M.REG
3440 ROL M.REG+1
3450 CLC
3460 .5 RTS
3470 -----
3480 * (1815) 101B 1599 1815 2D45
3490 HANG JMP HANG
3500 -----
1815- 4C 15 18 3490
3500

```

For Those Who Want the Most. From Those Who Make the Best. GS-RAM™

Now expand the IIcs' RAM and ROM with up to 8 MEG of "Instant On" memory with the all new GS-RAM!



GS-RAM has an all new design. A design that delivers higher performance including increased speed, greater expandability, and improved software.

More Sophisticated, Yet Easier to Use

GS-RAM works with all IIcs software. In fact any program that runs on Apple's smaller memory card runs on the GS-RAM. But with GS-RAM you can have more memory, improved performance, and almost unlimited expansion capabilities. We've designed the new GS-RAM to be easier to use too—you don't have to adjust the size of your RAM disk every time you use a DMA device. No other RAM card with more than 4 banks of memory installed can make the same claim.

More than Just Hardware

Each GS-RAM and GS-RAM Plus includes the most powerful set of IIcs software enhancements available anywhere. In fact, our nearest competitor offers only a fraction of the invaluable programs that we include with each GS-RAM card. This software includes the most powerful disk-caching program available, the GS-RAM Cache. The Cache will make most of your applications run up to 7 times faster. Also included is a diagnostic utility that lets you test your GS-RAM by graphically showing the location of any bad or improperly installed RAM chips. And for AppleWorks users, we give you our exclusive Expander program that dramatically enhances both the capabilities and speed of AppleWorks.

Making AppleWorks Even Better

Applied Engineering's Expander program eliminates AppleWorks internal memory limits allowing it to recognize up to 8 megabytes of desktop workspace. You can increase the limits from only 7,250 lines to 22,600 lines in the word processor and from 6,350 records to 22,600 records in the database. The Expander allows all of AppleWorks, including print functions, to automatically load into RAM. The clipboard size will increase from 255 to 2,042 lines maximum. GS-RAM will automatically segment larger files so you can save them onto multiple floppies. And

GS-RAM provides a built-in print buffer that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

Grow by Kilobytes or Megabytes

We offer GS-RAM in two configurations so you can increase your memory 256K at a time (GS-RAM) or a megabyte at a time (GS-RAM Plus). Both are IIcs compatible and both come with our powerful enhancement software. GS-RAM can hold up to 1.5 MEG of 256K chips and GS-RAM Plus can hold up to 6 MEG using 1 MEG chips. And since both use standard RAM chips (not high-priced SIMMs), you'll find expanding your GS-RAM or GS-RAM Plus easy, convenient, and very economical. For further expansion, you can plug a 2 MEG "piggyback" card into the GS-RAM's expansion port for up to 3.5 MEG of total capacity. Or up to a whopping 8 MEG on GS-RAM Plus. If a GS-RAM owner outgrows 3.5 MEG, he can easily upgrade to GS-RAM Plus for a nominal charge.

Permanent Storage for an "Instant On" Apple

With our RamKeeper™ back-up option, your GS-RAM or GS-RAM Plus will retain both programs and data while your IIcs is turned off! Now when you turn your IIcs back on, your favorite software is on your screen in under 4 seconds! With RamKeeper you can divide your IIcs memory into part "electronic hard disk" and part extended RAM. Even change the memory boundaries at any time—and in any way—you want. Because



Steve Wozniak, the creator of Apple Computer

"In quality, performance, compatibility, expandability and support, Applied Engineering's GS-RAM and GS-RAM Plus are number one."

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple, you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, Reliability is everything. That's why Applied Engineering uses state-of-the-art "GEL-CELLS" instead of Ni-Cad batteries (if Ni-Cads aren't discharged periodically, they lose much of their capacity). RamKeeper has about 6 hours of "total power failure" back-up time. That's 6 times the amount of other systems. But with power from your wall outlet, RamKeeper will back-up GS-RAM, GS-RAM Plus, or most other IIcs memory cards indefinitely. Should you ever have a "total power failure," RamKeeper switches to its 6-hour battery. When power returns, RamKeeper will automatically recharge the battery to full power. RamKeeper incorporates a dual-rate charger, status LED's, and advanced power reducing circuitry. RamKeeper comes complete with battery, software, and documentation.

GS-RAM's Got it All!

- 5-year warranty — parts & labor
- 6 RAM banks (most cards have 4)
- Memory expansion port
- ROM expansion port
- Ultra-fast disk caching on ProDOS 8 AND ProDOS 16.
- Expands AppleWorks internal limits
- Includes hi-res self test
- No soldered-in RAM chips
- Expandable to 8 MEG
- No configuration blocks to set
- RamKeeper back-up option allows permanent storage of program & data
- 15-day money-back guarantee
- Proudly made in the U.S.A.

GS-RAM with 256K	\$189
GS-RAM with 512K	\$259
GS-RAM with 1 MEG	\$399
GS-RAM with 1.5 MEG	\$539
GS-RAM with 2.5 to 3.5 MEG	CALL
GS-RAM Plus with 1-8 MEG	CALL
RamKeeper Option	\$179

Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside U.S.A.

AE APPLIED ENGINEERING™
The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006
Prices subject to change without notice

GS-RAM, GS-RAM Plus and RamKeeper are trademarks of Applied Engineering. Other brands and product names are registered trademarks of their respective holders.

```

3510 * (1818) 1021 1986 1A2D 1C02 26CC
1818- A2 10 3520 BEEP.AND.CLEAR.KEYBUF
181A- 20 E9 1F 3530 LDX #10 BEEP TOKEN
181D- 20 E0 1F 3540 JSR DISPLAY.TOKEN.X
1820- A9 01 3550 JSR CLEAR.KEYBUF
1822- 60 3560 LDA #01
3570 RTS
3580 *-----
3590 .ph $1837
3600 *-----
3610 * (1837) 1027 201C
3620 SHOW.HELP.STRING
1837- 20 02 D0 3630 JSR MEASURE.FREE.MEMORY
183A- 20 3E 1F 3640 JSR DISPLAY.AT
183D- 42 17 02
1840- 14 3650 .DA #42,#17,MSG..2 "Apple-? for Help"
1841- 60 3660 RTS
3670 *-----
3680 WARN.IF.FULLDESK
1842- AD F5 OF 3690 LDA KBYTES.DESKTOP.LEFT
1845- OD F6 OF 3700 ORA KBYTES.DESKTOP.LEFT+1
1848- D0 05 3710 BNE .1
184A- 20 50 18 3720 JSR SHOW.FULLDESK.WARNING
184D- A9 00 3730 LDA #00
184F- 60 3740 .1 RTS
3750 *-----
3760 * (1850) 102D 184A
3770 SHOW.FULLDESK.WARNING
1850- AD F3 OF 3780 LDA HIDE.FULLDESK.WARNING
1853- D0 16 3790 BNE .2 Don't display anything
1855- A6 96 3800 LDX WINDOW.BOTTOM
1857- E8 3810 INX
1858- 8E 5F 18 3820 STX .1 Store line number to print on
185B- 20 3E 1F 3830 JSR DISPLAY.AT
185E- FF 3840 .DA #FF
185F- 00 9A 14 3850 .1 .DA #00,MSG..7 "WARNING"
1862- 20 F1 1B 3860 JSR WAIT.FOR.SPACE.RETURN.OR.ESCAPE
1865- AD F4 OF 3870 LDA CHAR.FOR.BOTTOM.LINE
1868- 20 AE 20 3880 JSR DRAW.BOTTOM.LINE
186B- 60 3890 .2 RTS
3900 *-----
3910 * (186C) 1030
3920 SHOW.ENTER.COMMAND.MSG
186C- 20 F5 1F 3930 JSR DISPLAY.ON.LINE.23
186F- OF 14 3940 .DA MSG..3 "Type Entry or use Apple-commands"
1871- 60 3950 RTS
3960 *-----
3970 * (1872) 1033 238F
3980 CALL.THE.ORGANIZER
1872- A2 F8 3990 LDX #F8
1874- 9A 4000 TXS
1875- A9 20 4010 LDA #20
1877- 20 A1 11 4020 JSR LOAD.PROGRAM.SEGMENT.A
4030 *---Even though the above is a JSR, it never returns---
4040 *---from LOAD.PROGRAM.SEGMENT.A, because it enters---
4050 *---the ORGANIZER directly---
4060 *-----
4070 .ph $1BOB
4080 *-----
4090 * (1BOB) 1048 2765
4100 DRAW.TOP.AND.BOTTOM.LINES
1BOB- A2 03 4110 LDX #03 "HOME" TOKEN
1B0D- 20 E9 1F 4120 JSR DISPLAY.TOKEN.X
1B10- 20 26 D0 4130 JSR X.D026
1B13- 20 29 D0 4140 JSR DISPLAY.FUNCTION.AND.ESCAPE.MAP
1B16- A2 00 4150 LDX #00
1B18- A4 95 4160 LDY WINDOW.TOP
1B1A- 88 4170 DEY
1B1B- 20 23 18 4180 JSR MOVE.CURSOR.TO.XY
1B1E- A2 4F 4190 LDX #4F
1B20- A0 3D 4200 LDY #' PRINT 79 ='s above window
1B22- 20 BE 20 4210 JSR REPEAT.CHAR.Y.X.TIMES
1B25- A9 2D 4220 LDA #' PRINT 79 -'s below window
1B27- 20 AE 20 4230 JSR DRAW.BOTTOM.LINE
1B2A- 60 4240 RTS
4250 *-----
4260 * Mark that the current file has been changed.
4270 *-----
4280 * (1B2B) 104B
4290 POST.CHANGE.FLAG
1B2B- AD 6C 0C 4300 LDA FLAG.CURRENT.FILE
1B2E- 09 02 4310 ORA #02
1B30- 8D 6C 0C 4320 STA FLAG.CURRENT.FILE
1B33- 60 4330 RTS

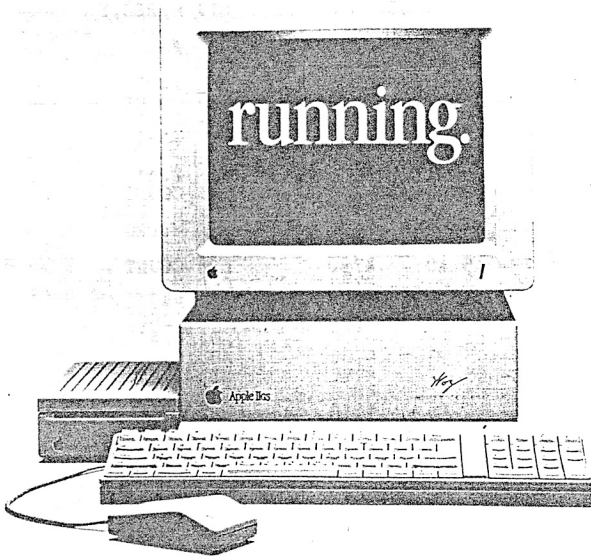
```

```

4350 * (1B34) 104E 1A09 1C35 2538 2550 2575
4360 MULTIPLY.X.BY.Y
1B34- 86 93 4370 STX MUL.X.FACTOR
1B36- 84 94 4380 STY MUL.Y.FACTOR
1B38- A9 00 4390 LDA #00
1B3A- 85 91 4400 STA M.REG
1B3C- A2 08 4410 LDX #8 8 bits
1B3E- 46 93 4420 .1 LSR MUL.X.FACTOR
1B40- 90 03 4430 BCC .2
1B42- 18 4440 CLC
1B43- 65 94 4450 ADC MUL.Y.FACTOR
1B45- 6A 4460 .2 ROR
1B46- 66 91 4470 ROR M.REG
1B48- CA 4480 DEX
1B49- D0 F3 4490 BNE .1
1B4B- 85 92 4500 STA M.REG+1
1B4D- 60 4510 RTS
4520 *-----
4530 * (1B4E) 1051
4540 MULTIPLY.PO.BY.P2
1B4E- 20 AE 18 4550 JSR GET.4.PARMS
1B51- B1 9A 4560 .1 LDA (P0),Y
1B53- 99 9E 00 4570 STA P4,Y
1B56- B1 9C 4580 LDA (P2),Y
1B58- 99 91 00 4590 STA M.REG,Y
1B5B- C8 4600 INY
1B5C- C0 01 4610 CPY #01
1B5E- F0 F1 4620 BEQ .1
1B60- A9 00 4630 LDA #00
1B62- 85 A0 4640 STA COUNT
1B64- 85 A1 4650 STA Z.A1
1B66- A2 11 4660 LDX #11
1B68- 18 4670 CLC
1B69- 66 A1 4680 .2 ROR Z.A1
1B6B- 66 A0 4690 ROR COUNT
1B6D- 66 92 4700 ROR M.REG+1
1B6F- 66 91 4710 ROR M.REG
1B71- 90 0D 4720 BCC .3
1B73- A5 A0 4730 LDA COUNT
1B75- 18 4740 CLC
1B76- 65 9E 4750 ADC P4
1B78- 85 A0 4760 STA COUNT
1B7A- A5 A1 4770 LDA Z.A1
1B7C- 65 9F 4780 ADC P5
1B7E- 85 A1 4790 STA Z.A1
1B80- CA 4800 .3 DEX
1B81- D0 E6 4810 BNE .2
1B83- 60 4820 RTS
4830 *-----
4840 .ph $1BAC
4850 *-----
4860 * (1BAC) 1057 1D1C 2404 263B 2653
4870 MOVE.BLOCK.UP
1BAC- A9 06 4880 LDA #06
1BAE- 20 B4 18 4890 JSR GET.A.PARMS
1BB1- A5 9F 4900 LDA P5
1BB3- F0 0B 4910 BEQ .1
1BB5- 18 4920 CLC
1BB6- 65 9B 4930 ADC P1
1BB8- 85 9B 4940 STA P1
1BBA- A5 9F 4950 LDA P5
1BBC- 65 9D 4960 ADC P3
1BBE- 85 9D 4970 STA P3
1BC0- A4 9E 4980 .1 LDY P4
1BC2- F0 08 4990 BEQ .3
1BC4- 88 5000 .2 DEY
1BC5- B1 9C 5010 LDA (P2),Y
1BC7- 91 9A 5020 STA (P0),Y
1BC9- 98 5030 TYA
1BCA- D0 F8 5040 BNE .2
1BCC- C6 9F 5050 .3 DEC P5
1BCE- 30 0E 5060 BMI .5
1BD0- C6 9B 5070 DEC P1
1BD2- C6 9D 5080 DEC P3
1BD4- 88 5090 .4 DEY
1BD5- B1 9C 5100 LDA (P2),Y
1BD7- 91 9A 5110 STA (P0),Y
1BD9- 98 5120 TYA
1BDA- D0 F8 5130 BNE .4
1BDC- F0 EE 5140 BEQ .3
1BDE- 60 5150 .5 RTS

```

In about the time it takes to read this headline, you can have the Finder up and



Now your favorite program can be ready to go seconds after you flip your Apple IIgs on.

With Applied Engineering's RamKeeper™ card, your IIgs retains stored programs and data when you turn your computer off.

RamKeeper powers up to two memory cards simultaneously when your Apple IIgs is off. And battery backup keeps power to the boards even during power failures. Your programs and data are stored in a permanent, "electronic hard disk," always ready to run.

Superior power backup.

Applied Engineering has the most experience in battery-



RamKeeper lets you keep programs and data in permanent, "electronic hard disk" memory. Turn your Apple IIgs on and you're ready to work.

backed memory for Apple computers. We were the first to offer battery-backed memory with our RamFactor™/RamCharger™ combination. Now RamKeeper sets the standard for IIgs memory backup.

Our experience shows in the way we designed and built RamKeeper. We used sealed Gel/

Cell batteries — far more reliable than Ni-Cads in this application. Ni-Cads lose much of their capacity if they're not discharged periodically. Just when you need them most, Ni-Cads could run out of power.

Our Gel/Cell pack, which is included in our price, gives you up to six hours of total power failure backup. That's about 6 times longer than other systems.

RamKeeper uses a Switching Power Supply — the same technology used by Apple for the IIgs power supply. This design uses energy much more efficiently to keep your Apple running cooler.

Our sealed Gel/Cell battery

stays outside your computer case. With other systems, the batteries are installed under the IIgs power supply where a leak could ruin computer circuitry.

Put two memory boards in the same slot.

You might have bought your IIgs with Apple's memory card. But now you want the features of Applied's GS-RAM™ card. RamKeeper efficiently resolves the dilemma.

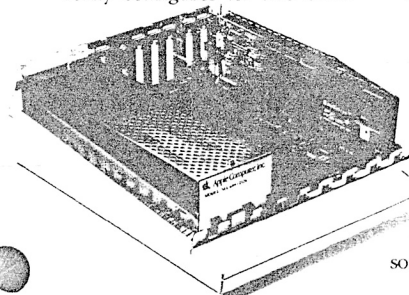
You can use RamKeeper with your current IIgs memory card and add another memory card — all in the same slot. Just attach your current memory card to one side of your new RamKeeper card, connect the second card to the other side and plug RamKeeper into the slot.

Of course RamKeeper works fine with just one memory card. But you can use two and still keep Slot 7 open with our optional Slot-Mover.

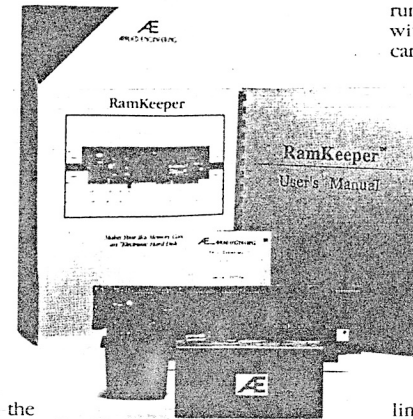
Makes all of your memory usable memory.

RamKeeper can power up to 16 Meg of memory. Other systems are limited to only 8 Meg. In addition, RamKeeper lets you mix and match different types of cards. For example, you can have a GS-RAM Plus™ using 1 Meg RAM chips and an Apple card using 256K RAM chips. Other systems are limited in the combinations they allow.

RamKeeper firmware automatically configures for two cards



RamKeeper is easy to install. Just plug it in. Even when you use two memory boards, you don't have jumpers. You can have two memory boards but use only one slot.



It all comes with RamKeeper ... board, Gel/Cell battery pack, easy-to-understand instructions, and Applied's powerful AppleWorks Expander software.

when the second card is installed. Other systems make you manually move jumpers.

RamKeeper configures memory linearly. Other systems don't, so they create memory gaps that can cause program crashes or keep some programs from using as much as half of your memory.

You easily decide how much memory you'll devote to ROM and to RAM from the IIgs Desk Accessories menu. You can configure Kilobytes or Megabytes of instant ROM storage for your favorite programs. And you can change ROM size any time without affecting stored files.

Protected from program crashes.

RamKeeper controlling firmware is in an EPROM. A program crash can't take out the operating software. With other systems, operating software is installed in RAM from a floppy. If the program crashes, it can take the operating software with it; and reinstalling the disk-based operating software destroys data in memory.

Verifies data security.

RamKeeper firmware uses optional startup checksums to verify that no data has been lost while power was off. The firmware also

runs ROM and RAM memory tests without disturbing data on the card.

Free AppleWorks Enhancement software.

Applied's powerful AppleWorks Enhancement software is free with RamKeeper. It makes AppleWorks faster and far more powerful by eliminating AppleWorks internal memory limits. Word processor limits go from only 7,250 lines to 22,600 lines. Database limits go from 6,350 records to 22,600 records. The clipboard size

limit is increased from 255 to 2,042 lines. It even automatically segments large files so you can save them on multiple floppies. No other company expands your IIgs' AppleWorks internal limits.

In addition, the most powerful disk-caching program available comes with the RamKeeper. The cache significantly increases access time to the Apple 3.5 Drive. Most applications will run up to 7 times faster.

The largest maker of Apple expansion boards.

Applied Engineering has sold more expansion boards than anyone else. And we've been in business 8 years, long enough to see the vast majority of our competitors come and go.

All of our products are crafted in the U.S.A. We back RamKeeper with a five year parts and labor warranty. And a 15-day, no questions asked, money back guarantee.

Only \$189.00.

See your dealer. Or call 214-241-6060, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA, C.O.D. welcome. Texas residents add 7% sales tax. Add \$10.00 outside U.S.A. Prices subject to change without notice.

AE APPLIED ENGINEERING™

The Apple enhancement experts.

P.O. Box 5100 • Carrollton, Texas 75011.

(214) 241-6060

```

5160 * (1BDF) 105A
5170 POP.ESCAPE.ROAD.MAP
5180 JSR MOVE.BLOCK.DOWN
5190
1BDF- 20 84 1B
1BE2- E8 0C FD
1BE5- 0C A8 00 5200 .DA ESCAPE.ROAD.MAP,ESCAPE.ROAD.MAP+21,168
1BE8- A9 00 5210 LDA #00 MARK LAST ENTRY EMPTY
1BEA- 8D 90 0D 5220 STA ESCAPE.ROAD.MAP+168
1BED- 20 29 0D 5230 JSR DISPLAY.FUNCTION.AND.ESCAPE.MAP
1BFO- 60 5240 RTS
5250 *-----*
5260 * (1BF1) 105D 1862 2389 2728 2ADB
5270 WAIT.FOR.SPACE.RETURN.OR.ESCAPE
1BF1- 20 E0 1F 5280 JSR CLEAR.KEYBUF
1BF4- 20 F5 1F 5290 JSR DISPLAY.ON.LINE.23
1BF7- 2E 14 5300 .DA MSG..4 "Press Space Bar to continue"
1BF9- A9 00 5310 LDA #0
1BFB- 85 88 5320 STA FLAG.FOUND.ESCAPE
1BFD- 20 35 1D 5330 .1 JSR AW.KEYIN
1C00- C9 20 5340 CMP #20
1C02- F0 0F 5350 BEQ .3 ...GOT A SPACE
1C04- C9 0D 5360 CMP #0D
1C06- F0 0B 5370 BEQ .3 ...GOT <RETURN>
1C08- C9 1B 5380 CMP #1B
1COA- F0 05 5390 BEQ .2 ...GOT <ESCAPE>
1C0C- 20 18 18 5400 JSR BEEP.AND.CLEAR.KEYBUF
1COF- DO EC 5410 BNE .1 ...ALWAYS
1C11- E6 88 5420 .2 INC FLAG.FOUND.ESCAPE RETURN FLAG.FOUND.ESCAPE=$01,
1C13- 60 5430 .3 RTS STATUS .NE.
5440 *-----*
5450 * (1C14) 1CB3 1CDD
1C14- 5460 PD.CHARCNT .BS 1
5470 * (1C15) 1C3F 1CF1
1C15- 5480 PRNTR.CRLF.FLAG .BS 1
5490 * (1C16) 1C6D 2309 2336
1C16- 5500 I.1C16 .BS 1 'e' if //e, or 'c' if //c
1C17- 5510 PRNTR.SETUP.STRING .BS 10
5520 *-----*
5530 * (1C21) 1060 1E22 1E32 1E39 1E48
5540 * Called from KEYIN when Apple-H is pressed,
5550 * and from JMP table at $1060
5560 * (A) = $00 Open Printer
5570 * $FE Print CRLF
5580 * $FF Close Printer
5590 * other A=#chars, P0,P1 is address of text.
5600 *-----*
1C21- A2 00 5610 PRNTR.DRIVER
1C23- 86 24 5620 LDX #00
1C25- 86 25 5630 STX MON.CH
1C27- AA 5640 STX MON.CV
1C28- F0 03 5650 TAX
1C2A- 4C AD 1C 5660 BEQ .1 A=00, Open Printer
5670 JMP .6 SOME OTHER FUNCTION
5680 *---Open Printer-----*
1C2D- AE 3D 0F 5690 .1 LDX CURRENT.PRINTER.NO
1C30- F0 78 5700 BEQ .5 None, or it would have been 1-3
1C32- CA 5710 DEX change range to 0,1,2
1C33- A0 24 5720 LDY #36 multiply by 36 bytes per printer
1C35- 20 34 1B 5730 JSR MULTIPLY.X.BY.Y
1C38- A6 91 5740 LDX M.REG
1C3A- BD 73 0F 5750 LDA X.OF73,X
1C3D- 29 20 5760 AND #20 non-zero means print LF after CR
1C3F- 8D 15 1C 5770 STA PRNTR.CRLF.FLAG
1C42- BD 71 0F 5780 LDA X.OF71,X
1C45- AA 5790 TAX
1C46- BD C6 0F 5800 LDA SLOTROM.MAP,X
1C49- 29 02 5810 AND #02
1C4B- F0 5D 5820 BEQ .5 Not a printer interface, for sure
1C4D- EC A8 0E 5830 CPX OPEN.PRINTER.FLAG
1C50- F0 58 5840 BEQ .5
1C52- 8E A8 0E 5850 STX OPEN.PRINTER.FLAG
1C55- 8A 5860 TXA
1C56- 09 C0 5870 ORA #C0
1C58- 85 37 5880 STA CSWH
1C5A- A9 00 5890 LDA #00
1C5C- 85 36 5900 STA CSWL
1C5E- AD 0A 1D 5910 LDA HANDLE.PRNTR.SETUP.STRING
1C61- 85 9A 5920 STA P0
1C63- AD 0B 1D 5930 LDA HANDLE.PRNTR.SETUP.STRING+1
1C66- 85 9B 5940 STA P1

```

```

1C68- A6 91 5950 LDX M.REG
1C6A- BD 70 0F 5960 LDA X.OF70,X
1C6D- CD 16 1C 5970 CMP I.1C16
1C70- D0 0F 5980 BNE .2
1C72- AD 0C 1D 5990 LDA HANDLE.OF61
1C75- 18 6000 CLC
1C76- 65 91 6010 ADC M.REG
1C78- 85 9A 6020 STA P0
1C7A- AD 0D 1D 6030 LDA HANDLE.OF61+1
1C7D- 69 00 6040 ADC #00
1C7F- 85 9B 6050 STA P1
1C81- A0 00 6060 .2 LDY #00
1C83- B1 9A 6070 .2 LDA (P0),Y
1C85- F0 23 6080 BEQ .5
1C87- 85 9C 6090 STA P2
1C89- 8D 82 C0 6100 STA RQM.D000
1C8C- C8 6110 .3 INY
1C8D- B1 9A 6120 LDA (P0),Y
1C8F- C9 09 6130 CMP #09
1C91- D0 10 6140 BNE .4
1C93- A9 89 6150 LDA #89
1C95- AE EF 0F 6160 LDX FLAG.APPLE.2C
1C98- F0 09 6170 BEQ .4 ...not an Apple //c
1C9A- AE A8 0E 6180 LDX OPEN.PRINTER.FLAG
1C9D- E0 02 6190 CPX #02
1C9F- D0 02 6200 BNE .4
1CA1- A9 01 6210 LDA #01
1CA3- 20 ED FD 6220 .4 JSR MON.COUT
1CA6- C4 9C 6230 CPY P2
1CA8- 90 E2 6240 BCC .3
1CAA- 4C 03 1D 6250 .5 JMP .13
6260 *-----*
1CAD- C9 FE 6270 .6 CMP #FE
1CAF- F0 33 6280 BEQ .11 A=FE, Print CRLF
1CB1- B0 4B 6290 BCS .12 A=FF, Close Printer
1CB3- 8D 14 1C 6300 STA PD.CHARCNT A=number chars to print
1CB6- 20 B2 18 6310 JSR GET.2.PARMS
1CB9- AD A8 0E 6320 LDA OPEN.PRINTER.FLAG
1CBC- F0 45 6330 BEQ .13
1CBE- 8D 82 C0 6340 STA RQM.D000
1CC1- A0 00 6350 LDY #00
1CC3- B1 9A 6360 .7 LDA (P0),Y
1CC5- C9 81 6370 CMP #81 Change "Apple" to "@"
1CC7- D0 04 6380 BNE .8 ...not an apple
1CC9- A9 0A 6390 LDA #40
1CCB- D0 06 6400 BNE .9
1CCD- C9 8D 6410 .8 CMP #8D ...always
1CCF- D0 02 6420 BNE .9 change <RETURN> to <SPACE>
1CD1- A9 20 6430 LDA #20
1CD3- C9 09 6440 .9 CMP #09 change 09 (ctrl-I) to 89
1CD5- D0 02 6450 BNE .10 ...not 09
1CD7- A9 89 6460 LDA #89
1CD9- 20 ED FD 6470 .10 JSR MON.COUT out to the printer, finally
1CDD- C8 6480 INY
1CDD- CC 14 1C 6490 CPY PD.CHARCNT
1CE0- 90 E1 6500 BCC .7 ...more to print
1CE2- B0 1F 6510 BCS .13 ...finished
6520 *---Print CRLF-----*
1CE4- AD A8 0E 6530 .11 LDA OPEN.PRINTER.FLAG
1CE7- F0 1A 6540 BEQ .13
1CE9- 8D 82 C0 6550 STA RQM.D000
1CEC- A9 0D 6560 LDA #0D
1CEE- 20 ED FD 6570 JSR MON.COUT
1CF1- AD 15 1C 6580 LDA PRNTR.CRLF.FLAG
1CF4- F0 0D 6590 BEQ .13
1CF6- A9 0A 6600 LDA #0A
1CF8- 20 ED FD 6610 JSR MON.COUT
1CFB- 4C 03 1D 6620 JMP .13
6630 *---Close Printer-----*
1CFE- A9 00 6640 .12 LDA #00
1D00- 8D A8 0E 6650 STA OPEN.PRINTER.FLAG
1D03- AD 83 C0 6660 .13 LDA RAM.D000
1D06- AD 83 C0 6670 LDA RAM.D000
1D09- 60 6680 RTS
6690 *-----*
1DOA- 17 1C 6700 * (1DOA,B) 1C5E 1C63
6710 HANDLE.PRNTR.SETUP.STRING .DA PRNTR.SETUP.STRING
6720 * (1DOC,D) 1C72 1C7A
1DOC- 61 0F 6730 HANDLE.OF61 .DA X.OF61
6740 *-----*

```



```

1DOE- 6750 * (1DOE) 1E10
      6760 .BS 1
      6770 -----
      6780 * (1DOF) 1063
1DOF- 20 B2 18 6790 PUSH.ESCAPE.ROAD.MAP
      6800 JSR GET.2.PARMS
1D12- A5 9A 6810 LDA P0
1D14- 8D 2A 1D 6820 STA .1 FILL IN ADDRESS
1D17- A5 9B 6830 LDA P1
1D19- 8D 2B 1D 6840 STA .1+1
1D1C- 20 AC 1B 6850 JSR MOVE.BLOCK.UP
1D1F- FD 0C E8
1D22- 0C A8 00 6860 .DA ESCAPE.ROAD.MAP+21,ESCAPE.ROAD.MAP,168
1D25- 20 F8 1E 6870 JSR MOVE.STRING
1D28- E8 0C 6880 .DA ESCAPE.ROAD.MAP
1D2A- 00 00 6890 .1 .DA 0000 FILLED IN
1D2C- 20 29 DO 6900 JSR DISPLAY.FUNCTION.AND.ESCAPE.MAP
1D2F- 60 6910 RTS
      6920 -----
      6930 .ph $1E80
      6940 -----
      6950 * (1E80) 1069 1911 193C 19B2 2025 2612 261A
1E80- AE AD OE 6960 MOVE.CURSOR.TO.TCOL.TROW
1E83- AC AE OE 6970 LDX TCOL
1E86- 20 23 18 6980 LDY TROW
1E89- 60 6990 JSR MOVE.CURSOR.TO.XY
      7000 RTS
      7010 -----
      7020 * (1E8A) 106C
1E8A- 8E AD OE 7030 SAVE.GOTO.XY
1E8D- 8C AE OE 7040 STX TCOL
1E90- 20 23 18 7050 STY TROW
1E93- 60 7060 JSR MOVE.CURSOR.TO.XY
      7070 RTS
      7080 -----
      7090 .ph $1EB4
      7100 -----
      7110 * (1EB4) 106F
1EB4- C9 61 7120 MAP.LOWER.TO.UPPER
1EB6- 90 06 7130 CMP #'a'
1EB8- C9 7B 7140 BCC .1 ...not lower-case
1EBA- B0 02 7150 CMP #'z'+1
1EBC- 29 DF 7160 BCS .1 ...not lower-case
1EBE- 60 7170 AND #$DF flip to upper-case
      7180 .1 RTS
      7190 -----
      7200 .ph $1F3E
      7210 -----
      7220 * Display string (P2,P3) at (P0,P1)
      7230 * JSR DISPLAY.AT
      7240 * .DA #column,#line
      7250 * .DA string.address
      7260 *
      7270 * If column value is negative but not $FF, then add $94
      7280 * which clears bit 7 and adds 20
      7290 * If column value is $FF then center the string in 80 columns
      7300 * -----
      7310 * (1F3E) 107B 183A 185B 200C 25E5 2713 271A 2ACD 2B70 2B77
      7320 * (1F3E) 2B8C 2B93 2BA1 2BA8 2D37 2D3E
1F3E- 20 AE 18 7330 DISPLAY.AT
1F41- A9 05 7340 JSR GET.4.PARMS
1F43- 8D 00 OA 7350 LDA #$05 Build string to set cursor position
1F46- A5 9A 7360 STA STR.A
1F48- 8D 01 OA 7370 LDA P0
1F4B- A5 9B 7380 STA STR.A+1
1F4D- 8D 02 OA 7390 LDA P1
1F50- A5 9C 7400 STA STR.A+2
1F52- 18 7410 LDA P2
1F53- 69 01 7420 CLC
1F55- 8D B2 OE 7430 ADC #1 Build address of 1st char after length
1F58- A5 9D 7440 STA X.OEB2
1F5A- 69 00 7450 LDA P3
1F5C- 8D B3 OE 7460 ADC #0
1F5F- A0 00 7470 STA X.OEB3
1F61- B1 9C 7480 LDY #0 Get length of string
1F63- 20 A0 1F 7490 LDA (P2),Y
1F66- 8D B4 OE 7500 JSR TRUNCATE.TO.79.IF.OVER.80
1F69- AD 01 OA 7510 STA X.OEB4
1F6C- 10 17 7520 LDA STR.A+1 Was P0 positive, meaning absolute column?
      7530 BPL .2 ...yes

```

```

1F6E- C9 FF 7540 CMP #$FF
1F70- DO OD 7550 BNE .1 ...not centering, so add $94
1F72- A9 50 7560 LDA #80 $FF means to center in 80 columns
1F74- 38 7570 SEC (80-length)/2
1F75- ED B4 OE 7580 SBC X.OEB4
1F78- 4A 7590 LSR
1F79- 8D 01 OA 7600 STA STR.A+1
1F7C- 4C 85 1F 7610 JMP .2
      7620 -----
1F7F- 18 7630 .1 CLC
1F80- 69 94 7640 ADC #$94
1F82- 8D 01 OA 7650 STA STR.A+1
1F85- 20 A9 1E 7660 .2 JSR POINT.PSTR.AT.OA00
1F88- A9 03 7670 LDA #3
1F8A- 20 D1 14 7680 JSR DISPLAY.STRING
1F8D- AD B2 OE 7690 LDA X.OEB2
1F90- 85 80 7700 STA PSTR
1F92- AD B3 OE 7710 LDA X.OEB3
1F95- 85 81 7720 STA PSTR+1
1F97- AD B4 OE 7730 LDA X.OEB4
1F9A- FO 03 7740 BEQ .3 null string
1F9C- 20 D1 14 7750 JSR DISPLAY.STRING
1F9F- 60 7760 .3 RTS
      7770 -----
      7780 * (1FA0) 1F63 20A7 20C2
1FA0- C9 51 7790 TRUNCATE.TO.79.IF.OVER.80
1FA2- 90 02 7800 CMP #81
1FA4- A9 4F 7810 BCC .1
1FA6- 60 7820 LDA #79
      7830 .1 RTS
      7840 -----
      7850 .ph $1FE9
      7860 -----
      7870 * (1FE9) 108A 181A 1970 19B7 19C2 1A1E 1A97 1AAB 1AC3 1AD8
      7880 * (1FE9) 1B0D 2015 2B5F
1FE9- 8E 00 OA 7890 DISPLAY.TOKEN.X
1FEC- 20 A9 1E 7900 STX STR.A
1FEF- A9 01 7910 JSR POINT.PSTR.AT.OA00
1FF1- 20 D1 14 7920 LDA #1
1FF4- 60 7930 JSR DISPLAY.STRING
      7940 RTS
      7950 -----
      7960 * JSR DISPLAY.ON.LINE.23
      7970 * .DA string
      7980 *
      7990 * 1. Display string starting at col. 0, line 23
      8000 * 2. Clear rest of the line
      8010 * 3. If Z.89 is $00, display "xxxxK Avail.";
      8020 * otherwise, display "Apple-? for Help"
      8030 * 4. Move cursor to end of the string.
      8040 * -----
      8050 * (1FF5) 1093 118D 186C 1931 1A45 1BF4 20D6 25F3
1FF5- 20 B2 18 8060 DISPLAY.ON.LINE.23
1FF8- A5 9A 8070 JSR GET.2.PARMS
1FFA- 8D 11 20 8080 LDA P0
1FFD- A5 9B 8090 STA .1 store address in parameter below
1FFF- 8D 12 20 8100 LDA P1
2002- B1 9A 8110 STA .1+1
2004- 8D AD OE 8120 LDA (P0),Y length of string
2007- A9 17 8130 STA TCOL later position cursor to end of string
2009- 8D AE OE 8140 LDA #23 on line 23
200C- 20 3E 1F 8150 STA TROW
200F- 00 17 8160 JSR DISPLAY.AT
2011- 00 00 8170 .DA #0,#23 Column 0, line 23
2013- A2 01 8180 .1 .DA 0000 String address, filled in
2015- 20 E9 1F 8190 LDX #$01 "Clear to End of Line" token
2018- A5 89 8200 JSR DISPLAY.TOKEN.X Print the token
201A- FO 06 8210 LDA Z.89
201C- 20 37 18 8220 BEQ .2 ...display "xxxxK Avail."
201F- 4C 25 20 8230 JSR SHOW.HELP.STRING Display "Apple-? for Help"
2022- 20 44 DO 8240 JMP .3
2025- 20 80 1E 8250 .2 JSR DISPLAY.K.AVAIL Display "xxxxK Avail."
2028- 60 8260 .3 JSR MOVE.CURSOR.TO.TCOL.TROW
      8270 RTS
      8280 -----

```

Klaxon Sound Effect.....Robert C. Moore

Here is an interesting little sound effect generator, which you might like to use to call attention to an operator error. It uses the simple Apple speaker, so it is compatible with all of the many models in the Apple II series and even the clones.

Lines 1110-1130 call on the SOUNDS subroutine to generate two bursts of sound. The first burst combines plays a higher note, the second a lower note. Both notes are combined inside SOUNDS with a pitch that is in between the two. The total effect sounds a little like a klaxon to me.

Bob S-C inserted line 1100, which turns the dee-daw sound into dee-daw-dee-daw. You might also enjoy experimenting with different pitches and durations. You can change the intermediate pitch by change the value 63 in lines 1190 and 1240.

You might notice that DURATION2 never gets initialized to anything. If you want to, you could store 0 there by inserting these lines:

```
1191 LDA #0
1192 STA DURATION2
```

However, you will not be able to tell the difference in how it sounds. The total time the horn blows is DURATION1 times 256, less up to 255 if DURATION2 starts out non-zero. In the worst case, with DURATION2 starting at 1, the horn will finish blowing about 5% sooner.

```
1010 *-----*
1020 * from Robert C. Moore, Laurel, Maryland.
1030 *-----*
00- 1040 DURATION1 .EQ 0
01- 1050 DURATION2 .EQ 1
02- 1060 PITCH .EQ 2
1070 *-----*
C030- 1080 SPEAKER .EQ $C030
1090 *-----*
0800- 20 03 08 1100 BOBSC JSR KLAXON
0803- A0 32 1110 KLAXON LDY #50 FIRST PRIMARY PITCH
0805- 20 0A 08 1120 JSR SOUNDS
0808- A0 50 1130 LDY #80 SECOND PRIMARY PITCH
1140 *-----*
1150 SOUNDS
080A- 84 02 1160 STY PITCH SAVE CALLER'S Y-PITCH
080C- A9 14 1170 LDA #20 LENGTH OF SOUND
080E- 85 00 1180 STA DURATION1
0810- A2 3F 1190 LDX #63 SECONDARY PITCH
1200 *-----*
0812- CA 1210 .2 DEX COUNT DOWN SECONDARY CYCLE
0813- D0 05 1220 BNE .3 ...NOT TIME FOR CLICK YET
0815- 2C 30 C0 1230 BIT SPEAKER ...CLICK NOW
0818- A2 3F 1240 LDX #63 START ANOTHER CYCLE
1250 *-----*
081A- 88 1260 .3 DEY COUNT DOWN PRIMARY CYCLE
081B- D0 05 1270 BNE .4 ...NOT TIME FOR CLICK YET
081D- 2C 30 C0 1280 BIT SPEAKER ...CLICK NOW
0820- A4 02 1290 LDY PITCH START ANOTHER CYCLE
1300 *-----*
0822- C6 01 1310 .4 DEC DURATION2 256*20 TIMES ALTOGETHER
0824- D0 EC 1320 BNE .2
0826- C6 00 1330 DEC DURATION1
0828- D0 E8 1340 BNE .2
082A- 60 1350 RTS
```

AppleWorks Segment Functions.....Steve Stephenson

I have been disassembling and patching AppleWorks since 1984 for CheckMate Technology, and have learned a few things about it. One interesting task was trying to figure out what each one of the segments in the SEG.M0 and SEG.M1 files does. As near as I can tell, here is the breakdown for AppleWorks 1.3:

- | | |
|--------------------------|----------------------------------|
| Seg.01 DB Common | Seg.24 SS Common |
| Seg.02 DB Main | Seg.25 SS Main |
| Seg.03 DB Help | Seg.26 SS Help |
| Seg.04 DB SingleLayout | Seg.27 SS Print |
| Seg.05 DB MultipleLayout | Seg.28 SS Layout |
| Seg.06 DB ChangeNames | Seg.29 SS Edit |
| Seg.07 DB Search | Seg.30 SS ToClipboard |
| Seg.08 DB Sort | Seg.31 SS Options |
| Seg.09 DB Report Main | |
| Seg.10 DB Report Util | Seg.32 DeskTop Manager |
| Seg.11 DB Report New | Seg.33 File Loader |
| Seg.12 DB Report Layout | Seg.34 Printer Setups |
| Seg.13 DB Report Open | Seg.35 Import DIF for DB |
| Seg.14 DB Report Output | Seg.36 File Saver |
| Seg.15 -doesn't exist- | Seg.37 Import Quick Files for DB |
| | Seg.38 Import ASCII Text for WP |
| Seg.16 WP Common | Seg.39 Import VisiCalc for SS |
| Seg.17 WP Main | |
| Seg.18 WP Help | Seg.40 Main Help |
| Seg.19 WP Print | |
| Seg.20 -doesn't exist- | Seg.41 Import Text for DB |
| Seg.21 -doesn't exist- | Seg.42 Import DIF for SS |
| Seg.22 SANE | |
| Seg.23 SANE | Seg.43 Disk Formatter |

I also wrote a program which will break out the segments from the SEG.M0 and SEG.M1 files into individual binary files, each with their proper SEG number, load address, and length.

There is no fancy prefix handling in the program, so the two source files and all the resultant files must all land in the same directory. I use a RAM disk of at least 600 blocks, for speed. There is also not any fancy error handling. If any error occurs, the hexadecimal error code returned by MLI will be displayed. If not, 00 will be displayed when it's finished.

The monthly AAL disk also includes a slightly different version of the Segment Splitter, which displays a line for each segment including the segment number, the file offset, starting address, and length.

```
1020 *-----*
1030 * AppleWorks Segment Splitter
1040 * by Steve Stephenson
1050 *-----*
BF00- 1060 MLI .EQ $BF00
BF0F- 1070 ERRCODE .EQ $BF0F
C0- 1080 CREATE .EQ $C0 ProDOS MLI Functions
C8- 1090 OPEN .EQ $C8
CA- 1100 READ .EQ $CA
CB- 1110 WRITE .EQ $CB
CC- 1120 CLOSE .EQ $CC
CE- 1130 SETMARK .EQ $CE
1140 *-----*
```

```

1150 .MA MLI Function, Parameter List
1160 JSR MLI
1170 .DA #]1,]2
1180 BCC :1 No Error
1190 JMP ERROR
1200 :1
1210 .EM
1220 *-----
1230 .MA STR
1240 .DA #-1--#-1 Number of bytes in string
1250 .AS -" ]1" String itself
1260 :1
1270 .EM
1280 *-----
0300- 1290 BUFFER.INDEX .EQ $300
06-- 1300 FILE.CNT .EQ $6
1310 *-----
1000- 1320 .OR $1000
1330 .TF B.SEG.SPLITTER
1340 *-----
1350 SEGMENT.SPLITTER
1000- A9 01 1360 LDA #1
1002- 85 06 1370 STA FILE.CNT
1380 PRELOOP
1004- 1390 >MLI OPEN,P.OPEN.SEGMX open seg.m0/1
100F- AD 38 11 1400 LDA P.OPEN.SEGMX+5 file refnum
1012- 8D 3A 11 1410 STA P.READ.INDEX+1
1015- 8D 47 11 1420 STA P.READ.SEGNN+1
1018- 8D 42 11 1430 STA P.SET.MARK+1
101B- 1440 >MLI READ,P.READ.INDEX read the index
1450 *-----
1026- A5 06 1460 MAINLOOP
1028- 0A 1470 LDA FILE.CNT find offset to this seg
1029- 65 06 1480 ASL by multiplying by three
102B- AA 1490 ADC FILE.CNT
102C- BD 00 03 1500 TAX
102F- 8D 43 11 1510 LDA BUFFER.INDEX,X get starting point in file
1032- BD 01 03 1520 STA P.SET.MARK+2 for SETMARK call
1035- 8D 44 11 1530 LDA BUFFER.INDEX+1,X
1038- BD 02 03 1540 STA P.SET.MARK+3
103B- 8D 45 11 1550 LDA BUFFER.INDEX+2,X
103E- 0D 44 11 1560 STA P.SET.MARK+4
1041- 0D 43 11 1570 ORA P.SET.MARK+3 if 000000, no segment exists
1044- DO 03 1580 ORA P.SET.MARK+2
1046- 4C DF 10 1590 BNE .1 ...there is a segment
1049- BD 03 03 1600 JMP NEXT ...not one, go to next one
104C- 1D 04 03 1610 .1 LDA BUFFER.INDEX+3,X
104F- 1D 05 03 1620 ORA BUFFER.INDEX+4,X
1052- DO 03 1630 ORA BUFFER.INDEX+5,X
1054- 4C DF 10 1640 BNE .2
1650 JMP NEXT
1660 *---Compute length of segment---
1057- 38 1670 .2 SEC
1058- BD 03 03 1680 LDA BUFFER.INDEX+3,X start of next seg
105B- FD 00 03 1690 SBC BUFFER.INDEX,X - start of this
105E- 8D 4A 11 1700 STA P.READ.SEGNN+4 = length of this
1061- 8D 64 11 1710 STA P.WRIT.SEGNN+4
1064- BD 04 03 1720 LDA BUFFER.INDEX+4,X
1067- FD 01 03 1730 SBC BUFFER.INDEX+1,X
106A- 8D 4B 11 1740 STA P.READ.SEGNN+5
106D- 8D 65 11 1750 STA P.WRIT.SEGNN+5
1070- 0D 4A 11 1760 ORA P.READ.SEGNN+4 if length = 0000, go to next
1073- DO 03 1770 BNE .3
1075- 4C DF 10 1780 JMP NEXT
1790 *---Read the Segment---
1078- 800 .3 >MLI SETMARK,P.SET.MARK move to the segment
1083- 810 >MLI READ,P.READ.SEGNN read it into main BUFFER
1820 *---Compute Start Address---
108E- 38 1830 SEC
108F- AD 00 1A 1840 LDA BUFFER ending addr of this seg
1092- ED 4A 11 1850 SBC P.READ.SEGNN+4 - length of this segment
1095- AA 1860 TAX = equals starting addr
1096- AD 01 1A 1870 LDA BUFFER+1
1099- ED 4B 11 1880 SBC P.READ.SEGNN+5
109C- 8D 54 11 1890 STA P.CREATE+6
109F- 8A 1900 TXA round to an even page
10A0- FO 05 1910 BEQ .4
10A2- EE 54 11 1920 INC P.CREATE+6
10A5- A9 00 1930 LDA #0
10A7- 8D 53 11 1940 .4 STA P.CREATE+5

```

```

1950 *---Create the Segment File---
1960 >MLI CREATE,P.CREATE create an individual seg.xx
1970 >MLI OPEN,P.OPEN.SEGNN open a seg.xx
1980 LDA P.OPEN.SEGNN+5 refnum of file
1990 STA P.WRIT.SEGNN+1
2000 STA P.CLOSE+1
2010 >MLI WRITE,P.WRIT.SEGNN write out a seg.xx
2020 >MLI CLOSE,P.CLOSE close seg.xx
2030 *-----
2040 NEXT
10DF- EE 32 11 2050 INC SNUMLO
10E2- AD 32 11 2060 LDA SNUMLO
10E5- C9 BA 2070 CMP #9#+1
10E7- 90 08 2080 BCC .1
10E9- EE 31 11 2090 INC SNUMHI
10EC- A9 B0 2100 LDA #0#
10EE- 8D 32 11 2110 STA SNUMLO
10F1- E6 06 2120 .1 INC FILE.CNT
10F3- A5 06 2130 LDA FILE.CNT
10F5- C9 0A 2140 CMP #10
10F7- 90 06 2150 BCC .2
10F9- FO 07 2160 BEQ .3
10FB- C9 2C 2170 CMP #44
10FD- B0 0C 2180 BCS .4
10FF- 4C 26 10 2190 .2 JMP MAINLOOP
1102- EE 2B 11 2200 .3 INC NAME.M1
1105- 20 14 11 2210 JSR CLOSEALL
1108- 4C 04 10 2220 JMP PRELOOP
110B- 20 14 11 2230 .4 JSR CLOSEALL
2240 *-----
2250 ERROR
110E- AD 0F BF 2260 LDA ERRCODE
1111- 20 DA FD 2270 JSR $FDCA
2280 *-----
2290 CLOSEALL
1114- A9 00 2300 LDA #0
1116- 8D 69 11 2310 STA P.CLOSE+1
1119- 2320 >MLI CLOSE,P.CLOSE
1124- 60 2330 RTS
2340 *-----
1125- 2350 NAME.MX >STR "SEG.M0#"
112B- 2360 NAME.M1 .EQ #-1
2370 *-----
112C- 2380 NAME.NN >STR "SEG.01#"
1132- 2390 SNUMLO .EQ #-1
1131- 2400 SNUMHI .EQ #-2
2410 *-----
2420 * Parameter Lists for MLI Calls
2430 *-----
1133- 03 25 11
1136- 00 12 00 2440 P.OPEN.SEGMX .DA #3,NAME.MX,BUFFER.SEGMX,#0
1139- 04 00 00
113C- 03 8C 00
113F- 00 00 2450 P.READ.INDEX .DA #4,#00,BUFFER.INDEX,140,0000
1141- 02 00 00
1144- 00 00 2460 P.SET.MARK .DA #2,#00,<000000
1146- 04 00 00
1149- 1A 00 00
114C- 00 00 2470 P.READ.SEGNN .DA #4,#00,BUFFER,0000,0000
114E- 07 2C 11
1151- C3 06 00
1154- 00 00 00
1157- 00 00 00 2480 P.CREATE .DA #7,NAME.NN,#$C3,$$06,0000,#00,0000,0000
115A- 03 2C 11
115D- 00 16 00 2490 P.OPEN.SEGNN .DA #3,NAME.NN,BUFFER.SEGNN,#00
1160- 04 00 00
1163- 1A 00 00
1166- 00 00 2500 P.WRIT.SEGNN .DA #4,#00,BUFFER,0000,0000
1168- 01 00 2510 P.CLOSE .DA #1,#00
2520 *-----
2530 .DUMMY
2540 .BS #+255/256*256-# force to page boundary
116A- 2550 BUFFER.SEGMX .BS $400
1200- 2560 BUFFER.SEGNN .BS $400
1600- 2570 BUFFER .EQ #
1A00- 2580 .ED
2590 *-----

```

More on AuxTypes in ProDOS CATALOGs.....Bob Sander-Cederlof

Last month I presented a patch for BASIC.SYSTEM and another for SCASM.SYSTEM to cause the AuxType to be displayed for all file types, rather than just for TXT and BIN files. My patch was not quite sufficient, it turns out. My patch only worked for files whose type is a named type, displayed in the catalog with a three-letter name. Files with types displayed as a hexadecimal number still do not show the AuxType. Since I "tested" my patches in directories which only had named filetypes, I did not see the error.

That was the bad news; the good news is that this month I will complete the job, and give you another simple patch to cause the rest of the AuxTypes to be displayed.

In BASIC.SYSTEM you need to patch \$A4F6 (which was \$44) to \$0E. You can do this with a POKE 42230,14. To put it back, POKE 42230,68. Or to make both last month's and this new change, permanently, do this:

```
BLOAD BASIC.SYSTEM,TSYS,A$2000
POKE 12051,17
POKE 12022,14
BSAVE BASIC.SYSTEM,TSYS,A$2000
```

The corresponding change in SCASM.SYSTEM is a little more difficult, because of the various versions out there. If you found last month's location, this new one is a JMP instruction about 14 lines before that one. Disassembling around \$AFD0, you should see:

```
CA      DEX
CA      DEX
CA      DEX
CA      DEX
10 F5   BPL ...
20 XX XX JSR ...
4C XX XX JMP ...      patch the JMP address to point
CA      DEX           to the BIT below.
BD XX XX LDA ...,X
20 XX XX JSR ...
DO F7   BNE ...
here 2C XX XX BIT ...
```

You need to determine the address of the BIT instruction, and make the JMP jump to it. If you are with me this far, I am sure you can figure out the details.

New Version 1.2 of BASIC.SYSTEM.....Bob Sander-Cederlof

When I receive a new version of something from Apple, my first impulse is to try to find out exactly what they changed. Especially, when for the first time in four years they update a program so important as BASIC.SYSTEM. And especially when there have been excellent articles published in the last four years clearly describing definite bugs, patches, and work-arounds.

I was very disappointed this morning after carefully analyzing the new version 1.2 of BASIC.SYSTEM. I started by BLOADing the old version 1.1 and then copying it into bank 2 of my IIGs. Then I BLOADED the new version 1.2 and used the monitor V-command to compare the two. There were a total of 24 bytes changed. Thirteen were inside the parameter block for a Get-File-Info call, so their value is irrelevant. One is a byte that is never referenced in any way. Three bytes were changed in the title screen, so that you see "1.2" instead of "1.1", and "COPYRIGHT APPLE 1983-87" instead of "COPYRIGHT APPLE, 1983-84". That leaves only seven bytes in the total update whose change has any significance. They have not fixed even ONE of the many published problems in BASIC.SYSTEM!

So what did they fix? The description sheet that came with the update said they were trying to fix a bug in the CATALOG command. A variable they call TOTENT, which happens to be at BCB9-BCBA, is used for a counter to control the loop which displays files names and info. When the directory is first opened the total number of files in the directory is copied into TOTENT. The original intention of the programmer was to decrement TOTENT after reading each file entry in the directory. When the counter reaches "zero" the catalog should be finished. Unfortunately, the program did not decrement the counter properly.

To make matters worse, the new code in version 1.2 does not fix the original bug. Instead, the patch just omits testing TOTENT altogether. Now if you have a long directory, delete most of the files leaving just a few file names in the first few entries, and CATALOG it in BASIC.SYSTEM, it will read all of the entries anyway. No real problem, just spins the disk a fraction of a second longer.

The original bug was not a very serious problem either. It only failed when the total number of active files in a directory was a multiple of 256, which seldom happens. In fact, it seldom happens that there are that many files in any one directory, because so many of the utilities and even AppleWorks get confused with large directories. The symptom you would see if you had exactly 256 files in a directory, as I understand it, is that you would get an "OUT OF DATA" error message at the end of the catalog instead of the "number of blocks" line. I suppose that could be unnerving, so the bug should be removed if possible.

The faulty decrement code is at the end of the Read Next Catalog Entry subroutine, at \$B215, and looks like this:

```

B215- DEC $BCB9
B218- BNE $B21D
B21A- DEC $BCBA
B21D- RTS

```

If the initial number in BCB9 (low byte) and BCBA (high byte) is not a multiple of 256, this code will always result in \$BCBA going negative when the total value has been counted down. But if the initial value IS a multiple of 256, it will take an extra 256 times to count it down to a negative value in \$BCBA. The end-of-loop test code is at \$B09E:

```

B09E- LDA $BCBA
B0A1- BPL $B078

```

The correct way to decrement the 16-bit value is like this:

```

LDA $BCB9
BNE .1
DEC $BCBA
.1 DEC $BCB9

```

This code results in both bytes being zero when it is counted all the way down. Code to test the TOTENT variable for zero already exists at the top of the loop in BASIC.SYSTEM:

```

B070- LDA $BCB9
B073- ORA $BCBA
B076- BEQ $B0A3

```

A little re-structuring of the code would result in even fewer bytes being required to do the decrement and loop control correctly. Instead, we have this strange wipe-out instead. Apple went further, and changed the branch at \$B076 to two NOP's, and the error branch following the call to Read Next Catalog Entry to terminate the catalog without error. Very interesting. I wonder if they know something I don't? Maybe the value in the directory which we get TOTENT from is sometimes incorrect? Maybe it is sometimes 0000 when there are really files? Why else NOP-out the instruction at \$B076? Well, I have never yet noticed such a problem. Have you? Notice that, with these patches, if you get a disk error when reading a directory block CATALOG will terminate without reporting the error: you just will not see the rest of the files.

The description also claimed to fix a problem which caused the CATALOG to prematurely terminate if a <space> was pressed after a control-S. I have never noticed any such problem in the old version, and was unable to make it happen today. But sure enough, it doesn't fail that way in the new version either. After all, they didn't change any of that code anyway!

Why didn't Apple confer with Ken Kashmarek, Cecil Fretwell, Sandy Mossberg, Don Worth, Pieter Lechner, Dennis Doms or others who have been so carefully analyzing ProDOS and BASIC.SYSTEM over the last four years?

Anyway, after all the above is said, maybe you still wish you had version 1.2. If so, you can turn version 1.1 into 1.2 like this:

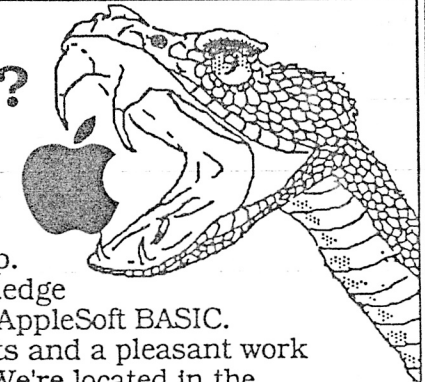
```

]BLOAD BASIC.SYSTEM,TSYS,A$2000
]CALL-151
*2282:B2           (was B1           )
*229A:A0           (was AC           )
*22A2:B7           (was B4           )
*3A76:EA EA       (was F0 28       )
*3A7C:26           (was 3A           )
*3A9E:EA A9 FF D0 (was AD BA BC 10)
*3D0G
]BSAVE BASIC.SYSTEM,TSYS,A$2000

```

The new version I have described above came on the newest IIgs system disk. This also included an update to ProDOS-8 called version 1.6. I personally never saw version 1.5, so I compared 1.6 to 1.4. There were way too many differences to determine what they really changed. Most of the differences appear to be due to a reassembly, so that the addresses of nearly everything internal have changed. A list of changes from 1.5 to 1.6 was included with the other documentation, but as I said I never saw version 1.5. Furthermore, after my experience with version 1.2 of BASIC.SYSTEM, I don't know what to believe. The one major fix they claim in 1.6 is only significant in the IIgs, and has to do with Desk Accessories determining whether ProDOS MLI was busy or not when the Desk Accessory was called up. If you are not using a IIgs, I think I would stick with version 1.4.

Do You Have Apple Knowledge?



If you do, Applied Engineering would like to put your knowledge to work. We're looking for someone to fill a position in our Technical Support group. You must have a strong working knowledge of AppleWorks, ProDOS, DOS 3.3, and AppleSoft BASIC. Applied Engineering offers good benefits and a pleasant work environment. Office is non-smoking. We're located in the Carrollton area.

So if you've got the knowledge and want to sink your teeth into a position with an ever-expanding company, give us a call at (214) 241-6060.

The Apple IIx Wish-O-Gram.....Jeff Creamer

I have decided to try doing my bit to help dig up the best ideas for what the next edition of the Apple II family ought to be. I'm sending the "Wish-O-Gram" to everyone I can think of who might have some influence on a new Apple II.

In the first Wish-O-Gram I gave my opinions that the the next Apple II should be packed with hardware "horsepower"; be able to emulate the Mac and/or the IBM PC; have standard provisions for 20+ meg hard disk; have an empty socket for a math coprocessor, or one built in; have video output compatible with EGA/VGA monitors.

If any of you readers of Apple Assembly Line have some thoughts on the subject, I would sure like to hear them. Send them to Jeff Creamer, Electronics Technology, Yavapai College, Prescott, AZ 86301.

EnterSoft:

Basic-like macros which make the complex simple. Don't re-write that multiplication routine for the hundredth time! Get EnterSoft instead! Do 8/16/32/64 bit Math/Input-Output/Graphics simply without all of the hassles. These routines are a must for the serious programmer who doesn't want to spend all of his/her time trying to re-invent the wheel. DOS 3.3 Version=\$30.00, ProDos Version=\$30.00, BOTH for only \$50.00. GET YOURS TODAY.

A Shape Table Program:

For once! A shape table program which is logically organized into its componet parts. Each section resides in its own program. The editor, disk access, Hi-Res section; each section is separate. Written almost entirely in Basic, it is easily modified. Not copyprotected! Put them on a Hard Disk, Ram Drive, anywhere! DOS 3.3 Version=\$20.00, ProDos Version=\$20.00, BOTH for \$30.00!

Send Check or Money Order To:	ProDos Upgrade for DOS 3.3 EnterSoft Owners - \$20.00
c/o Mark Manning Simulacron I/Baggy Game P.O. Box 58598 Webster, TX 77598	Thanks for the letters - Keep Writing!

Apple Assembly Line (ISSN 0889-4302) was published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228, from October, 1980 through May, 1988. Phone (214) 324-2050. This is the final issue. Back issues are \$1.80 each for Volumes 1-7, \$2.40 each from later Volumes (plus postage).

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)