MICRO

on the Apple

Volume 2

INCLUDES DISKETTE

MICRO on the Apple

MICRO on the Apple 2

Ford Cavallari, Editor

MICRO INK
Incorporated
P.O. Box 6502
Chelmsford, Massachusetts 01824

Notice

Cover Design and Graphics by Kate Winter

Every effort has been made to supply complete and accurate information. However, MICRO INK, Inc., assumes no responsibility for its use, nor for infringements of patents or other rights of third parties which would result.

MICRO on the Apple Series ISSN: 0275-3537
MICRO on the Apple Volume 2 ISBN: 0-938222-06-6
Printed in the United States of America
Printing 10 9 8 7 6 5 4 3 2 1
Floppy disk produced in the United States of America

To I.M.H.

Acknowledgements

The bulk of the credit for work done on this book goes out to exceptionally hard-working Special Projects Editor Marjorie Morse for her coordination of special projects operation, for her editing expertise, for her production and paste-up talent, and for her incredible patience with the MICRO-Lab operations. Since the compilation of this work also required extensive program generation and listing, I also wish to thank Darryl Wright, data entry specialist and precision programmer, for the hours of typing necessary to produce the diskette. Special thanks also go out to the entire MICRO staff, especially those who had direct involvement with this project. They are Emmalyn Bentley, the best typesetter in the hemisphere, and Paula Kramer, production professional. Thanks also to the publisher of MICRO, Robert Tripp, whose enthusiasm for MICRO made this project possible, to associate publishers Richard Rettig, for providing much advice on the entire *MICRO on the Apple* project, and Mary Grace Smith, for granting the spirit and autonomy needed to finish up this project. Finally, I wish to thank Ski, for being there again.

Contents

Introduction

MICRO magazine, the 6502/6809 Journal, has been offering software support to Apple users for over four years. With this book, we reaffirm our commitment to the Apple user, by presenting some of the most outstanding programs and articles which have appeared in MICRO over these years.

While MICRO continues to be the monthly source for new and innovative programs and articles, many of the older MICRO articles are still among the best material available for the Apple. Out of the pool of superb material, we have selected some of the best articles which we feel to be representative of MICRO, and have blended them together into this anthology.

MICRO has always catered to the serious computer user. Most of the pages in the magazine are filled with programs — programs which demonstrate some useful technique or perform some non-trivial task. This tradition of serious computing goes on at MICRO, and is reflected in this, the second volume, of *MICRO on the Apple*.

More than just another Apple book, *MICRO on the Apple 2* is an invaluable aid to the serious programmer, and a tool for the casual programmer to get serious with the Apple.

The *MICRO on the Apple* book series was conceived to distribute most effectively the wealth of Apple material available in MICRO. Each volume in the series brings together articles and programs, and presents them in logically defined chapters. All the material, even that which first appeared in early issues of MICRO, has been updated, either by the original author or by the MICRO staff. And all the programs related to these articles, whether Integer BASIC, Applesoft, or machine language, have been keyed-in, tested, and collected on a ready-to-use diskette.

This volume of the *MICRO on the Apple* series concentrates on the intermediate-to-advanced user, by presenting a host of indispensable aids for programming. The machine language utilities in the first chapter have been designed to ease the burden of 6502 programming. Similarly, the runtime utilities will facilitate advanced applications programming in Applesoft. The rest of the material in the book, from the recreational programs to the reference articles, all underscore the concept of good programming techniques.

Subsequent volumes of *MICRO on the Apple* will contain more comprehensive reference materials, tutorials, utilities, and applications programs, much of which will be original material not appearing in MICRO. MICRO magazine will maintain its monthly coverage of the Apple and the 6502. *MICRO on the Apple* will be the reference partner — the book you keep along with your reference manuals, next to your Apple.

Once again, a 13-sector diskette has been included with the book. The decision to include a 13-sector diskette was made because of the universal compatibility of 3.2 format and the large number of systems still without DOS 3.3. Through the use of Apple's MUFFIN program, this disk can easily be converted over to 3.3 format — and the programs will still work!

We hope that the approach which we have taken — collecting outstanding articles into a book and the accompanying programs onto a disk — will encourage the use of some of the routines you may have heard about but never had a chance to type in. We further hope that these routines afford you a chance to experiment with programming and explore some of the techniques and tricks explained in the articles. Lastly, we hope that *MICRO on the Apple 2* will give you the chance to catch up on the MICRO articles you might have missed, and will encourage you to check future issues of MICRO for the latest in sophisticated Apple material.

Ford Cavallari, Editor
October 1981

1
MACHINE LANGUAGE AIDS

INTRODUCTION

This chapter contains a group of utility programs designed to make machine language programming less tedious and less time consuming. Many of these utilities can work together, so the aspiring machine language programmer will be equipped with a formidable toolkit, indeed, after reading this chapter. ''Breaker: An Apple II Debugging Aid,'' by Richard Auricchio, facilitates the setting of break-points within programs, an invaluable capability for debugging large routines. ''Step and Trace for the Apple II Plus,'' by Craig Peterson, gives the Autostart Monitor ROM the stepping and tracing capabilities found only in the discontinued Old Monitor ROM. ''Tracer: A Debugging Tool for the Apple II,'' by R. Kovacs, enhances the step/trace capabilities of either your monitor or the Peterson program. These three routines form the debugging portion of the 'toolkit.'

Richard Suitor's ''Apple Integer BASIC Subroutine Pack and Load'' provides an easy method of binding machine language routines to Integer BASIC driver programs. This process can simplify program storage on either disk or tape. And, finally, R.M. Mottola's ''MEAN-14: A Pseudo-Machine Floating Point Processor for the Apple II'' provides a machine language alternative to Applesoft for floating point operations.

Breaker: An Apple II Debugging Aid

by Rick Auricchio

Machine language program development can often be speeded up through the use of breakpoints. While the Apple II does not have a breakpont capability built in, this program can provide that feature. Multiple breakpoints may be inserted into or deleted from any machine language program, in any place and at any time!

When debugging an Assembly language program, one of the easiest tools the programmer can use is the Breakpoint. In its most basic form, the Breakpoint consists of a hardware feature which stops the CPU upon accessing a certain address: a "deluxe" version might even use the Read/Write and Sync (instruction fetch) lines to allow stopping on a particular instruction, the loading of a byte, or the storing of a byte in memory. Since software is often easier to create than hardware (and cheaper for some of us!), a better method might be to implement the Breakpoint with software, making use of the BRK opcode of the 6502 CPU.

A Breakpoint, in practice is simply a BRK opcode inserted over an existing program instruction. When the user program's execution hits the BRK, a trip to the Monitor (via the IRQ vector $FFFE/FFFF) will occur. In the Apple, the Monitor saves the user program's status and registers, then prints the registers and returns control to the keyboard. The difficult part, however, comes when we wish to resume execution of the program: the BRK must be removed and the original instruction replaced, and the registers must be restored prior to continuing execution. If we merely replace the original opcode, however, the BRK will not be there should the program run through that address again.

The answer to this problem is BREAKER: a software routine to manage Breakpoints. What the debugger does is quite simple: it manages the insertion and removal of breakpoints, and it correctly resumes a user program after hitting a breakpoint. The original instruction will be executed automatically when the program is resumed!

Is it Magic?

No, it's not magic, but a way of having the computer remember where the breakpoints are! If the debugger knows where the breakpoints are, then it should also know what the original instruction was. Armed with that information, managing the breakpoints is easy. Here's how the debugger works.

During initialization, BREAKER is "hooked-in" to the APPLE monitor via the Control-Y user command exit, and via the COUT user exit. The control-Y exit is used to process debugger commands, and the COUT exit is used to "steal control" from the Monitor when a BRK occurs.

Breakpoint information is kept in tables: the LOCTAB is a table of 2-byte addresses—it contains the address at which a breakpoint has been placed. The ADTAB is a table of 1-byte low-order address bytes: it is used to locate a Break Table Entry (BTE). The BTE is 12 bytes long (only the first 9 are used, but 12 is a reasonably round number) and it contains the following items:

* Original user-program instruction
* JMP back to user-program
* JMP back for relative branch targets

When adding a breakpoint, we must build the BTE correctly, and place the user-program break address into the LOCTAB. There are eight (8) breakpoints allowed, so that we have a 16-byte LOCTAB, 8-byte ADTAB, and 96 bytes of BTE's.

As the breakpoint is added, the original instruction is copied to the first 3 bytes of the BTE, and it is "padded" with NOP instructions ($EA) in case it is a 1-or 2-byte instruction. A BRK opcode ($00) is placed into the user program in place of the original instruction's opcode (other instruction bytes are not altered). The next 3 bytes of the BTE will contain a JMP instruction back to the next user-program instruction.

If the original instruction was a Relative Branch, one more thing must be considered: if we remove the relative branch to the BTE, how will it branch correctly? This problem is solved by installing another JMP instruction into the BTE for a relative branch—back to the Target of the branch, which is computed by adding the original PC of the branch, +2, +offset. This Absolute address will be placed into the JMP at bytes 7-9 of the BTE. The offset which was copied from the original instruction will be changed to $04 so that it will now branch to that second JMP instruction within the BTE; the JMP will get us to the intended target of the original Relative Branch.

A call to the routine "INSDS2" in the Monitor returns the length and type of instruction for the "add" function. The opcode is supplied in the AC, and LENGTH & FORMAT are set appropriately by the routine.

Removal of a breakpoint involves simply restoring the original opcode, and clearing the LOCTAB to free this breakpoint's BTE.

Displaying of breakpoint prints the user-program address of a breakpoint, followed by the address of the BTE associated with the breakpoint (the BTE address is useful—its importance will be described later).

When the breakpoint is executed, a BRK occurs and the Apple Monitor gets control. The monitor will "beep" and print the user program's registers. During printing of the registers, BREAKER will take control via the COUT exit. (Remember, we get control on every character printed - but it's only important when the registers are being printed. That's when we're at a breakpoint). While it has control, BREAKER will grab the user-program's PC and save it (we must subtract 2 because of the action of the BRK instruction). If no breakpoint exists at this PC (we scan LOCTAB), then the Monitor is continued. If a breakpoint does exist here, then the BTE address is set as the "continue PC". In other words, when we continue the user program after the break, we will go to the BTE; the original instruction will now be executed, and we will branch back to the rest of the user program.

Using Breaker

The first thing to do is to load BREAKER into high memory. It must then be initialized via entry at the start address. This sets up the exits from the Monitor. After a Reset, you must re-initialize via "YcI" (Yc is Control-Y) to set up the COUT exit again. Upon entry at the start address, all breakpoints are cleared: after 'YcI", they remain in effect.

To add a breakpoint, type: aaaaYcA . This will add a breakpoint at address 'aaaa' in the user program. A 'beep' indicates an error; you already have a breakpoint at that address. To remove a breakpoint, type: aaaaYcR. This will remove the breakpoint at address 'aaaa' and restore the original opcode. A 'beep' means that there was none there to start with.

Run your user-program via the Monitor's "G" command. Upon hitting a breakpoint, you will get the registers printed, and control will go back to the monitor as it does normally. At this point, all regular Monitor commands are valid, including "YcA", "YcR", and "YcD" for BREAKER.

To continue execution type: YcG . This instructs BREAKER to resume execution at the BTE (to execute the original instruction), then to transfer control back to the user program. Do not resume via Monitor "G" command—it won't work properly, since the monitor knows nothing of breakpoints. To display all breakpoints, type: YcD. This will give a display of up to 8 breakpoints, with the address of the associated BTE for each one.

Caveats

Some care must be taken when using BREAKER to debug a program. First, there is the case of BREAKER not being initialized when you run the user program. This isn't a problem when you start, because you'll not be able to use the Yc commands. But if you should hit Reset during testing, you must re-activate via "YcI", otherwise BREAKER won't get control on a breakpoint. If you try a YcG, unpredictable things will happen. If you know that you hit a breakpoint while BREAKER was not active, you can recover. Simply do a "YcI", and then display the breakpoints (YcD). Resume the user-program by issuing a Monitor "G" command to the BTE for the breakpoint that was hit (since BREAKER wasn't around when you hit the breakpoint, you have to manually resume execution at the BTE). Now all is back to normal. You can tell if BREAKER is active by displaying locations $38 and $39. If not active, they will contain $F0 FD.

It's also important to note that any user program which makes use of either the Control-Y or COUT exits can't be debugged with BREAKER. Once these exits are changed, BREAKER won't get control when it's supposed to.

BREAKER Command Summary

Command	**Function**
aaaa Yc A	Add breakpoint at location aaaa. Won't allow you to add one over an already existing breakpoint. Maximum of 8 breakpoints allowed.
YC D	Display all breakpoints.
Yc I	Initialize after RESET key. Just sets up 'COUT' exit again without resetting any breakpoints.
aaaa Yc R	Remove breakpoint from location aaaa. Restores original opcode.

```
0800               ;************************
0800               ;*                      *
0800               ;*   BREAKER-DEBUGGER   *
0800               ;*    RICK AURICCHIO    *
0800               ;*                      *
0800               ;*       BREAKER        *
0800               ;*                      *
0800               ;*  COPYRIGHT (C) 1981  *
0800               ;*    MICRO INK, INC.   *
0800               ;* CHELMSFORD, MA 01824 *
0800               ;*  ALL RIGHTS RESERVED *
0800               ;*                      *
0800               ;************************
0800               ;
0800               ;ROUTINES TO HANDLE UP TO
0800               ;8 BREAKPOINTS, FOR USE IN
0800               ;DEBUGGING OF USER CODE...
0800               ;
0800               ;
0800               ; *** APPLE-2 MONITOR EQUATES
0800               ;
0800               FORMAT EPZ $2E
0800               LENGTH EPZ $2F
0800               A1L    EPZ $3C
0800               A1H    EPZ $3D
0800               A2L    EPZ $3E
0800               A2H    EPZ $3F
0800               A3L    EPZ $40
0800               A3H    EPZ $41
0800               ;
0800               CSWL   EPZ $36
0800               CSWH   EPZ $37
0800               ;
0800               INSDS2 EQU $F88E
0800               PRNTYX EQU $F940
0800               PRBYTE EQU $FDDA
0800               COUT   EQU $FDED
0800               RESET  EQU $FF65
0800               MON    EQU $FF69
0800               ;
0800               ; CHANGE 'LOWPAGE' TO LOCATE
0800               ; ELSEWHERE IN MEMORY. IT IS
0800               ; NOW SET FOR A 48K DOS SYSTEM.
0800               ;
0800               LOWPAG EQU $9300
0800               ;
9300                      ORG LOWPAG
9300                      OBJ $800
9300               ;
9300 4C3695        INIT   JMP INITX            ;=>INITIALIZATION ENTRY
9303               ;
9303               ; *** DATA AREAS
9303               ;
9303 00            FW1    BYT $00              ; 'FINDPC' WORK BYTE 1
9304 00            FW2    BYT $00              ; 'FINDPC' WORK BYTE 2
9305 00            PCL    BYT $00              ; 'GO' PC LO
9306 00            PCH    BYT $00              ; 'GO' PC HI
9307               ;
9307               ; SKELETON BREAK-TABLE ENTRY
9307               ;
9307 00            SKEL   BYT $00              ;SKELETON BTE
9308 EA                   NOP
9309 EA                   NOP                  ;NOPS FOR PADDING
930A 4C0000               JMP $00              ;JUMP BACK INLINE
930D 4C                   BYT $4C              ;JUMP OPCODE FOR BRANCHES
930E               ;
930E               ; LOW ADDRESS OF BTES KEPT IN ADTAB
930E               ;
930E 26            ADTAB  BYT BTE0             ;LO ADDRESS
```

```
930F 32               BYT BTE1
9310 3E               BYT BTE2
9311 4A               BYT BTE3
9312 56               BYT BTE4
9313 62               BYT BTE5
9314 6E               BYT BTE6
9315 7A               BYT BTE7
9316            ;
9316            ; LCCTAB CCNTAINS ADDRESS OF USER-PROGRAM INSTRUCTION
9316            ; WHERE WE PLACED THE BREAKPOINT IN THE FIRST PLACE
9316            ;
0826            LOCTAB DFS $10             ;SPACE FOR 16 PCH/L PAIRS
9326            ;
9326            ; BREAK-TABLE ENTRIES (BTE'S)
9326            ;
0832            BTE0   DFS $0C             ;12 BYTES RESERVED
083E            BTE1   DFS $0C
084A            BTE2   DFS $0C
0856            BTE3   DFS $0C
0862            BTE4   DFS $0C
086E            BTE5   DFS $0C
087A            BTE6   DFS $0C
0886            BTE7   DFS $0C             ;ENOUGH FOR 8 BREAKPOINTS
9386            ;
9386            ; END OF DATA AREAS
9386            ; *THE REST IS ROM-ABLE*
9386            ;
9386            ; ***********************************************
9386            ; *
9386            ; * NAME:    FINDPC
9386            ; * PURPOSE: CHECK IF PC IN FW1/FW2 MATCHES LOCTAB
9386            ; * RETURNS: CARRY SET IF YES; XREG=ADTAB INDEX 0-7
9386            ; *          CARRY CLR IF NOT; XREG=GARBAGE
9386            ; * VOLATILE:DESTROYS AC
9386            ; *
9386            ; ***********************************************
9386            ;
9386 A20F       FINDPC LDX #!15            ;BYTE-INDEX TC END OF TABLE
9388 AD0493     FPC00  LDA FW2             ;GET FOR COMPARE
938B DD1693            CMP LOCTAB,X        ;A PCH MATCH?
938E D008              BNE FPC02           ;=>NO. TRY NEXT2-BYTE ENTRY
9390 AD0393            LDA FW1             ;GET PCL NOW
9393 DD1593            CMP LOCTAB-1,X      ;A PCL MATCH?
9396 F006              BEQ FPC04           ;=>YES! WE HAVW BREAKPOINT!
9398 CA         FPC02  DEX                 ;BACK UP ONE
9399 CA                DEX                 ;AND ANOTHER
939A 10EC              BPL FPC00           ;=>DO ENTIRE TABLE SCAN
939C 18                CLC                 ;=>DONE; SCAN FAILED
939D 60                RTS
939E            ;
939E 48         FPC04  PHA                 ;HOLD AC
939F 8A                TXA                 ;HALVE VALUE IN X-REG
93A0 4A                LSR                 ;SINCE IT'S 2-BYTE INDEX
93A1 AA                TAX
93A2 68                PLA
93A3 38                SEC                 ;SET 'SUCCESS'
93A4 60
93A5            ;
93A5            ;
93A5            ; ***********************************************
93A5            ; *
93A5            ; * NAME:    BREAK
93A5            ; * PURPOSE: HANDLE ENTRY AT BRK AND PROCESS BRKPOINTS
93A5            ; * NOTE:    THIS ROUTINE GETS ENTERED ON *EVERY* 'COUT'
93A5            ; *          CALL-- IT KNOWS ABOUT BRK BECAUSE THE
93A5            ; * MONITOR'S REGISTERS ARE SETUP TO PRINT USER REG
93A5            ; * CONTENTS. AFTER PROCESSING IS DONE, IT RESTORESTHE
93A5            ; * MONITOR'S REGS AND RETURNS
93A5            ; *
93A5            ; ***********************************************
```

```
93A5               ;
93A5 E0FB          BREAK  CPX #$FB             ;IS XREG SET FOR EXAMINE
93A7 D027                 BNE BRKXX            ;=>NO GET OUT NOW.
93A9 C9A0          BRK02  CMP #$A0             ;IS AC SETUP CORRECTLY
93AB D023                 BNE BRKXX            ;=>NOPE. FALSE ALARM!
93AD A53C                 LDA A1L              ;GET USER PCL
93AF 38                   SEC                  ; AND BACK IT UP BY
93B0 E902                 SBC #$02             ; 2 BYTES SINCE BRK BUMPED
93B2 8D0393               STA FW1
93B5 A53D                 LDA A1H              ;GET PCH
93B7 E900                 SBC #$00             ;DO THE CARRY
93B9 8D0493               STA FW2              ; AND SAVE THAT TOO
93BC 208693               JSR FINDPC           ;A BREAKER OF OURS HERE?
93BF 900B                 BCC BRK04            ;=>NOPE. WE WON'T HANDLE
93C1 BD0E93               LDA ADTAB,X          ;YES, GET BTE ADDRESS THEN
93C4 8D0593               STA PCL              ; AND SET IT AS THE 'GO'
93C7 A993                 LDA /LOWPAG          ; PC FOR THE 'GO' COMMAND.
93C9 8D0693               STA PCH              ;(OUR PAGE FOR BTE'S)
93CC               ;
93CC A9A0          BRK04  LDA #$A0             ;SET AC BACK FOR MONITOR
93CE A2FB                 LDX #$FB             ;AND X-REG. TOO
93D0 4CF0FD        BRKXX  JMP $FDF0            ;=>NO. RIGHT BACK TO COUT
93D3               ;
93D3               ; ************************************
93D3               ; * PROCESS THE 'GO' COMMAND....    *
93D3               ; *     (RESUME USER EXECUTION)      *
93D3               ; * COMMAND FORMAT:(* CTRL-Y G)      *
93D3               ; ************************************
93D3               ;
93D3 AD0593        CMDGO  LDA PCL              ;GET RESUME PCL
93D6 853C                 STA A1L              ; AND SETUP TO SIMULATE
93D8 AD0693               LDA PCH              ; AN 'XXXX G' COMMAND
93DB 853D                 STA A1H
93DD 4CB9FE               JMP $FEB9            ;=>SAIL INTO MONITOR'S 'GO'
93E0               ;
93E0               ; ************************************
93E0               ; * WE GET CONTROL HERE ON THE       *
93E0               ; * CNTL-Y USER EXIT FROM THE        *
93E0               ; * MONITOR (ON KEY-INS).  ALL       *
93E0               ; * COMMANDS ARE SCANNED HERE;       *
93E0               ; * CONTROL WILL PASS TO THE         *
93E0               ; * APPROPRIATE ROUTINE.....         *
93E0               ; ************************************
93E0               ;
93E0 A2FF          KEYIN  LDX #$FF             ;CHAR INDEX
93E2 E8            KEYIN0 INX                  ;SET NEXT CHARACTER
93E3 BD0002               LDA $200,X           ;GET CHARACTER FROM BUFFER
93E6 C999                 CMP #$99             ;CONTROL-Y CHARACTER?
93E8 D0F8                 BNE KEYIN0           ;=>NO. KEEP SCANNING
93EA E8                   INX                  ;BUMP OVER CTRL-Y
93EB BD0002               LDA $200,X           ;GRAB COMMAND CHARACTER
93EE C9C7                 CMP #$C7             ;IS IT 'G' (GO)?
93F0               ;
93F0               ; A BRANCH TABLE WOULD BE NEATER,
93F0               ; BUT IT WOULD TAKE UP MORE CODE
93F0               ; FOR THE FEW OPTIONS WE HAVE...
93F0               ;
93F0 F0E1                 BEQ CMDGO            ;=>YES.
93F2 C9C1                 CMP #$C1             ;IS IT 'A' (ADD)?
93F4 F018                 BEQ CMDADD           ;=>YES.
93F6 C9C4                 CMP #$C4             ;IS IT 'D' (DISPLAY)?
93F8 F00B                 BEQ XXDISP           ;=>YES.
93FA C9D2                 CMP #$D2             ;IS IT 'R' (REMOVE)?
93FC F00A                 BEQ XXREMV           ;=>YES.
93FE C9C9                 CMP #$C9             ;IS IT 'I' (INIT)?
9400 F009                 BEQ XXINIT           ;=>YES.
9402 4C65FF        BADCMD JMP RESET            ;NOTHING, IGNORE IT!
9405               ;
9405 4CA894        XXDISP JMP CMDDSP           ;EXTENDED BRANCH
```

```
9408 4C0895   XXREMV JMP CMDRMV          ;EXTENDED BRANCH
940B 4C4F95   XXINIT JMP CMDINT          ;EXTENDED BRANCH
940E          ;
940E          ; **********************************
940E          ; *   PROCESS THE 'ADD' COMMAND    *
940E          ; * ADD A BREAKPOINT AT LOCATION   *
940E          ; * SPECIFIED IN COMMANDD.....     *
940E          ; * CMND FORMAT: (* AAAA CTRL-Y A)*
940E          ; **********************************
940E          ;
940E A000     CMDADD LDY #$00            ;CHECK OPCODE FIRST
9410 B13E            LDA (A2L),Y         ;OP AT AAAA A BRK ALREADY?
9412 F0EE            BEQ BADCMD          ;=>YES. ILLEGAL!
9414          ;
9414          ;SCAN LOCTAB FOR AN AVAILABLE BTE TO USE
9414          ;
9414 A20F            LDX #!15            ;BYTE INDEX TO LOCTAB END
9416 BD1693   ADD00  LDA LOCTAB,X        ;GET A BYTE
9419 D005            BNE ADD02           ;=>IN USE
941B BD1593          LDA LOCTAB-1,X      ;GET HI HALF
941E F006            BEQ ADD04           ;=>BOTH ZERO, USE IT!
9420 CA       ADD02  DEX                 ;MOVE BACK TO
9421 CA              DEX                 ; NEXT LOCTAB ENTRY
9422 10F2            BPL ADD00           ; AND KEEP TRYING
9424 30DC            BMI BADCMD          ;=>DONE? ALL FULL! REJECT
9426          ;
9426 A53E     ADD04  LDA A2L             ;GET AAAA VALUE
9428 9D1593          STA LOCTAB-1,X      ;SAVE LO HALF
942B 8D0B93          STA SKEL+4          ;STUFF LO ADDR INTO BTE
942E A53F            LDA A2H             ;GET AAAA VALUE
9430 9D1693          STA LOCTAB,X        ;SAVE HI HALF
9433 8D0C93          STA SKEL+5          ;STUFF HI ADDRESS INTO BTE
9436 8A              TXA                 ;GRAB INDEX FOR LOCTAB
9437 4A              LSR                 ;MAKE ADTAB INDEX
9438 AA              TAX                 ;AND STUFF BACK INTO X-REG
9439 A993            LDA /LOWPAG         ;BTE'S HI ADDRESS VALUE
943B 8541            STA A3H             ;HOLD IN WORK AREA
943D BD0E93          LDA ADTAB,X         ;GET BTE LO ADDR FROM ADTAB
9440 8540            STA A3L             ;SAVE IN WORK AREA
9442 A907            LDA #$07            ;7-BYTE MOVE FOR SKEL BTE
9444 B90793   ADD06  LDA SKEL,Y          ;GET SKEL BYTE
9447 9140            STA (A3L),Y         ;MOVE TO BTE
9449 88              DEY                 ;SET NEXT
944A 10F8            BPL ADD06           ;=> MOVE ENTILE SKELETON
944C C8              INY
944D B13E            LDA (A2L),Y         ;GET ORIGINAL OPCODE
944F 9140            STA (A3L),Y         ; INTO BTE......
9451 208EF8          JSR INSDS2          ;INSDS2 (TO DISASSEMBLE)
9454 A900            LDA #$00            ;SET BRK OPCODE
9456 913E            STA (A2L),Y         ; OVER ORIGINAL OPCODE
9458 A52F            LDA LENGTH          ;GET INSTRUCTION LENGTH
945A 38              SEC
945B          ;
945B          ; SET UP JMP TO NEXT INST. IN THE BTE
945B          ;
945B A004            LDY #$04
945D 7140            ADC (A3L),Y         ;ADD TO PC FOR DESTINATION
945F 9140            STA (A3L),Y         ;STUFF INTO BTE
9461 C8              INY
9462 B140            LDA (A3L),Y         ;RUN UP THE CARRY
9464 6900            ADC #$00            ; RIGHT HERE
9466 9140            STA (A3L),Y         ;STUFF ADDRESS INTO JMP
9468 A52E            LDA FORMAT          ;GET INSTRUCTION FORMAT
946A C99D            CMP #$9D            ;IS FORMAT=BRANCH
946C F016            BEQ ADDBRCH         ;=>YES. MORE TO DO
946E A52F            LDA LENGTH          ;LENGTH=1?
9470 F00F            BEQ CMDRET          ;=>YES. DONE
9472 6A              ROR                 ;LENGTH=2?
9473 B006            BCS ADDLEN2         ;=>YES
```

```
9475 A002           LDY #$02             ;LENGTH=3, 3RD BYTE TO BTE
9477 B13E           LDA (A2L),Y          ;GET INST 3RD BYTE
9479 9140           STA (A3L),Y          ;AND MOVE TO BTE
947B A001    ADDLEN LDY #$01             ;LENGTH=2, 2ND BYTE TO BTE
947D B13E           LDA (A2L),Y          ;GET INST 2ND BYTE
947F 9140           STA (A3L),Y          ;AND MOVE TO BTE
9481 4C69FF  CMDRET JMP MON              ;DONE, BACK TO MONITOR
9484         ;
9484         ; FOR BRANCHES, WE'VE GOTTA ADD A JMP FOR THE 'TRUE'
9484         ; CONDITION (SINCE WE MOVED BRANCH OUT OF PROGRAM)
9484         ;
9484 A001    ADDBRC LDY #$01             ;SET FOR 2ND BYTE
9486 B13E           LDA (A2L),Y          ;GET DESTINATION OFFSET
9488 18             CLC                  ;AND ADD 2 BYTES TO
9489 6902           ADC #$02             ;CONSTRUCT ABS ADDRESS
948B 653E           ADC A2L              ;ADD TO SUBJECT-INST
948D 853E           STA A2L
948F A53F           LDA A2H              ;CARRY IT
9491 6900           ADC #$00
9493 853F           STA A2H
9495 EA             NOP                  ;(PLACE HOLDER WASTE HERE)
9496 A904           LDA #$04             ;TRUE BRANCH TO +4
9498 9140           STA (A3L),Y          ;PUT INTO NEW OFFSET
949A A007           LDY #$07
949C A53E           LDA A2L              ;GET JMP ADDRESS
949E 9140           STA (A3L),Y          ;MOVE IT TO
94A0 C8             INY                  ;THE
94A1 A53F           LDA A2H              ; BTE FOR
94A3 9140           STA (A3L),Y          ; THE 'TRUE' BRANCH
94A5 B8             CLV                  ;SNEAKY BRANCH
94A6 50D9           BVC CMDRET           ; TO EXIT...
94A8         ;
94A8         ; *********************************
94A8         ; * DISPLAY ALL ACTIVE BRKPOINTS *
94A8         ; *  COMMAND FMT: (* CTRL-Y D)    *
94A8         ; *********************************
94A8         ;
94A8 A20F    CMDDSP LDX #!15             ;INDEX TO LOCTAB END
94AA BD1693  DISP00 LDA LOCTAB,X         ;GET A BYTE
94AD D00B           BNE DISP04           ;=>IN USE
94AF BD1593         LDA LOCTAB-1,X       ;TRY BOTH BYTES TO BE SURE
94B2 D006           BNE DISP04           ;=>DEFINITELY IN USE
94B4 CA      DSPNXT DEX                  ;SET NEXT ENTRY
94B5 CA             DEX                  ; IN LOCTAB
94B6 10F2           BPL DISP00           ;=>MORE TO GO
94B8 30C7           BMI CMDRET           ;=>DONE: EXIT TO MONITOR
94BA         ;
94BA A98D    DISP04 LDA #$8D             ;OUTPUT A CARRIAGE
94BC 20EDFD         JSR COUT             ; RETURN
94BF 8A             TXA                  ;GET INDEX
94C0 48             PHA                  ; SAVE IT
94C1 BC1693         LDY LOCTAB,X         ;GET SUBJECT-INST PCH
94C4 BD1593         LDA LOCTAB-1,X       ; AND ITS PCL
94C7 843B           STY $3B              ;SET UP PCH/PCL
94C9 853A           STA $3A
94CB AA             TAX
94CC 2040F9         JSR PRNTYX           ;PRINT Y,X BYTES IN HEX
94CF 68             PLA                  ;RESTORE INDEX
94D0 48             PHA
94D1 4A             LSR                  ;CONVERT TO ADTAB INDEX
94D2 AA             TAX
94D3 A9BC           LDA #$BC             ;'<' CHARACTER
94D5 20EDFD         JSR COUT             ;PRINT IT
94D8 A993           LDA /LOWPAG          ;BTE HI ADDRESS
94DA 853F           STA A2H              ;SET INDIRECT POINTER
94DC 20DAFD         JSR PRBYTE           ;PRINT HEX BYTE
94DF BD0E93         LDA ADTAB,X          ;GET BTE LOW ADDRESS
94E2 853E           STA A2L              ;SET INDIRECT POINTER
94E4 20DAFD         JSR PRBYTE           ;PRINT BTE FULL ADDRESS
```

```
94E7 A9BE           LDA #$BE         ;'>' CHARACTER
94E9 20EDFD         JSR COUT         ;PRINT IT
94EC          ;
94EC          ; DISSASSEMBLE THE ORIGINAL INSTRUCTION.
94EC          ; PICK UP ORIGINAL OPCODE FROM BTE,
94EC          ; ORIGINAL ADDRESDS FIELD FROM USER
94EC          ; PROGRAM LOCATION........
94EC          ;
94EC A9A0           LDA #$A0         ;PRINT ONE SPACE HERE
94EE 20EDFD         JSR COUT
94F1 A000           LDY #$00         ;INDEX
94F3 B13E           LDA (A2L),Y      ;GET OPCODE FROM BTE
94F5 20DAFD         JSR PRBYTE       ;PRINT OPCODE
94F8 B13E           LDA (A2L),Y      ;GET OPCODE FROM BTE
94FA 208EF8         JSR INSDS2       ; AND GET FORMAT/LENGTH
94FD 200495         JSR KLUGE        ;SNEAK INTO INSDSP @ F8D9
9500 68             PLA
9501 AA             TAX              ;RESTORE LOCTAB INDEX
9502 10B0           BPL DSPNXT       ;=>DISPLAY THE REST
9504          ;
9504          ;
9504          ; KLUGE ENTRY INTO SUBROUTINE WHICH
9504          ; FORCES JSR PRIOR TO A PHA INSTRUCTION.
9504          ; WE HAVE TO JSR TO THIS JMP!!
9504          ;
9504 48       KLUGE PHA              ;PUSH MNEMONIC INDEX
9505 4CD9F8         JMP $F8D9        ;CONTINUE WITH INSTDSP
9508          ;****END OF KLUGE****
9508          ;
9508          ;
9508          ; **********************************
9508          ; * REMOVE A BRKPOINT AT LOC AAAA *
9508          ; *  COMMAND FMT: (AAAA CTL-Y RR) *
9508          ; **********************************
9508          ;
9508 A53E     CMDRMV LDA A2L         ;GET ADDRESS LO
950A 8D0393         STA FW1          ;HOLD IT FOR FINDPC
950D A53F           LDA A2H          ;GET ADDRESS HI
950F 8D0493         STA FW2
9512 208693         JSR FINDPC       ;A BRKPOINT HERE??
9515 B003           BCS REMOV2       ;=>YES.
9517 4C65FF         JMP RESET        ;=>NO, BELL FOR YOU!
951A          ;
951A BD0E93   REMOV2 LDA ADTAB,X     ;GET THE LOCTAB ENTRY
951D 8540           STA A3L          ;HOLD IT
951F 8A             TXA              ;NOW CREATE LOCTAB INDEX
9520 0A             ASL
9521 AA             TAX
9522 A900           LDA #$00         ;CLEAR OUT THE APPROPRIATE
9524 A8             TAY              ;LOCTAB ENTRY FOR BKPT..
9525 9D1693         STA LOCTAB,X
9528 9D1793         STA LOCTAB+1,X
952B A993           LDA /LOWPAG      ;HI ADDRESS FOR BTE
952D 8541           STA A3H          ;HOLD FOR ADDRESSING
952F B140           LDA (A3L),Y      ;GET OPCODE OUT OF BTE
9531 913E           STA (A2L),Y      ; AND PULL BACK TO ORIGINAL
9533 4C69FF         JMP MON          ;=>ALL DONE.
9536          ;
9536          ;
9536          ; ******************************************
9536          ; *
9536          ; * INITIALIZATION CODE. ENTERED AT START
9536          ; * ADDR TO INITIALIZE.  IT CLEARS LOCTAB,
9536          ; * SETS UP THE CTL-Y AND COUT EXITS...
9536          ; *
9536          ; * AFTER EVERY RESET, MUST RESETUP WITH
9536          ; *        * CTL-Y I
9536          ; *
9536          ; ******************************************
```

```
9536           ;
9536 A94C      INITX  LDA #$4C         ;JMP CPCODE
9538 8DF803           STA $3F8         ;STUFF IN CTL-Y EXIT LOC
953B A993             LDA /KEYIN       ;KEYIN: HI ADDRESS
953D 8DFA03           STA $3FA         ;STUFF INTO JMP
9540 A9E0             LDA #KEYIN       ;KEYIN: LO ADDRESS
9542 8DF903           STA $3F9         ;STUFF INTO JMP ADDRESS
9545 A900             LDA #$00
9547 A20F             LDX #!15         ;INDEX INTO LOCTAB END
9549 9D1693    INIT00 STA LOCTAB,X     ;CLEAR IT OUT
954C CA               DEX              ;SO NO BREAKPOINTS
954D 10FA             BPL INIT00
954F           ;
954F           ; ENTER HERE AFTER HITTING RESET, PLEASE!
954F           ;
954F A9A5      CMDINT LDA #BREAK       ;BREAK: LO ADDRESS
9551 8536             STA CSWL         ;STUFF INTO 'COUT' EXIT HOOK
9553 A993             LDA /BREAK       ;BREAK: HI ADDRESS
9555 8537             STA CSWH         ;STUFF INTO 'COUT' EXIT HOOK
9557 4C69FF           JMP MON
                      END

***** END OF ASSEMBLY


           *************************
           *                       *
           * SYMBOL TABLE -- V 1.5 *
           *                       *
           *************************

LABEL. LOC.   LABEL. LOC.   LABEL. LOC.

** ZERO PAGE VARIABLES:

FORMAT 002E   LENGTH 002F   A1L    003C A1H    003D A2L    003E A2H    003F
A3L    0040   A3H    0041   CSWL   0036 CSWH   0037

** ABSOLUTE VARABLES/LABELS

INSDS2 F88E   PRNTYX F940
PRBYTE FDDA   COUT   FDED   RESET  FF65 MON    FF69 LOWPAG 9300 INIT   9300
FW1    9303   FW2    9304   PCL    9305 PCH    9306 SKEL   9307 ADTAB  930E
LOCTAB 9316   BTE0   9326   BTE1   9332 BTE2   933E BTE3   934A BTE4   9356
BTE5   9362   BTE6   936E   BTE7   937A FINDPC 9386 FPC00  9388 FPC02  9398
FPC04  939E   BREAK  93A5   BRK02  93A9 BRK04  93CC BRKXX  93D0 CMDGO  93D3
KEYIN  93E0   KEYIN0 93E2   BADCMD 9402 XXDISP 9405 XXREMV 9408 XXINIT 940B
CMDADD 940E   ADDC0  9416   ADD02  9420 ADD04  9426 ADD06  9444 ADDLEN 947B
CMDRET 9481   ADDBRC 9484   CMDDSP 94A8 DISP00 94AA DSPNXT 94B4 DISP04 94BA
KLUGE  9504   CMDRMV 9508   REMOV2 951A INITX  9536 INIT00 9549 CMDINT 954F

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0222
```

Step and Trace for the Apple II Plus

by Craig Peterson

If you miss the Step/Trace of the original Apple II on your new Apple II Plus, here is all you need to restore it.

Apple Computer's Apple II Plus is a pretty good machine. It has improved editing features over those of the standard Apple II and a better cursor control and stop list feature. And it's really nice to fire up the machine and be right in BASIC or DOS, or better yet, to be in the middle of a turn-key type program.

Furthermore, Applesoft BASIC is a standard feature, and I'm partial to it over Integer BASIC. But all of these improvements didn't come for free. There's only so much room in the ROM monitor, and certain of its features had to be sacrificed to make room for the new additions. As a result, the machine language Step/Trace capabilities of the older Apple II ended up on the cutting room floor.

A lot of people will probably never miss Step/Trace. Unless you are into assembly language programming, you probably don't need them. But if you do any assembly language programming, Step/Trace can be invaluable. They allow you to step through each machine language instruction, displaying all of the 6502 registers as you go along, so you can find any errors that might exist in the program, or even just see how the program works. Step does this one instruction at a time, and Trace does it continuously, without stopping (unless a BRK instruction is encountered).

Step-n-Trace Program

Well, fear not, Apple II Plus owners, Step-n-Trace is here. The Step-n-Trace (S&T) program essentially just adds the step-and-trace functions to the existing monitor of your Apple II Plus. The operation and use of the monitor is identical to that of the original Apple monitor. Type a hex address followed by one or more 'S's, to take steps through a program from that address. To trace from that address, type a hex address followed by a 'T'.

An improved feature of S&T over the original Apple trace is that all you have to do is press any key (for example, the space bar) to stop the trace. To continue tracing, type a 'T', and trace will continue from where it stopped. Or you can type an 'S' to take only one step. The prompt character used for S&T is an inverse '*' so you can distinguish it from the normal Apple monitor. S&T also includes all of the normal monitor commands in addition to step and trace. In fact, it actually uses many parts of the existing monitor to do its work.

How to Use the Program

To use Step-n-Trace, first load it and then type 'CALL 768', or 'BRUN' it from your disk. You will then have all of the monitor commands at your disposal, including step and trace. To get out of the program, just press 'RESET' on your Apple II Plus, or use CTRL-C, or CTRL-B and you will end up in BASIC.

Since the program resides in hex address $300 to $3E9, it loads over some of the DOS address pointers from $3D0 to $3E9. Generally, this doesn't cause any problems for me. However, this can be avoided by moving it to some other area of memory; but the jump addresses in lines 69, 75, 83, 91, 120, 168, and 169 will have to be revised accordingly. The assembler listing for S&T makes use of most of the same labels as the Apple monitor to make it easier to relate what's happening with the old monitor.

At this point, I should mention that the step-and-trace functions suffer from the same problems as the original Apple monitor, in that under certain conditions, the stack register will be displayed with an incorrect value. When this happens, for example, after JSR or RTS, the display will be corrected after the next instruction. Also, if the program manipulates the stack with the use of TXS instructions, the actual operation will probably be incorrect. Lastly, with DOS in effect, when a program is traced through the changing of an I/O hook (usually $36 or $37) the program trace will lock up because the output will have a partially incorrect jump indirect address, and your trace will fall off the edge of the earth. The frailties mentioned above are not nearly as restrictive as they may seem. All in all, S&T is a useful utility.

Exploring Applesoft with S&T

For those of you who have read this far, but don't really plan on doing any assembly language programming, here is how Applesoft works. First load Step-n-Trace and then enter the following BASIC program:

```
10 CALL 768: PRINT "HELLO"
20 END
```

Next type 'RUN', and you will be rewarded with the sound of the bell and an inverse '*' prompt character, telling you that you're in S&T. Next type 'FF58S'. From now on, each 'S' you type will step you through the operations of Applesoft. The first 'S' should display 'D823- 4C D2 D7 JMP $D7D2' on the screen, followed by the contents of the registers. This is the running return to Applesoft. As you 'S'tep or 'T'race through the instructions, you will see the colon ($3A), the print command token ($BA), the quotation ($22), the characters of the word 'HELLO'

($48,45,4C,4C,4F) and more pass through the A (accumulator) register, as Applesoft analyzes your program line.

With some study you'll begin to understand what Applesoft is doing. With some effort, you can actually find where the subroutines are located for the 'SIN', 'SQR', or any other function you're interested in. All of this is accomplished with the help of S&T.

So, if you're doing any assembly language work on an Apple II Plus, S&T can be of great help. If you're just interested in seeing how things actually run inside your Apple, Step-n-Trace can open a lot of interesting doors.

(Editor's Note: A slightly modified version of this program, Step-Trace.800, is also included on disk. Step-Trace.800 loads at $800 and does not employ the key stop feature found in Step-Trace [shown in listing]. As a result, Step-Trace.800 may be used with the TRACER program on Apple II Plus or Language Card systems. To accomplish this, initialize Step-Trace.800 and then TRACER.)

```
0800                ;*************************
0800                ;*                       *
0800                ;*      STEP-N-TRACE     *
0800                ;*     CRAIG PETERSON    *
0800                ;*                       *
0800                ;*       STEP-TRACE      *
0800                ;*                       *
0800                ;*  COPYRIGHT (C) 1981   *
0800                ;*     MICRO INK, INC.   *
0800                ;* CHELMSFORD, MA 01824  *
0800                ;*  ALL RIGHTS RESERVED  *
0800                ;*                       *
0800                ;*************************
0800                ;
0800                ;A PROGRAM TO FURNISH THE APPLE II
0800                ;PLUS WITH THE STEP AND TRACE CAPA-
0800                ;BILITIES OF THE STANDARD APPLE II.
0800                ;
0800                RTNL   EPZ $2C              ;RETURN ADDRESS LO
0800                RTNH   EPZ $2D              ;RETURN ADDRESS HI
0800                LGTH   EPZ $2F              ;LENGTH/DISPLACEMENT
0800                PRMP   EPZ $33              ;PROMPT CHARACTER
0800                YSAV   EPZ $34              ;PLACE TO SAVE Y
0800                PCL    EPZ $3A              ;PROGRAM COUNTER LO
0800                PCH    EPZ $3B              ;PROGRAM COUNTER HI
0800                XQT    EPZ $3C              ;USER INSTRUCTION
0800                STAT   EPZ $48              ;PROC STATUS REG
0800                ;
0800                ;
0800                KBRD   EQU $C000            ;KEYBOARD REGISTER
0800                INSD   EQU $F882            ;DISPLAY PRGRM CNTR
0800                DISA   EQU $F8D0            ;DISASEMBL INSTR
0800                ADJ2   EQU $F954            ;ADJUST PC-2
0800                ADJ3   EQU $F956            ;ADJUST PC-3
0800                REGD   EQU $FAD7            ;DISPLAY USER REGS
0800                RGDS   EQU $FADA            ;DISP REGS-NO CR
0800                GETL   EQU $FD67            ;GET INPUT LINE
0800                BL1    EQU $FE00            ;BLANK ROUTINE
0800                A1PC   EQU $FE75            ;COPY A1 TO PC
0800                BELL   EQU $FF3A            ;RING THE BELL
0800                RSTR   EQU $FF3F            ;RESTORE USER REGS
0800                SAVE   EQU $FF4A            ;SAVE USER REGS
0800                GETN   EQU $FFA7            ;GET ITEM,NONHEX
0800                TSUB   EQU $FFBE            ;PUSH AND GOTO SUB
0800                TSB1   EQU $FFC5            ;HANDLE THE MODE
0800                ZMOD   EQU $FFC7            ;ZERO THE MODE
0800                CHRT   EQU $FFCC            ; CHARACTER TABLE
0800                ;
0300                       ORG $0300
0300                       OBJ $0800
0300                ;
0300 D8             STRT   CLD                  ;SET HEX MODE
0301 203AFF                JSR BELL             ;RING THAT CHIME
0304 A92A           CONT   LDA '*               ;LOAD INVERSE *
0306 8533                  STA PRMP             ; AND STORE IN PRMP
0308 2067FD                JSR GETL             ;READ A LINE
030B 20C7FF                JSR ZMOD             ;SET MODE & Y=0
030E 20A7FF         NXTI   JSR GETN             ;GET ITEM,NONHEX
0311 8434                  STY YSAV             ;CHAR IN A-REG
0313 C9EC           TRYS   CMP #$EC             ;IS IT STEP?
0315 F00B                  BEQ ENT2             ;IF=STEP,GO ENT2
0317 C9ED           TRYT   CMP #$ED             ;IS IT TRACE?
0319 D00F                  BNE TRCR             ;IF<>TRACE,TRYCR
031B AD00C0                LDA KBRD             ;WAS KEY PRESSD?
031E 3024                  BMI AGIN             ;KEY ON,-->AGIN
0320 C634                  DEC YSAV             ;MAKES STEP RPT
0322 20C7FF         ENT2   JSR ZMOD             ;ENTRY FOR STEP
0325 204903                JSR STPZ             ;GO STEP OUT
0328 101A                  BPL AGIN             ;RTN TO INP LINE
032A C9C6           TRCR   CMP #$C6             ;IS IT A CR?
032C D009                  BNE MCMD             ;IF<>CR,TRY MCMD
032E 20C5FF                JSR TSB1
0331 2000FE                JSR BL1              ;HANDLE CR AS BLNK
0334 4C0403                JMP CONT             ;RETURN TO CONT
```

```
0337 A017    MCMD  LDY #$17      ;TRY MONITOR CMDS
0339 88      CHRS  DEY           ;SEARCH MON CHARS
033A 30C4          BMI STRT      ;NOT FOUND, GO START
033C D9CCFF        CMP CHRT,Y    ;CMP WITH TABLE
033F D0F8          BNE CHRS      ;NOT FOUND, ->CHRS
0341 20BEFF        JSR TSUB      ;FND, CALL SUB
0344 A434    AGIN  LDY YSAV      ;RESTORE Y
0346 4C0E03        JMP NXTI      ;GET NEXT COMMAND
0349 2075FE  STPZ  JSR A1PC      ;ADR TO PC
034C 20D0F8  STEP  JSR DISA      ;TAKE ONE STEP
034F 68            PLA           ;ADJUST TO USER
0350 852C          STA RTNL      ;STACK AND SAVE
0352 68            PLA           ;RTN ADR
0353 852D          STA RTNH
0355 A208          LDX #$08
0357 BDE103  XQIN  LDA INM1,X    ;INIT XEQ AREA
035A 953C          STA XQT,X
035C CA            DEX
035D D0F8          BNE XQIN
035F A13A          LDA (PCL,X)   ;JSR OPCODE BYTE
0361 F02C          BEQ XBRK      ;SPECIAL IF BREAK
0363 A42F          LDY LGTH      ;LENGTH FROM DASSY
0365 C920          CMP #$20
0367 F043          BEQ XJSR      ;HANDLE JSR,RTS,
0369 C960          CMP #$60      ;JMP,JMP( ),
036B F02F          BEQ XRTS      ; & RTI SPECIAL
036D C94C          CMP #$4C
036F F046          BEQ XJMP
0371 C96C          CMP #$6C
0373 F043          BEQ XJAT
0375 C940          CMP #$40
0377 F01F          BEQ XRTI
0379 291F          AND #$1F
037B 4914          EOR #$14
037D C904          CMP #$04      ;COPY USR INSTR
037F F002          BEQ XQ2       ;TO XEQ AREA
0381 B13A    XQ1   LDA (PCL),Y
0383 993C00  XQ2   STA XQT,Y
0386 88            DEY
0387 10F8          BPL XQ1
0389 203FFF        JSR RSTR      ;RESTOR USR REGS
038C 4C3C00        JMP XQT       ;XEQ USER OP
038F 2082F8  XBRK  JSR INSD      ;PRINT USER PC
0392 20DAFA        JSR RGDS      ;AND REGS
0395 4C0003        JMP STRT      ;THEN GO STRT
0398 18      XRTI  CLC
0399 68            PLA           ;SIMULATE RTI
039A 8548          STA STAT
039C 68      XRTS  PLA           ;RTS SIMULATION
039D 853A          STA PCL
039F 68            PLA
03A0 853B    PCN2  STA PCH
03A2 A52F    PCN3  LDA LGTH      ;UPDAT PC BY LEN
03A4 2056F9        JSR ADJ3
03A7 843B          STY PCH
03A9 18            CLC
03AA 9014          BCC NEWP
03AC 18      XJSR  CLC
03AD 2054F9        JSR ADJ2      ;UPDATE PC AND
03B0 AA            TAX           ;PUSH ONTO STAK
03B1 98            TYA           ;FOR JSR
03B2 48            PHA           ;SIMULATION
03B3 8A            TXA
03B4 48            PHA
03B5 A002          LDY #$02
03B7 18      XJMP  CLC
03B8 B13A    XJAT  LDA (PCL),Y
03BA AA            TAX           ;LOAD PC FOR JMP
03BB 88            DEY           ;& (JMP)
03BC B13A          LDA (PCL),Y   ;SIMULATION
03BE 863B          STX PCH
03C0 853A    NEWP  STA PCL
03C2 B0F3          BCS XJMP
03C4 A52D    RTNJ  LDA RTNH
03C6 48            PHA
```

```
03C7 A52C                 LDA RTNL
03C9 48                   PHA
03CA 4CD7FA               JMP REGD        ;DISPLAY USR REG
03CD 18           BRAN    CLC             ;BRANCH TAKEN,
03CE A001                 LDY #$01        ;ADD LEN+2 TO PC
03D0 B13A                 LDA (PCL),Y
03D2 2056F9               JSR ADJ3
03D5 853A                 STA PCL
03D7 98                   TYA
03D8 38                   SEC
03D9 B0C5                 BCS PCN2
03DB 204AFF       NBRN    JSR SAVE        ;NORML RTRN AFTR
03DE 38                   SEC             ;EXQING USER OP
03DF B0C1                 BCS PCN3        ;GO UPDATE PC
03E1 EA           INM1    NOP
03E2 EA           INIT    NOP
03E3 EA                   NOP             ;DUMMY FILL FOR
03E4 4CDB03               JMP NBRN        ;XEQ AREA
03E7 4CCD03               JMP BRAN
                         END

***** END OF ASSEMBLY


          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5  *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

RTNL    002C  RTNH    002D  LGTH    002F  PRMP    0033  YSAV    0034  PCL     003A
PCH     003B  XQT     003C  STAT    0048

** ABSOLUTE VARABLES/LABELS

KBRD    C000  INSD    F882  DISA    F8D0
ADJ2    F954  ADJ3    F956  REGD    FAD7  RGDS    FADA  GETL    FD67  BL1     FE00
A1PC    FE75  BELL    FF3A  RSTR    FF3F  SAVE    FF4A  GETN    FFA7  TSUB    FFBE
TSB1    FFC5  ZMOD    FFC7  CHRT    FFCC  STRT    0300  CONT    0304  NXTI    030E
TRYS    0313  TRYT    0317  ENT2    0322  TRCR    032A  MCMD    0337  CHRS    0339
AGIN    0344  STPZ    0349  STEP    034C  XQIN    0357  XQ1     0381  XQ2     0383
XBRK    038F  XRTI    0398  XRTS    039C  PCN2    03A0  PCN3    03A2  XJSR    03AC
XJMP    03B7  XJAT    03B8  NEWP    03C0  RTNJ    03C4  BRAN    03CD  NBRN    03DB
INM1    03E1  INIT    03E2

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:01D2
```

TRACER: A Debugging Tool for the Apple II

by R. Kovacs

The Apple's Step/Trace routines are handy, but you will find them even more useful when used in conjunction with this Tracer program.

The Apple II's monitor in ROM is crammed with many useful routines. These include memory interrogation and modification, keyboard input, CRT display output and cassette I/O. In addition, Apple has thoughtfully provided a number of routines related to assembly language programming. A single-pass assembler and disassembler are invaluable aids in writing and reviewing machine code. A step/trace feature allows you to control execution of your program during the software development phase.

The step routine executes a single instruction and displays its address, both Hex and disassembled code, the values of the A,X,Y,P registers and the stack pointer. You can modify any register and continue execution of either the next instruction or any arbitrary one.

Unfortunately, all this information uses up the display rather quickly such that at best only the 11 most recent steps are shown. It seemed to me that it would be useful to display more program counter history at the expense of other information.

The Program

The Tracer program was designed to operate in conjunction with Apple's step/trace routines to enhance their usefulness. It is basically a formatter which controls the information output to the screen. This routine will display up to 160 of the most recent instructions executed. This is in addition to the usual details (i.e. disassembled code and register displays) of the last instruction displayed. Features include single step and trace with paging. You can either continue execution or temporarily exit to modify registers or memory. Tracer also looks for the break code (00) and waits for your action after announcing the break with a double bell. The last instruction executed before the break was encountered will still be displayed.

Caution: It should be recognized that Tracer's display lags by one instruction. If the monitor is entered via reset, the current register values saved may be different due to the next instruction having executed. Thus you should check your values using the control-E monitor command.

A commented assembly listing is shown. The program is approximately 190 bytes long and is located starting at $300. It uses no additional page zero memory.

How it Works

Tracer controls what information is displayed on the screen by manipulating the characters generated by the step/trace routines. Tracer looks for certain key characters and sequences to determine when one instruction has been completed.

A slight complication arises out of the 2-line display format used by Apple. The character stream normally output to the screen after completion of a single step begins with a carriage return ($8D). It is then followed by a line of printout whose first 4 characters are the Hex Address of the instruction just executed. This line is terminated with another carriage return and the second line is output.

Tracer looks for the carriage return which marks the beginning of the first line by diverting all characters to Tracer via the COUT hook. Subsequent characters are stored in a buffer. The second line is recognized by a carriage return followed by a space ($A0). The next carriage return is used to output the 4 character Hex address from the buffer (plus a space) to the screen using the monitor COUT routines ($FDF0). These routines take care of wraparound and scrolling to display up to 160 addresses in an 8 by 20 line format.

Since the buffer happens to be part of screen memory, then it too is displayed. The buffer region is protected by moving the bottom of the scrolling window.

The control Y function is used to initialize Tracer via a jump at $3F8. It clears the screen, sets the scrolling window and sets the COUT hook at $36 and $37 to divert all characters normally displayed on the screen to Tracer.

Directions

Tracer is relatively simple to use:

1. Load Tracer starting at $300. (Don't forget the Control-Y jump at 3FB: 4C 00 03.)

2. Run the program via the monitor by typing: Yc XXXX T where Yc is a Control-Y and XXXX is the address where debugging is to begin. The screen will clear, Tracer will become hooked via COUT and tracing begins as the specified address.

3. Tracer is initialized to single step and will halt after displaying the familiar step/trace information at the bottom of the screen. Additional steps are

executed by depressing the space bar. The addresses of previously executed instructions will begin to accumulate in the upper part of the display.

4. One page of instructions can be executed by depressing the return key instead of the space bar. Control can be retained immediately by hitting any key.

5. Of course hitting reset returns the user back to the monitor where registers and memory can be manipulated if needed. Tracer can be reentered by typing: Yc T.

Figure 1: This example illustrates Tracer's output format while looping through Apple's WAIT routine at $FCA8. The normal step/trace output for the current instruction is at the bottom of the screen and the previous 160 addresses of program counter are listed above.

Oldest

↑

160 Previously Executed Addresses

↓

Most Recent

```
FCA9 FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAE
FCAF FCB1 FCA9 FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAE
FCAF FCB1 FCA9 FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAE FCAF FCB1
FCA9 FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAE FCAF FCB1 FCA9 FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAC FCAA FCAC FCAE FCAF FCB1 FCA9 FCAA
FCAC FCAA FCAC FCAA FCAC FCAA FCAC FCAA
FCAA-   E9 01        SBC   #$01
 A=05 X=00 Y=00 P=31 S=99
```

Normal Apple Step/Trace Display

```
0800          ;*************************
0800          ;*                       *
0800          ;*        TRACER         *
0800          ;*       R. KOVACS       *
0800          ;*                       *
0800          ;*  COPYRIGHT (C) 1981   *
0800          ;*    MICRO INK, INC.    *
0800          ;* CHELMSFORD, ME 01824  *
0800          ;*  ALL RIGHTS RESERVED  *
0800          ;*                       *
0800          ;*************************
0800          ;
0800          ;
0800          ;ENTER VIA CONTROL-Y FOLLOWED BY XXXXT
0800          ;WHERE XXXX IS THE ADDRESS TO BEGIN TRACING
0800          ;
0800          WNDBTM EPZ $23           ;BOTTOM OF SCROLLING WINDOW
0800          PCL    EPZ $3A           ;PGM COUNTER
0800          ;
0800          WINDOW EQU $FB3C         ;SET NORMAL SCROLL WINDOW
0800          BELL   EQU $FBDD         ;TOGGLE SPEAKER
0800          CLEAR  EQU $FC58         ;CLEAR SCREEN, HOME CURSOR
0800          COUT   EQU $FDF0         ;OUTPUT CHAR TO SCREEN
0800          READ   EQU $C000         ;KEYBOARD STROBE
0800          RESET  EQU $C010         ;RESET KEYBOARD
0800          ;
0800          BUFF   EQU $0750         ;LINE #22-COL #0
0800          BUFF1  EQU $07D0         ;     #23     #0
0800          ;
0800          ;***********************************
0800          ;SET UP CONTROL-Y JUMP TO $3F8
0800          ;
03F8                 ORG $03F8
03F8                 OBJ $08F8
03F8          ;
03F8 4C0003          JMP TRINIT
03FB          ;
03FB          ;***********************************
03FB          ;
03FB          ;TRACER INITIALIZATION
03FB          ;
0300                 ORG $0300
0300                 OBJ $0800
0300          ;
0300 203CFB   TRINIT JSR WINDOW        ;CLEAR ENTIRE SCREEN
0303 2058FC          JSR CLEAR
0306 A915            LDA #$15          ;SET SCROLL WINDOW
0308 8523            STA WNDBTM
030A A91C            LDA #TRACER       ;SET COUT HOOK
030C 8536            STA $36           ;TO TRACER
030E A903            LDA /TRACER
0310 8537            STA $37
0312 A91F            LDA #$1F          ;INIT CH FOR EVEN PAGING
0314 8524            STA $24
0316 A902            LDA #$02          ;INIT PGCNT FOR
0318 8DBB03          STA PGCNT         ;SINGLE STEP
031B 60              RTS
031C          ;
031C          ;***********************************
031C          ;
031C 8DB703   TRACER STA SAVEA         ;SAVE A & Y
031F 8CB803          STY SAVEY         ;REGISTERS
0322 2CBA03          BIT CRFLG         ;WAS LAST CHAR A CR?
0325 301C            BMI CR            ;YES
0327 C98D            CMP #$8D          ;IS THIS CHAR A CR?
0329 F00C            BEQ SETCR         ;YES
032B ACB903   STORE  LDY BPTR          ;LOAD BUFF POINTER
032E 995007          STA BUFF,Y        ;NO, SO STORE IT
0331 C8              INY               ;INC POINTER
```

```
0332 8CB903          STY BPTR          ;  & SAVE IT
0335 D005            BNE DONE          ;BRANCH ALWAYS
0337 A080    SETCR   LDY #$80          ;SET CR FLAG
0339 8CBA03          STY CRFLG
033C ADB703  DONE    LDA SAVEA         ;RESTORE
033F ACB803          LDY SAVEY         ;REGISTERS
0342 60              RTS               ;RETURN TO MONITOR
0343 A000    CR      LDY #$00          ;RESET CR FLAG
0345 8CBA03          STY CRFLG
0348 C9A0            CMP #$A0          ;IS NEXT CHAR A SPACE?
034A D007            BNE ADDR-2        ;NO
034C A080            LDY #$80          ;ADJ PTR TO NEXT
034E 8CB903          STY BPTR          ;LINE ON SCREEN
0351 D0D8            BNE STORE         ;BRANCH ALWAYS
0353 A000            LDY #$00          ;INIT BUFF POINTER
0355 B95007  ADDR    LDA BUFF,Y
0358 20F0FD          JSR COUT          ;OUTPUT IT
035B C8              INY
035C C004            CPY #$04          ;FINISHED PRINTING 4 CHAR
035E 90F5            BCC ADDR          ;NO
0360 A9A0            LDA #$A0
0362 20F0FD          JSR COUT          ;OUTPUT A SPACE
0365         ;
0365         ;CHECK FOR BREAK
0365         ;
0365 A000            LDY #$00
0367 B13A            LDA (PCL),Y       ;GET OPCODE
0369 F00C            BEQ KEY1          ;PAUSE IF BREAK
036B         ;
036B         ;LOOK FOR KEYBOARD INPUT
036B         ;
036B CEBB03  KEY     DEC PGCNT         ;CHECK PAGING
036E F00D            BEQ KEY2
0370 2C00C0          BIT READ          ;ANY KEYBOARD INPUTS?
0373 300D            BMI KEY3          ;YES
0375 1020            BPL TRACE
0377 20DDFB  KEY1    JSR BELL          ;SOUND BELL FOR BRK
037A 20DDFB          JSR BELL
037D A0A0    KEY2    LDY #$A0          ;RESET PAGE COUNTER
037F 8CBB03          STY PGCNT         ;AND PAUSE
0382 8D10C0  KEY3    STA RESET
0385 2C00C0  KEY4    BIT READ          ;LOOP UNTIL ANOTHER
0388 10FB            BPL KEY4          ;KEY IS HIT
038A         ;
038A         ;TEST INPUT FOR TRACE, STEP OR QUIT
038A         ;
038A AD00C0          LDA READ          ;LOAD CHARACTER
038D C98D            CMP #$8D          ;'RETURN' TO CONTINUE TRACE
038F F006            BEQ TRACE
0391 C9A0            CMP #$A0          ;'SPACE' TO SINGLE STEP
0393 F005            BEQ STEP
0395 D0E3            BNE KEY1+3        ;NO MATCH, TRY AGAIN
0397 8D10C0  TRACE   STA RESET         ;RESET KEYBOARD STROBE
039A EA      STEP    NOP
039B         ;
039B         ;FILL PROTECTED FIELD WITH SPACES
039B         ;
039B A9A0            LDA #$A0          ;ASCII SPACE
039D A027            LDY #$27          ;40 CHAR/LINE
039F 995007  FILL    STA BUFF,Y
03A2 99D007          STA BUFF1,Y
03A5 88              DEY
03A6 10F7            BPL FILL
03A8         ;
03A8 ADB703          LDA SAVEA
03AB A000            LDY #$00          ;RESET BUFF POINTER
03AD 8CB903          STY BPTR
03B0 C9B0            CMP #$B0          ;IS 1ST CHAR 0-9/A-F ?
03B2 9088            BCC DONE          ;NO
```

```
03B4 4C2B03  139         JMP STORE            ;YES, OUTPUT IT
03B7         140  ;
03B7         141  ;
03B7 00      142  SAVEA   HEX 00
03B8 00      143  SAVEY   HEX 00
03B9 00      144  BPTR    HEX 00
03BA 00      145  CRFLG   HEX 00
03BB 00      146  PGCNT   HEX 00
            147           END

***** END OF ASSEMBLY


          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

WNDBTM 0023  PCL    003A

** ABSOLUTE VARABLES/LABELS

WINDOW FB3C  BELL   FBDD  CLEAR  FC58 COUT   FDF0
READ   C000  RESET  C010  BUFF   0750 BUFF1  07D0 TRINIT 0300 TRACER 031C
STORE  032B  SETCR  0337  DONE   033C CR     0343 ADDR   0355 KEY    036B
KEY1   0377  KEY2   037D  KEY3   0382 KEY4   0385 TRACE  0397 STEP   039A
FILL   039F  SAVEA  03B7  SAVEY  03B8 BPTR   03B9 CRFLG  03BA PGCNT  03BB

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0102
```

Apple Integer BASIC Subroutine Pack and Load

by Richard F. Suitor

Oftentimes Apple programmers find themselves writing machine language subroutines which will be called from an integer BASIC program. Storing these subroutines in the same file as the BASIC driver programs can get messy. This program enables you to include a BASIC program and machine language subroutines in one file which may be easily saved to disk or tape.

The first issue of CONTACT, the Apple Newsletter, gave a suggestion for loading assembly language routines with a BASIC program. Simply summarized, one drops the pointer of the BASIC beginning below the assembly language portion, adds a BASIC instruction that will restore the pointer and SAVEs. The procedure is simple and effective but has two limitations. First, it is inconvenient if BASIC and the routines are widely separated (and is very tricky if the routines start at $800, just above the display portion of memory). Second, a program so saved cannot be used with another HIMEM, and is thus inconvenient to share or to submit to a software exchange.

The subroutine presented here avoids these difficulties at the expense of the effort to implement it. It is completely position independent; it may be moved from place to place in core with the monitor move command and used at the new location without modification. It makes extensive use of SWEET 16, the 16-bit interpreter supplied as part of the Apple Monitor ROM.

How to Use Pack and Load

To use the routine from Apple Integer BASIC, CALL MKUP, where MKUP is 128 (decimal) plus the first address of the routine. The prompt shown is "@". Respond with the hex limits of the routine to be stored, as BBBB.EEEE (BBBB is the beginning address, EEEE is the ending; the same format that the monitor uses). Several groups may be specified on one line separated by spaces or several lines. Type S after the last group to complete the pack and return to BASIC. The program can now be saved.

To load, enter BASIC and LOAD. When complete, RUN. The first RUN will move all routines back to their original location and return control to BASIC. It will not RUN the program; subsequent RUNs will.

A LIST of the program after calling MKUP and before the first RUN will show one BASIC statement (which initiates the restoration process) and gibberish. If this is done, RESET followed by CTRL-C will return control to BASIC.

WARNING #1: The routine must be placed in memory where it will not overwrite itself during the pack. The start of the routine must be above HIMEM (e.g. in the high resolution display region) or $17A + 4*N + W below the start of the BASIC program, where N is the number of routines stored and W is the total number of words in all of these routines. Also, those routines that are highest in memory should be packed first to avoid overwriting during pack or restore. Otherwise it is not necessary to worry about overwriting during the restore process; only $1A words just below the BASIC program are used.

WARNING #2: Do not attempt to edit the program after calling MKUP. If editing is necessary, RUN once to unpack, then edit and call MKUP again.

How Pack and Load Works

The routine first packs the restore routine just below the BASIC program. It then packs other routines as requested, with first address and number of bytes (words). When S is given, it packs itself with the information to restore LOMEM and the beginning of the BASIC program. The first $46 words of the routine form a BASIC statement which will initiate the restoration process when RUN is typed.

If a particular HIMEM is needed by the program (e.g. for high resolution programs) it must be entered before LOADing. The LOMEM will be reset by the restoration process to the value it had when MKUP was called.

Some convenient load and entry points are:

BASO (load)	MKUP entry	
hex	hex	decimal
800	880	2176
A90	B10	2832
104C	10CC	4300
2050	20D0	8400
3054	30D4	12500
6000	6080	24704
9000	9080	−28544

Program on disk BLOADS at $9400. MKUP is at $9480, −27520 decimal.

Editor's note: Due to a special request by the author, MICRO encourages the use and distribution of this subroutine. However, please make sure proper credit is placed on any copies: "This PACK and LOAD Subroutine was written by Richard F. Suitor and first published in an early issue (#6) of MICRO, the 6502/6809 Journal."

Appendix to Subroutine Pack and Load

When the subroutine Pack and Load was first written, I had in mind a utility that would allow the user to easily pack and unpack subroutines (we had only cassette storage then) before running a program. After using it awhile, it became clear to many people that, after a program was debugged, it would be nice if it unpacked and ran in one operation. Alan Hill, who has contributed many significant programs for the Apple, was the first to point out to me that a JMP to $EFEC instead of $E003 would accomplish this. In the meantime, Apple switched to pushing Applesoft instead of Integer BASIC, a reasonable enough decision, but exasperating to those who had invested a lot of effort in developing Integer BASIC software. Apple still supplies the Integer BASIC in both ROM and language card forms, but both of these cost money. A person on a limited budget who has purchased an Apple Plus can obtain software versions from either IAC-associated clubs or from Apple Pugetsound Program Library Exchange (A.P.P.L.E.) (304 Main Ave. S., Suite 300, Renton, WA 98055).

Unfortunately this was a development which I had not foreseen when I wrote this routine. The routine returns to ROM addresses which I believed immutable; now those with Apple Plus versions can obtain versions of Integer BASIC for which programs packed with this routine will badly fail.

The enclosed routine will solve their problem and the problem of those programmers who wish to change the return vector to automatically RUN or not. It is a routine to change the address to which the UNPACK procedure returns upon completion.

The desired address is entered into locations 0 and 1. For example, if you want to use the address $EFEC, from the monitor you:

```
*0:EC EF
```

or from BASIC you:

```
POKE 0,236
POKE 1,239
```

To accomplish the change this routine, and the program to be changed, must be in memory. The program must be LOADed, but not run. The routine is shown at location 800 ($320), but will run correctly anywhere. BLOAD the routine, set up locations 0 and 1, then CALL 800 to accomplish the change. You may save the changed program.

The addresses which you may wish to use are:

Purpose	ROM Version	Disk Version
Back to BASIC	$E003	$03D0
Unpack & RUN	$EFEC	($9D58)

The last entry, to unpack and RUN from a disk version, means you put the contents of $9D58 into 0 and the contents of $9D59 into 1. This method should be used for the A.P.P.L.E. version of Integer. Please note that although the locations $9D58,9 are the same for any 48K disk-based system, the contents of the locations may differ. Thus, a version of a program prepared in this way is least likely to be able to be run on another system. The version that is most likely to be "universally" usable is one using the address $3D0. This choice has the disadvantage that it will not unpack and RUN, but it will fail only on a cassette system or on a disk system that has had page 3 overwritten. For these systems, enter the monitor and type 3D0:4C 03 E0. (Note: this will enable a 3D0G to return to BASIC, but will not restore a disconnected DOS.)

However, using the routine given in this program, any "packed" program can be loaded and altered to run on the user's system, and then saved.

Editor's Note: The Pack-Load routine requires that SWEET-16 be resident in your Apple. Even after the modifications mentioned in this Appendix are made, if SWEET-16 is not available, the unpacking and packing processes will fail. Thus, if your version of Integer BASIC does not include SWEET-16 in the proper locations, Subroutine Pack and Load will not work.

```
0320- D8 18 A5 CA 69 54 85 18
0328- A5 CB 69 01 85 19 A0 00
0330- 38 A5 4C F1 18 48 A5 4D
0338- C8 F1 18 AA 68 38 E9 03
0340- 85 18 B0 01 CA 86 19 A5
0348- 01 91 18 88 A5 00 91 18
0350- 60
```

```
0800        ;************************
0800        ;*                      *
0800        ;* PACK AND LOAD SUBRTN *
0800        ;*  RICHARD F. SUITOR   *
0800        ;*                      *
0800        ;*      PACK-LOAD       *
0800        ;*                      *
0800        ;*  COPYRIGHT (C) 1981  *
0800        ;*    MICRO INK, INC.   *
0800        ;* CHELMSFORD, MA 01824 *
0800        ;*  ALL RIGHTS RESERVED *
0800        ;*                      *
0800        ;************************
0800        ;
0800        ;
0800        ; INTEGER BASIC ROUTINE TO PACK AND RELOAD
0800        ; MACHINE LANGUAGE SUBROUTINES AND/OR TABLES
0800        ;
0800        ; CALL BASO+128(DEC) = MKUP TO PACK EXISTING
0800        ; ROUTINES AT THE START OF BASIC
0800        ;
0800        ; RUNNING THE PACKED PROGRAM WILL UNPACK THE
0800        ; PACKED ROUTINES AND RETURN TO BASIC (>)
0800        ;
0800        ; CHANGE THE LAST INSTRUCTION OF THE LISTING
0800        ; TO 'JMP BRUN' TO UNPACK AND RUN IN ONE OPERATION.
0800        ; NOTE: THIS STEP NOT NORMALLY TAKEN UNTIL
0800        ; PROGRAM DEVELOPMENT IS COMPLETE!
0800        ;
0800        ; PROGRAM WILL RUN ANYWHERE IN MEMORY
0800        ;
0800        ;
9400               ORG $9400
9400               OBJ $800
9400        ;
9400        ;
9400        ACCL   EPZ $00              ;R0, ACCUMULATOR
9400        BSOL   EPZ $02              ;R1
9400        TABL   EPZ $04              ;R2
9400        TBCL   EPZ $06              ;R3
9400        HIMS   EPZ $08              ;R4
9400        LMRT   EPZ $0A              ;R5
9400        BPRG   EPZ $0C              ;R6
9400        FRML   EPZ $0E              ;R7
9400        NBYT   EPZ $10              ;R8
9400        BPR2   EPZ $12              ;R9
9400        PTLL   EPZ $14              ;R10
9400        XTAB   EPZ $16              ;R11
9400        SKPL   EPZ $18              ;R12, SW16 STACK PTR
9400        MODE   EPZ $31
9400        YSAV   EPZ $34
9400        PRMP   EPZ $33              ;PRCMPT
9400        LMML   EPZ $4A              ;INTEGER LOMEM
9400        HIML   EPZ $4C              ;INTEGER HIMEM
9400        LMWL   EPZ $CC              ;INTEGER END OF VARIABLES
9400        BBSL   EPZ $CA              ;BOTTOM OF PROGRAM
9400        JSRL   EPZ $CE              ;CALL VECTOR
9400        BSC2   EQU $E003            ;BASIC
9400        BRUN   EQU $EFEC            ;RUN BASIC
9400        BUFF   EQU $0200            ;INPUT BUFFER
9400        SW16   EQU $F689            ;SWEET16 ENTRY
9400        GTNM   EQU $FFA7            ;GET # FROM BUFF.
9400        PBL2   EQU $F94A            ;PRINT BLANKS
9400        COUT   EQU $FDED            ;OUTPUT CHAR.
9400        BELL   EQU $FF3A            ;BEEP
9400        GTLN   EQU $FD67            ;INPUT A LINE
9400        ;
9400        ; BASIC STATEMENT TO START CODE REPLACEMENT
9400        ; PROCESS...
9400        ;
9400        ; 0 POKE 1,76: POKE 2,(PEEK(202)+70) MOD 256:
9400        ;   POKE 3,(PEEK(203) + (PEEK(202)+70)/256):
9400        ;   CALL 1
9400        ;
```

```
9400 460000     BASO   HEX 460000
9403 64B101            HEX 64B101
9406 0065B7            HEX 0065B7
9409 4C0003            HEX 4C0003
940C 64B2              HEX 64B2
940E 020065            HEX 020065
9411 382E3F            HEX 382E3F
9414 B2CA              HEX B2CA
9416 007212            HEX 007212
9419 B74600            HEX B74600
941C 721F              HEX 721F
941E B20001            HEX B20001
9421 0364B3            HEX 0364B3
9424 0300              HEX 0300
9426 65382E            HEX 65382E
9429 3FB2CB            HEX 3FB2CB
942C 0072              HEX 0072
942E 12382E            HEX 12382E
9431 3FB2CA            HEX 3FB2CA
9434 0072              HEX 0072
9436 12B746            HEX 12B746
9439 007215            HEX 007215
943C B200              HEX B200
943E 017203            HEX 017203
9441 4DB101            HEX 4DB101
9444 0001              HEX 0001
9446                ;
9446                ; INITIALIZE POINTERS
9446                ;
9446 D8         PTBK   CLD
9447 A201              LDX #1
9449 B5CA       PT02   LDA BBSL,X           ;R1 IS START OF PACKED PROG.
944B 9502              STA BSOL,X
944D B54C              LDA HIML,X           ;R4 IS END (HIMEM)
944F 9508              STA HIMS,X
9451 CA                DEX
9452 10F5              BPL PT02
9454 2089F6            JSR SW16
9457 105201            SET R0,PTLP-BASO
945A 185701            SET R8,PTLP+5-BASO
945D A1                ADD R1               ;SET R7 TO CURRENT START OF
945E 37                STO R7               ; PACKED DATA (BBSL+PTLP-BASO)
945F 67                LDD @R7
9460 35                STO R5               ; PUT IN R5
9461 67                LDD @R7              ;PUT ORIGINAL LENGTH OF PROGRAM
9462 36                STO R6               ; IN R6
9463 24                LDR R4
9464 B6                SUB R6               ;CALCULATE START OF ORIGINAL
9465 36                STO R6               ; PROGRAM AND PUT IN R6
9466 1A1100            SET RA,ST16+1-PLP1
9469 BA                SUB RA               ;CURRENT LOCATION OF ENTRY
946A 3A                STO RA               ;TO RESTORE LOOP IN RA
946B 67                LDD @R7              ;BASO LOCATION TO LEAVE ROUTINE
946C 33                STO R3
946D 00                RTN
946E A201              LDX #1
9470                ;
9470                ; RESTORE ORIGINAL LOMEM AND START
9470                ; OF ORIGINAL PROGRAM...
9470                ;
9470 B50A       PT04   LDA LMRT,X
9472 954A              STA LMML,X
9474 95CC              STA LMWL,X
9476 B50C              LDA BPRG,X
9478 95CA              STA BBSL,X
947A CA                DEX
947B 10F3              BPL PT04
947D 6C1400            JMP (PTLL)
9480                ; JMP (RA) = PLP1
9480                ;
9480                ; SECTION TO PERFORM PACK
9480                ;
9480 A201       MKUP   LDX #1
9482 B54A       MK21   LDA LMML,X
9484 950A              STA LMRT,X           ;R5=LOMEM
```

```
9486 B5CA              LDA BBSL,X          ;R9,R6=START
9488 9512              STA BPR2,X          ; OF PROGRAM
948A 950C              STA BPRG,X
948C B5CE              LDA JSRL,X          ;R2=MKUP LOCATION
948E 9504              STA TABL,X
9490 B54C              LDA HIML,X          ;R4=HIMEM
9492 9508              STA HIMS,X
9494 CA                DEX
9495 10EB              BPL MK21
9497            ;
9497            ; INIT AND PACK THE RESTORE LOOP AT PTLP
9497            ;
9497 2089F6            JSR SW16
949A 24                LDR R4
949B B9                SUB R9
949C 39                STO R9              ;LENGTH OF PROGRAM
949D 118000            SET R1,MKUP-BASO
94A0 22                LDR R2
94A1 B1                SUB R1
94A2 31                STO R1              ;BASO LOCATION
94A3 105201            SET RO,PTLP-BASO
94A6 A1                ADD R1
94A7 32                STO R2              ;PTLP LOCATION
94A8 181800            SET R8,ST16-PTLP
94AB A8                ADD R8
94AC 33                STO R3              ;ST16 LOCATION
94AD E3                INR R3              ;END (ST16) + 1
94AE 1C5000            SET RC,$50          ;SW16 STACK
94B1 0C42              BSB MV52            ;PACK RESTORE LOOP
94B3 00         MK22   RTN
94B4 A9C0       MK01   LDA #$C0            ;'@'
94B6            ;
94B6            ; GET LIMITS AND PACK PROGRAMS
94B6            ;
94B6 8533              STA PRMP            ;PROMPT IS '@'
94B8 A900              LDA #0
94BA 8531              STA MODE
94BC 2067FD            JSR GTLN            ;GET COMMAND
94BF 8616              STX XTAB            ;END OF COMMAND
94C1 A000              LDY #0
94C3 B90002            LDA BUFF,Y
94C6 C9D3              CMP #$D3            ;'S', STOP?
94C8 F068              BEQ MK10            ;YES
94CA 20A7FF     MK06   JSR GTNM            ;START OF RANGE
94CD C9A7              CMP #$A7            ;F(.) (SEE MON.)
94CF F010              BEQ MK02
94D1 98         MERR   TYA                 ;ERROR IF HERE
94D2 AA                TAX
94D3 204AF9            JSR PBL2            ;ERROR INDICATOR
94D6 A9DE              LDA #$DE            ;'^'
94D8 20EDFD            JSR COUT
94DB 203AFF            JSR BELL
94DE 18         MK05   CLC
94DF 90D3              BCC MK01
94E1 E631       MK02   INC MODE
94E3 20A7FF            JSR GTNM            ;END OF RANGE
94E6            ;
94E6            ; A1 & A3 NOW HAVE 1ST #, A2 SECOND
94E6            ; SET UP MOVE TO JUST BELOW (BBSL)
94E6            ; AND LOWER BBSL
94E6            ;
94E6 2089F6            JSR SW16
94E9 011E              BRA SM02
94EB 183C00     MV51   SET R8,$3C
94EE 68                LDD @R8
94EF 32                STO R2              ;R2=A1
94F0 68                LDD @R8
94F1 33                STO R3              ;R3=A2
94F2 B2                SUB R2              ;A2-A1
94F3 38                STO R8
94F4 E3                INR R3
94F5 83         MV52   POP @R3             ;MOVE FROM (R3) DOWN TO (R2)
94F6 96                STP @R6             ;TO (R6) AND DOWN
94F7 23                LDR R3
94F8 D2                CPR R2
```

```
94F9 07FA              BNZ MV52
94FB 28                LDR R8            ;LENGTH-1
94FC 33                STO R3
94FD 180800            SET R8,8
9500 88                POP @R8           ;PREFACE PACKED ROUTINE
9501 96                STP @R6           ;BY LENGTH-1 AND BY
9502 88                POP @R8           ;STARTING ADDRESS
9503 96                STP @R6
9504 88                POP @R8
9505 96                STP @R6
9506 88                POP @R8
9507 96                STP @R6
9508 0B                RSB
9509 0CE0       SM02   BSB MV51
950B 00         SM03   RTN
950C C9EC       MK09   CMP #$EC          ;F(S) STOP?
950E F022              BEQ MK10          ;YES
9510 C9C6              CMP #$C6          ;F(CR) END OF LINE?
9512 F0A0              BEQ MK01          ;YES, GET NEW COMM.
9514 C999              CMP #$99          ;F( ) ;BLANK?
9516 F003              BEQ MK12          ;YES
9518 D0B7              BNE MERR          ;ERROR IF OTHER
951A C8         MK11   INY
951B B90002     MK12   LDA BUFF,Y        ;GET NEXT COMM. CHAR
951E C416              CPY XTAB          ;END OF LINE?
9520 B092              BCS MK01          ;YES, GET ANOTHER
9522 C9A0              CMP #$A0          ;BLANK
9524 F0F4              BEQ MK11
9526 C98D              CMP #$8D          ;CR.
9528 F08A              BEQ MK01
952A C9D3              CMP #$D3          ;'S'
952C F004              BEQ MK10
952E C631              DEC MODE
9530 F098              BEQ MK06          ;ALWAYS
9532                   ;
9532                   ; PACK 1ST PART AND CLEAN UP
9532                   ;
9532 2089F6     MK10   JSR SW16
9535 21                LDR R1
9536 32                STO R2            ;BASO LOCATION
9537 185201            SET R8,PTLP-BASO
953A A8                ADD R8
953B 37                STO R7            ;PTLP LOCATION
953C 25                LDR R5            ;PACK:
953D 77                STD @R7           ; LOMEM
953E 29                LDR R9
953F 77                STD @R7           ; ORIGINAL LENGTH OF PROGRAM
9540 21                LDR R1            ; BASO LOCATION
9541 77                STD @R7           ;ONTO END OF 1ST PART
9542 27                LDR R7            ;OF ROUTINE
9543 33                STO R3
9544 0CAF              BSB MV52          ;PACK BASO-PTLP PLUS ABOVE VARS.
9546 66         SM04   LDD @R6           ;STRIP PREFACE
9547 66                LDD @R6           ;LEAVING BASIC STATEMENT
9548 00                RTN
9549 A50C              LDA BPRG
954B 85CA              STA BBSL          ;R6 IS NEW START
954D A50D              LDA BPRG+01       ;OF PROGRAM
954F 85CB              STA BBSL+01
9551 60                RTS
9552                   ;
9552                   ; RESTORE LOOP --THIS LOOP DOES THE ACTUAL
9552                   ; UNPACKING AND IS ALWAYS JUST IN FRONT OF
9552                   ; THE ORIGINAL BASIC PROGRAM...
9552                   ;
9552 2089F6     PTLP   JSR SW16
9555 61         PLPO   LDD @R1
9556 33                STO R3            ;DESTINATION
9557 61                LDD @R1
9558 38                STO R8            ;LENGTH
9559 00                RTN
955A 2089F6     PLP1   JSR SW16
955D 41         MV60   LDR @R1           ;UNPACK
955E 53                STO @R3
955F F8                DCR R8
```

```
9560 04FB     300           BIP MV60
9562 21       301           LDR R1
9563 D6       302           CPR R6             ;AT END YET?
9564 05EF     303           BIM PLP0           ;NOT YET
9566 00       304    PLP2   RTN
9567 4C03E0   305           JMP BSC2
956A          306    ; OR JMP BRUN TO RUN AUTOMATICALLY
956A 00       307    ST16   HEX 00
             308          END

***** END OF ASSEMBLY


          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

ACCL   0000  BSOL   0002  TABL   0004  TBCL   0006  HIMS   0008  LMRT   000A
BPRG   000C  FRML   000E  NBYT   0010  BPR2   0012  PTLL   0014  XTAB   0016
SKPL   0018  MODE   0031  YSAV   0034  PRMP   0033  LMML   004A  HIML   004C
LMWL   00CC  BBSL   00CA  JSRL   00CE

** ABSOLUTE VARABLES/LABELS

BSC2   E003  BRUN   EFEC  BUFF   0200
SW16   F689  GTNM   FFA7  PBL2   F94A  COUT   FDED  BELL   FF3A  CTLN   FD67
BAS0   9400  PTBK   9446  PT02   9449  PT04   9470  MKUP   9480  MK21   9482
MK22   94B3  MK01   94B4  MK06   94CA  MERR   94D1  MK05   94DE  MK02   94E1
MV51   94EB  MV52   94F5  SM02   9509  SM03   950B  MK09   950C  MK11   951A
MK12   951B  MK10   9532  SM04   9546  PTLP   9552  PLP0   9555  PLP1   955A
MV60   955D  PLP2   9566  ST16   956A

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:01DA
```

Mean 14: A Pseudo-Machine Floating Point Processor for the Apple II

by R.M. Mottola

Modelled after the Sweet 16, this program supports a large variety of mathematical operations on five-byte floating point values. This 'processor' can greatly simplify and enhance your mathematical processing power.

In the beginning of the life of the Apple II computer, an obstacle had to be overcome in the writing of the firmware. As we know, the 6502 is an eight bit microprocessor, but all too frequently routines require numeric operations involving double precision integers. Repeating common operations every time the routines are required could be done, but it is not very space efficient. For that matter, performing the requisite register set-ups to use some general purpose subroutines can also deplete available memory space, if the routines are called frequently. What was needed was an arithmetic processor that could handle two-byte integers. So, a pseudo-machine processor is a machine language program that behaves like a processor.

This elegant solution is called the "Sweet 16 Pseudo-Machine Interpreter" and is known and used by many Apple programmers. It lives from $F689 to F7FA on the FO Integer BASIC ROM found in regular Apple II computers. From a software point of view, the interpreter is used very much like you would use a microprocessor. Programming it requires the use of various instructions and operands. Hand assembly is easy because the instruction set isn't long and the format of the operators is very straightforward. A popular resident asembler, the Lisa assembler by Randall Hyde, will even assemble Sweet 16 mnemonics.

The Mean 14 pseudo-machine floating point processor was modelled after the Sweet 16. It too is programmed like a hardware processor. Instead of being designed to process two-byte integers, the Mean 14 can perform many mathematical operations on five-byte floating point values. These values are formatted in the standard Applesoft variable representation described in the Applesoft manual.

The Mean 14 processor was written to facilitate floating point machine language programming on an Apple II Plus or a standard Apple II with Applesoft ROM card. Since Apple does not provide any documentation for the floating point routines in Applesoft, it is pretty difficult for those wishing to write floating point routines in assembly language. Even knowing the locations and entry requirements of those routines is only partially helpful if either complex or repetitive functions must be performed. Of course, you could always write your more involved functions in Applesoft BASIC, but the Mean 14 will always perform at least ten times as fast and probably much more. The reason for this is simply that the Mean 14 has little of the interpreter overhead that Applesoft has. Using the example of adding two values, if Applesoft is used, and the values are represented as variables which have not been used before, Applesoft must allocate space for them first. And if arrays have been dimensioned, they must be moved up to make space for the new variables. If the variables or arrays happen to collide with strings, then string ''house-cleaning'' must take place. In machine terms, all this takes an awful lot of time. As an added kicker, even more time must be allowed if you use constants instead of variables.

On the other hand, Mean 14 doesn't have to do all of this. Its interpreter overhead is very small and since you, the programmer, supply the operand either by specifying pointers or, in the Immediate Mode, by actually supplying the floating point value, the floating point routines don't have to search for or convert anything. Mean 14 spends its time processing numbers — not trying to find them or converting ASCII strings into them.

What Mean 14 Does

Mean 14 is a very simple kind of interpreter. You give it a number and it looks it up, in a table, where it picks up the address of the subroutine which performs the specific function required. Most of those functions already exist in Applesoft. Some require set-ups to make entry and exit easier. In all cases, the instruction set has been designed to make straight-line machine language floating point arithemetic a lot easier.

That last line indicates one of the possible shortcomings of the Mean 14 for your particular floating point requirement. It can process data only in a straight line. At present, it contains no conditionals in the instruction set. This apparent problem isn't really all that bad when you actually use the Mean 14. For my own applications, I've found that testing, branching, and loop operations can best be handled outside of Mean 14, in 6502 assembly language. This is because, relative to the amount of time it takes even the simplest floating point operation to execute, all sorts of branching and testing—including entries and exits into and out of Mean 14—can be accomplished very quickly. For this reason, conditionals were left out of the Mean 14's instruction set. But that certainly doesn't mean that you couldn't add them if your particular application required them.

Using Mean 14

Making use of the Mean 14 processor in your machine language programs is easy. The only prerequisite, besides a working knowledge of assembly language, is a fundamental knowledge of the format of Applesoft variables.

1. Note that Mean 14 and the Applesoft subroutines that it calls could leave any and all registers in an undeterminable state. If you need certain registers in

specific states, it's a good idea to write yourself both a Save and a Restore routine and remember to JSR to the Save before entering Mean 14. You could even add these routines to the Mean 14 entry and exits if you like.

2. Enter Mean 14 with a JSR to MEAN 14 ($8E00 in the source listing provided). All code between this JSR and a Mean 14 "RET" will be interpeted by the Mean 14 processor. Remember that byte sequence is a function of the addressing mode. In the Implied mode, any operator is followed by the next operator. In Immediate mode, an operator is immediately followed by a five byte operand (constant) in Applesoft floating point variable format. In the Absolute mode, the operator must be followed by a two byte pointer to the first memory location containing a floating point value. In the Indirect mode, the operator is followed by a pointer, which points to a pointer, which points to a floating point value. Remember, all pointers must be in standard 6502 low-byte, high-byte order.

3. Consider the following section of code:

```
2000 SUB1      STY YSAVE       ; SAVE Y
2002           STX XSAVE       ; SAVE X
2004           JSR MEAN 14     ; ENTER MEAN 14
2007           DFB C0 00 03    ; *LDA $300
200A           DFB C4 05 03    ; *ADD $305
200D           DFB 45 81 00
2010           DFB 00 00 00    ; *SUB #1
2013           DFB 0C          ; *ABS
2014           DFB 81 40 03    ; *STA ($340)
2017           DFB 11          ; *RET
2018           LDX XSAVE       ; RESTORE X
201A           LDY YSAVE       ; RESTORE Y
201C           RTS
```

Both the X and Y registers were saved before entering Mean 14 in this example. To make the code representation less confusing, it's a good idea to show the Mean 14 mnemonic equivalents of the defined bytes in the comments field. I like to designate them with an asterisk but any appropriate scheme should do.

4. If your machine language routines are to be called from BASIC and if values obtained from Mean 14 operations will be used by BASIC, you might want to store values directly into the memory locations allocated to Applesoft variables. This will make the results of your machine language calculations directly available to BASIC. Although there are subroutines in Applesoft to find a variable by its name, they can take a lot of time to execute. An easier approach is to "know" where your variables are by allocating them first, in your BASIC program. Thus, if the first line of your program is:

```
10 A=0:B=0:C=0:D=0
```

then you'll know that the first variable is A, the second is B, etc. The pointer at locations $69,$69A tells you the beginning of the simple variable space, so you should be all set.

5. Be careful to avoid floating point errors such as Overflow and Division by Zero, as Applesoft routines tend to dump you into BASIC if an error occurs.

Format Of Mean 14 Operators

Mean 14 instructions are represented as single byte numeric values. Two quantities are represented in this byte — instruction and addressing mode. Since there was room to spare (there are only four addressing modes and twenty odd instructions) a very simple scheme was devised to include both. There are also many unused values so the instruction set could easily be expanded. An instruction is represented with the two high order bits indicating the addressing mode and the lower six bits indicating the operation

7 6	5 4 3 2 1 0
Addressing Mode	Instruction

Mean 14 Addressing Modes

The Mean 14 pseudo-machine processor instructions use four different addressing modes. They are:

IMMEDIATE
ABSOLUTE
INDIRECT
IMPLIED

IMMEDIATE — Just like any processor, the Mean 14 instructions that allow immediate addressing use the value following an operator in memory for the operand. Since we deal with floating point values, the five memory locations following the operator must contain the floating point operand. This must be in Applesoft variable format.

EX. Load FPAC1 with the value "0"

40	00 00 00 00 00	LDA#0
OPERATOR	OPERAND	SYMBOLIC

ABSOLUTE — The two bytes that follow the instruction (operator) in the absolute mode must contain the address of the first byte of the desired buffer. The value of the byte pointed at, and the values of pointer must be in low byte, high byte format.

EX. Store FPAC1 in locations $1F00-$1F04

C1	00 1F	STA $1F00-$1F04
OPERATOR	OPERAND	SYMBOLIC

INDIRECT — In this addressing mode, the two bytes that follow the operator must contain the address of a two byte pointer which points to the first byte of the buffer. This addressing mode is useful when loop processing a number of variables. It allows the pointer to the variable to be changed and, since the pointer is not a part of the Mean 14 object code, you needn't write self modifying code to perform a loop. Again, both the operand and the pointer must be represented in the low byte, high byte format.

EX. Store FPAC1 in $2FF0-$2FF4

81 00 20 STA($2000)

Where $2000,$2001 point at $2FF0

IMPLIED — Certain instructions perform operations which do not involve variables. These include register functions and exits from Mean 14.

EX. Transfer FPAC1 to FPAC2
02 TAB

EX. Exit Mean 14
11 RET

```
                    MEAN 14 INSTRUCTION SET

LDA   Load FPAC1 with memory                  M --> FPAC1

        IMMEDIATE = $40
        ABSOLUTE  = $C0
        INDIRECT  = $80

---------------------------------------------------------------

STA   Store FPAC1 in memory                   FPAC1 --> M

        ABSOLUTE  = $C1
        INDIRECT  = $81

---------------------------------------------------------------

TAB   Transfer FPAC1 to FPAC2           FPAC1 --> FPAC2

        IMPLIED   = $02

---------------------------------------------------------------

TBA   Transfer FPAC2 to FPAC1           FPAC2 --> FPAC1

        IMPLIED   = $03

---------------------------------------------------------------

ADD   Add memory to FPAC1               M + FPAC1 --> FPAC1

        IMMEDIATE = $44
        ABSOLUTE  = $C4
        INDIRECT  = $84

---------------------------------------------------------------

SUB   Subtract FPAC1 from memory      M - FPAC1 --> FPAC1

        IMMEDIATE = $45
        ABSOLUTE  = $C5
        INDIRECT  = $85
```

```
------------------------------------------------------------------------

MUL   Memory times FPAC1                    M * FPAC1 --> FPAC1

        IMMEDIATE = $46
        ABSOLUTE  = $C6
        INDIRECT  = $86

------------------------------------------------------------------------

DIV   Memory divided by FPAC1               M / FPAC1 --> FPAC1

        IMMEDIATE = $47
        ABSOLUTE  = $C7
        INDIRECT  = $87

------------------------------------------------------------------------

NOP   No operation                                      MPC + 1

        IMPLIED   = $08

------------------------------------------------------------------------

SQR   Square root of FPAC1                   √FPAC1 --> FPAC1

        IMPLIED   = $09

------------------------------------------------------------------------

EXP     FPAC2 raised to the power          FPAC2 ^ M --> FPAC1
         of memory

        IMMEDIATE = $4A
        ABSOLUTE  = $CA
        INDIRECT  = $8A
------------------------------------------------------------------------

INT   Integer value of FPAC1            INT ( FPAC1 ) --> FPAC1

        IMPLIED   = $0B

------------------------------------------------------------------------

ABS   Absolute value of FPAC1           ABS ( FPAC1 ) --> FPAC1

        IMPLIED   = $0C
------------------------------------------------------------------------

SGN   Value of the sign of              SGN ( FPAC1 ) --> FPAC1
        FPAC1

        IMPLIED   = $0D

------------------------------------------------------------------------

LOG   Natural log of FPAC1              LOG ( FPAC1 ) --> FPAC1

        IMPLIED   = $0E
```

```
-------------------------------------------------------------------

CVA   Convert two-byte integer                     M% --> FPAC1
         in Applesoft integer variable
         format to its floating point
         equivalent.

        ABSOLUTE   = $CF
        INDIRECT   = $8F

-------------------------------------------------------------------

CVB   Convert two-byte integer                  ML,MH --> FPAC1
         in 6502 format to its floating
         point equivalent.

        ABSOLUTE   = $D0
        INDIRECT   = $90

-------------------------------------------------------------------

RET   Exit MEAN 14                                  MPC --> PC

        IMPLIED    = $11
```

```
0800              ;*************************
0800              ;*                       *
0800              ;* MEAN-14 FP PROCESSOR *
0800              ;*      R.M. MOTTOLA     *
0800              ;*                       *
0800              ;*        MEAN-14        *
0800              ;*                       *
0800              ;*  COPYRIGHT (C) 1981   *
0800              ;*   MICRO INK, INC.     *
0800              ;* CHELMSFORD, MA 01824 *
0800              ;*  ALL RIGHTS RESERVED *
0800              ;*                       *
0800              ;*************************
0800              ;
0800              ;*SOFTWARE ADDRESSES
0800              ;*
0800              TEMPL  EPZ $1E
0800              TEMPH  EPZ $1F
0800              MPCL   EPZ $4C
0800              MPCH   EPZ $4D
0800              FPAC1  EPZ $9D
0800              FPAC2  EPZ $A5
0800              ;
0800              ;
0800              ;
0800              ;FIRMWARE ADDRESSES
0800              ;
0800              ;
0800              INT>FP EQU $E2F2
0800              FPSUB  EQU $E7A7
0800              FPADD  EQU $E7BE
0800              FPLOG  EQU $E941
0800              FPMUL  EQU $E97F
0800              FPDIV1 EQU $EA66
0800              FPLOAD EQU $EAF9
0800              FPSTR  EQU $EB2B
0800              TR2>1  EQU $EB53
0800              TR1>2  EQU $EB63
0800              FPSGN  EQU $EB90
0800              FPABS  EQU $EBAF
0800              FPINT  EQU $EC23
0800              FPSQR  EQU $EE8D
0800              FPEXP  EQU $EE94
0800              ;
8E00                     ORG $8E00
8E00                     OBJ $800
8E00              ;
8E00              ;MEAN 14 PSEUDO-MACHINE
8E00              ;FLOATING POINT PROCESSOR
8E00              ;
8E00 68           MEAN14 PLA               ;GET M14 CODE LOCATION
8E01 854C                STA MPCL          ;FROM RETURN ADDRESS
8E03 68                  PLA
8E04 854D                STA MPCH
8E06 205F8E              JSR PCINC
8E09 200F8E       M14A   JSR M14B
8E0C 4C098E              JMP M14A
8E0F A000         M14B   LDY #$0
8E11 B14C                LDA (MPCL),Y      ;GET ONE INSTRUCTION
8E13 AA                  TAX
8E14 293F                AND #$3F          ;GET CORRECT SUBROUTINE
8E16 0A                  ASL               ;ADDRESS FROM TABLE
8E17 A8                  TAY
8E18 C8                  INY
8E19 B9A08E              LDA SUBTBL,Y      ;AND SHOVE IT
8E1C 48                  PHA
8E1D 88                  DEY
8E1E B9A08E              LDA SUBTBL,Y
8E21 48                  PHA
8E22 205F8E              JSR PCINC         ;INCREM. M14 P.C. COUNT
8E25 8A                  TXA
8E26 29C0                AND #$C0          ;GET ADDRESSING MODE
8E28 F034                BEQ M14G          ;IMPLIED?
8E2A 1020                BPL M14D          ;IMMEDIATE?
8E2C 2940                AND #$40
```

```
8E2E D013              BNE M14C           ;ABSOLUTE?
8E30 B14C              LDA (MPCL),Y       ;INDIRECT
8E32 851E              STA TEMPL          ;GET POINTER TO ADDRESS
8E34 C8                INY                ;OF OPERAND
8E35 B14C              LDA (MPCL),Y
8E37 851F              STA TEMPH
8E39 88                DEY
8E3A B11E              LDA (TEMPL),Y
8E3C 48                PHA
8E3D C8                INY
8E3E B11E              LDA (TEMPL),Y
8E40 48                PHA
8E41 9013              BCC M14E
8E43 B14C      M14C    LDA (MPCL),Y       ;GET ADDRESS OF
8E45 48                PHA                ;OPERAND
8E46 C8                INY
8E47 B14C              LDA (MPCL),Y
8E49 48                PHA
8E4A 900A              BCC M14E
8E4C A54C      M14D    LDA MPCL           ;SAVE P.C. AS ADDRESS
8E4E 48                PHA                ;OF IMMEDIATE OPERAND
8E4F A54D              LDA MPCH
8E51 48                PHA
8E52 A905              LDA #$5            ;AND OFFSET P.C. 5 BYTES
8E54 9002              BCC M14F
8E56 A902      M14E    LDA #$2            ;OFFSET P.C. 2 BYTES
8E58 20618E    M14F    JSR PCADD
8E5B 68                PLA                ;PULL OPERAND ADDRESS
8E5C           ;AND TRANSFER
8E5C A8                TAY                ;TO A AND Y REGS FOR SUBS
8E5D 68                PLA
8E5E 60        M14G    RTS                ;JMP VIA RTS
8E5F           ;
8E5F A901      PCINC   LDA #$1
8E61 18        PCADD   CLC
8E62 654C              ADC MPCL
8E64 854C              STA MPCL
8E66 9003              BCC PC1
8E68 E64D              INC MPCH
8E6A 18                CLC
8E6B A000      PC1     LDY #$0
8E6D 60                RTS
8E6E           ;
8E6E AA        STR     TAX
8E6F 4C2BEB            JMP FPSTR
8E72 851E      CONV1   STA TEMPL
8E74 841F              STY TEMPH
8E76 A000              LDY #$0
8E78 B11E              LDA (TEMPL),Y
8E7A 48                PHA
8E7B C8                INY
8E7C B11E      C1A     LDA (TEMPL),Y
8E7E A8                TAY
8E7F 68                PLA
8E80 20F2E2            JSR INT>FP
8E83 A5A2              LDA FPAC1+$5
8E85 1007              BPL NOOP
8E87 A9C4              LDA #VALUE1
8E89 A08E              LDY /VALUE1
8E8B 20BEE7            JSR FPADD
8E8E 60        NOOP    RTS
8E8F 851E      CONV2   STA TEMPL
8E91 841F              STY TEMPH
8E93 A001              LDY #$1
8E95 B11E              LDA (TEMPL),Y
8E97 48                PHA
8E98 88                DEY
8E99 F0E1              BEQ C1A
8E9B 68        RETURN  PLA                ;PULL MEAN 14 RETURN
8E9C 68                PLA                ;ADDRESS FROM STACK
8E9D 6C4C00            JMP (MPCL)
8EA0           ;
8EA0           ;
8EA0           ;SUBROUTINE ADDRESS TABLE
8EA0           ;
```

```
8EA0 F8EA    SUBTBL ADR FPLOAD-$1
8EA2 6D8E           ADR STR-$1
8EA4 62EB           ADR TR1>2-$1
8EA6 52EB           ADR TR2>1-$1
8EA8 BDE7           ADR FPADD-$1
8EAA A6E7           ADR FPSUB-$1
8EAC 7EE9           ADR FPMUL-$1
8EAE 65EA           ADR FPDIV1-$1
8EB0 8D8E           ADR NOOP-$1
8EB2 8CEE           ADR FPSQR-$1
8EB4 93EE           ADR FPEXP-$1
8EB6 22EC           ADR FPINT-$1
8EB8 AEEB           ADR FPABS-$1
8EBA 8FEB           ADR FPSGN-$1
8EBC 40E9           ADR FPLOG-$1
8EBE 718E           ADR CONV1-$1
8EC0 8E8E           ADR CONV2-$1
8EC2 9A8E           ADR RETURN-$1
8EC4         ;
8EC4         ;FLOATING POINT CONSTANTS
8EC4         ;
8EC4 910000  VALUE1 HEX 9100000000       ; % 65536
8EC7 0000
8EC9         ;
8EC9         ;
8EC9         ;
8EC9         ;
8EC9         LENGTH EQU *-MEAN14
                    END

***** END OF ASSEMBLY


          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5  *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

TEMPL  001E  TEMPH  001F  MPCL   004C  MPCH   004D  FPAC1  009D  FPAC2  00A5


** ABSOLUTE VARABLES/LABELS

INT>FP E2F2  FPSUB  E7A7  FPADD  E7BE  FPLOG  E941  FPMUL  E97F  FPDIV1 EA66
FPLOAD EAF9  FPSTR  EB2B  TR2>1  EB53  TR1>2  EB63  FPSGN  EB90  FPABS  EBAF
FPINT  EC23  FPSQR  EE8D  FPEXP  EE94  MEAN14 8E00  M14A   8E09  M14B   8E0F
M14C   8E43  M14D   8E4C  M14E   8E56  M14F   8E58  M14G   8E5E  PCINC  8E5F
PCADD  8E61  PC1    8E6B  STR    8E6E  CONV1  8E72  C1A    8E7C  NOOP   8E8E
CONV2  8E8F  RETURN 8E9B  SUBTBL 8EA0  VALUE1 8EC4  LENGTH 00C9

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:015A
```

2
I/O ENHANCEMENTS

INTRODUCTION

In order to communicate with your computer, an I/O device is a necessity. The keyboard and video output are the standard I/O devices of the Apple, with a printer being another commonly-found output device. Obviously, any enhancements to the I/O capabilities will promote a better interface between programmer and Apple. In this chapter, some enhancements are described which should make working with your Apple a bit easier.

''Screen Write/File,'' by Bruce Baxter, provides a method to directly save and retrieve text screens to and from the disk. This technique can often save valuable program memory space. ''Bi-Directional Scrolling,'' by Roger Wagner, allows scrolling through memory either backwards or forwards. Any portion of memory may be scrolled through and viewed (in ASCII) with this routine. ''Apple II Integer BASIC Program List by Page,'' by David Partyka, lets the user list through an Integer BASIC program page-by-page on the Apple video screen.

The following two routines will be of special interest to printer owners. ''Paged Printer Output for the Apple,'' by Gary Little, provides for printer output to be divided into variable size pages. It also allows a pause for single sheet paper feed. And ''Hex Printer,'' by LeRoy Moyer, facilitates machine language disassembly listings on you printer.

Screen Write/File Routine

by B.E. Baxter

Here is a useful and instructive routine which makes it simple to edit the Apple screen and save the screen image on disk.

The screen write/file routine is a simple 73-byte device to take control away from the monitor and write directly to the screen. All of the escape editing capabilities are supported so that it is very easy to enter and modify up to and including 21 lines of text. It is equally easy to save the screen image to disk after completion of text entry.

How it Works

The source code is straightforward and makes liberal use of monitor routines. Upon entry the cursor is homed and placed on line 1 (not zero). The block labeled KEY continually polls the keyboard and outputs characters through COUT (VIDOUT [$FBFD] could also be used if printer services are not wanted). The limited editing facilities of the monitor are invoked by typing (escape) followed by one of the command characters. Keyboard entry of CNTL Q is used to exit the routine and return to BASIC via $3D0. Automatic exit is also obtained at line 43. Upon exit, the bell will sound and the BASIC prompt character will appear with the file parameters displayed at the end of the line. At this point the file must be saved using the command, (BSAVE File name) A$0400, L$03CF (RETURN). The parenthetical expressions must be typed by the user; that is, type BSAVE file name, then trace over the remainder of the line with the right arrow to place it into the keyboard buffer and at the end of the line press RETURN. Although I do not find it necessary, a monitor MOVE to page 2 could be set up and inserted between lines 57 and 58 of the source listing. This would provide back-up in case BSAVE command is messed up. The object code is assembled at $0350 and is $49 bytes long.

Command Summary

In summary, the usage commands are:

Entry to Routine

From BASIC	Call 848
From Monitor	$0350G

Exit to BASIC Mode

User	(Control) Q
Automatic	Line 43

Edit Screen (See Apple Ref. Materials)

(Escape)	@: Home cursor (Clear text)
	A: Advance cursor
	B: Backspace cursor
	C: Move cursor down 1 line
	D: Move cursor up 1 line
	E: Clear from cursor to end of line
	F: Clear from cursor to end of screen

Save Screen Image

[BSAVE file name]A$0400,L$03CF[CR]   [] = typed by user

Of course it doesn't make much sense to idly write to the screen without some useful purpose. I use the routine to create instruction and documentation files. These files are especially valuable for object code utilities by providing ready access to usage and entry point information. Once the file has been created, it can be handled just like any other file. BLOADing (file name) will immediately display its contents on the screen without requiring any otherwise useful memory. Instruction/print statements in BASIC programs can therefore be eliminated to be replaced by deferred execution BLOAD disk commands for a very efficient use of main memory.

```
0800          1  ;************************
0800          2  ;*                      *
0800          3  ;*     SCREEN WRITER    *
0800          4  ;*      BRUCE BAXTER    *
0800          5  ;*                      *
0800          6  ;*      SCREEN-WRITE    *
0800          7  ;*                      *
0800          8  ;*   COPYRIGHT (C) 1981 *
0800          9  ;*    MICRO INK, INC.   *
0800         10  ;* CHELMSFORD, MA 01824 *
0800         11  ;*  ALL RIGHTS RESERVED *
0800         12  ;*                      *
0800         13  ;************************
0800         14  ;
0800         15  ;
0350         16         ORG $350
0350         17         OBJ $800
0350         18  ;
0350         19  ;
0350         20  ;
0350         21  CV     EPZ $25
0350         22  POS    EPZ $09
0350         23  ;
0350         24  COUT   EQU $FDED
0350         25  HOME   EQU $FC58
0350         26  TABV   EQU $FB5B
0350         27  RDCHAR EQU $FD35
```

```
0350              CROUT  EQU $FD8E
0350              BELL   EQU $FF3A
0350              ;
0350  2058FC             JSR HOME
0353  208EFD             JSR CROUT
0356              ;
0356  2035FD      KEY    JSR RDCHAR
0359  C991               CMP #$91
035B  F00C               BEQ QUIT
035D  A625               LDX CV
035F  E016               CPX #$16
0361  F006               BEQ QUIT
0363  20EDFD             JSR COUT
0366  4C5603             JMP KEY
0369              ;
0369  A916        QUIT   LDA #$16
036B  8525               STA CV
036D  205BFB             JSR TABV
0370  203AFF             JSR BELL
0373  A9E4               LDA #$E4
0375  8509               STA POS
0377  A907               LDA #$07
0379  850A               STA POS+1
037B  A000               LDY #$00
037D              ;
037D  B98A03      OUT    LDA DATA,Y
0380  9109               STA (POS),Y
0382  C8                 INY
0383  C00F               CPY #$0F
0385  D0F6               BNE OUT
0387  20D003             JSR $03D0
038A  A0C1A4      DATA   ASC " A$0400,L$03CF "
038D  B0B4B0
0390  B0ACCC
0393  A4B0B3
0396  C3C6A0
                         END

***** END OF ASSEMBLY


          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.   LABEL. LOC.   LABEL. LOC.

** ZERO PAGE VARIABLES:

CV      0025   POS     0009

** ABSOLUTE VARABLES/LABELS

COUT    FDED   HOME    FC58   TABV    FB5B   RDCHAR FD35
CROUT   FD8E   BELL    FF3A   KEY     0356   QUIT   0369   OUT    037D   DATA   038A


SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0072
```

Bi-Directional Scrolling

by Roger Wagner

Everyone knows that a teletype only moves the paper in one direction — up. Likewise, the Apple display only scrolls one way — up. Now you can have scrolling in both directions with these routines.

By using the following machine language routines, it is possible to scroll either text/gr page in either direction.

The up-scroll routine is derived from Apple Computer's Reference Manual with the difference being that a zero-page location is referred to in order to determine which page to scroll. The down scroll routine makes similar use of the same zero-page byte.

How to Use the Program

To use the routine a few entry conditions must be met:

1. Load the binary routine into the $300 page of memory starting at $300.

2. Set pointers 6,7 and 8,9. If you want to bring new information onto the screen from RAM as you scroll, locations 6,7 must point to the location in memory where the data to be loaded onto the top line of the screen will come from when you scroll the screen page down. Similarly 8,9 point to the place in memory to get the data for the bottom line when you scroll up.

   If you want to use this routine to directly view memory, the easiest way to set the pointers 6,7 and 8,9 is to set 8 and 9 to the address you want to start viewing at. Put the low order byte in 8 and the high order in 9 then scroll up 25 times. (The screen height plus 1.) Then set 6,7 to the same value as 8,9 were originally (i.e., the low and high byte bring the starting address). Last of all, scroll back down one line to bring the starting address line into position as the first line of text visible at the top of the screen.

   If you do not want new data brought onto the screen, then 6,7 and 8,9 will have to point to a part of memory that contains 40 blank space characters. One way to do this is to freeze one blank line on either page 1 or 2, and then set 6,7 and 8,9 to that location. These pointers must be reset to that value each time the scroll is done. This is because normally the scroll routine updates 6,7 and 8,9 by the screen width so as to remain synchronized with the screen display. Another technique is to just clear the top or bottom line to blanks each time a scroll is done.

3. Location 5 must hold a 4 for page 1 scrolling, and an 8 for page 2.

4. Now when you want the screen to scroll just 'CALL 768' to scroll up, and '845' to scroll down.

Special Notes:

If you are going to use page 2 of text/gr in Integer BASIC, be sure to protect the variables with a 'LOMEM': 3072. This may be done before running the program, or if you know how, put as an early line in the program.

To use page 2 in Applesoft is more difficult, but can be done. First, location $3AB in the machine code must be changed from $05 to $1F. Also, you must POKE 31 with a 4 or 8 as compared to the POKE 5 in Integer.

The real rub is that Applesoft programs normally begin in memory at $800 (hex) which conflicts with page 2 use. The way around this is to do a 'POKE 104, 12:POKE 3072, 0' before loading your program. After loading do a 'CALL 54514' (unnecessary with DOS 3.2). Unless you do a 'RESET', 'Control-B, other Applesoft programs will continue to load in at this higher location. Unfortunately, use of page 2 with the RAM version of Applesoft is to my knowledge impossible. (Sorry....)

If you wish to move the scrolling routine, the only location-dependent aspects of the code are 5 'JSR's and 1 'JMP' within it. Since these operations always reference absolute addresses they will have to be rewritten. Of course, if you have a relocate utility, it is that much easier.

For further enlightenment, see the sample Integer BASIC program which makes use of the scrolling routine. Have fun!

Location Dependent

```
$303:  JSR  $39E
 319:  JSR   39E
 34A:  JMP   39C
 353:  JSR   39E
 369:  JSR   39E
 39E:  JSR   3A6
```

If page 2 of text/gr is to be used, it must be protected by a 'LOMEM:3072' for Integer BASIC, or a 'special load' (as described in article) when using Applesoft.

Note: $3AB must be changed from $05 to $1F for Applesoft.

```
0800                ;****************************
0800                ;*                          *
0800                ;* APPLE SCROLLING ROUTINE *
0800                ;*       ROGER WAGNER       *
0800                ;*                          *
0800                ;*          SCROLL          *
0800                ;*                          *
0800                ;*   COPYRIGHT (C) 1981     *
0800                ;*    MICRO INK, INC.       *
0800                ;*  CHELMSFORD, MA 01824    *
0800                ;*   ALL RIGHTS RESERVED    *
0800                ;*                          *
0800                ;****************************
0800                ;
0800                ; THIS WILL LET EITHER
0800                ; PAGE SCROLL IN EITHER
0800                ; DIRECTION.  IT IS PRI-
0800                ; MARILY DESIGNED TO FEED
0800                ; NEW SCREEN DATA IN FROM
0800                ; A GIVEN RANGE OF RAM.
0800                ;
0800                ;
0800                ;
0800                       OBJ $800
0300                       ORG $300
0300                ;
0300                ;
0300                WNDLFT EPZ $20
0300                WNDWID EPZ $21
0300                WNDTOP EPZ $22
0300                WNDBTM EPZ $23
0300                CH     EPZ $24
0300                CV     EPZ $25
0300                BASL   EPZ $28
0300                BASH   EPZ $29
0300                BAS2L  EPZ $2A
0300                BAS2H  EPZ $2B
0300                PAGE   EPZ $05
0300                ;* FOR APPLESOFT USE PAGE EQU $1F
0300                ;* PAGE MUST HOLD $04 FOR PG 1,
0300                ;* $08 FOR PG 2
0300                SCRNTP EPZ $06
0300                ;* $06, $07 = LO/HI BYTES
0300                ;* OF START OF LINE JUST BEFORE
0300                ;* TOP LINE
0300                SCRNBM EPZ $08
0300                ;* $08,$09 = LO/HI BYTES
0300                ;* OF START OF LINE JUST AFTER
0300                ;* BOTTOM LINE
0300                ;*
0300                ;*
0300 A522           SCROLL LDA WNDTOP
0302 48                    PHA
0303 209E03                JSR VTABZ
0306 A528           NXTLN  LDA BASL
0308 852A                  STA BAS2L
030A A529                  LDA BASH
030C 852B                  STA BAS2H
030E A421                  LDY WNDWID
0310 88                    DEY
0311 68                    PLA
0312 6901                  ADC #$01
0314 C523                  CMP WNDBTM
0316 B00D                  BCS LDBTM
0318 48                    PHA
0319 209E03                JSR VTABZ
031C B128           NXTCHR LDA (BASL),Y
031E 912A                  STA (BAS2L),Y
0320 88                    DEY
```

```
0321 10F9                  BPL NXTCHR
0323 30E1                  BMI NXTLN
0325 A000        LDBTM     LDY #00
0327 B108        LD2       LDA (SCRNBM),Y
0329 9128                  STA (BASL),Y
032B C8                    INY
032C C421                  CPY WNDWID
032E 90F7                  BCC LD2
0330 18          CRRCT     CLC
0331 A506                  LDA SCRNTP
0333 6521                  ADC WNDWID
0335 8506                  STA SCRNTP
0337 A507                  LDA SCRNTP+1
0339 6900                  ADC #00
033B 8507                  STA SCRNTP+1
033D 18                    CLC
033E A508                  LDA SCRNBM
0340 6521                  ADC WNDWID
0342 8508                  STA SCRNBM
0344 A509                  LDA SCRNBM+1
0346 6900                  ADC #00
0348 8509                  STA SCRNBM+1
034A 4C9C03                JMP VTAB
034D             ;*
034D             ;*
034D 38          SCRLDN    SEC
034E A523                  LDA WNDBTM
0350 E901                  SBC #$01
0352 48                    PHA
0353 209E03                JSR VTABZ
0356 A528        NXTLN2    LDA BASL
0358 852A                  STA BAS2L
035A A529                  LDA BASH
035C 852B                  STA BAS2H
035E A421                  LDY WNDWID
0360 88                    DEY
0361 68                    PLA
0362 E900                  SBC #$00
0364 C522                  CMP WNDTOP
0366 300D                  BMI LDTOP
0368 48                    PHA
0369 209E03                JSR VTABZ
036C B128        NXTCR2    LDA (BASL),Y
036E 912A                  STA (BAS2L),Y
0370 88                    DEY
0371 10F9                  BPL NXTCR2
0373 30E1                  BMI NXTLN2
0375 A000        LDTOP     LDY #$00
0377 B106        LT2       LDA (SCRNTP),Y
0379 9128                  STA (BASL),Y
037B C8                    INY
037C C421                  CPY WNDWID
037E 90F7                  BCC LT2
0380 38          CRRT2     SEC
0381 A506                  LDA SCRNTP
0383 E521                  SBC WNDWID
0385 8506                  STA SCRNTP
0387 A507                  LDA SCRNTP+1
0389 E900                  SBC #00
038B 8507                  STA SCRNTP+1
038D 38                    SEC
038E A508                  LDA SCRNBM
0390 E521                  SBC WNDWID
0392 8508                  STA SCRNBM
0394 A509                  LDA SCRNBM+1
0396 E900                  SBC #00
0398 8509                  STA SCRNBM+1
039A 60                    RTS
```

```
039B 00        138          BRK
039C           139   ;*
039C           140   ;*
039C A525      141   VTAB   LDA CV
039E 20A603    142   VTABZ  JSR BASCLC
03A1 6520      143          ADC WNDLFT
03A3 8528      144          STA BASL
03A5 60        145          RTS
03A6           146   ;*
03A6           147   ;*
03A6 48        148   BASCLC PHA
03A7 4A        149          LSR
03A8 2900      150          AND #$00
03AA 0505      151          ORA PAGE
03AC 8529      152          STA BASH
03AE 68        153          PLA
03AF 2918      154          AND #$18
03B1 9002      155          BCC BSCLC2
03B3 697F      156          ADC #$7F
03B5 8528      157   BSCLC2 STA BASL
03B7 0A        158          ASL
03B8 0A        159          ASL
03B9 0528      160          ORA BASL
03BB 8528      161          STA BASL
03BD 60        162   END    RTS
              163          END

***** END OF ASSEMBLY


            *************************
            *                       *
            * SYMBOL TABLE -- V 1.5 *
            *                       *
            *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

WNDLFT 0020  WNDWID 0021  WNDTOP 0022 WNDBTM 0023 CH     0024 CV     0025
BASL   0028  BASH   0029  BAS2L  002A BAS2H  002B PAGE   0005 SCRNTP 0006
SCRNBM 0008

** ABSOLUTE VARABLES/LABELS

SCROLL 0300  NXTLN  0306  NXTCHR 031C LDBTM  0325 LD2    0327
CRRCT  0330  SCRLDN 034D  NXTLN2 0356 NXTCR2 036C LDTOP  0375 LT2    0377
CRRT2  0380  VTAB   039C  VTABZ  039E BASCLC 03A6 BSCLC2 03B5 END    03BD


SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0102
```

```
  1 REM    ************************
  2 REM    *                      *
  3 REM    * APPLE SCROLLING RTNE *
  4 REM    *     ROGER WAGNER     *
  5 REM    *                      *
  6 REM    *       SCROLLER       *
  7 REM    *                      *
  8 REM    *  COPYRIGHT (C) 1981  *
  9 REM    *    MICRO INK, INC.   *
 10 REM    * CHELMSFORD, MA 01824 *
 11 REM    *  ALL RIGHTS RESERVED *
 12 REM    *                      *
 13 REM    ************************
 14 REM
 16 REM
 18 LOMEM:3072
 20 REM   OR SET LOMEM MANUALLY BEFORE RUNNING.
 30 CALL -936: INPUT "PAGE 1 OR 2?",PAGE
 40 PRINT "INPUT ADDRESS (<32767) TO START AT:": INPUT A
 50 REM   TO SCROLL WITHOUT BRINGING IN NEW DATA ENTER '0' FOR ADDRESS.
 60 IF A#0 THEN 100: TEXT : CALL -936: POKE 34,1:
    REM   FREEZE ONE BLANK LINE AT TOP OF SCREEN
 70 VTAB 12: PRINT "(SAMPLE PG. 1 SCREEN DATA)"
 75 POKE 5,PAGE*4: IF PAGE=2 THEN POKE -16299,0
 80 POKE 6,0: POKE 7,4: POKE 8,0: POKE 9,4:
    REM   BRING NEW SCREEN DATA FROM  THAT BLANK LINE
 90 GOTO 150
100 LB=A MOD 256:HB=A/256
110 POKE 5,PAGE*4: IF PAGE=2 THEN POKE -16299,0
120 POKE 8,LB: POKE 9,HB
130 FOR I=1 TO 25: CALL 768: NEXT I
140 POKE 6,LB: POKE 7,HB
150 KEY= PEEK (-16384): POKE -16368,0
160 IF KEY=149 THEN CALL 768: REM   RT. ARROW KEY TO SCROLL UP
170 IF KEY=136 THEN CALL 845: REM   LFT. ARROW KEY TO SCROLL DOWN
180 IF KEY#136 AND KEY#149 OR A#0 THEN 190: POKE 6,0: POKE 7,4: POKE 8,
    0: POKE 9,4: REM   RESET 6,7 & 8,9 TO POINT AT BLANK LINE
190 IF KEY#177 THEN 200: POKE 5,4: POKE -16300,0: REM   '1' FOR PAGE 1
200 IF KEY#178 THEN 210: POKE 5,8: POKE -16299,0: REM   '2' FOR PAGE 2
210 IF KEY#216 THEN 150: POKE -16300,0: TEXT : CALL -868: PRINT "BYE":
    END
```

Apple II Integer BASIC Program List by Page

by Dave Partyka

Viewing long program listings on the Apple's small video display has been a consistent source of frustration to the programmer. The solution implemented here allows the user to view listings page-by-page.

If you own an Apple II, I'm sure you feel there could be a better way to list a program. Now you either list the whole program and watch it go by faster than you can read it, or you list it by line numbers. When you list it by line numbers, you may get two lines or you may get more lines than will fit on the screen.

Using the assembler program listed, and the Integer BASIC of the Apple II, you can list your Integer BASIC programs one page (screen) at a time with a page number at the bottom of each. Pressing just about any key (except B, P, or S) will clear the screen and display the next page adding one to the page number. By pressing keys you display your program a page at a time, not only two lines here, or too many lines there.

The B, P, or S keys are special function keys. The B key (for beginning) will clear the screen and display your program from the first page. This comes in handy when you're in the middle or near the end of the display and you want to see some subroutines or anything else at the beginning. Just press the B key and you are at the beginning, ready to start over.

The next key P (for page) will clear the screen and start displaying your program, stopping at the page number you keyed in. For example, if you are at page 25 and you want to back up 2 pages, you press P0023. P will clear the screen and the Apple will beep as you key in the four digits. You have to enter four digits so the leading zeros are necessary. After the last digit is pressed, your program will be displayed from the beginning, stopping at page 23. This is faster than pressing the B key and other ones until you get to page 23.

The last key, S (for Stop) gets you out of the list program and back to the Apple II BASIC. This key is used when you find a place in your program where you want to add or delete a line. If you don't press the S key and you try to do anything, as soon as you press a key the next page will be displayed.

There are two ways to activate this program. From monitor press CTRL-Y then the RETURN key, or from BASIC type CALL 1016 then press the RETURN key. As long as you don't use the area from hex 300 to 3FF, this program will remain in memory. Once the list program is activated, it is entered only when the screen display reaches the bottom of the screen. If the end of your program ends anywhere but the bottom of the screen, the Apple II will return to BASIC but the list program will still be activated. To deactivate the list program, type CALL 1016, press the RETURN key, then press the S key for stop, or press the RETURN key to skip to the bottom of the page and press the S key to stop.

If you ran a BASIC program and the list program is still activated, then the results you get will depend on your program. Some programs won't be affected at all. Others will stop if the listing reaches the bottom of the screen. Pressing a key will start the program again. Other programs might be able to make use of this assembler routine by stopping the display at the bottom of the screen.

Using this assembler program, you'll find it easier to de-bug your programs or just follow the flow of any program.

```
0800                ;*************************
0800                ;*                       *
0800                ;*     LIST BY PAGE      *
0800                ;*     DAVID PARTYKA     *
0800                ;*                       *
0800                ;*       PAGE LIST       *
0800                ;*                       *
0800                ;*  COPYRIGHT (C) 1981   *
0800                ;*    MICRO INK, INC.    *
0800                ;* CHELMSFORD, MA 01824 *
0800                ;*  ALL RIGHTS RESERVED *
0800                ;*                       *
0800                ;*************************
0800                ;
0800                ;
0800                ;
0800                BASL    EPZ $28         ;LEFT CHAR POS ON LINE
0800                BASH    EPZ $29
0800                CSWL    EPZ $36         ;MONITOR OUTPUT  HOOK
0800                CSWH    EPZ $37
0800                ;
0800                KBD     EQU $C000       ;KEYBOARD INPUT
0800                KBDSTB  EQU $C010       ;KEYBOARD STROBE
0800                BASIC2  EQU $E003       ;BASIC WARM ENTRY
0800                LIST    EQU $E04B       ;BASIC LISTING ROUTINE
0800                BELL    EQU $FBDD       ;MONITOR BELL ROUTINE
0800                HOME    EQU $FC58       ;MONITOR CLEARSCREEN
0800                COUT1   EQU $FDF0       ;CHARACTER OUTPUT ROUTINE
0800                SAVE    EQU $FF4A       ;REGISTER SAVE ROUTINE
0800                RESTOR  EQU $FF3F       ;REGISTER RESTORE ROUTINE
0800                ;
0300                        ORG $0300
0300                        OBJ $0800
0300                ;
0300                ;
0300 A922           INIT    LDA #MAIN       ;LOAD BEGINNING
0302 8536                   STA CSWL        ;ADDRESS OF MAIN
0304 A903                   LDA /MAIN       ;PROGRAM IN USER
0306 8537                   STA CSWH        ;OUTPUT LOCATIONS.
0308 20E603         BEG     JSR HLOD        ; LOAD HIGH VALUES.
030B A900           ZPNO    LDA #$00        ;MOVE ZEROS TO
030D 8DF403                 STA PGHI        ;PAGE COUNT
0310 8DF503                 STA PGLO        ;LOCATIONS.
0313 2058FC                 JSR HOME        ;CLEAR SCREEN.
0316 204BE0                 JSR LIST        ;START BASIC LIST.
0319                ;
0319 209603                 JSR ADD1        ;ADD1 TO PAGE#.
031C 20E603                 JSR HLOD        ;LOAD PAGE HOLD WITH FF.
031F 4C03E0                 JMP BASIC2      ;RETURN TO BASIC CONTROL.
0322 204AFF         MAIN    JSR SAVE        ;SAVE REGISTERS
0325 A528                   LDA BASL        ;CHECK SCREEN ADDRESS
0327 4529                   EOR BASH        ;FOR 07 DO THE
0329 C9D7                   CMP #$D7        ;24TH LINE.
032B D051                   BNE DISP        ;IF NOT = BRANCH.
032D 209603                 JSR ADD1        ;ADD 1 TO PAGE #.
0330                ;
0330 ADF603                 LDA PHOLD       ;CHECK PAGE HOLD,
0333 C9FF                   CMP #$FF        ;IF = FF THEN THE P
0335 F019                   BEQ NPRES       ; KEY WASN'T PRESSED.
0337 ADF403                 LDA PGHI        ;COMPARE PAGE #
033A CDF603                 CMP PHOLD       ;WITH PAGE HOLD,
033D D008                   BNE CLR         ;IF EQUAL
033F ADF503                 LDA PGLO        ;BRANCH TO THE
0342 CDF703                 CMP PHOLD+1     ;LOOP ROUTINE
0345 F006                   BEQ LOOPR       ;ELSE
0347 2058FC         CLR     JSR HOME        ;CLEAR SCREEN
034A 4C8103                 JMP RR          ;CONTINUE PRINTING.
034D 20E603         LOOPR   JSR HLOD        ;LOAD PAGE HOLD WITH FF.
0350 2C00C0         NPRES   BIT KBD         ;LOOP UNTIL A
```

```
0353 10FB              BPL NPRES      ;KEY IS PRESSED.
0355 AD00C0            LDA KBD        ;WHEN KEY IS PRESSED
0358 8D10C0            STA KBDSTB     ;CLEAR KEY STROBE
035B C9D3              CMP #$D3       ;AND COMPARE FOR S.
035D D00B              BNE CMPB       ;IF NOT = BRANCH.
035F A9F0              LDA #$F0       ;IF S STORE
0361 8536              STA CSWL       ;NORMAL ADDRESS
0363            ;
0363 A9FD              LDA #$FD       ;IN THE USER
0365 8537              STA CSWH       ;OUTPUT LOCATIONS.
0367 4C03E0            JMP BASIC2     ;RETURN TO BASIC CONTROL.
036A C9C2       CMPB   CMP #$C2       ;B KEY PRESSED?
036C F09A              BEQ BEG        ;IF YES BRANCH.
036E C9D0              CMP #$D0       ;P KEY PRESSED?
0370 D00C              BNE DISP       ;IF NO BRANCH.
0372 A200              LDX #$00       ;IF YES THEN GET
0374 20CF03            JSR GTPG       ;2 DIGITS OF PAGE#
0377 E8                INX            ;UP INDEX AND
0378 20D203            JSR GTPG1      ;GET NEXT TWO DIGITS.
037B 4C0B03            JMP ZPNO       ;JUMP TO ZERO PAGE #.
037E 2058FC     DISP   JSR HCME       ;CLEAR SCREEN.
0381 203FFF     RR     JSR RESTOR     ;RESTORE REGISTERS
0384 4CF0FD            JMP COUT1      ;DISPLAY ROUTINE.
0387 A8         PRINT  TAY            ;SAVE ACCUM. AND
0388 290F              AND #$0F       ;CONVERT LOW ORDER
038A 09B0              ORA #$B0       ;BYTE TO DECIMAL AND
038C 9DF407            STA $7F4,X     ;PRINT PAGE #.
038F 98                TYA            ;GET ACCUM. AND
0390 6A                ROR            ;ROTATE
0391 6A                ROR            ;HIGH ORDER
0392 6A                ROR            ;BYTE TO THE
0393 6A                ROR            ;LOW ORDER
0394 CA                DEX            ;BYTE AND
0395 60                RTS            ;RETURN.
0396 F8         ADD1   SED            ;SET DECIMAL MODE.
0397 18                CLC            ;CLEAR CARRY FLAG.
0398 ADF503            LDA PGLO       ;ADD
039B 6901              ADC #$01       ;1
039D 8DF503            STA PGLO       ;TO
03A0 ADF403            LDA PGHI       ;THE
03A3 6900              ADC #$00       ;PAGE
03A5 8DF403            STA PGHI       ;NUMBER.
03A8 D8                CLD            ;CLEAR DECIMAL MODE.
03A9 A203              LDX #$03       ;SET IND-X.
03AB ADF503            LDA PGLO       ;GET PAGE # LOW.
03AE 208703            JSR PRINT      ;PRINT 1ST DIGIT.
03B1 208703            JSR PRINT      ;PRINT 2ND DIGIT.
03B4 ADF403            LDA PGHI       ;GET PAGE # HIGH.
03B7 208703            JSR PRINT      ;PRINT 3RD DIGIT.
03BA 208703            JSR PRINT      ;PRINT 4TH DIGIT.
03BD 60                RTS            ;RETURN.
03BE 2C00C0     KEY    BIT KBD        ;LOOP UNTIL A
03C1 10FB              BPL KEY        ;KEY IS PRESSED.
03C3 20DDFB            JSR BELL       ;RING BELL
03C6 AD00C0            LDA KBD        ;GET KEY
03C9 8D10C0            STA KBDSTB     ;CLEAR STROBE
03CC 290F              AND #$0F       ;DROP HIGH ORDER
03CE 60                RTS            ;HALF AND RETURN.
03CF 2058FC     GTPG   JSR HOME       ;CLEAR SCREEN.
03D2            ;
03D2 20BE03     GTPG1  JSR KEY        ;GET PAGE #.
03D5 0A                ASL            ;SHIFT LOW ORDER
03D6 0A                ASL            ;HALF TO THE
03D7 0A                ASL            ;HIGH ORDER
03D8 0A                ASL            ;HALF.
03D9 9DF603            STA PHOLD,X    ;STORE IN PAGE HOLD.
03DC 20BE03            JSR KEY        ;GET NEXT NUMBER.
03DF 5DF603            EOR PHOLD,X    ;COMBINE WITH
03E2 9DF603            STA PHOLD,X    ;PREVIOUS # AND STORE
```

```
03E5 60        139          RTS                      ;IN PAGE HOLD, RETURN.
03E6           140   ;
03E6 A9FF      141   HLOD   LDA #$FF                 ;PUT HIGH VALUES
03E8 8DF603    142          STA PHOLD                ;IN PAGE HOLD
03EB 8DF703    143          STA PHOLD+1              ;LOCATIONS THEN
03EE 60        144          RTS                      ;RETURN.
03EF           145   ;
03EF 000000    146          HEX 0000000000
03F2 0000
03F4 00        147   PGHI   HEX 00                   ;PAGE # HIGH
03F5 00        148   PGLO   HEX 00                   ;PAGE # LOW
03F6 0000      149   PHOLD  HEX 0000                 ;PAGE HOLD
03F8           150   ;
03F8 4C0003    151   CTRLY  JMP INIT
              152          END

***** END OF ASSEMBLY


           *************************
           *                       *
           * SYMBOL TABLE -- V 1.5 *
           *                       *
           *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

BASL    0028  BASH    0029  CSWL    0036  CSWH    0037

** ABSOLUTE VARABLES/LABELS

KBD     C000  KBDSTB  C010
BASIC2  E003  LIST    E04B  BELL    FBDD  HOME  FC58 COUT1 FDF0  SAVE   FF4A
RESTOR  FF3F  INIT    0300  BEG     0308  ZPNC  030B MAIN  0322  CLR    0347
LOOPR   034D  NPRES   0350  CMPB    036A  DISP  037E RR    0381  PRINT  0387
ADD1    0396  KEY     03BE  GTPG    03CF  GTPG1 03D2 HLOD  03E6  PGHI   03F4
PGLO    03F5  PHOLD   03F6  CTRLY   03F8

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:011A
```

Paged Printer Output for the Apple

by Gary Little

Improve the format of your printed output by adding a page mode to your system.

If you have ever sent output to a printer you have probably become very annoyed when the output continued from the very end of one page and then on to the next. Wouldn't it be nice if the printer would automatically scroll to the top of a new page when it got near the bottom of the previous one? PAGER does it for you; it will count the number of line feeds that are sent by Apple to the printer. When this total reaches 54, twelve blank lines are generated to automatically bring you to the top of the next eleven-inch page. PAGER can be used from within a program or from immediate-execution mode. It is extremely useful for LISTing long programs page by page.

How to Use the Program

PAGER was written for use with a serial printer that is connected to the Apple serial interface card. If PAGER is to be used in conjunction with a parallel printer connected to the Apple parallel interface card, two bytes of the routine must first be changed. To make these changes, load PAGER, and then enter the following two commands from BASIC:

```
POKE 785,2
POKE 812,2
```

The modified program should then be saved.

To change the number of lines that are printed before PAGER causes the paper to scroll to the top of the next page, enter the command POKE 798,LP from BASIC, where LP is the required number of lines per page.

To change page length, enter POKE 804,PL from BASIC, where PL is six times the length of the page (in inches). For example, for an eleven inch page, PL = 66. Note that PL must be greater than LP.

Output to the printer can be stopped after each page is printed by entering a POKE 822,1 command before activating PAGER. To proceed after a page has been printed, simply press any key on the keyboard. This 'page pause' feature must be used when you're feeding each piece of paper to the printer manually. To turn off the 'page pause', enter a POKE 822,0 command.

Instructions for Use Within a Program

Use the following sequence to turn the printer on and off from within a BASIC program:

```
 5  D$=CHR$(4)
10  PRINT D$;"PR#1"
20  LW=132 : REM LINE WIDTH
30  PRINT CHR$(9);LW;"N" : PRINT CHR$(9); "K"
40  CALL 768 : REM TURN ON PAGER
 .
 .  (Generate Output)
 .
50  PRINT D$;"PR#0" : REM TURN PRINTER OFF
```

If DOS is not being used, change line 10 to PR#1 and line 50 to PR#0 and delete line 5. If a serial printer is being used, delete lines 20 and 30.

Instructions for Use Outside a Program

If a serial printer is involved, PAGER can be activated by a CALL 768 from BASIC. It can be deactivated by a PR#0. If a parallel printer is involved, PAGER can be activated by performing the following four steps:

1. Enter PR#1
2. Enter CTRL-I 132N (132 or other line width).
3. Enter CTRL-I K
4. Enter CALL 768

It can be deactivated by a PR#0.

Additional Notes:

1. Remember to set the DIP switches on the serial printer interface card for the appropriate baud rate and line width before activating PAGER.

2. Remember to adjust the paper in the printer so that the first line printed will be at the desired starting position before activating PAGER.

3. Make sure that a PRINTed line will not exceed the line width which has been set for the printer. If it does, then the overflow will appear on the next line and this line will not be taken into account by PAGER.

```
0800          ;************************
0800          ;*                      *
0800          ;*     PAGED PRINTER    *
0800          ;*      GARY LITTLE     *
0800          ;*                      *
0800          ;*         PAGE         *
0800          ;*                      *
0800          ;*  COPYRIGHT (C) 1981  *
0800          ;*    MICRO INK, INC.   *
0800          ;* CHELMSFORD, MA 01824 *
0800          ;*  ALL RIGHTS RESERVED *
0800          ;*                      *
0800          ;************************
0800          ;
0800          ;
0800          ;
0800          ;
0800          ;POSITION PAPER IN PRINTER
0800          ;THEN CALL 768 FROM BASIC
0800          ;TO ACTIVATE THIS ROUTINE.
0800          ;TO DE-ACTIVATE, ENT PR#0.
0800          ;
0800          ;PAGE PAUSE FEATURE:
0800          ;  POKE 822,0 TURN OFF
0800          ;  POKE 822,1 TURN ON
0800          ;LINES PRINTED PER PAGE:
0800          ;  PCKE 798,LP
0800          ;
0800          ;PAGE LENGTH:
0800          ;  PCKE 804,PL
0800          ;
0800          ;DESCRIPTION:
0800          ;  THIS ROUTINE WILL SEND 'PL-LP'
0800          ; BLANK LINES TO THE PRINTER AFTER
0800          ; 'LP' LINES HAVE BEEN SENT BY THE
0800          ; USER.
0800          ;
0800          ;DEFAULTS:
0800          ;  LP=54
0800          ;  PL=66 (11" PAPER)
0800          ;  PAGE PAUSE OFF
0800          ;
0800          ;
0800          ;
0800          ;
0800          COUNT  EPZ $06            ;LINE COUNT STORAGE
0800          CSWL   EPZ $36            ;OUTPUT HOOK
0800          DOS    EQU $3EA           ;DOS I/O UPDATE HOOK
0800          KBD    EQU $C000          ;KEYBOARD
0800          STRB   EQU $C010          ;KEYBOARD STROBE
0800          PRINT  EQU $C100          ;PR#1 SERIAL OUTPUT
0800          ;
0800          ;
0300                 CRG $300
0300                 OBJ $800
0300          ;
0300          ;
0300 A90F            LDA #START         ;SET OUTPUT HOCK
0302 8536            STA CSWL           ;TC START OF RCUTINE.
0304 A903            LDA /START         ;
0306 8537            STA CSWL+1
0308 A900            LDA #$00           ;ZERO THE LINE COUNTER.
030A 8506            STA COUNT
030C 4CEA03          JMP DOS            ;GIVE NEW HOOK TO DOS.
030F 48       START  PHA                ;RCUTINE STARTS HERE.
0310 2000C1          JSR PRINT          ;SEND CHARACTER TO PRINTER.
0313 68              PLA
0314 C98D            CMP #$8D           ;CARRIAGE RETURN?
0316 F001            BEQ LINE           ;BRANCH IF IT IS.
0318 60       NEXT   RTS
0319 E606     LINE   INC COUNT          ;INCREMENT LINE COUNT.
031B A506            LDA COUNT
031D C936            CMP #$36           ;LINE COUNT =54?
031F D0F7            BNE NEXT           ;IF NOT, THEN RETURN.
```

```
0321 A506       BLANK  LDA COUNT
0323 C942              CMP #$42        ;PAGE LENGTH MET?
0325 F00A              BEQ LOOP
0327 E606              INC COUNT       ;INCREMENT THE COUNTER
0329 A98A              LDA #$8A        ;LOAD A LINE FEED
032B 2000C1            JSR PRINT       ;AND SEND IT TO THE PRINTER
032E 38                SEC
032F B0F0              BCS BLANK
0331 A900       LOOP   LDA #$00        ;ZERO THE COUNTER.
0333 8506              STA COUNT
0335 A900              LDA #$00        ;CHANGE TO LDA #$01 TO
0337 F008              BEQ DONE        ;GET 'PAGE PAUSE'.
0339 2C00C0     AGAIN  BIT KBD         ;WAIT FOR KEYPRESS
033C 10FB              BPL AGAIN       ;BEFORE CONTINUING.
033E 2C10C0            BIT STRB        ;CLEAR KEYBOARD STROBE.
0341 60         DONE   RTS
                       END

***** END OF ASSEMBLY


          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5 *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

COUNT   0006   CSWL    0036

** ABSOLUTE VARABLES/LABELS

DOS     03EA   KBD     C000   STRB    C010   PRINT   C100
START   030F   NEXT    0318   LINE    0319   BLANK   0321   LOOP    0331   AGAIN   0339
DONE    0341

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:007A
```

Hexadecimal Printer

by LeRoy Moyer

This simple program permits you to specify the limits within which you want the Apple II disassembler to operate.

When using the disassembler in the Apple II to print out machine language code, you normally type in the starting address and then a number of L's. There are two problems with using this method to print out a machine language program. The first is that if the machine language program does not happen to be a multiple of 20 instructions, there is probably going to be a collection of unwanted garbage printed at the bottom of the desired machine code. The second problem occurs when the program being printed is fairly long. Do you type in 50 to 51 L's to get all of the desired code? The program presented here solves both of these problems by decoding and outputing the disassembled machine language code that lies between two hexadecimal addresses.

After loading the program, using it is very easy. First, turn on the printer with a control P and then type 800G (return). The screen will clear and prompt you with the header ''STARTING ADDRESS''. Enter the hexadecimal address of the first instruction to be decoded and then hit return. A similar prompting question will be asked for the ending address and after entering the ending address the program will start outputting the disassembled code beginning at the starting address and continuing until the ending address.

The code presented here is transportable in that only two addresses (4 bytes) need to be changed to relocate the program anywhere in memory. These locations are the addresses for the data that prints out the program's two lines of text. Data for this text is stored starting at lines $86B and $87D in the program listing and this data is used in the lines at $806 and $828 respectively.

Several Apple monitor subroutines are used in this program and two of them deserve some comment. The first is the GETNUM ($FFA7) subroutine that converts a number stored as ASCII characters in the input buffer ($200), indexed by the Y register, into a two byte hexadecimal number. This routine converts ASCII characters until it encounters a character that is a non-hexadecimal number. A carriage return ($8D) is used in this program for the terminator. The resulting hexadecimal address is stored at location A2L ($3E) and A2H ($3F) in the usual low byte, high byte order for addresses required by the 6502.

The second routine that deserves some comment is the INSTDSP ($F8D0) routine. This routine disassembles an instruction and outputs it to the screen. The address that is used to direct the subroutine to the op-code to be disassembled is stored in PCL ($3A) and PCH ($3B). After returning from INSTDSP, a number that is one less than the length of the instruction is stored in location LENGTH ($2F). The address in the pointer ($3A, $3B) is not changed by INSTDSP and hence the length of the instruction needs to be added to the pointer to get to the location of the next op-code (lines 58 to 64 in the program listing).

If you don't want the initial lines of text printed out on your printer then insert a printer turn-on routine between lines 55 and 56 of the assembled program listing. Hopefully this routine will be useful in making your machine language print-outs look neater in the future.

```
0800             1  ;*************************
0800             2  ;*                       *
0800             3  ;* HEXIDECIMAL PRINTER   *
0800             4  ;*     LEROY MOYER       *
0800             5  ;*                       *
0800             6  ;*     HEX PRINTER       *
0800             7  ;*                       *
0800             8  ;*  COPYRIGHT (C) 1981   *
0800             9  ;*   MICRO INK, INC.     *
0800            10  ;* CHELMSFORD, MA 01824  *
0800            11  ;*  ALL RIGHTS RESERVED  *
0800            12  ;*                       *
0800            13  ;*************************
0800            14  ;
0800            15  ;DECODE BETWEEN ADR
0800            16  ;
0800            17  FINA    EPZ $FE
0800            18  APA2    EPZ $3E
0800            19  LENG    EPZ $2F
0800            20  APPC    EPZ $3A
0800            21  ;
0800            22          ORG $800
0800            23          OBJ $800
0800            24  ;
0800 2058FC     25  STAR    JSR $FC58     ;CLEAR SCREEN
0803 A200       26          LDX #$00      ;OUTPUT FIRST HEADER LINE
0805 BD6B08     27  DBA2    LDA TIT1,X    ;"STARTING ADDRESS"
0808 F008       28          BEQ DBA1
080A 0980       29          ORA #$80
080C 20EDFD     30          JSR $FDED
080F E8         31          INX
0810 D0F3       32          BNE DBA2
0812 206FFD     33  DBA1    JSR $FD6F     ;KEYBOARD INPUT OF STARTING ADDRESS
0815 A000       34          LDY #$00
0817 20A7FF     35          JSR $FFA7     ;CHANGE TO HEXIDECIMAL ADDRESS
081A A53E       36          LDA APA2      ;MOVE HEXADECIMAL ADDRESS TO
081C 853A       37          STA APPC      ; APPC ($3A)
081E A53F       38          LDA APA2+01
0820 853B       39          STA APPC+01
0822 208EFD     40          JSR $FD8E     ;PRINT LINE FEED
0825 A200       41          LDX #$00      ;PRINT SECOND HEADER LINE
0827 BD7D08     42  DBA4    LDA TIT2,X    ;   "ENDING ADDRESS"
082A F008       43          BEQ DBA3
082C 0980       44          ORA #$80
082E 20EDFD     45          JSR $FDED
0831 E8         46          INX
0832 D0F3       47          BNE DBA4
0834 206FFD     48  DBA3    JSR $FD6F     ;KEYBOARD INPUT OF ENDING ADDRESS
```

```
0837 A000        LDY #$00
0839 20A7FF      JSR $FFA7      ;CHANGE TO HEXADECIMAL ADDRESS
083C A53E        LDA APA2       ;MOVE HEXADECIMAL ADDRESS TO
083E 85FE        STA FINA       ; FINA ($FE) FINAL ADDRESS
0840 A53F        LDA APA2+01
0842 85FF        STA FINA+01
0844 208EFD      JSR $FD8E      ;PRINT LINE FEED
0847 20D0F8 DBA5 JSR $F8D0      ;DISASSEMBLE ONE LINE
084A E62F        INC LENG       ;INCREMENT BYTE FOR LENGTH
084C 18          CLC
084D A53A        LDA APPC       ;ADDLENGTH OF INSTRUCTION TO
084F 652F        ADC LENG       ;ADDRESS THAT IS POINTER FOR
0851 853A        STA APPC       ;OP CODE TO BE DISASSEMBLED
0853 A53B        LDA APPC+01
0855 6900        ADC #$00
0857 853B        STA APPC+01
0859 38          SEC
085A A53A        LDA APPC       ;SUBTRACT FINAL ADDRESS TO SEE IF
085C E5FE        SBC FINA       ; THE END HAS BEEN REACHED
085E A53B        LDA APPC+01
0860 E5FF        SBC FINA+01
0862 90E3        BCC DBA5
0864 208EFD      JSR $FD8E      ;PRINT LINE FEED
0867 208EFD      JSR $FD8E      ;PRINT LINE FEED
086A 60          RTS            ;RETURN TO MONITOR
086B D3D4C1 TIT1 ASC "STARTI"   ;DATA FOR FIRST HEADER LINE
086E D2D4C9
0871 CEC7A0      ASC "NG ADD"
0874 C1C4C4
0877 D2C5D3      ASC "RESS "
087A D3A0
087C 00          HEX 00
087D 0D     TIT2 HEX 0D         ;DATA FOR SECOND HEADER LINE
087E C5CEC4      ASC "ENDING"
0881 C9CEC7
0884 A0C1C4      ASC " ADDRE"
0887 C4D2C5
088A D3D3A0      ASC "SS "
088D 00          HEX 00
                 END

***** END OF ASSEMBLY


          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5  *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

FINA   00FE  APA2   003E  LENG   002F APPC 003A

** ABSOLUTE VARABLES/LABELS

STAR   0800  DBA2   0805
DBA1   0812  DBA4   0827  DBA3   0834 DBA5 0847  TIT1 086B TIT2 087D


SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0072
```

3
RUNTIME UTILITIES

INTRODUCTION

Runtime utilities are defined as the family of programs which assist in the execution of other programs. Such a utility usually is linked to the host program at execution time, and runs concurrently with it as a subroutine. In this chapter, several runtime utilities for Applesoft and Integer BASIC programs are described which will enhance the programming power of your Apple.

Robert Zant's "Common Variables on the Apple II" discusses how to set up a common variable space shared between BASIC programs. Both Integer and Applesoft versions are presented. Gary Morris' "PRINT USING for Applesoft" article presents an implementation of the popular PRINT USING statement for Applesoft. "Searching String Arrays," by Gary Little, presents a machine language array searching routine which is an order of magnitude faster than the BASIC equivalent would be.

The next two utilities make use of the ampersand feature in Applesoft and are both powerful enhancements to the Applesoft language. "Applesoft and Matrices," by Cornelis Bongers, provides for full matrix operations using BASIC arrays. "AMPER-SORT," by Alan Hill, implements automatic sorting of arrays, whether numeric or string.

Finally, "Apple II Trace List Utility," by Alan Hill, presents a means of interactively tracing an Integer BASIC program while storing the trace information.

Common Variables on the Apple II

by Robert F. Zant

Modular software designs rely on common variables to pass data between interrelated programs. Two short subroutines emulate the DOS CHAIN capability by allowing use of common variables under Integer or Applesoft BASIC, without a disk.

The solution of complex problems often leads to the writing of several interrelated programs. Furthermore, the programs usually use several of the same variables — called common variables. This is accomplished in most systems by not destroying the common variables when a new program is loaded. Thus, the value of a variable can be defined in one program and used in subsequent programs.

There is no true facility with the Apple II for using common variables. The CHAIN command in DOS comes close to providing the capability, but it saves all variables instead of just saving designated common variables. Also, it can only be used with Integer BASIC programs run under DOS. No facility for common variables is provided for non-disk systems or for Applesoft programs.

Creating a Common Variable Space

The following machine language routines can be used to pass all variables to succeeding programs. Integer BASIC and Applesoft versions are provided. Both versions are used as follows:

1. Load the machine language routine before the first BASIC program is executed.

2. In each BASIC program except the last program, "CALL 774" immediately before termination or before the DOS command to RUN the next program.

3. In each BASIC program except the first program, "CALL 770" before executing any statement that affects or uses variables. Do not reDIMension variables in subsequent programs.

Since all variables are saved whether they are needed or not, main storage is used most efficiently if the same set of variable names is used in all programs. This, of course, is required for the variables that are intended to be common for all programs. Other main storage is reclaimed by the reuse of the names of ''non-common'' variables.

String variables will not always be saved correctly in Applesoft. If the string value was read from disk, tape or keyboard, the value will be saved. If the string value is defined in an assignment statement (e.g. A$ = ''XXX''), the value will not be available to subsequent programs.

The Programs

The routine for Integer BASIC is very simple. The variable table pointer is simply saved and restored. The Applesoft version, however, is a little more complex. The Applesoft version of the routine moves all non-string variables to high RAM, just under the strings. Then, when called at the beginning of the next program via ''CALL 770'', the routine moves the variables back down to the end of the new program.

```
0800            ;*************************
0800            ;*                       *
0800            ;*   COMMON VARIABLES    *
0800            ;*     ROBERT ZANT       *
0800            ;*                       *
0800            ;*      COM-VAR-I        *
0800            ;*                       *
0800            ;*  COPYRIGHT (C) 1981   *
0800            ;*   MICRO INK, INC.     *
0800            ;* CHELMSFORD, MA 01824  *
0800            ;*  ALL RIGHTS RESERVED  *
0800            ;*                       *
0800            ;*************************
0800            ;
0800            ;
0800            ;FOR INTEGER BASIC
0800            ;
0800            CL     EPZ $1A
0800            CH     EPZ $AB
0800            ;
0800            ;
0302                   ORG $302
0302                   OBJ $800
0302            ;
0302            ;
0302 4C0F03            JMP RECALL          ;ENTRY 770
0305 00                BRK
0306 A5CC              LDA $CC             ;ENTRY 774 - SAVE VARIABLES
0308 851A              STA CL              ;SAVE END OF
030A A5CD              LDA $CD             ;VARIABLE TABLE
030C 85AB              STA CH
030E 60                RTS                 ;BACK TO BASIC
030F A51A       RECALL LDA CL              ;ENTRY 770 - RECALL VARIABLES
0311 85CC              STA $CC             ;RESET END OF
0313 A5AB              LDA CH              ;VARIABLE TABLE
0315 85CD              STA $CD
0317 60                RTS                 ;BACK TO BASIC
                       END

          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

CL     001A  CH     00AB

** ABSOLUTE VARABLES/LABELS

RECALL 030F

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:002A

0800            ;*************************
0800            ;*                       *
0800            ;*   COMMON VARIABLES    *
0800            ;*     ROBERT ZANT       *
0800            ;*                       *
0800            ;*      COM-VAR-A        *
0800            ;*                       *
0800            ;*  COPYRIGHT (C) 1981   *
0800            ;*   MICRO INK, INC.     *
0800            ;* CHELMSFORD, MA 01824  *
0800            ;*  ALL RIGHTS RESERVED  *
0800            ;*                       *
0800            ;*************************
0800            ;
0800            ;
```

```
0800                  ;
0800                  ;FOR APPLESOFT II BASIC
0800                  ;
0800                  ;
0800                  DL      EPZ $18
0800                  DH      EPZ $19
0800                  CL      EPZ $1A
0800                  CH      EPZ $1B
0800                  EL      EPZ $1C
0800                  EH      EPZ $1D
0800                  A1L     EPZ $3C
0800                  A1H     EPZ $3D
0800                  A2L     EPZ $3E
0800                  A2H     EPZ $3F
0800                  A4L     EPZ $42
0800                  A4H     EPZ $43
0800                  ;
0800                  ;
0302                          ORG $302
0302                          OBJ $800
0302                  ;
0302                  ;
0302 4C5603                   JMP RECALL    ;ENTRY 770
0305 00                       BRK
0306 38                       SEC           ;ENTRY 774 - SAVE NUMERICS
0307 A56F                     LDA $6F       ;COMPUTE ADDRESSES FOR MOVE
0309 8518                     STA DL        ; SAVE START OF STRING ADDRESS
030B E56D                     SBC $6D       ;END OF NUMERICS
030D 851A                     STA CL        ;TEMPORARY STORAGE
030F A570                     LDA $70
0311 8519                     STA DH
0313 E56E                     SBC $6E
0315 851B                     STA CH        ;TEMPORARY STORAGE
0317 18                       CLC
0318 A51A                     LDA CL
031A 6569                     ADC $69       ;START OF NUMERICS
031C 851A                     STA CL        ;TEMP STORAGE
031E A51B                     LDA CH
0320 656A                     ADC $6A
0322 851B                     STA CH
0324 A61A                     LDX CL        ;SUBTRACT ONE
0326 D002                     BNE A1
0328 C61B                     DEC CH        ;START OF COMMON
032A CA               A1      DEX
032B 861A                     STX CL
032D 8642                     STX A4L       ;SET UP MOVE
032F A51B                     LDA CH
0331 8543                     STA A4H
0333 A569                     LDA $69       ;START OF VARIABLES
0335 853C                     STA A1L
0337 A56A                     LDA $6A
0339 853D                     STA A1H
033B A56D                     LDA $6D       ;END OF VARIABLES
033D 853E                     STA A2L
033F A56E                     LDA $6E
0341 853F                     STA A2H
0343 A000                     LDY #$00
0345 202CFE                   JSR $FE2C             ;USE MONITOR MOVE ROUTINE
0348 38                       SEC                   ;COMPUTE DISPLACEMENT
0349 A56B                     LDA $6B               ; TO ARRAYS
034B E569                     SBC $69
034D 851C                     STA EL
034F A56C                     LDA $6C
0351 E56A                     SBC $6A
0353 851D                     STA EH
0355 60                       RTS                   ;BACK TO BASIC
0356 A51A             RECALL  LDA CL                ;ENTRY 770 - RECALL
0358 853C                     STA A1L               ;SET UP MOVE
035A A51B                     LDA CH
```

```
035C 853D            STA A1H
035E A518            LDA DL
0360 856F            STA $6F                 ;START OF STRINGS
0362 853E            STA A2L
0364 A519            LDA DH
0366 8570            STA $70
0368 853F            STA A2H
036A A569            LDA $69                 ;STYART OF NUMERICS
036C 8542            STA A4L
036E A56A            LDA $6A
0370 8543            STA A4H
0372 A000            LDY #$00
0374 202CFE          JSR $FE2C               ;USE MONITOR MOVE ROUTINE
0377 18              CLC                     ;COMPUTE START
0378 A569            LDA $69                 ;OF ARRAYS
037A 651C            ADC EL
037C 856B            STA $6B
037E A56A            LDA $6A
0380 651D            ADC EH
0382 856C            STA $6C
0384 38              SEC                     ;COMPUTE END OF NUMERICS
0385 A56F            LDA $6F
0387 E51A            SBC CL
0389 856D            STA $6D
038B A570            LDA $70
038D E51B            SBC CH
038F 856E            STA $6E                 ;TEMP STORAGE
0391 18              CLC
0392 A56D            LDA $6D
0394 6569            ADC $69
0396 856D            STA $6D                 ;TEMP VALUE
0398 A56E            LDA $6E
039A 656A            ADC $6A
039C 856E            STA $6E                 ;TEMP VALUE
039E A56D            LDA $6D                 ;SUBTRACT ONE
03A0 D002            BNE A2
03A2 C66E            DEC $6E                 ;END OF NUMERICS
03A4 C66D       A2   DEC $6D
03A6 60              RTS                     ;BACK TO BASIC
                    END
```

```
          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

DL    0018  DH    0019  CL    001A  CH  001B  EL    001C  EH  001D
A1L   003C  A1H   003D  A2L   003E  A2H 003F  A4L   0042  A4H 0043

** ABSOLUTE VARABLES/LABELS

A1    032A  RECALL 0356 A2    03A4

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:008A
```

PRINT USING for Applesoft

by Gary A. Morris

One of the minor but annoying problems with BASIC is the format of output. The program here permits user-defined formatting of the output for Applesoft, and can be easily modified for other flavors of BASIC.

When I started using my Apple for business programming, my biggest headache was formatting output for reports. I started out using various BASIC subroutines that barely performed the needed job and required a lot of overhead. Tired of using MID$, LEFT$, RIGHT$, and STR$, I decided to write a general-purpose print formatter using the USR function in Applesoft.

The routine is written entirely in assembly language, which is ideal for handling this sort of problem. It is called from BASIC by assigning the string variable ED$, the edit pattern showing how you want the output formatted. During a print statement when you use the USR function, the argument is evaluated and then printed in the format specified by the current value of ED$.

In the sample BASIC program (in figure 1) line 10 loads the machine language program into RAM at $300-$3A9. Then line 20 puts a "JMP $0300" at $000A, which is used by Applesoft to find the routine to be used. Lines 10 and 20 are only needed once at the beginning of a program. Line 30 assigns an edit pattern to the variable ED$. Line 40 is a sample print statement that uses the USR function. Line 50 assigns a value to *X* (that we want printed) rounded off to two decimal places, and line 60 does this. If you wanted to round to three places, the 100 would be changed to 1000 and the edit pattern would have to be changed to allow three digits after the decimal point. Note that any valid expression could be within the parentheses of the USR function.

The routine works by taking the number that Applesoft would normally print out and filling up the edit pattern with those characters from right to left, skipping over decimal points, commas and special characters.

The output of the routine may be used wherever a BASIC PRINT statement can be used, such as printing to a disk file, to a printer, or just to the screen. It is especially desirable for creating fixed-length records in files.

The edit pattern can be fairly complex, as in figure 1, or it can be simply blanks. Using a blank pattern will cause the number to be right-justified within the number of blanks in the edit pattern. If the number is too large to fit in the edit pattern, the left-most digits will be truncated. Any special characters ($,.'' ÷ %:*) in the edit pattern will be skipped, and the digits will fill in over blanks or numeric digits in the pattern.

The zeros are used in the edit pattern so that, if the number is small, there will always be zeros between the decimal point and the right-most column. If the number is too small to fill past the comma(s), then the extra commas will be replaced with blanks. When using an edit pattern with a decimal point, the argument for the function must be a whole number, or two decimal points will result. The edit pattern must be less than or equal to 16 characters in length. If it is greater, it will be cut off at 16.

The machine language program was written so that it can be located anywhere in addressable memory space. It is completely relocatable. That is, no changes are needed to run it at another address. It requires 169 ($A9) bytes of RAM. The program uses the same zero page locations that are assigned to Applesoft so that there are no conflicts. It also uses 752-767 ($2F0-$2FF) as a buffer to perform editing. This area is in the input buffer and is not used during printing (except when printing DOS commands).

How It Works

Starting with the PRINT statement, the argument for the USR is evaluated and placed in the floating point accumulator by the BASIC interpreter. Then a JSR is made to $000A, where we have a JMP to the start of our subroutine.

At the beginning of the machine language subroutine, the Applesoft floating point accumulator is converted (lines $300-$30B) into a character string, in the format that Applesoft would normally print it out. This is done by the Applesoft subroutines FPSTR1 and FPSTR2 (my names). These routines leave the resulting string at the bottom of the page used for the stack ($100).

The routine then searches ($30C-$32C) the variable table to find ED$. When found, its value is moved ($32D-$336) to the buffer area ($2F0-$2FF).

After the program has all the necessary data, it starts to work. The length of the unformatted number is found ($337-$340); and this number (an ASCII string right now) is then moved (lines $341-$34D) into the buffer, one character at a time, from right to left. The current character in the pattern is checked and, if it is a special character, it is skipped. Minus signs are carried over any digits in the pattern so that they will be on the left of the number. This process continues until we run out of characters to put in the pattern (or the pattern fills up), at which time any leftover commas are covered up (lines $37A-$390) with blanks.

Finally the program is ready to print out the result. The lines at $391-$39D print out all of the number, except the last digit (I'll explain this in a moment), using the output routine in Applesoft. This output routine does all of the necessary checking and conversion so that Applesoft's SPEED, INVERSE, and FLASH functions will work. The routine also sets the most significant bit of all outgoing ASCII characters.

The USR function must return a value to the BASIC program, which will be printed out by the BASIC interpreter, because we are in a PRINT statement. The last character of the buffer (which must be a digit) is taken and converted to an integer in the *Y* register and passed to Applesoft's integer to floating conversion routine ($39E-$3A8). This routine converts the integer (passed in the *A,Y* registers) into floating point in the floating point accumulator, which is just where we need it to pass back to BASIC.

Hardware Requirements

This program requires an Apple II Plus, an Apple II with an Applesoft card, or an Apple II with a language card. It will work in any memory size system. A disk drive is not required.

If the appropriate changes are made to the JSRs and JMP in the machine language routine, the program can be used with RAM Applesoft (which loads in at $0800-2FFF). After keying in the code, make the following modifications to the equate table and it will run with RAM Applesoft instead:

```
FPSTR1 = $252B
FPSTR2 = $1BDE
COUT   = $135F
INTFP  = $1AEB
FIND   = $184C
```

```
10  REM  PRINT USING DEMO
15  REM
20  POKE 10,76: POKE 11,0: POKE 12,3
30 ED$ = "$   ,   0.00"
40  PRINT "SUB TOTAL..."; USR (3495)
50 X = 12345.67899
60  PRINT "NET TOTAL..."; USR (INT(X*100 +.5))
70  END
```

Figure 1

```
0800          ;*************************
0800          ;*                       *
0800          ;*      PRINT USING      *
0800          ;*      GREG MORRIS      *
0800          ;*                       *
0800          ;*  COPYRIGHT (C) 1981   *
0800          ;*    MICRO INK, INC.    *
0800          ;* CHELMSFORD, MA 01824  *
0800          ;*  ALL RIGHTS RESERVED  *
0800          ;*                       *
0800          ;*************************
0800          ;
0800          ;
0800          ;
0800          ;THE USR FUNCTION REQUIRES A JMP TO
0800          ;THE START OF THE ROUTINE. IF 'START'
0800          ;EQUALS THE ADDRESS WHERE THE ROUTINE
0800          ;IS LOADED THEN THE FOLLOWING WILL SET
0800          ;UP THE JMP:
0800          ;
0800          ; 10 POKE 10,76
0800          ; 20 POKE 11,START-INT(START/256)*256
0800          ; 30 POKE 12,INT(START/256)
0800          ;
0800          ;VARIABLES:
0800          AFLAG  EPZ $52              ;FLAG FOR APPLESOFT
0800          NAME   EPZ $81              ;VARIABLE NAME
0800          PNTR   EPZ $83              ;PNTR TO EDIT PATTERN
0800          VARBLE EPZ $9B              ;POINTER TO VARIABLE
0800          LENGTH EPZ $D0              ;PATTERN LENGTH
0800          ;
0800          ;
0800          ;
0300                 ORG $300             ;ORG AT $0300 (RELOCATABLE)
0300                 OBJ $800
0300          ;
0300          ;
0300          ;
0300          BUFFER EQU $02F0            ;EDIT BUFFER
0300          STRING EQU $0100            ;NUMBER PUT HERE AS
0300          ;  A CHARACTER STRING
0300          ;
0300          ;ROM APPLESOFT SUBROUTINE ADDRESSES:
0300          FPSTR1 EQU $ED34            ;FLOATING TO STRING
0300          FPSTR2 EQU $E3E7            ;CONVERSION ROUTINES
0300          COUT   EQU $DB5C            ;PRINT AN ASCII CHAR
0300          INTFP  EQU $E2F2            ;INT TO FP CONVERSION
0300          FIND   EQU $E053            ;FIND A VARIABLE
0300          ;
0300          ;RAM APPLESOFT SUBROUTINE ADDRESSES:
0300          ;FPSTR1 EQU $252B ;FLOATING TO STRING
0300          ;FPSTR2 EQU $1BDE ;CONVERSION ROUTINES
0300          ;COUT EQU $135F ;PRINT AN ASCII CHAR
0300          ;INTFP EQU $1AEB ;INT TO FP CONVERSION
0300          ;FIND EQU $184C ;FIND A VARIABLE
0300          ;
0300          ;FIRST CONVERT FLOATING POINT ACCUM
0300          ;TO AN ASCII STRING
0300 A552     START  LDA AFLAG            ;SAVE THE FLAG
0302 48              PHA
0303 2034ED          JSR FPSTR1           ;CONVERT FLOATING
0306 20E7E3          JSR FPSTR2           ;POINT TO STRING
0309 68              PLA
030A 8552            STA AFLAG            ;RESTORE FLAG
030C          ;
030C          ;NOW FIND THE VARIABLE (ED$) THAT
030C          ;HAS THE EDIT PATTERN.
030C A945     SEARCH LDA #'E'             ;BASIC VARIABLE
030E A2C4            LDX #$C4             ;NAME IS ED$
0310 8581            STA NAME
0312 8682            STX NAME+1
0314 2053E0          JSR FIND
0317 A004            LDY #4
0319 B19B            LDA (VARBLE),Y       ;GET ADDR HI
```

```
031B 8584           STA PNTR+1
031D 88             DEY
031E B19B           LDA (VARBLE),Y      ;GET ADDR LO
0320 8583           STA PNTR
0322 88             DEY
0323 B19B           LDA (VARBLE),Y      ;GET LENGTH
0325 C910           CMP #16
0327 9002           BCC LENOK           ;MAXIMUM LENGTH
0329 A910           LDA #16             ;ALLOWED IS 16!!!
032B 85D0    LENOK  STA LENGTH
032D         ;MOVE THE PATTERN TO THE BUFFER
032D A8             TAY
032E 88             DEY
032F B183    LOOP2  LDA (PNTR),Y
0331 99F002         STA BUFFER,Y
0334 88             DEY
0335 10F8           BPL LOOP2
0337         ;FIND THE STRING END
0337 A000           LDY #0
0339 B90001  LOOP   LDA STRING,Y        ;GET CHAR
033C F003           BEQ NEXT2
033E C8             INY
033F D0F8           BNE LOOP
0341         ;
0341         ;MOVE STRING TO THE BUFFER, FROM
0341         ;RIGHT TO LEFT, FILLING OVER NUM-
0341         ;BERS BUT SKIPPING COMMA'S AND
0341         ;PERIODS.  IF WE COME TO A MINUS
0341         ;SIGN, THEN KEEP GOING LEFT UNTIL
0341         ;THE PATTERN HAS A BLANK OR A COM-
0341         ;MA, THEN KEEP GOING LEFT STORING
0341         ;BLANKS IN THE BUFFER UNTIL IT ENDS
0341         ;OR WE COME TO A DOLLAR SIGN.
0341         ;
0341 A6D0    NEXT2  LDX LENGTH          ;FIELD WIDTH
0343 88      EDLOOP DEY
0344 B90001         LDA STRING,Y        ;GET A CHARACTER
0347 48             PHA                 ;SAVE IT
0348 68      CHECK  PLA
0349 48             PHA
034A C92D           CMP #'-'            ;IF A MINUS THEN
034C D00E           BNE DIGIT
034E BDEF02  MINUS  LDA BUFFER-1,X
0351 C92D           CMP #'-'
0353 9016           BCC DROPIT
0355 CA      SKIPIT DEX
0356 D0F0           BNE CHECK
0358 68             PLA
0359 18             CLC
035A 9035           BCC DONE
035C BDEF02  DIGIT  LDA BUFFER-1,X
035F C920           CMP #' '
0361 F008           BEQ DROPIT
0363 C93A           CMP #':'
0365 F0EE           BEQ SKIPIT
0367 C930           CMP #'0'
0369 90EA           BCC SKIPIT
036B 68      DROPIT PLA                 ;GET IT BACK
036C 9DEF02         STA BUFFER-1,X
036F CA             DEX
0370 F01F           BEQ DONE
0372 C000           CPY #0              ;END OF STRING?
0374 D0CD           BNE EDLOOP
0376 E8             INX
0377 18             CLC
0378 9010           BCC NEXT1
037A BDEF02  BLANK  LDA BUFFER-1,X      ;BLANK FROM
037D C924           CMP #'$'            ;HERE TO $
037F F010           BEQ DONE
0381 C92E           CMP #'.'
0383 B005           BCS NEXT1
0385 A920           LDA #' '
0387 9DEF02         STA BUFFER-1,X
038A CA      NEXT1  DEX
038B F004           BEQ DONE
```

```
038D E4D0            CPX LENGTH
038F 90E9            BCC BLANK
0391 A201     DONE   LDX #1
0393 BDEF02   LOOP4  LDA BUFFER-1,X      ;PRINT THE
0396 205CDB          JSR COUT            ;OUTPUT BUFFER
0399 E8              INX                 ;EXCEPT LAST CHAR
039A E4D0            CPX LENGTH
039C 90F5            BCC LOOP4
039E          ;TAKE THE LAST CHAR FROM THE BUF-
039E          ;FER, CONVERT IT TO FLOATING AND
039E          ;RETURN IT TO APPLESOFT TO BE PRINTED.
039E BDEF02          LDA BUFFER-1,X
03A1 4930            EOR #'0'
03A3 A8              TAY                 ;LO ORDER BYTE
03A4 A900            LDA #0              ;HI CRDER BYTE
03A6 4CF2E2          JMP INTFP           ;CONVERT & RETURN
                    END

***** END OF ASSEMBLY
```

```
          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

AFLAG  0052  NAME   0081  PNTR   0083  VARBLE 009B  LENGTH 00D0

** ABSOLUTE VARABLES/LABELS

BUFFER 02F0
STRING 0100  FPSTR1 ED34  FPSTR2 E3E7  COUT   DB5C  INTFP  E2F2  FIND   E053
START  0300  SEARCH 030C  LENOK  032B  LOOP2  032F  LOOP   0339  NEXT2  0341
EDLOOP 0343  CHECK  0348  MINUS  034E  SKIPIT 0355  DIGIT  035C  DRCPIT 036B
BLANK  037A  NEXT1  038A  DONE   0391  LOOP4  0393

SYMBCL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:00F2
```

Searching String Arrays

by Gary B. Little

This machine language program makes searching a large string array considerably faster and easier.

Have you ever wanted to search through a string array to see if it contains a particular phrase? If you have, it's probable that you have written a rather short loop routine in Applesoft to do this. However, if you have a few thousand comparisons to make, the Applesoft version may take an undesirable length of time to grind out the desired results.

A much faster search can be carried out on the Apple II by using a search routine written in 6502 assembly language. Such a program is shown here.

The SEARCH Routine

To understand exactly how the program works it is necessary to analyze the method by which the Apple stores variables in its memory. The details are found on page 137 of the *Applesoft II BASIC Programming Reference Manual*. For a one-dimensional string array, the storage pattern is as follows:

```
NAME (2 bytes)
OFFSET pointer to next variable (2 bytes)
No. of dimensions (1 byte)
Size 1st dimension (2 bytes)
String$(0)—length (1 byte)
          —address low (1 byte)
          —address high (1 byte)
.
.
String$(N) (3 bytes)
```

N is the size of the 1st dimension. If the string array is the first array variable defined in a program, the memory location of the first byte of the trio of bytes, reserved for the Cth array variable, is given by PEEK(107) + 256*PEEK(108) + 7 + 3*C (where 0<=C<=N). This is because the pointer to the beginning of the array space, and also to the beginning of the string array variable map, is found at $6B,$6C (107,108) and there are 7 + 3*C bytes before the three Cth array variable bytes.

If the phrase to be searched for (the search variable) is the first simple variable defined in a program, the memory location of the first byte of the three bytes reserved for the length and location of the string is given by PEEK(105) + 256*PEEK(106) + 2. This is because the pointer to the beginning of the simple variable space, and also to the beginning of the simple variable map, is found at $69,$6A (105,106). There are two bytes before the three variable bytes.

To carry out the search, it is simply necessary to compare the string pointed to by SV + 3,SV + 4 (where SV = PEEK(105) + 256*PEEK(106)) with the string pointed to by AV + 8 + 3*C,AV + 9 + 3*C (where AV = PEEK(107) + 256*PEEK(108) and C runs from 0 to N). This is precisely what is done in this assembly language routine.

The time savings that can be realized by using the routine can be seen by running the Applesoft demo program LISTed. For example, an assembly language search of 2,000 string array variables takes only one second, whereas the same search done in Applesoft takes 19 seconds!

Using the Search Routine

To use the search routine from within an Applesoft program, the following procedure must be followed:

1. POKE the length of, and the two pointers to, the search phrase into locations 0,6,7, respectively. This is done in line #210 of the demo program.
2. POKE the number of the array variable from which the search is to proceed ('C') in locations 30,31 (low,high). This is done in line #220.
3. POKE the number of the array variable, at which the search is to end, ('N') in locations 28,29 (low,high). This is done in line #230.
4. POKE the location of the trio of bytes for the Cth array variable in locations 8,9 (low,high). This is done in line #240.
5. CALL 768 to start the assembly language search routine. When control returns to Applesoft the array number that has satisfied the search will be returned in locations 30,31. If PEEK(30) + 256*PEEK(31) is greater than N, then the search has failed. If not, then a match has been made with R$(C) where C = PEEK(30) + 256*PEEK(31) and R$ is the array that is being searched.
6. To continue the search to the end of the array, increment C and repeat the above process.

The routine, as written, does not search for exact matches with the string array variables. If the leftmost part of a string array variable is the same as the search phrase, a match is considered to have occurred.

A useful application of this search routine is to use it in conjunction with a mailing list database program. In this way, the search time for an individual record can be cut down dramatically.

```
1  REM  *************************
2  REM  *                       *
3  REM  * STRING SEARCH ROUTINE*
4  REM  *      GARY LITTLE      *
5  REM  *                       *
6  REM  *  COPYRIGHT (C) 1981   *
7  REM  *    MICRO INK, INC.    *
8  REM  * CHELMSFORD, MA 01824 *
9  REM  *  ALL RIGHTS RESERVED *
10  REM *                       *
11  REM *************************
12  REM
14  REM
100 S$ = "": REM  MUST BE FIRST DEFINED SIMPLE VARIABLE
110 N = 2000: DIM R$(N): REM  MUST BE FIRST DEFINED ARRAY VARIABLE
120  GOSUB 1000: REM  LOAD SEARCH ROUTINE
130  DEF  FN MD(X) = X - 256 *  INT (X / 256)
140  TEXT : HOME : PRINT  TAB( 8);: INVERSE : PRINT "STRING ARRAY
      SEARCH  DEMO": NORMAL
150  PRINT : PRINT "RANDOM STRINGS:": PRINT
160  FOR I = 1 TO N:R$(I) =  CHR$ (65 + 26 *  RND (1)) +  CHR$
     (65 + 26 *  RND (1)): PRINT R$(I);" ";: NEXT I: PRINT : PRINT
170  INPUT "ENTER SEARCH STRING: ";S$: PRINT
180 SV = AV:C = 1
190 SV =  PEEK (105) + 256 *  PEEK (106)
200 AV =  PEEK (107) + 256 *  PEEK (108)
210  POKE 0, PEEK (SV + 2): POKE 6, PEEK (SV + 3): POKE 7, PEEK(SV + 4)
220  POKE 30, FN MD(C): POKE 31, INT (C / 256)
230  POKE 28, FN MD(N): POKE 29, INT (N / 256)
240  POKE 8, FN MD(AV + 7 + 3 * C): POKE 9, INT ((AV + 7 + 3 * C)/ 256)
250  CALL 768
260 C =  PEEK (30) + 256 *  PEEK (31)
270  IF C > N THEN 300
280  PRINT S$;" MATCHES #";C;" (PHRASE: ";R$(C);")"
290 C = C + 1: IF C <  = N THEN 190
300  PRINT : PRINT "MACHINE LANGUAGE SEARCH COMPLETED"
310  PRINT : INPUT "PRESS 'RETURN' FOR APPLESOFT SEARCH: ";A$: PRINT
320  FOR I = 1 TO N
330  IF S$ =  LEFT$ (R$(I), LEN (S$)) THEN  PRINT S$;" MATCHES #";I;"
      (PH  RASE: ";R$(I);") "
340  NEXT I: PRINT : PRINT "APPLESOFT SEARCH COMPLETED": END
1000  FOR I = 768 TO 849: READ X: POKE I,X: NEXT I: RETURN
1010  DATA  32,74,255,160,0,177,8,133,1,200,177,8,133,26,200,177,8,133,
       27,165,1,197,0,48,15,160,0,177,6,209
1020  DATA  26,208,7,200,196,0,240,16,208,243,165,30,197,28,208,11,165,
       31,197,29,208,5,230,31,76,63,255,24,165,8
1030  DATA  105,3.144,2,230,9,133,8,24,165,30,105,1,144,2,230,31,133,
       30,56,176,177
```

```
0800               ;*************************
0800               ;*                       *
0800               ;*SEARCHING STRING ARRAYS*
0800               ;*      GARY LITTLE      *
0800               ;*                       *
0800               ;*     STRING SEARCH     *
0800               ;*                       *
0800               ;*   COPYRIGHT(C) 1981   *
0800               ;*    MICRO INK, INC.    *
0800               ;*  CHELMSFORD, MA 01824 *
0800               ;*   ALL RIGHTS RESERVED *
0800               ;*                       *
0800               ;*************************
0800               ;
0800               ;
0800               ;
0800               ;
0800               LENS    EPZ $0       ;LENGTH OF SEARCH PHRASE
0800               LENR    EPZ $1       ;LENGTH OF STRING ARRAY VARIABLE
0800               SP      EPZ $6       ;POINTER TO SEARCH PHRASE
0800               RP      EPZ $8       ;POINTER TO ARRAY VARIABLE TABLE
0800               RL      EPZ $1A      ;POINTER TO ARRAY VARIABLE
0800               NL      EPZ $1C      ;ENDING ARRAY NUMBER
0800               CL      EPZ $1E      ;STARTING ARRAY NUMBER AND COUNTER
0800               SAVE    EQU $FF4A    ;SAVE REGISTERS
0800               RSTORE  EQU $FF3F    ;RESTORE REGISTERS
0800               ;
0300                       ORG $300
0300                       OBJ $800
0300               ;
0300 204AFF                JSR SAVE     ;SAVE REGISTERS
0303 A000          LOOP    LDY #$00
0305 B108                  LDA (RP),Y   ;GET LENGTH OF VARIABLE
0307 8501                  STA LENR     ;AND STORE
0309 C8                    INY
030A B108                  LDA (RP),Y   ;GET POINTER (LO)
030C 851A                  STA RL       ;AND SAVE
030E C8                    INY
030F B108                  LDA (RP),Y   ;GET POINTER (HI)
0311 851B                  STA RL+1     ;AND SAVE
0313 A501                  LDA LENR     ;IF LENGTH OF SEARCH
0315 C500                  CMP LENS     ;PHRASE EXCEEDS LENGTH
0317 300F                  BMI NOPE     ;OF VARIABLE, SEARCH FAILS
0319 A000                  LDY #$00
031B B106          AGAIN   LDA (SP),Y   ;COMPARE THE PHRASES
031D D11A                  CMP (RL),Y   ;LETTER BY LETTER
031F D007                  BNE NOPE     ;FAILS IF NOT EQUAL
0321 C8                    INY
0322 C400                  CPY LENS
0324 F010                  BEQ RTS1     ;SUCCESS!
0326 D0F3                  BNE AGAIN
0328 A51E          NOPE    LDA CL       ;COMPARE COUNTER
032A C51C                  CMP NL       ;TO ENDING ARRAY NUMBER
032C D00B                  BNE LOOP1
032E A51F                  LDA CL+1
0330 C51D                  CMP NL+1
0332 D005                  BNE LOOP1    ;DONE IF EQUAL
0334 E61F                  INC CL+1
0336 4C3FFF        RTS1    JMP RSTORE
0339 18            LOOP1   CLC
033A A508                  LDA RP       ;SET POINTER TO NEXT
033C 6903                  ADC #$03     ;TRIO OF ARRAY BYTES
033E 9002                  BCC N1
0340 E609                  INC RP+1
0342 8508          N1      STA RP
0344 18                    CLC
0345 A51E                  LDA CL       ;INCREMENT COUNTER
```

```
0347 6901      68          ADC #$01
0349 9002      69          BCC N2
034B E61F      70          INC CL+1
034D 851E      71   N2     STA CL
034F 38        72          SEC
0350 B0B1      73          BCS LOOP              ;CHECK NEXT ARRAY VARIABLE
              74          END

***** END OF ASSEMBLY


          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

LENS    0000  LENR   0001  SP     0006  RP     0008  RL     001A  NL 001C
CL      001E

** ABSOLUTE VARABLES/LABELS

SAVE    FF4A  RSTORE FF3F  LOOP   0303  AGAIN  031B  NOPE   0328
RTS1    0336  LOOP1  0339  N1     0342  N2     034D

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0092
```

Applesoft and Matrices

by Cornelis Bongers

This machine language program performs the most commonly used special matrix operations, as well as most Applesoft operations. The program can be linked to Applesoft by means of the & statement. Two advantages of using this program rather than a BASIC subroutine are a significant increase in execution speed (on the average a factor 5) and greater convenience. The required system configuration for the program is a 48K Apple with Applesoft in ROM (or in the Language Card).

For those who are not accustomed to working with matrices, a matrix is a block of numbers. Several operations can be performed on a matrix or a pair of matrices. For instance, adding two matrices A and B together, we obtain a matrix C, whose elements consist of the sums of the corresponding elements of A and B. Thus if,

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 1 & 4 \\ 4 & -2 & 1 \end{bmatrix}$$

and

$$B = \begin{bmatrix} 2 & 4 & 7 \\ 1 & 8 & -6 \\ 5 & 0 & 1 \end{bmatrix}$$

then the sum of A and B is

$$C = \begin{bmatrix} 3 & 7 & 12 \\ 3 & 9 & -2 \\ 9 & -2 & 2 \end{bmatrix}$$

It will be clear that A, B, and C can be represented by three 2-dimensional arrays in BASIC. When A and B have to be added, the following BASIC routine may be used:

```
100 FOR I=1 TO N: FOR J=1 TO M: C(I,J)=A(I,J)+B(I,J): NEXTJ,I
```

where N and M are both equal to 3 in our example. When using the machine language program, this routine can be replaced by the statement:

```
100 & C=A+B
```

Note that by using the latter statement, the names of the matrices are irrelevant. In the BASIC routine the names of the matrices always must be A, B, and C to comply with the names of the BASIC arrays.

Applesoft Operations

Except for comparison, SCRN(, and CHR$, all the Applesoft operators and functions that can be used on real variables or expressions are available for matrix operations. There are, however, some restrictions on the syntax of the matrix statement. First, no more than 3 matrices may be used in a matrix statement. Second, single-valued expressions (or variables) must be put between brackets. Another restriction is that matrices used in an & statement must have two dimensions. Each of these dimensions must be larger than 0 and smaller than 255. Furthermore, each matrix appearing in an & statement must have been dimensioned previously by means of a DIM statement. For the exact syntax of the matrix statement we refer to the 'Instructions' section of the article. Some examples are listed below.

Example 1:

```
10 DIM A(10,10): B=1
20 &A=(B): A=RND(A): A=A*(10): A=INT(A)
```

In this example, the array A is set equal to 1. Next, the RND function is performed on all elements of A, so that A now contains random numbers between 0 and 1. Then A is multiplied by 10, and the INT function is executed on each element of A. After the execution of line 20, A is thus filled with random numbers between 0 and 9. Note that the statement A=(RND(1)) puts all elements of A equal to the same random number.

Example 2:

```
10 DIM A(5,6), B(5,6), C(5,6)
20 B=3
30 &A=(3): B=(2): C=A*B: C=C  (B)
```

The statement C=A*B multiplies the corresponding elements of A and B and stores the result in the corresponding elements of C. After the execution of this statement, all elements of C are therefore equal to 6. Note that for a successful execution of the statement, A, B, and C must have the same dimension (or order). By means of the last statement, all elements of C are raised to the third power. If, instead of the statement C=C ^(B), the statement C=C ^B is used, all elements of C will become equal to the second power of 6, because now the *matrix* B instead of the *variable* B is taken.

Matrix Operations

Although the operations and functions used in the examples above can be handy sometimes, they hardly justify the writing of a machine language program. The real usefulness of the program is, therefore, not its ability to perform Applesoft functions and operations, but rather to handle some specific matrix operations as well. The following operations are implemented:

1. A = IDN(*aexpr*) where A must be a square matrix and 1 < = *aexpr* < = N if N is the order of A. This statement puts A equal to a matrix consisting of zeros and ones. If *aexpr* equals one, A becomes the identity matrix. For larger values of *aexpr*, the columns of the identity matrix will be rotated *aexpr* – 1 positions to the left. For instance, if A and B are square matrices of order 3, then A = IDN(1) and B = IDN(2) return.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

2. A = TRN(B) puts A equal to the transpose of B. If B is of order p by q, then A must be of order q by p. Putting a matrix equal to its own transpose (i.e. A = TRN(A)) is not allowed. For instance, if B equals,

$$B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 0 \end{bmatrix}$$

then A = TRN(B) will return

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 0 \end{bmatrix}$$

3. A = B.C puts A equal to the matrix product of B and C. If B is of order p by q, then the first dimension of C must equal q. In case the second dimension of C equals r (thus C is q by r), the matrix A has to be of the order p by r. Furthermore, the matrix on the left of the "=" sign may not equal one of the matrices on the right of the "=." As an example, we can multiply the matrices A and B in the example above by means of the statement &C = A.B. This leads to

$$C = \begin{bmatrix} 35 & 14 \\ 14 & 20 \end{bmatrix}$$

4. A = MIN(B), A = MAX(B) or A = ABM(B) put A respectively equal to the minima, the maxima, or the absolute maxima of the columns of B. The overall maximum, minimum, or absolute maximum of B is stored in A(0,1). If B is of order p by q, then A must be of order q by 1.

5. A = INV(B) puts A equal to the inverse of B and stores the determinant of B in A(0,0). A and B must be square and of the same order. The statement D = INV(C), where C equals the matrix above, returns for instance,

$$D = \begin{bmatrix} .0396825397 & -.0277777778 \\ -.0277777778 & .0694444444 \end{bmatrix}$$

At the execution of the inverse statement, values stored in the 0th row of the target matrix will be destroyed since this row is used to store some pointers. To obtain the inverse of a matrix A, the statement A = INV(A) also may be used. Finally, zeros on the main diagonal of the matrix to be inverted are allowed.

6. A = NEINV(B) gives the same result as A = INV(B) except that the program continues if a division by zero occurs when B is singular. When using NEINV, it is recommended to check the determinant of B (in A(0,0)) after execution of the statement. When B is singular, the determinant will be zero.

7. A = PNT (*aexpr*) displays the matrix A. For each element of A, *aexpr* positions are reserved, and a carriage return is generated after each row. If *aexpr* equals zero, the elements of A are separated by a blank.

An Application

An interesting application of matrix algebra is the linear model. The linear model can be used to analyze the influence of a number of variables, called the independent variables, on another variable, called the dependent variable. The model has the form,

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots b_m x_m + u ,$$

where y denotes the dependent variable, and x_1, x_2, etc., denote the independent variables.

The last term, u, represents the influence of factors that were not included in the model. Usually this term is called the residual. As an example, suppose that we want to establish the relationship between the annual regional sales of a particular product (y), the number of times advertised (x_1) and the number of people living in the region (x_2). The available data are given in the table below.

Obs. No.	**Y Sales**	**X_1 Advert.**	**X_2 Popul.**
1	118	8	583
2	138	9	692
3	104	5	1082
4	65	1	836
5	46	1	628
6	61	2	244
7	48	1	632
8	66	2	172
9	78	5	319
10	69	2	383

In matrix algebra the model can be written as,

$$Y = X.B + U,$$

where B (the unknown coefficients) is of order 3 by 1 and Y (the sales), and U (the residuals) are of order 10 by 1. The matrix X is of order 10 by 3. The elements of the first column of X are equal to one (to account for b_0) whereas the second and third columns correspond to the columns under the heading $\mathbf{X_1}$, and $\mathbf{X_2}$ in the table. To fit the equation to the data, the least squares principle is used, which means that the coefficients are chosen such that the sum of the squares of the elements of U is minimized. This leads to the following solution for B,

$$B = (X'.X)^{-1}X'.Y$$

where X denotes the transpose of X. A BASIC program to compute the least squares solution is presented in listing 1, with the results of the example. The least squares equation shows that the sales increase by 9.5 for each additional advertisement (other things being equal) whereas an increase of 100 in the population of the region increases the sales by 1.6 (other things being equal).

The application given in this section was kept simple purposely. The linear model, for instance, can easily be extended with a tremendous amount of statistics which may (or may not) simplify the analysis of the data. Also the application presented gives only a narrow view on the wide field of problems in which matrix algebra may be useful. Examples include computations with Markov-type problems and the location of the maximum (or minimum) of a function of several variables by means of the Newton method.

The Machine Language Program

The program is about $700 bytes long and starts at $8900. The end is at $8FF2, which means that the area $9000-$9600 is free for other routines. *(Editor's Note: This program is not listed, but is saved on the disk in object form as MATRICES.)*

It can be connected to an Applesoft program by means of the command : BRUN matrices or, if you don't have a disk, by the monitor command : 8900 G. In the latter case you must enter Applesoft *via* the warm start (i.e., Control-C). The BRUN or 8900 G command executes the initialization routine at the start of the program that sets HIMEM to the appropriate value and installs the & vector. In case the & vector is destroyed during execution of a program, the matrix program can be reconnected by the command CALL 35072.

The program extensively uses zero page locations to increase execution speed. However, as a consequence, the ON ERR flag will be temporarily cleared during the execution of an & line since the matrix routines use the storage space of the ON ERR pointers. After the execution of the & line, the ON ERR flag and pointers are restored to their original values. Apart from zero page locations, the control Y and the & vector are used, which implies that values stored at $3F5 - $3FA will be destroyed.

In Case of an Error

If the interpreter returns an error message during the execution of an & line, there is either a bug in your statement or a bug in my program. In the first case, the error is probably caused by the violation of one of the following conditions:

1. Only matrices containing reals are allowed in the & line.

2. Matrices used in an & statement must have 2 dimensions.

3. Each dimension of a matrix must be larger than 0 and smaller than 255.

4. The orders of the matrices should satisfy the conditions in the "instructions" section of this article.

5. Each matrix appearing in an & statement must have been dimensioned earlier in the program by a DIM statement.

6. ON ERR doesn't work during the execution of an & line.

Although the other case (i.e. a bug in my program) seems at this time highly improbable to me since the program was heavily tested for several months, I am well aware that there are some kinds of bugs that can, as it seems, only be discovered by other people. Therefore, if you find one, I would appreciate it very much if you let me know.

Finally, a utility package which contains, among others, the matrix program, will be released soon. This utility package resides in the second 4K bank of the Language Card, and it will use only $300 bytes of 'normal' RAM.

Instructions

This section contains the matrix expressions that can be executed by means of the & line. The syntax of the line is:

```
& matrix expression: matrix expression: etc.
```

The following operators and functions may be used:

```
operator := +, −, *, /, ∧, AND,OR
function := SGN, INT, ABS, USR, FRE, PDL, POS, SQR, RND, LOG, EXP,
            COS, SIN, TAN, ATN, PEEK
```

Unless stated otherwise, matrices appearing in an & statement must have the same order, and matrix names on the left of the "=" sign can be chosen equal to matrix names on the right of the "=". The matrix expressions that are allowed follow.

I. Applesoft Operations and Functions with:

1.1 1 matrix and 1 expression
A = *(aexpr)*

Example:
A = (−1/2), B = (Z%)

1.2 2 matrices
A = B
A = −B
A = NOT B
A = function(B)

Example:
A = SIN(B)

1.3 2 matrices and 1 expression
A = B operator *(aexpr)*

Example:
A = B∧(COS(−3))

1.4 3 matrices
A = B operator C

Example:
A = B/C

II. Specific Matrix Operations

2.1 A = IDN*(aexpr)* — Identity: A must be square and 1< = *aexpr* < = order of A.

2.2 A = TRN(B) — Transpose: if B is of order p by q, then A must be of order q by p. A = TRN(A) is not allowed.

2.3 A = B.C — Multiplication: if B is of order p by q and C of order q by r, then A must be of order p by r. A = A.C or A = C.A is not allowed.

2.4 A = MIN(B), A = MAX(B), A = ABM(B) — Minimum, maximum or absolute maximum: if B is of order p by q then A must be of order q by 1. After execution A(0,1) contains the overall minimum, maximum or absolute maximum of B.

2.5 A = INV(B) — Inverse: A and B must be square and of the same order. After execution, A(0,0) contains the determinant of B.

2.6 A = NEINV(B) — Inverse: same as INV, except that singularity of B doesn't stop the program.

2.7 A = PNT(*aexpr*) — Print: if *aexpr* = 0 the elements are separated by a blank, else *aexpr* positions are reserved for each element.

```
1  REM  ************************
2  REM  *                      *
3  REM  * MATRICES & APPLESOFT *
4  REM  *     BY C. BONGERS    *
5  REM  *                      *
6  REM  *      MATRIX DEMO     *
7  REM  *                      *
8  REM  *  COPYRIGHT (C) 1981  *
9  REM  *   MICRO INK, INC.    *
10  REM * CHELMSFORD, MA 01824 *
11  REM * ALL RIGHTS RESERVED  *
12  REM *                      *
13  REM ************************
14  REM
15  REM  THE LINEAR MODEL
16  REM
18  HOME
20  INPUT "NUMBER OF OBSERVATIONS ? ";N
30  INPUT "NUMBER OF INDEPENDENT VARIABLES ? ";M:M1 = M + 1
40  IF M1 >  = N THEN  PRINT : PRINT "TOO FEW OBSERVATIONS ": STOP
50  DIM X(N,M1),XA(M1,N),Y(N,1),B(M1,1),E(N,1),EA(1,N),S(M1,M1)
60  DIM V1(1,1),V2(1,1),H(M1,1),J(1,N)
70  PRINT : PRINT "INPUT THE ELEMENTS OF THE Y-VECTOR": PRINT
80  FOR I = 1 TO N
90  PRINT "ELEMENT ";I;" ? ";: INPUT "";Y(I,1):X(I,1) = 1
100  NEXT I
110  FOR J = 2 TO M1
120  PRINT : PRINT "INPUT THE ELEMENTS OF THE X";J - 1;"-VECTOR": PRINT
130  FOR I = 1 TO N
140  PRINT "ELEMENT ";I;" ? ";: INPUT "";X(I,J)
150  NEXT I,J
160  REM  CALCULATE RESULTS
170  & XA = TRN(X):S = XA.X:S = NEINV(S):H = XA.Y:B = S.H
180  IF S(0,0) = 0 THEN  PRINT "THE S-MATRIX IS SINGULAR": STOP
190  PRINT : PRINT "THE LEAST SQUARES EQUATION EQUALS ": PRINT
200  PRINT "Y = ";B(1,1);
210  FOR J = 2 TO M1: IF B(J,1) >  = 0 THEN  PRINT "+";
220  PRINT B(J,1);"*X";J - 1;
230  NEXT : PRINT : PRINT
240  & E = X.B:EA = TRN(E):E = Y - E
250  PRINT "** THE TABLE OF RESIDUALS **": PRINT
260  PRINT "NO"; TAB( 4);"OBSERVED Y"; TAB( 16);"ESTIMATED Y";
     TAB( 29);"    RESIDUAL"
270  FOR I = 1 TO N
280  PRINT I; TAB( 4);Y(I,1); TAB( 16);EA(1,I); TAB( 29);E(I,1)
290  NEXT I: PRINT
300  & EA = TRN(E):V1 = EA.E
310  PRINT "STANDARD DEV. RESIDUALS: "; SQR (V1(1,1) / (N - M1))
320  & J = (1):V2 = J.Y:V2 = V2 / (N):E = Y - (V2(1,1)):EA = TRN(E):
    V2 = EA.E
330 R = (V2(1,1) - V1(1,1)) / V2(1,1): IF R < 0 THEN R = 0
340  PRINT "R^2";: HTAB (24): PRINT ": "; SQR (R)
350  END
```

AMPER-SORT

by Alan G. Hill

Here's a fast machine language sort utility for the Apple II that handles integer, floating point, and character records. Because it is callable from BASIC, this sort routine is a worthwhile addition to any software library.

A sort utility is usually one of the first programs needed for records management application programs. If the utility is written in BASIC and runs under an interpreter, one quickly discovers that the sort is painfully slow on a micro. The sort program presented here, written in machine language for the Apple II with Applesoft ROM, will certainly remedy that problem. While no speed records will be set, it will run circles around BASIC, sorting 900 integer, 700 floating point, or 300 30-character records in about 60 seconds.

The & Connection

Speed is not the only beauty of AMPER-SORT. As its name implies, the BASIC-to-machine language interface utilizes the powerful, but not-widely-known, feature of Applesoft—the Ampersand. What is the Ampersand and why is it so useful? Consider the following example of how a BASIC program passes sort parameters to AMPER-SORT.

```
100 &SRT#(AB$,0,10,7,10,A,1,5,D)
```

This statement, when embedded in a BASIC program or entered as an immediate command, will command AMPER-SORT to sort AB$(0) through AB$(10) in ascending order based on the 7th to 10th characters and in descending order for the 1st through 5th characters. Of course, POKEs could be used to pass parameters from other 6502 BASICs, but there's something more professionally pleasing about the Ampersand interface.

There is no user documentation from Apple on the Ampersand feature. I first read of the feature in the October 1978 issue of *CALL APPLE*. When the Applesoft interpreter encounters an ampersand (&) character at the beginning of a BASIC statement, it does a JSR $3F5. If the user has placed a JMP instruction there, a link is made to the user's machine language routine. Apple has thoughtfully provided some ampersand handling routines described in the November and December issues of *CALL APPLE*. The routines enable your machine language routine to examine and convert the characters or expressions following the ampersand. Here are the routines used in AMPER-SORT.

CHRGET ($00B1)

This routine will return, in the accumulator, the next character in the statement.

The first character is in the accumulator when the JSR $3F5 occurs. The zero flag is set if the character is an end-of-line token (00) or statement terminator ($3A). The carry flag is set if the character is non-numeric, and cleared if it is numeric. The character pointer at $B8 and $B9 is advanced automatically so that the next JSR $B1 will return the next character. A JSR $B7 will return a character without advancing the pointer.

FRMNUM ($DD67)

This routine evaluates an expression of variables and constants in the ampersand statement from the current pointer to the next comma. The result is placed in the floating point accumulator.

GETADR ($E752)

This routine will convert the floating point accumulator to a two-byte integer and place it in $50 and $51. FRMNUM and GETADR are used by AMPER-SORT to retrieve the sort parameters and convert each to an integer.

GETBYT ($E6F8)

This routine will retrieve the next expression and return it as a one-byte integer in the X-register.

It is the user's responsibility to leave the $B8 and $B9 pointer at the terminator.

Exploration of Parameters

Parameters are passed to AMPER-SORT in the following form:

```
100 &SRT#(AB$,B,E,7,10,A,1,5,D)
```

where:

AB$ Is the variable name of the string array to be sorted. The general form is XX$ for string arrays, XX% for integer arrays, and XX for floating point arrays.

B is a variable, constant or expression containing the value of the subscript element where the sort is to begin; e.g. AB$(B).

E is a variable or constant or expression containing the value of the subscript element where the sort is to end; e.g., AB$(E). B and E are useful when the AB$ array is partially filled or has been sectioned into logically separate blocks that need to be sorted independently.

7 is a variable, constant or expression specifying the beginning position of the major sort field.

10 is a variable, constant or expression specifying the ending position of the major sort field.

A is a character specifying that the major sort field is to be sorted in ascending order.

1 is a variable, constant or expression specifying the beginning position of the first minor sort field.

5 is a variable, constant or expression specifying the ending position of the first minor sort field.

D is a character specifying that the first minor sort field is to be sorted in descending order.

Using AMPER-SORT

The &SRT command will sort character, integer or floating point arrays and can be used in either the immediate or deferred execution mode similar to other Applesoft BASIC commands. Of course, the named array must have been previously dimensioned and initialized in either case.

A. Character Arrays
   1. Equal or unequal element lengths
   2. Some or all elements
   3. Ascending or descending order
   4. A major sort field and up to 4 minor sort fields

Examples:

```
10 DIM NA$(500)

100 &SRT#(NA$,0,500,1,5,A)
200 &SRT#(NA$,0,500,1,5,A,6,10,D,11,11,A)
299 F% =0:L=10
300 &SRT=(NA$,F%,L,10,15,D)
```

Line 100 sorts on positions 1 through 5 in ascending order for all 501 elements of NA$(500).

Line 200 is the same as Line 100 except that minor sort fields are specified. The sort sequence on positions 1-5 is in ascending order, positions 6-10 are in descending order, and position 11 is ascending order.

Line 299 and 300 sort on positions 10-15 in descending order for NA$(0) through NA$(10).

B. Integer and Floating Point Arrays
   1. Some or all elements
   2. Ascending order only. (Step through the array backwards if needed in descending order.)

Examples:

```
10 DIM AB%(100),FP(100)

100 &SRT#(AB%,0,100)
299 S=50: E=100
300 &SRT#(AB%,S,E)
399 X=49
400 &SRT#(FP,0,X)
```

Line 100 sorts all 101 elements of AB%(100) in ascending order. Lines 299 and 300 sort from AB%(50) through AB%(100), while lines 399 and 400 sort from FP(0) through FP(49).

Limited editing has been included in the parameter processing code. Therefore, you must be careful to observe such rules as:

1. 0≤B< E≤ maximum number of AB$ elements.
2. AB$ must be a scalar array; e.g., AB$(10), not AB$(20,40).
3. The sort array name must be less than 16 characters, only the first two count, and they must be unique.
4. The maximum number of sort fields is 5.
5. The beginning sort field position must not be greater than the ending sort field position.

Options:

1. Constants, variables, or expressions may be used for subscript bounds and sort positions.
2. The &SRT command may be used in immediate or deferred execution mode.

Some editing checks are made. You will notice this when you get a "?SYNTAX ERROR IN LINE XXX" error message. You will also get a "VARIABLE XXX NOT FOUND" message if the routine cannot find the AB$ variable name in variable space.

The AMPER-SORT program is listed in its entirety. A BASIC demo program is also shown.

```
10  REM *************************
20  REM *                       *
30  REM *     AMPER-SORT        *
40  REM *     ALLEN HILL        *
45  REM *                       *
50  REM *    AMPERSORT DEMO     *
55  REM *                       *
60  REM *  COPYRIGHT (C) 1981   *
70  REM *   MICRO INK, INC.     *
80  REM * CHELMSFORD, MA 01824 *
90  REM *  ALL RIGHTS RESERVED *
100  REM *                      *
110  REM ************************
1000  GOTO 10000
1050  REM  CHARACTER SORT
1060 CH$ = "ABCDWXYZ":L =  LEN (CH$) - 1
1070 N% = 8
1080  DIM AB$(N%)
1090  FOR I = 0 TO N%
1100 C$ =  MID$ (CH$, INT ( RND (1) * L) + 1,1)
1110 B$ =  MID$ (CH$, INT ( RND (1) * L) + 1,1)
1120  FOR J = 1 TO 3
1130 C$ = C$ + C$:B$ = B$ + B$
1140  NEXT J
1150 AB$(I) = B$ + C$
1160  NEXT I
1170  GOSUB 1240
1180  REM  SORT HALF ASCENDING
1190  REM  SORT HALF DESCENDING
1200  & SRT#(AB$,0,N%,1,8,A,9,16,D)
1210  GOSUB 1260
1220  GOTO 11000
1230  REM  PRINT ROUTINE
1240  PRINT "     BEFORE"
1250  GOTO 1270
1260  PRINT "     AFTER": PRINT "ASCEND  DESCEND"
1270  FOR I = 0 TO N%
1280  PRINT AB$(I): NEXT I: RETURN
2000  REM  INTEGER SORT
2010 N% = 8
2020  DIM IN%(N%)
2030  FOR I = 0 TO N%
2040 IN%(I) = 7500 -  INT ( RND (1) * 15000)
2050  NEXT I
2060  GOSUB 2120
2070  REM    SORT
2080  & SRT#(IN%,0,N%)
2090  GOSUB 2130
2100  GOTO 11000
2110  REM  PRINT ROUTINE
2120  HTAB 10: PRINT "BEFORE": GOTO 2140
2130  HTAB 10: PRINT "AFTER"
2140  FOR I = 0 TO N%
2150  PRINT IN%(I): NEXT I: RETURN
3000  REM  FLOATING POINT
3010 T% = 8
3020  DIM FP(T%)
3030  FOR I = 0 TO 8
3040 FP(I) = 1000 *  RND (1) *  SIN (I * 7.16)
3050  NEXT I
3060  GOSUB 3120
3070  REM  SORT
3080  & SRT#(FP,0,T%)
3090  GOSUB 3130
3100  GOTO 11000
3110  REM  PRINT ROUTINE
3120  HTAB 10: PRINT "BEFORE": GOTO 3140
3130  HTAB 10: PRINT "AFTER"
3140  FOR I = 0 TO T%
3150  PRINT FP(I): NEXT I: RETURN
9999  REM
10000  REM  ** &SORT DEMO **
10010  REM  SAVE ROOM FOR
10020  REM  SORT ROUTINE
10030  HIMEM: 20992: REM  $5200
```

```
10040 D$ =  CHR$ (4)
10050  PRINT D$;"BLOAD AMPERSORT,A$5200"
10060  REM  SET UP '&' HOOK
10070  REM  AT $3F5:JMP $5200
10080  POKE 1013,76: POKE 1014,0: POKE 1015,82
10090  HOME : CLEAR
10100  VTAB 8: HTAB 15: PRINT "SORT DEMO"
10110  PRINT : HTAB 15: PRINT "SELECTIONS"
10120  PRINT : HTAB 10: PRINT "1 INTEGER SORT"
10130  HTAB 10: PRINT "2 FLOATING POINT SORT"
10140  HTAB 10: PRINT "3 CHARACTER SORT"
10150  HTAB 10: PRINT "4 EXIT"
10160  VTAB 17: INPUT "SELECTION ";SE%
10170  IF SE% < 0 OR SE% > 4 THEN 10090
10180  ON SE% GOTO 2000,3000,1050,10190
10190  END
11000  PRINT "HIT ANY KEY TO RETURN TO MENU"
11010  WAIT  - 16384,128
11020  POKE  - 16368,0
11030  GOTO 10090
```

```
0800      ;************************
0800      ;*                      *
0800      ;*     AMPER-SORT       *
0800      ;*     BY ALAN HILL     *
0800      ;*                      *
0800      ;*      AMPERSORT       *
0800      ;*                      *
0800      ;*  COPYRIGHT (C) 1981  *
0800      ;*    MICRC INK, INC.   *
0800      ;* CHFLMSFCRD, MA 01824 *
0800      ;*  ALL RIGHTS RESERVEC *
0800      ;*                      *
0800      ;************************
0800      ;
0800      ;
0800      NAPT   EPZ $D0
0800      NMS1   EPZ $D4
0800      ASI1   EPZ $D6
0800      CSI1   EPZ $D8
0800      ASI2   EPZ $DA
0800      CSI2   EPZ $DC
0800      IIII   EPZ $DE
0800      NNNN   EPZ $E0
0800      FSTR   EPZ $E2
0800      FLEN   EPZ $E7
0800      DISP   EPZ $EC
0800      JJJJ   EPZ $ED
0800      LENI   EPZ $EF
0800      LENJ   EPZ $F0
0800      TYPE   EPZ $F1
0800      ZZ50   EPZ $50
0800      ZZ6B   EPZ $6B
0800      CHRG   EPZ $B1
0800      GETB   EQU $E6F8      ;APPLESOFT EVALUATION ROUTINE 'GETBY
0800      SNER   EQU $DEC9      ;OUTPUTS "SYNTAX ERROR"
0800      FRNM   EQU $DD67      ;APPLESCFT EXPRESSION EVALUATOR ROUTI
0800      GETA   EQU $E752      ;APPLESOFT FP->INT RCUTINE 'GETADR'
0800      MPLY   EQU $558A      ;RELOCATED OLD MON. MULTIPLY ROUTINE
0800      COUT   EQU $FDED      ;APPLESOFT OUTPUT ROUTINE
0800      ;
0800      ;
5200             ORG $5200
5200             OBJ $0800
5200      ;
5200      ;
5200      ;PRCCESS '&'
5200      ;
```

```
5200                ;
5200 48             SORT   PHA              ;ENTER WITH FIRST CHAR
5201 20E654                JSR SVZP         ;SAVE A WORK AREA IN ZERO PAGE
5204 68                    PLA
5205 A200                  LDX #$00
5207 DD2C55         SR01   CMP SRTS,X       ;EDIT FOR 'SRT#('
520A D046                  BNE ERRX         ;SIGNAL 'SYNTAX ERROR'
520C 20B100                JSR CHRG         ;GET NEXT CHARACTER
520F E8                    INX
5210 E005                  CPX #$05
5212 D0F3                  BNE SR01
5214 A200                  LDX #$00         ;OK SO FAR
5216 F003                  BEQ VNAM
5218 20B100         SR04   JSR CHRG         ;GET ANOTHER CHARACTER
521B C92C           VNAM   CMP ',           ;LOOP TO GET ARRAY NAME
521D F00A                  BEQ SR05
521F 9D7255                STA NAME,X       ;SAVE NAME
5222 E8                    INX
5223 E010                  CPX #$10         ;16 CHARACTERS IS LONG
5225 D0F1                  BNE SR04         ;ENOUGH FOR A NAME
5227 F029                  BEQ ERRX         ;SIGNAL ERROR
5229 CA             SR05   DEX
522A BD7255                LDA NAME,X       ;WHAT TYPE
522D C924                  CMP '$
522F F024                  BEQ CHAR         ;CHARACTER
5231 C925                  CMP '%
5233 D015                  BNE FP00         ;FLOATING POINT
5235                ;
5235                ; INTEGER SORT
5235                ;
5235 A201           INTE   LDX #$01         ;INTEGER
5237 A980           INT1   LDA #$80
5239 1D7255                ORA NAME,X       ;NEG. ASCII
523C 9D7255                STA NAME,X
523F CA                    DEX
5240 10F5                  BPL INT1
5242 A902                  LDA #$02         ;INITIALIZE DISPLACEMENT
5244 85EC                  STA DISP
5246 A901                  LDA #$01
5248 D019                  BNE SR06
524A                ;
524A                ;
524A                ;FLOATING POINT SORT
524A                ;
524A                ;
524A A905           FP00   LDA #$05
524C 85EC                  STA DISP
524E A902                  LDA #$02
5250 D011                  BNE SR06
5252                ;
5252                ;
5252 4CA552         ERRX   JMP ERRO
5255                ;
5255                ;
5255                ;CHARACTER SORT
5255                ;
5255 A980           CHAR   LDA #$80
5257 0D7355                ORA NAME+01      ;NEG. ASCII
525A 8D7355                STA NAME+01
525D A903                  LDA #$03
525F 85EC                  STA DISP
5261 A900                  LDA #$00
5263                ;
5263                ; ** SET UP SORT LIMITS **
5263                ;
5263 85F1           SR06   STA TYPE         ;0=CH,1=INT,2=FP
5265 20B100                JSR CHRG         ;NOW GET SUBSCRIPTS
5268 2067DD                JSR FRNM         ; AND PUT IN F.P. ACC.
526B 2052E7                JSR GETA         ;CONVERT TO INTEGER
526E A550                  LDA ZZ50
5270 85DE                  STA IIII         ;FIRST SUBSCRIPT
5272 A551                  LDA ZZ50+01
5274 85DF                  STA IIII+01
5276 20B100                JSR CHRG
5279 2067DD                JSR FRNM
```

```
527C 2052E7            JSR GETA
527F A550              LDA ZZ50
5281 85D4              STA NMS1           ;LAST SUBSCRIPT INTO N-1
5283 18                CLC
5284 6901              ADC #$01
5286 85E0              STA NNNN           ;N
5288 A551              LDA ZZ50+01
528A 85D5              STA NMS1+01
528C 6900              ADC #$00
528E 85E1              STA NNNN+01
5290 A5F1              LDA TYPE
5292 D059              BNE TERM           ;BRANCH NOT CHARACTER SORT
5294 F015              BEQ SR16
5296            ;
5296            ; **** ERROR ****
5296            ;
5296 A200       ERR3   LDX #$00
5298 BD3155     SR11   LDA MSG1,X         ;ARRAY VARIABLE NAME
529B 0980              ORA #$80           ;NOT FOUND
529D 20EDFD            JSR COUT           ;NOTIFY USER
52A0 E8                INX
52A1 E017              CPX #$17
52A3 D0F3              BNE SR11
52A5 200955     ERRO   JSR RSZP           ;RESTORE ZERO PAGE AND
52A8 4CC9DE            JMP SNER           ;SIGNAL SYNTAX ERROR
52AB            ;
52AB            ; *** GET SORT FIELDS ***
52AB A000       SR16   LDY #$00
52AD 8C8955            STY SAVY
52B0 20B100     SR17   JSR CHRG           ;GET NEXT CHARACTER
52B3 20F8E6            JSR GETB
52B6 CA                DEX
52B7 AC8955            LDY SAVY
52BA 96E2              STX FSTR,Y         ;START COLUMN-1
52BC 20B100            JSR CHRG
52BF 20F8E6            JSR GETB
52C2 AC8955            LDY SAVY
52C5 96E7              STX FLEN,Y         ;END COLUMN
52C7 20B100            JSR CHRG
52CA 90D9              BCC ERRO           ;SHOULD BE 'A'OR'D'
52CC C944              CMP 'D
52CE F004              BEQ SR07           ;DESCENDING
52D0 A9FF              LDA #$FF           ;ASCENDING
52D2 3002              BMI SR09
52D4 A900       SR07   LDA #$00
52D6 998255     SR09   STA UPDN,Y         ;SAVE SEQUENCE
52D9 C8                INY
52DA 8C8955            STY SAVY
52DD 20B100            JSR CHRG
52E0 C929              CMP ')
52E2 F006              BEQ LAST
52E4 C92C              CMP ',
52E6 F0C8              BEQ SR17           ;LOOP FOR NEXT SORT FIELDPARMS
52E8 D0BB              BNE ERRO
52EA 8C8855     LAST   STY PRSN           ;NO. OF SORT FIELDS
52ED 20B100     TERM   JSR CHRG           ;MUST BE TERMINATOR
52F0 D0B3              BNE ERRO           ;IT WASN'T
52F2            ;
52F2            ;SEARCH SORT ARRAY NAME
52F2            ;
52F2 A000       MC20   LDY #$00
52F4 B16B              LDA (ZZ6B),Y
52F6 CD7255            CMP NAME
52F9 D008              BNE MC22
52FB C8                INY                ;FOUND FIRST CHARACTER
52FC B16B              LDA (ZZ6B),Y
52FE CD7355            CMP NAME+01
5301 F02B              BEQ SETN           ;FOUND BOTH
5303 18         MC22   CLC                ;KEEP LOOKING
5304 A002              LDY #$02
5306 B16B              LDA (ZZ6B),Y
5308 656B              ADC ZZ6B
530A 48                PHA
530B C8                INY
```

```
530C B16B              LDA (ZZ6B),Y
530E 656C              ADC ZZ6B+01
5310 856C              STA ZZ6B+01
5312 68                PLA
5313 856B              STA ZZ6B
5315 C56D              CMP $6D
5317 A56C              LDA ZZ6B+01
5319 E56E              SBC $6E
531B B003              BCS SR27            ;NO LUCK, OUT OF BOUNDS
531D 4CF252            JMP MC20
5320            ;
5320            ; ****** NAME NOT FOUND *******
5320            ;
5320 A202       SR27   LDX #$02
5322 BD7255     SR28   LDA NAME,X
5325 9D3B55            STA VARI+1,X        ;PUT NAME IN BUFFER
5328 CA                DEX
5329 10F7              BPL SR28
532B 4C9652            JMP ERR3            ;SEND A MESSAGE
532E            ;
532E            ; ***** INITIALIZE ARRAY POINTER ***
532E            ;
532E 18         SETN   CLC                 ;FOUND VARIABLE NAME OF
532F A56B              LDA ZZ6B            ;ARRAY TO BE SORTED.
5331 6907              ADC #$C7            ;COMPUTE ADDRESS OF
5333 8552              STA $52             ;STRING LENGTH BYTE.
5335 A56C              LDA ZZ6B+01
5337 6900              ADC #$00
5339 8553              STA $53
533B A5DE              LDA IIII            ;(6B,6C)+7+DISP*IIII
533D 8550              STA ZZ50
533F A5DF              LDA IIII+01
5341 8551              STA ZZ50+01
5343 A5EC              LDA DISP
5345 8554              STA $54
5347 A900              LDA #$00
5349 8555              STA $55
534B 208A55            JSR MPLY            ;ROM MULTIPLY ROUTINE
534E A550              LDA ZZ50
5350 85D6              STA ASII            ;SAVE ADDRESS FOR MUCH USE
5352 A551              LDA ZZ50+01
5354 85D7              STA ASII+01
5356 4C6653            JMP SR22
5359            ;
5359            ;****** BEGIN SORT *******
5359            ;
5359            ;** FOR I =II TO N-1 LOOP **
5359            ;
5359 18         CONI   CLC
535A A5D6              LDA ASII
535C 65EC              ADC DISP            ;NEXT I ADDRESS
535E 85D6              STA ASII
5360 A5D7              LDA ASII+01
5362 6900              ADC #$00
5364 85D7              STA ASII+01
5366 A001       SR22   LDY #$01
5368 B1D6              LDA (ASII),Y        ;GET ADDRESS OF THE
536A 85D8              STA CSII            ;CHARACTER STRING
536C C8                INY
536D B1D6              LDA (ASII),Y
536F 85D9              STA CSII+01
5371 18                CLC
5372 A5D6              LDA ASII            ; ALSO NEED ADDRESS OF
5374 65EC              ADC DISP            ;ADJACENT ELEMENT FOR
5376 85DA              STA ASI2            ;BUBBLE SORT COMPARISON
5378 A5D7              LDA ASII+01
537A 6900              ADC #$00
537C 85DB              STA ASI2+01
537E 18                CLC
537F A5DE              LDA IIII
5381 6901              ADC #$01
5383 85ED              STA JJJJ            ;J=I+1
5385 A5DF              LDA IIII+01
5387 6900              ADC #$00
5389 85EE              STA JJJJ+01
```

```
538B 4C9B53              JMP SR24
538E              ;
538E              ;**** FOR J=I+1 TO N LOOP ****
538E              ;
538E 18           CONJ   CLC
538F A5DA                LDA ASI2
5391 65EC                ADC DISP             ;INCREMENT AB$(J) ADDRESS
5393 85DA                STA ASI2
5395 A5DB                LDA ASI2+01
5397 6900                ADC #$00
5399 85DB                STA ASI2+01
539B A001         SR24   LDY #$01
539D B1DA                LDA (ASI2),Y
539F 85DC                STA CSI2             ;GET NEW STRING ADDRESS
53A1 C8                  INY
53A2 B1DA                LDA (ASI2),Y
53A4 85DD                STA CSI2+01
53A6 A5F1                LDA TYPE
53A8 F003                BEQ CHST             ;CHARACTER SORT
53AA 4C2F54              JMP NCHH
53AD              ;
53AD              ;*** CHARACTER SORT ***
53AD              ;
53AD A000         CHST   LDY #$00
53AF B1D6                LDA (ASII),Y         ;STRING LENGTH
53B1 F052                BEQ MC40             ;NULL STRING: SKIP
53B3 85EF                STA LENI             ;SAVE LEN (AB$(I))
53B5 B1DA                LDA (ASI2),Y
53B7 F04C                BEQ MC40
53B9 85F0                STA LENJ             ;SAVE LEN(AB$(J))
53BB A200                LDX #$00
53BD B4E2         SR29   LDY FSTR,X           ;STARTING SORT COLUMN
53BF BD8255       MC33   LDA UPDN,X           ;SEQUENCE
53C2 300C                BMI ASND             ;BRANCH ASCENDING
53C4 B1D8                LDA (CSII),Y         ;CHARACTER BY CHARACTER
53C6 D1DC                CMP (CSI2),Y         ;COMPARISON FOR DESCENDING
53C8 B014                BGE MC26             ;POSSIBLE SWAP
53CA 20C154              JSR SWAP             ;DEFINITE SWAP
53CD 4C0554              JMP MC40             ;NEXT RECORD
53D0 B1D8         ASND   LDA (CSII),Y         ;ASCENDING
53D2 D1DC                CMP (CSI2),Y
53D4 902F                BLT MC40             ;NO SWAP: NEXT RECORD
53D6 F019                BEQ MC27             ;POSSIBLE SWAP
53D8 20C154       MC25   JSR SWAP             ;SWAP
53DB 4C0554              JMP MC40             ;NEXT RECORD
53DE D025         MC26   BNE MC40             ;NO SWAP
53E0 C8                  INY                  ;LOOK AT REMAINING CHARACTER
53E1 C4EF                CPY LENI
53E3 F006                BEQ MC39             ;UP TO THE LIMITS OF UNTIL
53E5 C4F0                CPY LENJ
53E7 F016                BEQ MC29             ;WE FIND A REASON TO SWAP
53E9 900F                BLT MC28
53EB C4F0         MC39   CPY LENJ
53ED 90E9                BLT MC25             ;SWAP
53EF F00E                BEQ MC29             ;NO SWAP
53F1 C8           MC27   INY
53F2 C4EF                CPY LENI
53F4 F009                BEQ MC29
53F6 C4F0                CPY LENJ
53F8 F0DE                BEQ MC25
53FA 98           MC28   TYA
53FB D5E7                CMP FLEN,X           ;END OF SORT FIELD?
53FD D0C0                BNE MC33             ;BRANCH NO
53FF E8           MC29   INX
5400 EC8855              CPX PRSN             ;YES, ANY MORE FIELDS?
5403 D0B8                BNE SR29
5405              ;
5405              ;***** NEXT J ********
5405              ;
5405 E6ED         MC40   INC JJJJ
5407 D002                BNE MC38
5409 E6EE                INC JJJJ+01          ;J=J+1
540B A5ED         MC38   LDA JJJJ
540D C5E0                CMP NNNN             ;J=N?
540F A5EE                LDA JJJJ+01
```

```
5411 E5E1                SBC NNNN+01
5413 9014                BCC JMPJ           ;BRANCH NO
5415             ;
5415             ;*** NEXT I ****
5415             ;
5415 E6DE                INC IIII
5417 D002                BNE MC41
5419 E6DF                INC IIII+01        ;I=I+1
541B A5DE        MC41    LDA IIII
541D C5D4                CMP NMS1           ;I=N-1?
541F A5DF                LDA IIII+01
5421 E5D5                SBC NMS1+01
5423 9007                BCC JMPI           ;BRANCH NO
5425             ;
5425             ;***** SORT DONE *******
5425             ;
5425 200955      SDON    JSR RSZP           ;RESTORE ZERO PAGE
5428 60                  RTS
5429 4C8E53      JMPJ    JMP CONJ
542C 4C5953      JMPI    JMP CONI
542F 18          NCHH    CLC                ;NOT A CHARACTER SORT SO
5430 6A                  ROR                ;IT MUST BE INTEGER OR F. P.
5431 B003                BCS INTC           ;IT'S INTEGER
5433 4C6D54              JMP FPCC           ;IT'S FLOATING POINT
5436             ;
5436             ;***** INTEGER SORT ******
5436             ;
5436 A001        INTC    LDY #$01
5438 B1D6                LDA (ASII),Y       ;ASCENDING ORDER ONLY
543A D1DA                CMP (ASI2),Y
543C 88                  DEY                ;COMPARE IN%(I) WITH IN%(J)
543D B1D6                LDA (ASII),Y
543F F1DA                SBC (ASI2),Y
5441 9022                BCC NCSP           ;POSSIBLE SWAP
5443 B1D6                LDA (ASII),Y
5445 51DA                EOR (ASI2),Y
5447 30BC                BMI MC40
5449             ;
5449             ;**** SWAP I WITH J *******
5449             ;
5449 C8          SWIN    INY
544A B1DA                LDA (ASI2),Y
544C 48                  PHA
544D 88                  DEY
544E B1DA                LDA (ASI2),Y       ;SWAP IN%(I) WITH IN%(J)
5450 48                  PHA
5451 B1D6                LDA (ASII),Y
5453 91DA                STA (ASI2),Y
5455 C8                  INY
5456 B1D6                LDA (ASII),Y
5458 91DA                STA (ASI2),Y
545A 88                  DEY
545B 68                  PLA
545C 91D6                STA (ASII),Y
545E C8                  INY
545F 68                  PLA
5460 91D6                STA (ASII),Y
5462 4C0554              JMP MC40           ;NEXT RECORD
5465 B1D6        NCSP    LDA (ASII),Y
5467 51DA                EOR (ASI2),Y
5469 30DE                BMI SWIN           ;SWAP
546B 1098                BPL MC40
546D             ;
546D             ; **** FLOATING POINT SORT ****
546D             ;
546D A000        FPCC    LDY #$00
546F B1D6        FP01    LDA (ASII),Y
5471 D1DA                CMP (ASI2),Y
5473 900B                BCC MBSP
5475 F002                BEQ FP02
5477 B01D                BCS FPSP           ;THIS BIT OF CONVOLUTED
5479 C8          FP02    INY                ;LOGIC TELLS ME IF
547A C005                CPY #$05           ;FP(I) IS GREATER THAN,
547C D0F1                BNE FP01           ;EQUAL TO, OR LESS THAN
547E F03E                BEQ JM40           ;FP(J).
```

```
5480 A001     422  MBSP  LDY #$01
5482 B1D6     423        LDA (ASI1),Y          ;A TRUTH TABLE HELPS
5484 31DA     424        AND (ASI2),Y
5486 11DA     425        ORA (ASI2),Y
5488 3020     426        BMI FP03
548A 88       427        DEY
548B B1DA     428        LDA (ASI2),Y
548D D02F     429        BNE JM40
548F C8       430        INY
5490 B1D6     431        LDA (ASI1),Y
5492 1016     432        BPL FP03
5494 3028     433        BMI JM40
5496 A001     434  FPSP  LDY #$01
5498 B1D6     435        LDA (ASI1),Y
549A 31DA     436        AND (ASI2),Y
549C 11D6     437        ORA (ASI1),Y
549E 301E     438        BMI JM40
54A0 88       439        DEY
54A1 B1D6     440        LDA (ASI1),Y
54A3 D005     441        BNE FP03
54A5 C8       442        INY
54A6 B1DA     443        LDA (ASI2),Y
54A8 1014     444        BPL JM40
54AA A004     445  FP03  LDY #$04
54AC B1D6     446  FP04  LDA (ASI1),Y          ;SAVE FP(I) IN STACK
54AE 48       447        PHA
54AF 88       448        DEY
54B0 10FA     449        BPL FP04
54B2 C8       450  FP08  INY
54B3 B1DA     451        LDA (ASI2),Y
54B5 91D6     452        STA (ASI1),Y          ;SWAP
54B7 68       453        PLA
54B8 91DA     454        STA (ASI2),Y
54BA C004     455        CPY #$04
54BC D0F4     456        BNE FP08
54BE 4C0554   457  JM40  JMP MC40              ;NEXT RECORD
54C1 A000     458  SWAP  LDY #$00
54C3 B1D6     459        LDA (ASI1),Y
54C5 48       460        PHA                   ;ROUTINE TO SWAP THE
54C6 C8       461        INY
54C7 A5D8     462        LDA CSI1              ;CHARACTER POINTERS FOR
54C9 91DA     463        STA (ASI2),Y
54CB C8       464        INY                   ;CHARACTER SORT.
54CC A5D9     465        LDA CSI1+01
54CE 91DA     466        STA (ASI2),Y
54D0 A5DD     467        LDA CSI2+01
54D2 91D6     468        STA (ASI1),Y
54D4 85D9     469        STA CSI1+01
54D6 88       470        DEY
54D7 A5DC     471        LDA CSI2
54D9 91D6     472        STA (ASI1),Y
54DB 85D8     473        STA CSI1
54DD 88       474        DEY
54DE B1DA     475        LDA (ASI2),Y
54E0 91D6     476        STA (ASI1),Y
54E2 68       477        PLA
54E3 91DA     478        STA (ASI2),Y
54E5 60       479        RTS
54E6 A200     480  SVZP  LDX #$00              ;SAVE SOME OF APPLESOFT'S
54E8 B5D0     481  MC51  LDA NAPT,X            ;ZERO PAGE. SORT ROUTINE
54EA 9D4855   482        STA ZPSV,X            ;NEEDS SOME ROOM TO WORK.
54ED E8       483        INX
54EE E022     484        CPX 22
54F0 D0F6     485        BNE MC51
54F2 A56B     486        LDA ZZ6B              ;ALSO $6B.6C
54F4 8D7055   487        STA SV6B
54F7 A56C     488        LDA ZZ6B+01
54F9 8D7155   489        STA SV6B+01
54FC A200     490        LDX #$00
54FE B550     491  MC55  LDA ZZ50,X            ;ALSO $50.55
5500 9D6A55   492        STA SV50,X
5503 E8       493        INX
5504 E006     494        CPX #$06
5506 D0F6     495        BNE MC55
5508 60       496        RTS
```

```
5509 A200     497  RSZP  LDX #$00          ;RESTORE ZERO PAGE DATA
550B BD4855   498  MC61  LDA ZPSV,X
550E 95D0     499        STA NAPT,X
5510 E8       500        INX
5511 EC22     501        CPX 22
5513 D0F6     502        BNE MC61
5515 AD7055   503        LDA SV6B
5518 856B     504        STA ZZ6B
551A AD7155   505        LDA SV6B+01
551D 856C     506        STA ZZ6B+01
551F A200     507        LDX #$00
5521 BD6A55   508  MC65  LDA SV50,X
5524 9550     509        STA ZZ50,X
5526 E8       510        INX
5527 E006     511        CPX #$06
5529 D0F6     512        BNE MC65
552B 60       513        RTS
552C          514  ;
552C 535254   515  SRTS  ASC 'SRT#('
552F 2328
5531 8D       516  MSG1  HEX 8D
5532 564152   517        ASC 'VARIABLE'
5535 494142
5538 4C45
553A 202020   518  VARI  HEX 2020202020
553D 2020
553F 4E4F54   519        ASC 'NCT FOUND'
5542 20464F
5545 554E44
5548 000000   520  ZPSV  HEX 0000000000000000
554B 000000
554E 0000
5550 000000   521        HEX 0000000000000000
5553 000000
5556 0000
5558 000000   522        HEX 0000000000000000
555B 000000
555E 0000
5560 000000   523        HEX 0000000000000000
5563 000000
5566 0000
5568 0000     524        HEX 0000
556A 000000   525  SV50  HEX 000000000000
556D 000000
5570 0000     526  SV6B  HEX 0000
5572 000000   527  NAME  HEX 0000000000000000
5575 000000
5578 0000
557A 000000   528        HEX 0000000000000000
557D 000000
5580 0000
5582 000000   529  UPDN  HEX 0000000000
5585 0000
5587 00       530  INDS  HEX 00
5588 00       531  PRSN  HEX 00
5589 00       532  SAVY  HEX 00
             533        END

***** END OF ASSEMBLY
```

```
          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5 *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

NAPT   00D0  NMS1   00D4  ASII   00D6  CSII   00D8  ASI2   00DA  CSI2   00DC
IIII   00DE  NNNN   00E0  FSTR   00E2  FLEN   00E7  DISP   00EC  JJJJ   00ED
LENI   00EF  LENJ   00F0  TYPE   00F1  ZZ50   0050  ZZ6B   006B  CHRG   00B1


** ABSOLUTE VARABLES/LABELS

GETB   E6F8  SNER   DEC9  FRNM   DD67  GETA   E752  MPLY   558A  COUT   FDED
SORT   5200  SR01   5207  SR04   5218  VNAM   521B  SR05   5229  INTE   5235
INT1   5237  FP00   524A  ERRX   5252  CHAR   5255  SR06   5263  ERR3   5296
SR11   5298  ERRO   52A5  SR16   52AB  SR17   52B0  SR07   52D4  SR09   52D6
LAST   52EA  TERM   52ED  MC20   52F2  MC22   5303  SR27   5320  SR28   5322
SETN   532E  CONI   5359  SR22   5366  CONJ   538E  SR24   539B  CHST   53AD
SR29   53BD  MC33   53BF  ASND   53D0  MC25   53D8  MC26   53DE  MC39   53EB
MC27   53F1  MC28   53FA  MC29   53FF  MC40   5405  MC38   540B  MC41   541B
SDON   5425  JMPJ   5429  JMPI   542C  NCHH   542F  INTC   5436  SWIN   5449
NOSP   5465  FPCC   546D  FP01   546F  FP02   5479  MBSP   5480  FPSP   5496
FP03   54AA  FP04   54AC  FP08   54B2  JM40   54BE  SWAP   54C1  SVZP   54E6
MC51   54E8  MC55   54FE  RSZP   5509  MC61   550B  MC65   5521  SRTS   552C
MSG1   5531  VARI   553A  ZPSV   5548  SV50   556A  SV6B   5570  NAME   5572
UPDN   5582  INDS   5587  PRSN   5588  SAVY   5589

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0332
```

Apple II
Trace List Utility

by Alan G. Hill

The Integer BASIC trace command provides useful information for program debugging. But the format in which this information is presented (a barrage of line numbers) is not terribly pleasant or easy to use. This utility enhances the trace command's capabilities by providing a more legible output format and a capability for saving line numbers on longer tracings.

Did you ever use the TRACE function in Integer BASIC, only to give up in despair after looking at a screen full of line numbers? Try it without a printer and you may never use TRACE again! Here's the utility that will put TRACE back into your debugging repertoire (for those of us who need a little help getting it right).

The utility presented here will list each BASIC program source statement line by line in the order executed. There's no need to refer back and forth between TRACE line numbers and the source program listing. Two versions are presented: Version 1 is a real-time utility; i.e. each statement is listed immediately prior to execution so you can follow the program's logical sequence. You can slow the execution rate down or even temporarily halt execution while you scan the screen. Version 2 only saves the line numbers of the last 100 lines executed for listing later. Version 2 could be useful in tracing a full-screen graphics program.

The Technique

The program utilizes the DOS COUT hook at $AA53, $AA54 to intercept and suppress TRACE printing. All other printing continues normally with one exception (see Warning #1). Before returning to the BASIC interpreter, the line number is picked up and pushed into an array (TR) in the variables area above LOMEM. If the number is the same as the previous line number, a zero line number is placed in the stack with the line number of a FOR I = 1 to 1000: NEXT I delay loop, for instance. When the number changes, it will be placed in the stack. The most recent 100 line numbers are saved. Tracing is performed under user control by the normal TRACE/NOTRACE statements. In Version 2, the lines may then be listed after the test program ends. The technique in Version 1 is similar with one distinction. The trace intercept routine transfers control to the utility program to list the line as soon as it is put in the stack.

How the TRACE Intercept Routine Works

The output pointer in $AA53, $AA54 is initialized by the utility to the address ($300) of the Trace Intercept Routine. Each character is examined by TIR as it comes through if the TRACE flag is up (bit 7 of $A0 on). If off, TIR jumps back to the normal print utility at $FDF0. If the character is a # ($A3), it is assumed that a line number follows. Every line number in the stack is pushed down and the current line number is placed at the top. Location $DC,DD points to the BASIC line about to be executed. The line number is in the second and third bytes. In Version 2, TIR returns to the interpreter. In the real-time version (Version 1), control is next transferred to the utility program at line 30020. TIR expects that the address of line 30010 has been saved in $15,16 by the utility programs CALL 945 in line 30010. TIR first saves the contents of $DC,DD and then replaces it with the contents of $15,16. It also saves the address of the current statement within the BASIC line. That is, the contents of $E0,E1 are saved at $1B,1C. TIR can now transfer control back to the interpreter's continue entry point by a JMP $E88A which then executes line 30020 of the utility. The current line of the test program is listed; the BASIC pointers are restored by the CALL 954 in line 30090; the return address is popped; and control is returned to the test program through $E881. Fait accompli.

As mentioned previously, the TR array is used to save the line numbers. The array is set up the first time TIR is entered. Note that TR is intentionally not DIMensioned in the utility. TIR must handle that task since a RUN of the test program will reset the variables area pointer ($CC,CD) back to LOMEM.

Programming the Routines

TIR starts at $300. It could be relocated if the absolute references in the POKE and CALL statements are changed. Also note that the LIST statement in lines 30060 and 32040 will not be accepted by the Syntax checker. They must first be coded as PRINT statements, located, and changed to LIST tokens ($74) using the monitor. This is more easily done if these lines are coded and the tokens changed before the remaining lines are entered. See example below for the case where HIMEM is 32768:

```
NEW
30060 PRINT EXECLINE
32040 PRINT TR (I)
CALL – 151 (to enter Monitor)
*7FEC:74
*7FF9:74
(enter Control/C)
LIST
30060 LIST EXECLINE
32040 LIST TR (I)
```

Using the Utility

1. After coding the assembler and BASIC utility programs, the test program is then appended.

2. Create a line 0 that will be used to indicate that a line has successively executed. For example, code:

```
0 REM ***ABOVE LINE REPEATED***
```

3. **Run** the utility of your choice:

RUN 30000 Version 1 (Real-time list)
or RUN 32000 Version 2 (Post-execution list)

4. Insert the TRACE/NOTRACE statements wherever desired in test program. Just enter the TRACE command directly if you want to trace the entire program. Also see Warning #1.

5. RUN the test program.

6. Display the results:

   A. **Real-time Version:** The lines will be listed automatically as executed. Note the FOR:NEXT loop in line 30090 can be adjusted to control the execution rate. The upper limit could be PDL(0), thereby giving you run-time control over the execution rate. Note also that execution can be forced to pause by depressing paddle switch 0. Execution will resume when the switch is released.

   B. **Post-execution Version:** After stopping or ending the program, enter a GOTO 32020 command. The first page of statements will be displayed. Enter a "C" to display additional pages, a "T" to reset for another test run, or an "E" to return to BASIC. Note that even if you have traced with Version 1, you can still display the last 100 lines with Version 2.

Sample Run

Test Program

```
  0 REM *** REPEATED ***
 10 TRACE
 30 GOSUB 100+RND(3) *10
 40 FOR I=1 TO 10: NEXT I
 50 GOTO 30
100 PRINT "LINE 100":RETURN
110 PRINT "LINE 110":RETURN
```

```
120 PRINT "LINE 120":POP
125 NO TRACE:END
  > RUN 30000
  > RUN
```

Trace Output

```
 30 GOSUB 100+RND(3)*10
110 PRINT "LINE 110":RETURN LINE 110
 30 GOSUB 100+RND(3)*10
 40 FOR I=1 TO 10:NEXT I
  0 REM *** REPEATED ***
 50 GOTO 30
 30 GOSUB 100+RND(3)*10
120 PRINT "LINE 120":POP LINE 120
125 NO TRACE:END
  >
```

For a slow motion game of "BREAKOUT", trace it with the real-time version!

Hints and Warnings

It's usually a good idea to deactivate TIR after the test program has ended by hitting Reset and Control-C and entering NOTRACE. Don't try to trace the test program without first running the utility program at line 30000 or 32000.

To increase the debugging power of the real-time trace utility, make liberal use of the push button to halt program execution. With practice and the proper choice of the delay loop limit in line 30090, you can step through the program one line at a time. Enter a Control-C while the push button is depressed and execution will be STOPPED AT 30070. You can then use the direct BASIC commands to PRINT and change the current value of the program's variables. Enter CON and execution will resume. The game paddles must be installed for the program to work correctly.

With additional logic in the utility program, you can create specialized tracing such as stopping after a specified sequence of statements has been detected. Return via a CALL 958 if you don't want TRACE turned back on.

Tracing understandably slows the execution rate of your program, but you probably aren't concerned with speed at this point. However, the wise use of TRACE/NOTRACE will help move things along. Also, when encountering a delay loop such as FOR I = 1 to 3000: NEXT I, you may want to help it along by stopping with a Control-C entering I = 2999, and CONtinuing.

Warning #1: There must be **no** PRINT statement with a # character in the output. TIR assumes that a # is the beginning of a trace sequence. Either remove the # or bracket the PRINT statement with a NOTRACE/TRACE pair.

Warning #2: There must be no variable names in the test program identical to those in Version 1. The TR variable name must be unique in both versions.

Warning #3: Line 0 in the test program should be a REMark statement as described above to avoid confusion. Line 0 is listed when a line is successively repeated.

Warning #4: Once TRACE has been enabled, the test program must not dynamically reset the variables pointer ($CC,CD) with a CLR or POKE unless it first disables TRACE and resets $13,14; e.g., 100 NOTRACE:CLR: POKE 19, 0: POKE 20,0: TRACE is OK.

Extensions

The primary motivation for this program was to improve the TRACE function in Integer BASIC. However, you can imagine other uses of a program that gains control as each statement is executed—maybe the kernel of a multiprogramming executive.

```
29970 REM  ************************
29971 REM  *                      *
29972 REM  *  TRACE LIST UTILITY  *
29973 REM  *    BY ALAN G. HILL   *
29974 REM  *                      *
29975 REM  *      TRACE LIST      *
29976 REM  *                      *
29977 REM  *  COPYRIGHT (C) 1981  *
29978 REM  *    MICRO INK, INC.   *
29979 REM  * CHELMSFORD, MA 01824 *
29980 REM  *  ALL RIGHTS RESERVED *
29981 REM  *                      *
29982 REM  ************************
29983 REM
29984 REM
29985 PRINT : PRINT "'RUN 31000' APPEND": PRINT "'RUN 30000' REAL-TIME LIST"
      : PRINT "'RUN 32000' POST-EXEC SETUP"
29986 PRINT "'GOTO 32020' POST-EXEC LIST": VTAB 20: INPUT "'RETURN' WHEN READY
       TO APPEND",A$
29995 GOTO 31000
29998 REM  'RUN 30000' REAL-TIME
30000 NOTRACE : POKE 54,768 MOD 256: POKE 55,768/256: POKE 19,0: POKE 20,
      0: POKE 787,76: POKE 788,211: POKE 789,3: POKE 790,234: CALL -22447

30004 PRINT "ENABLE TRACE IN YOUR PROGRAM": PRINT "AND 'RUN'."
30005 REM  TRACE VER1.0 11-28-78
30006 REM  TRACE VER1.1 3-6-79
30007 REM  ADD DISK APPEND CAPABILITY
30010 CALL 945: END
30020 EXECLINE=TR(0): IF EXECLINE#0 THEN 30050
30030 IF RRRRR=1 THEN 30070
30040 RRRRR=1: GOTO 30060
30050 RRRRR=0
30060 LIST EXECLINE
30070 IF PEEK (-16287)>127 THEN 30070
30075 IF EXECLINE=0 THEN 30090
30080 FOR JJJJJ=1 TO 150: NEXT JJJJJ
30090 CALL 954: REM   BACK TO TEST PGM
30100 END
31000 DIM A$(30)
31001 VTAB 24
31002 INPUT "APPEND ",A$
31005 IF A$#"" THEN 31030
31010 POKE 0, PEEK (76): POKE 1, PEEK (77): POKE 76, PEEK (202): POKE 77,
       PEEK (203): CALL -3873: POKE 76, PEEK (0): POKE 77, PEEK (1): END
```

```
31030 POKE 0, PEEK (76): POKE 1, PEEK (77): POKE 76, PEEK (202): POKE 77,
       PEEK (203): PRINT "LOAD ";A$;",V": POKE 76, PEEK (0): POKE 77, PEEK
      (1)
31031 PRINT "'RUN 30000' REAL-TIME": PRINT "'RUN 32000' POST TIME": END
31999 REM  'RUN 32000' POST-EXEC
32000 POKE 54,768 MOD 256: POKE 55,768/256: POKE 19,0: POKE 20,0: POKE 787
      ,169: POKE 788,127: POKE 789,133: POKE 790,5: CALL -22447
32010 PRINT "TRACE SET UP. ENABLE TRACE IN YOUR PGM": END
32020 NOTRACE : POKE 54,240: POKE 55,253: IF PEEK (20)#0 THEN 32030: PRINT
      "TRACE NOT ON IN YOUR PGM": GOTO 32090
32030 CALL -936: FOR I=100 TO 1 STEP -1: IF TR(I)=-1 THEN 32060
32040 LIST TR(I)
32050 IF PEEK (37)>18 THEN 32090
32060 NEXT I
32070 GOTO 32090
32080 CALL -936: IF I>1 THEN 32060
32090 PRINT : PRINT "C/T/E ?"
32100 KEY= PEEK (-16384): IF KEY<128 THEN 32100: POKE -16368,0: IF KEY=212
       THEN 32000: IF KEY=195 THEN 32080: END
```

Editor's Note: The main listing was omitted from the text due to space limitations. The machine language program appears on the disk as TRACE INTERRUPT.

4
GRAPHICS AND GAMES

INTRODUCTION

No book on the Apple would be complete without a chapter exploring the recreational capabilities of the machine. The two features of the Apple which have exhibited the most recreational potential are the graphics and sound generation. This section includes programs which utilize both these capabilities, and additionally includes a fun space maze game!

David Allen's ''A Versatile Hi-Res Function Plotter'' uses high-resolution graphics to plot curves for any user-defined function. ''Apple II Hi-Res Picture Compression,'' by Bob Bishop, allows the user to compress any image on the graphics screen by taking advantage of redundancy. The discussion of the pixel technique used is very revealing. ''An Apple Flavored Lifesaver,'' by Greg Tibbetts, is a version of the popular ''Life'' simulation which allows pattern storage on disk.

''Applayer Music Interpreter,'' by Dick Suitor, implements a sophisticated music generation system for the Apple using no additional hardware. Several sample tunes are provided, as are the necessary instructions for generating music of your own. William Shryock's ''Improved Star Battle Sound Effects'' is another tonemaking routine. Though much shorter than the previous one, it has nonetheless provided hours of amusement to many.

Finally, the space-maze game entry in this chapter is ''Galacti-Cube'' by Bob Bishop. Written in Integer BASIC, the game challenges you to find the exit to the 'giant cube' floating through space!

A Versatile Hi-Res Function Plotter

by David P. Allen

One of the obvious uses for Apple Hi-Res capability is to plot various mathematical functions. The program presented here is very general purpose and permits the user to simply plot any expression as a function of angle from 1 to 360 degrees. A modification is included which will permit the program to be used on an Atari as well.

A few years ago when scientific calculators first made their appearance, I was enchanted by the ease with which calculations using transcendental functions could be accomplished. This prompted me to dust off the old trigonometry book and delve into some basics through which I had once passed somewhat painfully. Maybe pain isn't the word. Probably boredom and drudgery would be better words. Log and function tables are probably the only documents with less magnetism than the Little Rock telephone book. I expect that many a budding mathematics curiosity has atrophied over the dryness of log tables.

With the power and freedom of this nifty calculator at hand I suddenly found myself unfettered by the yoke of boredom and I swiftly recovered much of my early curiosity by travelling quickly through basic trigonometry. Gone were the stumbling blocks of look-up tables and I was able to move down many diversionary "what if's" to see what really happens when certain values change in mathematical formulae.

But as exciting as all this was, and because much of mathematics requires visual images, I looked forward to a time when, with the help of a small computer, I could generate graphs and figures as well as numbers to excite and satisfy my curiosity.

And so it was that after acquiring an Apple II computer, one of my first exercises was to develop a program which would use Apple's excellent high-resolution graphics to plot the path of a variety of mathematical expressions. This program is the result and I have had much, much fun with it.

The program was developed on an Apple II with 48K of RAM and an Applesoft ROM card. The entire program takes only slightly more than 3K of RAM, depending on the complexity of the function being plotted.

Those who do not have the Applesoft ROM card may still use this program by changing line 480 to read "HGR2" instead of "HGR". Under these circumstances the function plotted formula will not be printed at the bottom of the screen. All other functions work as described.

The heart of the program is line 1010 which contains the function being explored. A typical function is listed here. When run, the program first defines some trigonometric and hyperbolic functions which are not directly available in Applesoft BASIC. It then proceeds to plot the *X* and *Y* axes. As currently arranged, the expression under investigation is plotted as a function of changing angle, from 1 to 360 degrees. By changing lines 670 and 900, other independent variables could be introduced. The program is completely protected against off-scale plotting and automatically scales itself for the range of independent variables selected.

When the plot is completed the program dutifully presents a print-out of the function and awaits your pleasure at the push of the return key. It then presents you with a helpful list of all of the additional functions defined by the program in addition to those resident in Applesoft BASIC. Line 1010 is listed and the cursor invites your screen editing of this line for further variations.

A word of caution: any attempt to plot mathematical "no-no's" like square roots or logs of negative values will earn you a quick error message. Do not despair. Use of the ABS command will quickly get you back in business when these values crop up!

This program has all kinds of tinkering possibilities. You might try surrounding line 1010 with a FOR...NEXT loop to introduce other variable changes and to allow longer expressions than you can conveniently type into line 1010 all at once. Just beware! This program is subtly laced with a curious narcotic which has been known to keep the user awake all night! Have fun!

```
10  REM  ************************
12  REM  *                      *
14  REM  *   FUNCTION PLOTTER   *
16  REM  *    DAVID P. ALLEN    *
18  REM  *                      *
20  REM  *       FNPLOTTER      *
22  REM  *                      *
24  REM  *  COPYRIGHT (C) 1981  *
26  REM  *   MICRO INK, INC.    *
28  REM  * CHELMSFORD, MA 01824 *
30  REM  *  ALL RIGHTS RESERVED *
32  REM  *                      *
34  REM  ************************
140  REM
150  REM
180  REM  THIS PROGRAM PLOTS A
190  REM  CURVE FOR ANY EXPRESSION
200  REM  AS A FUNCTION OF INCREAS-
210  REM  ING ANGLE FROM 1 TO 360
220  REM  DEGREES.
230  REM  CHANGE LINE 1010 TO A
240  REM  FUNCTION YOU WISH TO
250  REM  PLOT.
260  REM
270  REM
280  REM  *** DEFINE FUNCTIONS ***
290  REM
300  DEF  FN SCH(X) = 2 / ( EXP (X) +  EXP ( - X)): REM  SECH(X)
310  DEF  FN CCH(X) = 2 / ( EXP (X) -  EXP ( - X)): REM  CSCH (X)
320  DEF  FN CTH(X) =  EXP ( - X) / ( EXP (X) -  EXP ( - X)) * 2 + 1:
      REM COTH(X)
330  DEF  FN SEC(X) = 1 /  COS (X): DEF  FN CSC(X) = 1 /  SIN (X): DEF
     FN COT(X) = 1 /  TAN (X)
340  DEF  FN SNH(X) = ( EXP (X) -  EXP ( - X)) / 2: REM  SINH(X)
350  DEF  FN COH(X) = ( EXP (X) +  EXP ( - X)) / 2: REM  COSH(X)
360  DEF  FN TAH(X) =  -  EXP ( - X) / ( EXP (X) +  EXP ( - X)) * 2 + 1:
      REMTANH(X)
370  REM
380  REM
390  REM  ** PLOT GRAPH AXES **
400  REM
410  HOME
420  REM
430  REM  MOVE CURSOR TO BOTTOM
440  REM  LINE.
450  REM
460  VTAB 24
470  REM
480  HGR
490  HCOLOR= 7
500  HPLOT 0,80 TO 279,80
510  HPLOT 0,16 TO 0,143
520  FOR I = 0 TO 279 STEP 70
530  HPLOT I,78 TO I,82: HPLOT 279,78 TO 279,82
540  NEXT I
550  FOR I = 16 TO 144 STEP 16
560  HPLOT 0,I TO 4,I
570  NEXT I
580  REM
590  REM  FLAGS FOR FIRST PLOT
600  REM  AND SCALE.
610  REM
620 F = 0:G = 0
630  REM
640  REM  R1 AND R2 MAY BE SET
650  REM  FOR OTHER LIMITS.
660  REM
670 R1 = 1:R2 = 360
680  REM
```

```
690  REM
700  REM  ** BEGIN PLOT **
710  REM
720  REM  CHANGE STEP FOR MORE
730  REM  OR LESS RESOLUTION.
740  REM  IF R1>R2 THEN STEP
750  REM  MUST BE NEGATIVE.
760  REM
770  FOR I = R1 TO R2 STEP 5
780  REM
790  REM  NEXT 3 STEPS ESTABLISH
800  REM  HORIZONTAL SCALE.
810  REM
820  IF  ABS (R1) >  =  ABS (R2) THEN R =  ABS (R1)
830  IF  ABS (R2) >  =  ABS (R1) THEN R =  ABS (R2)
840  IF G = 0 THEN S = 70 * 4 / R:G = 1
850 X = I:Y = 0
860  REM
870  REM  CONVERTS DEGREES TO
880  REM  RADIANS.
890  REM
900 X = X * 3.14159 / 180
910  REM
920  REM  PREVENTS CRASHING WHEN
930  REM  X=0.
940  REM
950  IF X = 0 THEN X = .00001
960  REM
970  REM
980  REM  NEXT LINE DESCRIBES
990  REM  FUNCTION TO BE PLOTTED
1000  REM
1010 Y1 =  SIN (X) +  COS (2 * X)
1020 Y = Y + Y1
1030 Y = Y * 20
1040  REM
1050  REM  SCALES X
1060  REM
1070 X = I * S
1080  REM
1090  REM  RELATES PLOT TO X AXIS
1100  REM
1110 Y =  - Y + 80
1120  REM
1130  REM  SUBROUTINE PREVENTS
1140  REM  OFF-SCALE CRASHING.
1150  REM
1160  GOSUB 1830
1170  REM
1180  REM  PLOTS FIRST POINT.
1190  REM
1200  IF F = 0 THEN  HPLOT X,Y:F = 1
1210  HPLOT  TO X,Y
1220  NEXT I
1230  PRINT : LIST 1010
1240  REM
1250  REM  BLANKS OUT LINE #
1260  REM  AFTER LISTING
1270  REM  LINE 1010.
1280  REM
1290  POKE 1616,160: POKE 1617,160: POKE 1618,160: POKE 1619,160
1300  REM
1310  REM  WAITING FOR YOUR PLEASURE!
1320  REM  PUNCH 'RETURN'
1330  REM  TO CONTINUE!
1340  REM
1350  POKE  - 16368,0: WAIT  - 16384,128
1360  REM
1370  REM
```

```
1380  REM  THROWS PREVIOUS KEYSTROKE
1390  REM  AWAY WITH
1400  REM  'GET Z$'!
1410  REM
1420  GET Z$
1430  REM
1440  REM  CLEAR SCREEN AND
1450  REM  PRINT FUNCTIONS FOR
1460  REM  REMINDER.
1470  REM
1480  TEXT : HOME
1490  PRINT  TAB( 9);"SECANT = FN SEC(X)"
1500  PRINT  TAB( 9);"COSEC = FN CSC(X)"
1510  PRINT  TAB( 9);"COTAN = FN COTAN(X)"
1520  PRINT  TAB( 9);"SINH = FN SNH(X)"
1530  PRINT  TAB( 9);"COSH = FN COH(X)"
1540  PRINT  TAB( 9);"TANH = FN TAH(X)"
1550  PRINT  TAB( 9);"SECH = FN SCH(X)"
1560  PRINT  TAB( 9);"CSCH = FN CCH(X)"
1570  PRINT  TAB( 9);"COTH = FN CTH(X)"
1580  REM
1590  REM  NOW WE SET UP LINE
1600  REM  1010 FOR EDITING.
1610  REM  'POKE 32, 2' MOVES
1620  REM  MARGIN SO CURSOR CAN
1630  REM  FIT IN FRONT.
1640  REM
1650  VTAB (12)
1660  PRINT " CHANGE LINE 1010 AS DESIRED AND"
1670  PRINT "RUN AGAIN!"
1680  POKE 32,2
1690  LIST 1010
1700  REM
1710  REM  NOW WE RESTORE MARGIN
1720  REM  AND MOVE CURSOR IN
1730  REM  FRONT OF LINE #.
1740  REM
1750  POKE 32,0
1760  POKE 37,13: POKE 36,0
1770  REM
1780  END
1790  REM
1800  REM  SCALE ANTI-CRASHING
1810  REM  SUBROUTINE.
1820  REM
1830  IF X < 0 THEN X = 0
1840  IF X > 279 THEN X = 279
1850  IF Y < 0 THEN Y = 0
1860  IF Y > 159 THEN Y = 159
1870  RETURN
```

Apple II Hi-Res Picture Compression

by Bob Bishop

Every Apple owner is aware of the wonderful pictures that can be made with Hi-Res graphics. An interesting technique is presented which allows greater efficiency in encoding picture information, and produces additional special effects.

Almost every Apple II owner has, by now, seen examples of how the Apple II can display digitized photographs in its Hi-Res graphics mode. These images consist of 192 × 280 arrays of dots all of the same intensity. By clustering these dots into groups (such as in "dithering"), it is even possible to produce pictures having the appearance of shades of gray. Several "slide shows" of these kinds of pictures have been created by both Bill Atkinson and myself and are available through various sources, such as the Apple Software Bank. A typical "slide show" consists of about 11 pictures on a standard 13-sector disk.

Each Hi-Res picture must reside in one of the two Hi-Res display areas before it can be seen. The first area, \$2000-\$3FFF, is called the *primary* display buffer; the second area, \$4000-\$5FFF, is called the *secondary* display buffer. It is obvious that each of these display areas are 8K bytes long. Consequently, Hi-Res pictures are usually stored as 8K blocks of data, exactly as they appear in a display buffer. But do they have to be stored that way?

If you look closely at a Hi-Res picture, you can almost always detect small regions that look very similar to other small regions elsewhere in the picture. For example, Hi-Res displays usually contain regions of pure white or pure black. In the case of dithered pictures, the illusion of gray may be caused by micro-patterns of dots that are similar to other gray patterns somewhere else. Clearly, Hi-Res pictures tend to contain a lot of redundancy. If there were some way of removing this redundancy then it would be possible to store Hi-Res pictures in less than the customary 8K bytes of memory.

Suppose we were to divide the display into small rectangular clusters, each 7 bits wide, by 8 bits high. Then a picture would consist of 24 rows of these picture elements ("pixels"), with 40 of them per row. (Note the resemblance to the Apple

II's TEXT mode of 24 lines, 40 columns per line!) The total number of pixels that would be needed to define a Hi-Res picture would then be 40 times 24, or 960. However, not all 960 pixels would be unique if there were redundancy in this picture.

To try out these ideas, I used Atkinson's LADY BE GOOD picture (from the Apple Magic Lantern—Slide Show 2) shown in figure 1, and wrote a program to extract all the different pixels. I found that only 662 of the 960 pixels were unique. This meant that almost one third of the picture was redundant!

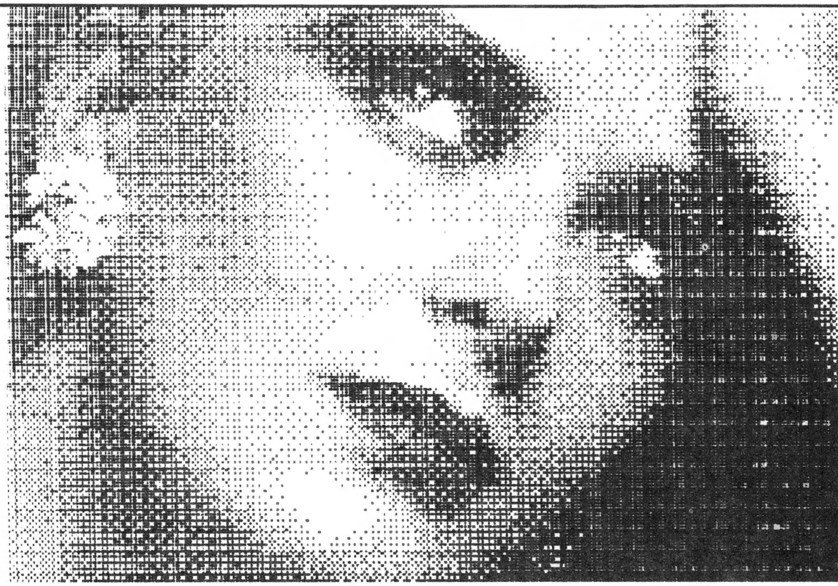
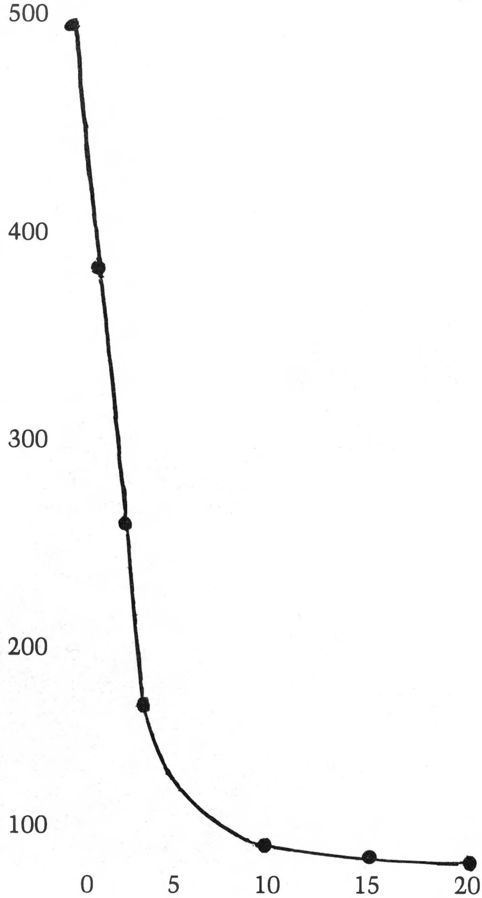


Figure 1: **(Max errors/pixel = 0)**

The next question that came to mind was: of the 662 unique pixels, how 'unique' were they? Was it possible that there might be two or more pixels that were almost the same, except for maybe one or two dots that differed? If so, then it could be possible to regard these as being identical 'for all practical purposes' since the error in the resulting picture would hardly be noticed.

To examine this possibility, I modified my program to extract only those pixels that differed by more than a specified MAX ERRORS/PIXEL. Table 1 shows the result. If we allow, at most, 1 dot to be wrong in any one pixel, then we need only 492 pixels to define the picture, which is only about half of the original 960 pixels! As we allow more and more errors per pixel, the number of pixels required to reconstruct the picture decreases accordingly, until we reach 28 errors/pixel. At this point we are allowing half of the dots to be wrong. Since total black and total white are always included in every pixel set (to prevent black or white areas from becoming dotted), pictures with MAX ERRORS/PIXEL greater than or equal to 28 can always be composed of no more than two pixels, namely the black and white pixels.



max errors per pixel	number of pixels
0	662
1	492
2	334
3	245
4	178
5	135
6	100
7	75
8	53
9	44
10	34
11	29
12	24
13	18
14	15
15	11
16	10
17	7
18	7
19	6
20	5
22	4
24	4
26	4
28	2

***Table 1:* Max Errors/Pixel**

Suppose we now try to reconstruct the original picture from our extracted pixel set. Clearly, the fewer pixels we have available for synthesizing, the poorer the result will be. Figures 2 through 5 show the results of synthesizing LADY BE GOOD with MAX ERRORS/PIXEL of 3, 7, 14, and 28. The number of pixels used in each case was 245, 75, 15, and 2, respectively. Notice that the difference in quality between figures 1 and 2 is not all that objectionable. The advantage that figure 2 has is that it can be stored in less than 3K bytes of memory! (245 pixels at 8 bytes/pixel, plus 960 bytes to define which pixels go where.)

Thus it is clearly possible to store an 8K Hi-Res picture in considerably less than 8K bytes, if you are willing to accept a little loss in the image quality. By using this principle, I have produced a "Super Slide Show" containing 33 pictures on a single disk. (Copies may be obtained from Apple's Software Bank.)

The Compression Program

Listings 1 and 2 show the compression routines (and some associated data tables), and require an Apple II with at least 32K bytes of memory. The routines consist of two basic parts—the "analysis" portion, and the "synthesis" portion.

The analysis routine ($0B00) searches the primary Hi-Res display buffer ($2000-$3FFF) and compares each pixel there with the pixels in its own current pixel table (which starts at $0600) looking for a "match". If it finds a pixel in the table that matches to within the specified MAX ERRORS/PIXEL (location $10), it calls a match and proceeds to the next pixel in the picture. If it fails to find a match, it adds the pixel to its current pixel table and then proceeds.

The synthesis routine ($0B80) works in the other direction. It first compares each pixel of the primary buffer with each pixel in the pixel table to find the best match. It then places this pixel in the corresponding location in the secondary Hi-Res buffer, thus synthesizing the best approximation to the primary picture as it can by using the pixels in its pixel table. (Since the analysis routine doesn't know where its pixel table originated, it is possible to snythesize one picture from another picture's pixels! The result is usually surprisingly good.)

The routines are very easy to use. Simply load the picture to be compressed into $2000-$3FFF, set MAX ERRORS/PIXEL into $10, and then call the routine at $0B00. When the routine returns, locations $07 and $08 contain the number of extracted pixels in the form: NUMBER = 1 + (contents of $07) + 40* (contents of $08).

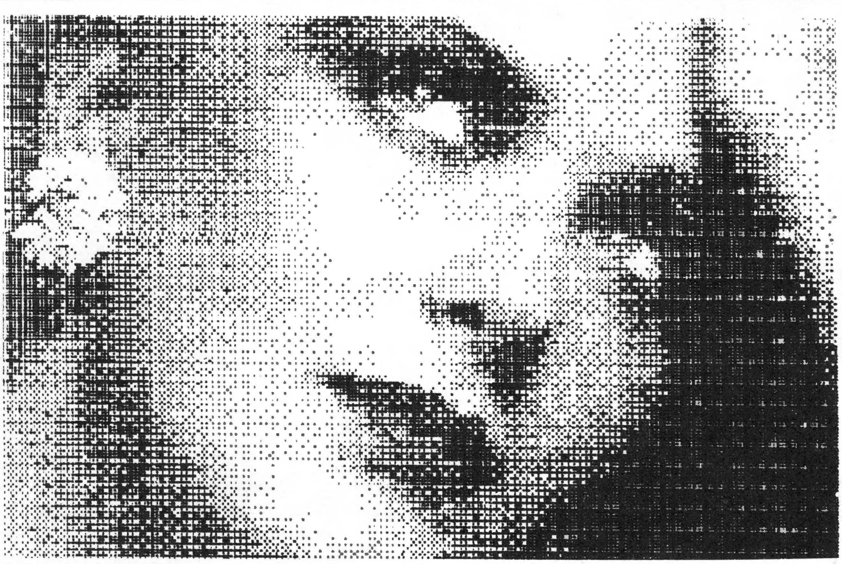


Figure 2: **(Max errors/pixel =3)**

To synthesize the picture from the extracted pixels, simply call the routine at $0B80. When the routine returns, the reconstructed picture will be in the secondary Hi-Res buffer ($4000-$5FFF).

If you have a 48K Apple and a disk, you can use the BASIC program shown in listing 3. This program calls the compression routines (listings 1 and 2) in a more user-oriented way so that they are even easier to use. The program displays a menu of options that let you:

L — Load a picture from disk into the primary Hi-Res buffer.
1 — Display the picture currently in the primary Hi-Res buffer.
2 — Display the picture currently in the secondary Hi-Res buffer.
A — Analyze the primary picture (create the pixel table).
S — Synthesize the primary picture using the current pixel table.
D — Issue disk commands.
X — Transfer the compressed picture to disk drive number 2.

None of the selections require you to hit RETURN; just hit the corresponding character. When specifying "L", the program will ask you for the name of the file to be loaded. When specifying "A", you will be asked for the maximum error per pixel that you will allow. (This does require a RETURN.) The "D" command will give a colon (:) as the prompt character and will allow you to issue disk commands. It will continue in this mode until you give it a null command (hit RETURN) at which time it will return to the menu. The "X" command saves the compressed picture (960 bytes) and its corresponding pixel table (up to 2K bytes) onto a disk file. (I will leave it up to the interested reader to figure how to "uncompress" this data.)

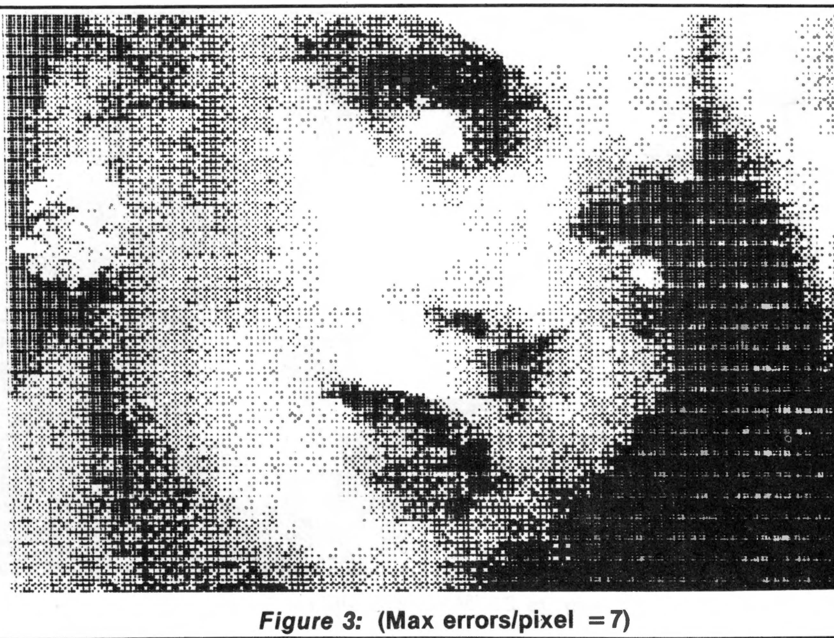


Figure 3: **(Max errors/pixel = 7)**

Figure 4: **(Max errors/pixel = 14)**

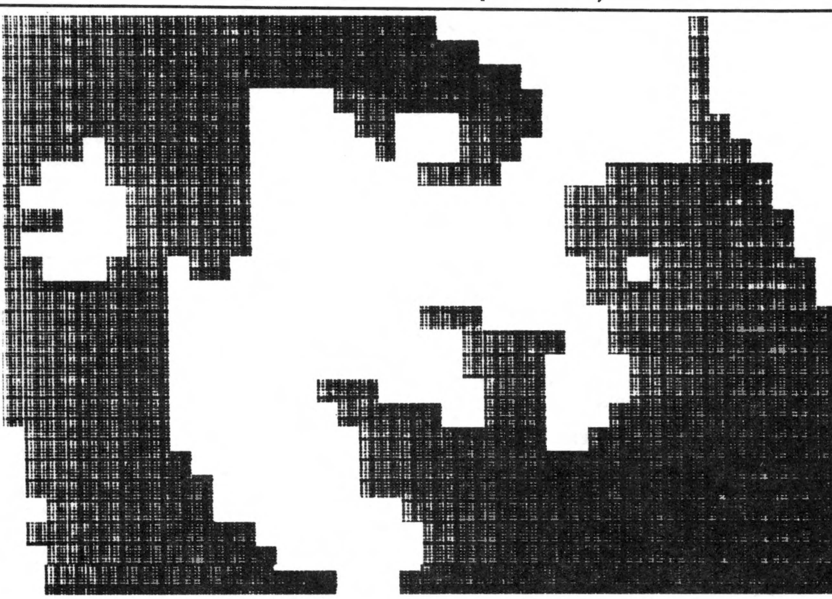


Figure 5: **(Max errors/pixel = 28)**

While the methods here work pretty well, they may not represent the optimum way of compressing Apple II picture data. For example, my choice of 7 × 8 dots/pixel was somewhat arbitrary. Is it posible to get better compression ratios by choosing smaller (or larger) pixel sizes? Or, given a picture that was reconstructed from a given set of *n* pixels, is it possible to find another set of *n* pixels that gives a better result?

```
0C00- 00 00 00 00 00 00 00 00
0C08- 80 80 80 80 80 80 80 80
0C10- 00 00 00 00 00 00 00 00
0C18- 80 80 80 80 80 80 80 80
0C20- 00 00 00 00 00 00 00 00
0C28- 80 80 80 80 80 80 80 80
0C30- 00 00 00 00 00 00 00 00
0C38- 80 80 80 80 80 80 80 80
0C40- 28 28 28 28 28 28 28 28
0C48- A8 A8 A8 A8 A8 A8 A8 A8
0C50- 28 28 28 28 28 28 28 28
0C58- A8 A8 A8 A8 A8 A8 A8 A8
0C60- 28 28 28 28 28 28 28 28
0C68- A8 A8 A8 A8 A8 A8 A8 A8
0C70- 28 28 28 28 28 28 28 28
0C78- A8 A8 A8 A8 A8 A8 A8 A8
0C80- 50 50 50 50 50 50 50 50
0C88- D0 D0 D0 D0 D0 D0 D0 D0
0C90- 50 50 50 50 50 50 50 50
0C98- D0 D0 D0 D0 D0 D0 D0 D0
0CA0- 50 50 50 50 50 50 50 50
0CA8- D0 D0 D0 D0 D0 D0 D0 D0
0CB0- 50 50 50 50 50 50 50 50
0CB8- D0 D0 D0 D0 D0 D0 D0 D0
```

```
0D00- 20 24 28 2C 30 34 38 3C
0D08- 20 24 28 2C 30 34 38 3C
0D10- 21 25 29 2D 31 35 39 3D
0D18- 21 25 29 2D 31 35 39 3D
0D20- 22 26 2A 2E 32 36 3A 3E
0D28- 22 26 2A 2E 32 36 3A 3E
0D30- 23 27 2B 2F 33 37 3B 3F
0D38- 23 27 2B 2F 33 37 3B 3F
0D40- 20 24 28 2C 30 34 38 3C
0D48- 20 24 28 2C 30 34 38 3C
0D50- 21 25 29 2D 31 35 39 3D
0D58- 21 25 29 2D 31 35 39 3D
0D60- 22 26 2A 2E 32 36 3A 3E
0D68- 22 26 2A 2E 32 36 3A 3E
0D70- 23 27 2B 2F 33 37 3B 3F
0D78- 23 27 2B 2F 33 37 3B 3F
0D80- 20 24 28 2C 30 34 38 3C
0D88- 20 24 28 2C 30 34 38 3C
0D90- 21 25 29 2D 31 35 39 3D
0D98- 21 25 29 2D 31 35 39 3D
0DA0- 22 26 2A 2E 32 36 3A 3E
0DA8- 22 26 2A 2E 32 36 3A 3E
0DB0- 23 27 2B 2F 33 37 3B 3F
0DB8- 23 27 2B 2F 33 37 3B 3F
```

```
1000- 00 01 01 02 01 02 02 03
1008- 01 02 02 03 02 03 03 04
1010- 01 02 02 03 02 03 03 04
1018- 02 03 03 04 03 04 04 05
1020- 01 02 02 03 02 03 03 04
1028- 02 03 03 04 03 04 04 05
1030- 02 03 03 04 03 04 04 05
1038- 03 04 04 05 04 05 05 06
1040- 01 02 02 03 02 03 03 04
1048- 02 03 03 04 03 04 04 05
1050- 02 03 03 04 03 04 04 05
1058- 03 04 04 05 04 05 05 06
1060- 02 03 03 04 03 04 04 05
1068- 03 04 04 05 04 05 05 06
1070- 03 04 04 05 04 05 05 06
1078- 04 05 05 06 05 06 06 07
```

```
  1 REM  *************************
  2 REM  *                       *
  3 REM  * PICTURE  COMPRESSION *
  4 REM  *    BY ROBERT BISHOP   *
  5 REM  *                       *
  6 REM  *       COMPRESS        *
  7 REM  *                       *
  8 REM  *  COPYRIGHT (C) 1981   *
  9 REM  *    MICRO INK, INC.    *
 10 REM  * CHELMSFORD, MA 01824 *
 11 REM  *  ALL RIGHTS RESERVED *
 12 REM  *                       *
 13 REM  *************************
 14 REM
 15 DIM A$(40)
 20 ANAL=11*256:SYN=ANAL+128:PRESS=4096+2*256+8*16
 30 FLAG=0:XFLAG=0
 50 PRINT "BLOAD PIXEL STUFF"
100 CALL -936: POKE -16300,0: POKE -16303,0
110 TAB 17: PRINT "M E N U"
120 TAB 17: PRINT "-------": PRINT
130 PRINT : PRINT "   L - LOAD PICTURE FROM DISK"
140 PRINT : PRINT "   A - ANALYZE PICTURE INTO PIXELS"
150 PRINT : PRINT "   S - SYNTHESIZE PICTURE FROM PIXELS"
160 PRINT : PRINT "   1 - DISPLAY ORIGINAL PICTURE"
170 PRINT : PRINT "   2 - DISPLAY SYNTHESIZED PICTURE"
```

```
 180 PRINT : PRINT "   D - ISSUE DISK COMMANDS"
 190 PRINT : PRINT "   X - SAVE COMPRESSED PICTURE TO DISK"
 195 VTAB 20: PRINT "SELECTION: "
 200 REM READ KEYBOARD
 210 CHAR= PEEK (-16384)
 220 IF CHAR<128 THEN 210
 230 POKE -16384+16,0
 300 ID=0
 310 IF CHAR= ASC("L") THEN ID=1
 320 IF CHAR= ASC("A") THEN ID=2
 330 IF CHAR= ASC("S") THEN ID=3
 340 IF CHAR= ASC("1") THEN ID=4
 350 IF CHAR= ASC("2") THEN ID=5
 360 IF CHAR= ASC("D") THEN ID=6
 370 IF CHAR= ASC("X") THEN ID=7
 400 IF ID=0 THEN 100
 500 GOTO 1000*ID
1000 VTAB 20: TAB 12: CALL -958: PRINT "LOAD PICTURE"
1005 POKE -16300,0: POKE -16303,0
1010 VTAB 22: INPUT "FILE NAME: ",A$
1015 IF A$="" THEN 100
1020 VTAB 22: PRINT "BLOAD ";A$;",A$2000,D1"
1050 GOTO 100
2000 VTAB 20: TAB 12: CALL -958: PRINT "ANALYZE PICTURE"
2005 POKE -16300,0: POKE -16303,0
2010 VTAB 22: INPUT "MAX ERRORS/PIXEL:",MAXERR
2020 POKE 16,MAXERR: CALL ANAL
2025 FLAG=1:XFLAG=0:NUMBER=40* PEEK (8)+ PEEK (7)+1
2030 VTAB 22: PRINT "THERE ARE ";NUMBER;" PIXELS WITH MAX ERROR = ";MAXERR
2035 POKE -16384+16,0
2040 IF PEEK (-16384)<128 THEN 2040
2050 GOTO 100
3000 VTAB 20: TAB 12: PRINT "SYNTHESIZE PICTURE"
3005 POKE -16300,0: POKE -16303,0: VTAB 22: CALL -958
3010 FOR K=1 TO 500: NEXT K
3020 IF FLAG THEN 3050
3030 VTAB 22: PRINT "THERE ARE NO PIXELS DEFINED YET!"
3040 GOTO 3060
3050 CALL SYN
3055 XFLAG=1
3060 POKE -16384+16,0
3070 IF PEEK (-16384)<128 THEN 3070
3080 IF PEEK (-16384)= ASC("1") THEN 210
3085 IF PEEK (-16384)= ASC("2") THEN 210
3090 GOTO 100
4000 POKE -16304,0: POKE -16302,0: POKE -16300,0: POKE -16297,0
4050 GOTO 200
5000 POKE -16304,0: POKE -16302,0: POKE -16299,0: POKE -16297,0
5050 GOTO 200
6000 VTAB 20: TAB 12: CALL -958: PRINT "DISK COMMAND"
6005 POKE -16300,0: POKE -16303,0
6010 VTAB 22: INPUT ":",A$
6015 IF A$="" THEN 100
6020 VTAB 22: TAB 2: PRINT "";A$
6030 PRINT : PRINT : PRINT
6040 GOTO 6010
7000 VTAB 20: TAB 12: CALL -958: PRINT "SAVE COMPRESSED PICTURE"
7005 POKE -16300,0: POKE -16303,0
7010 IF XFLAG THEN 7025
7015 VTAB 22: PRINT "NO PICTURE HAS BEEN SYNTHESIZED YET!"
7020 GOTO 7040
7025 IF NUMBER<=256 THEN 7060
7030 VTAB 22: PRINT "THERE ARE TOO MANY (";NUMBER;") PIXELS"
7040 POKE -16384+16,0
7045 IF PEEK (-16384)<128 THEN 7045
7050 GOTO 100
7060 VTAB 22: INPUT "FILE NAME: ",A$
7065 IF A$="" THEN 100
7070 CALL PRESS
7080 VTAB 22: PRINT "BSAVE ";A$;",A$8000,L";960+2+8*NUMBER;",D2"
7090 GOTO 100
```

```
0800                ;************************
0800                ;*                      *
0800                ;*  PICTURE COMPRESSION *
0800                ;*     ROBERT BISHOP    *
0800                ;*                      *
0800                ;*       PICT COMP      *
0800                ;*                      *
0800                ;*  COPYRIGHT (C) 1981  *
0800                ;*    MICRO INK, INC.   *
0800                ;* CHELMSFORD, MA 01824 *
0800                ;*  ALL RIGHTS RESERVED *
0800                ;*                      *
0800                ;************************
0800                ;
0800                ;
0800                ;
0800                XAT     EPZ $0000
0800                YAT     EPZ $0001
0800                ZAT     EPZ $0002
0800                XTO     EPZ $0003
0800                YTO     EPZ $0004
0800                ZTO     EPZ $0005
0800                SCOR    EPZ $0006
0800                XMAX    EPZ $0007
0800                YMAX    EPZ $0008
0800                XTMP    EPZ $0009
0800                YTMP    EPZ $000A
0800                BEST    EPZ $000B
0800                AT      EPZ $000C
0800                TO      EPZ $000E
0800                ERR     EPZ $0010
0800                XIN     EPZ $0011
0800                YIN     EPZ $0012
0800                PROD    EPZ $0013
0800                HGRL    EQU $0C00
0800                HGRH    EQU $0D00
0800                BITS    EQU $1000
0800                BELL    EQU $FF3A
0800                ;
0800                ;
0B00                        ORG $B00
0B00                        OBJ $800
0B00                ;
0B00 209311         BILD    JSR INIT
0B03 A900                   LDA #$00
0B05 8500                   STA XAT
0B07 8501                   STA YAT
0B09 A901                   LDA #$01
0B0B 8502                   STA ZAT
0B0D A903                   LDA #$03
0B0F 8505                   STA ZTO
0B11 A900           BLUP    LDA #$00
0B13 8503                   STA XTO
0B15 8504                   STA YTO
0B17 202311         LUPE    JSR COMP
0B1A A510                   LDA ERR
0B1C C506                   CMP SCOR
0B1E B01F                   BCS GOOD
0B20 A503                   LDA XTO
0B22 C507                   CMP XMAX
0B24 D006                   BNE NEXT
0B26 A504                   LDA YTO
0B28 C508                   CMP YMAX
0B2A F005                   BEQ OVER
0B2C 20F10B         NEXT    JSR NUTO
0B2F D0E6                   BNE LUPE
0B31 20F10B         OVER    JSR NUTO
0B34 200011                 JSR MOVE
0B37 A503                   LDA XTO
0B39 8507                   STA XMAX
0B3B A504                   LDA YTO
0B3D 8508                   STA YMAX
0B3F E600           GOOD    INC XAT
0B41 A500                   LDA XAT
0B43 C928                   CMP #$28
```

```
0B45 D0CA             BNE BLUP
0B47 A900             LDA #$00
0B49 8500             STA XAT
0B4B E601             INC YAT
0B4D A501             LDA YAT
0B4F C918             CMP #$18
0B51 D0BE             BNE BLUP
0B53 4C3AFF           JMP BELL
0B56           ;
0B56           ; RECONSTRUCTION
0B56           ;
0B80                  ORG $B80
0B80                  OBJ $880
0B80           ;
0B80 A900      RCON   LDA #$00
0B82 8D50C0           STA $C050
0B85 8D52C0           STA $C052
0B88 8D55C0           STA $C055
0B8B 8D57C0           STA $C057
0B8E 8503             STA XTO
0B90 8504             STA YTO
0B92 A903             LDA #$03
0B94 8502             STA ZAT
0B96 A9FF      RLUP   LDA #$FF
0B98 850B             STA BEST
0B9A A900             LDA #$00
0B9C 8500             STA XAT
0B9E 8501             STA YAT
0BA0 A901             LDA #$01
0BA2 8505             STA ZTO
0BA4 202311    LOOP   JSR COMP
0BA7 A506             LDA SCOR
0BA9 C50B             CMP BEST
0BAB B00A             BCS CONT
0BAD 850B             STA BEST
0BAF A500             LDA XAT
0BB1 8509             STA XTMP
0BB3 A501             LDA YAT
0BB5 850A             STA YTMP
0BB7 A500      CONT   LDA XAT
0BB9 C507             CMP XMAX
0BBB D006             BNE INC
0BBD A501             LDA YAT
0BBF C508             CMP YMAX
0BC1 F010             BEQ SEND
0BC3 E600      INC    INC XAT
0BC5 A500             LDA XAT
0BC7 C928             CMP #$28
0BC9 D0D9             BNE LOOP
0BCB A900             LDA #$00
0BCD 8500             STA XAT
0BCF E601             INC YAT
0BD1 D0D1             BNE LOOP
0BD3 A509      SEND   LDA XTMP
0BD5 8500             STA XAT
0BD7 A50A             LDA YTMP
0BD9 8501             STA YAT
0BDB A902             LDA #$02
0BDD 8505             STA ZTO
0BDF 200012           JSR STOR
0BE2 200011           JSR MOVE
0BE5 20F10B           JSR NUTO
0BE8 A504             LDA YTO
0BEA C918             CMP #$18
0BEC D0A8             BNE RLUP
0BEE 4C3AFF           JMP BELL
0BF1 E603      NUTO   INC XTO
0BF3 A503             LDA XTO
0BF5 C928             CMP #$28
0BF7 D006             BNE RET
0BF9 A900             LDA #$00
0BFB 8503             STA XTO
0BFD E604             INC YTO
0BFF 60        RET    RTS
0C00           ;
```

```
0C00           ; MOVE A PIXEL FROM XAT,YAT,ZAT
0C00           ; TO XTO,YTO,ZTO....
0C00           ;
1100                   ORG $1100
1100                   OBJ $E00
1100           ;
1100 8A        MOVE    TXA
1101 48                PHA
1102 98                TYA
1103 48                PHA
1104 205411            JSR PREP
1107 A400      MLUP    LDY XAT
1109 B10C              LDA (AT),Y
110B A403              LDY XTO
110D 910E              STA (TO),Y
110F A50D              LDA AT+1
1111 6904              ADC #$04
1113 850D              STA AT+1
1115 A50F              LDA TO+1
1117 6904              ADC #$04
1119 850F              STA TO+1
111B CA                DEX
111C D0E9              BNE MLUP
111E 68                PLA
111F A8                TAY
1120 68                PLA
1121 AA                TAX
1122 60                RTS
1123           ;
1123           ; COMPARE PIXEL AT XAT,YAT,ZAT
1123           ; TO XTO,YTO,ZTO
1123           ;
1123 8A        COMP    TXA
1124 48                PHA
1125 98                TYA
1126 48                PHA
1127 205411            JSR PREP
112A A900              LDA #$00
112C 8506              STA SCOR
112E A400      CLUP    LDY XAT
1130 B10C              LDA (AT),Y
1132 A403              LDY XTO
1134 510E              EOR (TO),Y
1136 297F              AND #$7F
1138 A8                TAY
1139 B90010            LDA BITS,Y
113C 6506              ADC SCOR
113E 8506              STA SCOR
1140 A50D              LDA AT+1
1142 6904              ADC #$04
1144 850D              STA AT+1
1146 A50F              LDA TO+1
1148 6904              ADC #$04
114A 850F              STA TO+1
114C CA                DEX
114D D0DF              BNE CLUP
114F 68                PLA
1150 A8                TAY
1151 68                PLA
1152 AA                TAX
1153 60                RTS
1154           ;
1154 A502      PREP    LDA ZAT
1156 6A                ROR
1157 6A                ROR
1158 6A                ROR
1159 6A                ROR
115A 2960              AND #$60
115C 850D              STA AT+1
115E A505              LDA ZTO
1160 6A                ROR
1161 6A                ROR
1162 6A                ROR
1163 6A                ROR
1164 2960              AND #$60
```

```
1166 850F           STA TO+1
1168 A501           LDA YAT
116A 0A             ASL
116B 0A             ASL
116C 0A             ASL
116D AA             TAX
116E BD000C         LDA HGRL,X
1171 850C           STA AT
1173 BD000D         LDA HGRH,X
1176 291F           AND #$1F
1178 650D           ADC AT+1
117A 850D           STA AT+1
117C A504           LDA YTO
117E 0A             ASL
117F 0A             ASL
1180 0A             ASL
1181 AA             TAX
1182 BD000C         LDA HGRL,X
1185 850E           STA TO
1187 BD000D         LDA HGRH,X
118A 291F           AND #$1F
118C 650F           ADC TO+1
118E 850F           STA TO+1
1190 A208           LDX #$08
1192 60             RTS
1193          ;
1193 20C00C   INIT  JSR $0CC0
1196 A97F           LDA #$7F
1198 8D0160         STA $6001
119B 8D0164         STA $6401
119E 8D0168         STA $6801
11A1 8D016C         STA $6C01
11A4 8D0170         STA $7001
11A7 8D0174         STA $7401
11AA 8D0178         STA $7801
11AD 8D017C         STA $7C01
11B0 A900           LDA #$00
11B2 8508           STA YMAX
11B4 A901           LDA #$01
11B6 8507           STA XMAX
11B8 60             RTS
11B9          ;
1200                ORG $1200
1200 98       STCR  TYA
1201 48             PHA
1202 A503           LDA XTO
1204 8511           STA XIN
1206 A504           LDA YTO
1208 8512           STA YIN
120A 202C12         JSR X40
120D A513           LDA PROD
120F 850E           STA TO
1211 18             CLC
1212 A514           LDA PROD+1
1214 6980           ADC #$80
1216 850F           STA TO+1
1218 A500           LDA XAT
121A 8511           STA XIN
121C A501           LDA YAT
121E 8512           STA YIN
1220 202C12         JSR X40
1223 A513           LDA PROD
1225 A000           LDY #$00
1227 910E           STA (TO),Y
1229 68             PLA
122A A8             TAY
122B 60             RTS
122C A512     X40   LDA YIN
122E 8513           STA PROD
1230 A900           LDA #$00
1232 8514           STA PROD+1
1234 0613           ASL PROD
1236 2614           ROL PROD+1
1238 0613           ASL PROD
```

```
123A 2614          ROL PROD+1
123C 0613          ASL PROD
123E 2614          ROL PROD+1
1240 A513          LDA PROD
1242 0613          ASL PROD
1244 2614          ROL PROD+1
1246 0613          ASL PROD
1248 2614          ROL PROD+1
124A 6513          ADC PROD
124C 8513          STA PROD
124E A514          LDA PROD+1
1250 6900          ADC #$00
1252 8514          STA PROD+1
1254 A513          LDA PROD
1256 6511          ADC XIN
1258 8513          STA PROD
125A A514          LDA PROD+1
125C 6900          ADC #$00
125E 8514          STA PROD+1
1260 60            RTS
                  END

***** END OF ASSEMBLY


          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5  *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

XAT   0000  YAT   0001  ZAT   0002  XTO   0003  YTO   0004  ZTO   0005
SCOR  0006  XMAX  0007  YMAX  0008  XTMP  0009  YTMP  000A  BEST  000B
AT    000C  TO    000E  ERR   0010  XIN   0011  YIN   0012  PROD  0013

** ABSOLUTE VARABLES/LABELS

HGRL  0C00  HGRH  0D00  BITS  1000  BELL  FF3A  BILD  0B00  BLUP  0B11
LUPE  0B17  NEXT  0B2C  OVER  0B31  GOOD  0B3F  RCON  0B80  RLUP  0B96
LOOP  0BA4  CONT  0BB7  INC   0BC3  SEND  0BD3  NUTO  0BF1  RET   0BFF
MOVE  1100  MLUP  1107  COMP  1123  CLUP  112E  PREP  1154  INIT  1193
STOR  1200  X40   122C

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0172
```

An Apple Flavored Lifesaver

by Gregory L. Tibbetts

The game of LIFE is made a little easier with this flexible storage program which provides for translation, rotation, and reversal of patterns.

John Conway's game of LIFE has one of the largest followings of any computer simulation ever devised. My own interest dates back to my first ''cellular excursion'' in 1972, on a Hewlett-Packard 2000c machine. Since then I've collected half a dozen versions and have played with several more, all widely different in execution. One serious drawback nearly every version shares, however, is the sheer drudgery of entering from 2 to 200 sets of coordinates each time a simulation is to be run. I've seen several programs with systems to capture coordinates for a given figure—some plain and some incredibly complex. All of these though, are hampered by the fact that LIFE devotees rarely input the same pattern at exactly the same location and orientation twice, and they usually like to combine figures for interactive effects. One system attempting to circumvent these problems had over 120 individual figures on paper tape, most duplicated up to 8 times for different orientations, and all marked and cataloged. Now that's dedication!

Being basically lazy myself (after all, I bought a computer to save myself work), I decided that I needed a few simple routines that would let me name and save figures to disk, and then call them back to the screen at virtually any location, at any reasonable orientation, and in combination with any other pattern on file. My goal then, and the subject of this article, is simply to make LIFE a little easier (pun intended).

The platform I chose to build my routine on is an excellent machine code/Integer BASIC hybrid program written by Dick Suitor entitled ''Life for Your Apple.'' It appeared in *MICRO on the Apple, Volume I.* Probably the best and most versatile of all the versions I have seen, it has features like variable generation speed, the ability to set random cells alive in a selected field, and the use of contrasting color to show cell development.

My first task was to come up with a method of storing and retrieving the figures. The obvious solution was to save the *x,y* coordinates in a sequential text file. To make the figures completely relocatable however, I needed a way to make the stored coordinates independent of the screen coordinates. The method I chose was to select an arbitrary centerpoint for the figure, prior to input. Then as each coordinate set was typed in, the *x*, *y* values of the center point would be subtracted from the *x*, *y* values of the point being entered. The result is a set of codified *x*, *y* values, positive and negative, which are relative only to the centerpoint, and therefore totally independent of their current screen location. All that's required to relocate the figure then, is to change the centerpoint when calling the figure back from storage.

This method, in conjunction with Apple's system of screen coordinates, does introduce an irregularity which will become important as we proceed. In normal coordinate systems *x* values increase as we move to the right, and *y* values increase as we go up. With the Apple II, *y* values increase as we descend on the screen. Further, all screen coordinates are positive, while the codified values may be positive or negative, since they essentially make up a coordinate grid of their own, with the *x* (horizontal) and *y* (vertical) axes intersecting at the chosen centerpoint. Unlike normal grids, therefore, *y* values will be negative above this *x* axis and positive below it. It will be necessary to keep this in mind, as it is the codified values we will be manipulating in the coming paragraphs when we determine how to reorient the figures.

This second task—finding a way to bring the stored figure back to the screen in a different attitude than originally entered—was somewhat more difficult than simply making it relocatable. However, it quickly became clear that all possible orientations could be achieved by reversing the figure, rotating it, or both.

Rotation is obtained by moving each point clockwise around the center some distance (depending on the degree of rotation), while reversal takes the two dimensional image and flips it over, as one would turn over a playing card. Obviously reversal requires us to know which axis the figure is to be reversed around.

Defining an algorithm to rotate and reverse the figures was an interesting exercise, (actually three exercises and three algorithms). I'm sure that somewhere in the field of coordinate mathematics there exists specific rules for such operations. Being more a tinkerer than a scholar, however, I chose to discover those rules by trial and error. Armed with graph paper and pencil, I defined a center, an *x* and *y* axis, and began examining what happened to various sets of coordinates when the points they described were reversed or rotated. The first thing I discovered was that for any single set of coordinates, rotation or reversal involved only two operations: either the unsigned magnitudes of the *x* and *y* values being swapped, or the signs of one or both values being changed. One, or a combination of these two alterations will produce all feasible orientations. I also learned that rotations in other than 90^0 increments were not feasible for the purposes of the LIFE game, but the proof of that is left as an exercise for the reader.

The reversal mechanism turned out to be the simplest. A little paper and pencil work showed that no matter which axis was used for reversal, any point remained the same distance from each axis when reversed. The magnitudes of the

x and y values then must remain the same. The signs, however, do not. A reversal around the y axis, for example, sends points from the upper right quadrant $(+x, -y)$ to the upper left quadrant $(-x\ -y)$, and from lower right $(+x, +y)$ to lower left $(-x, +y)$. Obviously then, reversal on the y axis changes the sign of the x values only. By the same token, an x axis reversal changes the sign of the y values only. Translated into a sequence of program steps this mechanism is implemented in program lines 1070-1110 and 350-400. I also resolved the further question of whether multiple reversals were desirable, that is, two reversals around one axis, or one around each. I determined they were not, but as a second exercise, for fun, the reader may wish to prove why they were not.

Rotation was a little harder as the cases of 90⁰, 180⁰, and 270⁰ rotation all had to be allowed for. Easiest to discover was the 180⁰ process. Just as in the reversal case, a point rotated 180⁰ still remains the same distance from each axis, and therefore, the x and y magnitudes remain the same. Signs however, do not follow the same pattern as during reversal. Since the points in the upper right quadrant $(+x,\ -y)$ move to the lower left $(-x,\ +y)$, lower right$(+x,\ +y)$ to upper left $(-x,\ -y)$ and vice versa, it becomes clear that both x and y values must change sign. A 180⁰ rotation therefore is accomplished by simply multiplying the two values by -1. This is implemented in lines 1030-1060 and 320-340.

A 90⁰ rotation is not so straight-forward. It is best seen by using the example of a clock face with the x axis running through the 9 and 3, and the y axis through the 12 and 6. A 90⁰ rotation of this clock face moves the point at numeral 1 to the position of numeral 4. For the first time, the magnitude of the x and y values have changed. The distance of the point from the y axis in its original position has become the distance from the x axis after rotation and vice versa. What happens in a 90⁰ rotation then, is that the magnitudes of x and y are simply exchanged. The signs, unfortunately, do not follow such a clearcut pattern. Nevertheless, a pattern does exist. I found it by examining the four quadrants in sequence and noting what happens to their associated x and y signs. Starting at the upper right $(+x,\ -y)$ and moving to the lower right produces $(+x,\ +y)$. Another 90⁰ rotation produces $(-x,\ +y)$, and the final rotation $(-x,\ -y)$. Study here shows that the sign of x in the original quadrant is the sign y will have in the new quadrant. Since the magnitude of x becomes the magnitude of y also, we can simply give y the signed value of x for every point to be rotated. You can also see that the sign of the new x value is the opposite of the old y value. To get the new x value we must multiply the old signed value of y by -1. These two steps complete the 90⁰ algorithm and it is implemented in lines 1030-1060 and 270-310. To keep the program as short as possible, 270⁰ rotations were made by using the 90⁰ and 180⁰ subroutines together. This completes the screen output design.

Disk storage is achieved by saving the x and y arrays into a sequential text file; each figure to a separate file. Though this is somewhat wasteful of disk space, I set it up this way to avoid complex file management routines, and to allow for easy renaming and catalog display. The final step was to insert tests in the plot sequence to prevent range errors from crashing the program if a center point was selected that would cause the figure to plot off the screen, and having to restart the program from scratch. The original centerpoint is not stored with the codified values, and consequently is not available for later examination.

The program as it appears in the listing, is set up to run on a 48K Apple II, using Apple DOS to store and retrieve the patterns. The instructions for setting up the program, however, are universal with respect to RAM size. I believe that the program could also be converted to use a cassette-based DOS imitator as off-line storage, but that is beyond the scope of this article. (Editor's Note: See Robert Stein's "Cassette Operating System" article, in the Hardware section.) The machine code runs resident at $800 (2048), and the program has been modified to load both sections as a unit, and relocate the machine portion when run. (Editor's Note: Both separate BASIC and Machine Language sections, as well as the combined version, are saved on disk.)

The program is completely automated and self-prompting, therefore I have only a few helpful hints.

First, patterns are best developed on, and input from graph paper numbered along the top and side to match the screen. This gives a backup as well as a hard copy visual image to check the screen output. Second, the centerpoint you select to input the figure is not automatically set as a live cell. Consequently, it can literally be any point on the screen. You must remember though, that all figures are rotated and reversed around this relative center and, therefore, it should be chosen with care. Third, with really large figures where the choice of center point is critical to keep from plotting the figure off screen, it is helpful to include the center coordinates in the figure name as a guide during recall. Last, due to the finite field limits established by Mr. Suitor's program, known patterns may not behave normally if they contact the edge. Gliders for example, turn to boxes as they hit the edge, rather than continue to move off screen. This is no cause for alarm; simply a fact of Life.

For fun, create a pattern file with the coordinates listed below. Name this figure PULSAR SEED, and use an initial centerpoint of say 19,19. When you run it the results may surprise you. In any case, have fun!

(x,y); (10,8); (9,9); (11,9); (9,10); (11,10); (9,11); (10,11); (11,11); (9,12); (11,12); (9,13); (11,13); (10,14); (99,99).

```
0800                ;************************
0800                ;*                      *
0800                ;*   APPLE LIFESAVER    *
0800                ;* GREGORY L. TIBBETTS  *
0800                ;*                      *
0800                ;*      LIFESAVER       *
0800                ;*                      *
0800                ;*  COPYRIGHT (C) 1981  *
0800                ;*   MICRO INK, INC.    *
0800                ;* CHELMSFORD, MA 01824 *
0800                ;*  ALL RIGHTS RESERVED *
0800                ;*                      *
0800                ;************************
0800                ;
0800                ;
0800                ;
0800 A505           LBLI    LDA $0005
0802 8503                   STA $0003
0804 A504                   LDA $0004
0806 8502                   STA $0002
0808 18                     CLC
0809 6980                   ADC #$80
080B 8504                   STA $0004
080D A505                   LDA $0005
080F 6900                   ADC #$00
0811 C908                   CMP #$08
0813 D00C                   BNE LBLA
0815 A504                   LDA $0004
0817 6927                   ADC #$27
0819 C952                   CMP #$52
081B 1008                   BPL LBLB
081D 8504                   STA $0004
081F A904                   LDA #$0004
0821 8505           LBLA    STA $0005
0823 18                     CLC
0824 60             LBLR    RTS
0825 38             LBLB    SEC
0826 B0FC                   BCS LBLR
0828 20CA08                 JSR LBLS
082B 200008         LBLX    JSR LBLI
082E 9001                   BCC LBLC
0830 60                     RTS
0831 A027           LBLC    LDY #$27
0833 98                     TYA
0834 AA                     TAX
0835 A900           LBLH    LDA #$00
0837 994009                 STA $0940,Y
083A 997009                 STA $0970,Y
083D B102                   LDA ($02),Y
083F F00F                   BEQ LBLE
0841 100A                   BPL LBLD
0843 FE4009                 INC $0940,X
0846 FE7009                 INC $0970,X
0849 2908                   AND #$08
084B F003                   BEQ LBLE
084D FE4009         LBLD    INC $0940,X
0850 B104           LBLE    LDA ($04),Y
0852 F00F                   BEQ LBLG
0854 1003                   BPL LBLF
0856 FE7009                 INC $0970,X
0859 2908           LBLF    AND #$08
085B F006                   BEQ LBLG
085D FE7009                 INC $0970,X
0860 FE4009                 INC $0940,X
0863 88             LBLG    DEY
0864 CA                     DEX
0865 10CE                   BPL LBLH
0867 A026                   LDY #$26
0869 18                     CLC
086A AD6709                 LDA $0967
086D 6D6609                 ADC $0966
0870 8506                   STA $0006
0872 AD9709                 LDA $0997
0875 6D9609                 ADC $0996
```

```
0878 8507               STA $0007
087A 18          LBLW   CLC
087B A506               LDA $0006
087D 793F09             ADC $093F,Y
0880 38                 SEC
0881 F94209             SBC $0942,Y
0884 8506               STA $0006
0886 C903               CMP #$03
0888 F00E               BEQ LBLK
088A 9004               BCC LBLJ
088C C904               CMP #$04
088E F00E               BEQ LBLL
0890 B102        LBLJ   LDA ($02),Y
0892 F00A               BEQ LBLL
0894 2985               AND #$85
0896 5004               BVC LBLM
0898 B102        LBLK   LDA ($02),Y
089A 0930               ORA #$30
089C B102        LBLM   LDA ($02),Y
089E 18          LBLL   CLC
089F A507               LDA $0007
08A1 796F09             ADC $096F,Y
08A4 38                 SEC
08A5 F97209             SBC $0972,Y
08A8 8507               STA $0007
08AA C903               CMP #$03
08AC F00E               BEQ LBLP
08AE 9004               BCC LBLN
08B0 C904               CMP #$04
08B2 F00E               BEQ LBLT
08B4 B104        LBLN   LDA ($04),Y
08B6 F00A               BEQ LBLT
08B8 29F8               AND #$F8
08BA 5004               BVC LBLV
08BC B104        LBLP   LDA ($04),Y
08BE 0903               ORA #$03
08C0 9104        LBLV   STA ($04),Y
08C2 88          LBLT   DEY
08C3 F002               BEQ LBLU
08C5 10B3               BPL LBLW
08C7 4C2B08      LBLU   JMP LBLX
08CA A904        LBLS   LDA #$04
08CC 8505               STA $0005
08CE A900               LDA #$00
08D0 8504               STA $0004
08D2 8D6809             STA $0968
08D5 8D8809             STA $0988
08D8 60                 RTS
08D9 20CA08             JSR LBLS
08DC 200008      LABD   JSR LBLI
08DF 9001               BCC LBLY
08E1 60                 RTS
08E2 A027        LBLY   LDY #$27
08E4 B102        LBLO   LDA ($02),Y
08E6 F00A               BEQ LBLZ
08E8 297F               AND #$7F
08EA C910               CMP #$10
08EC 3002               BMI LABA
08EE 0980               ORA #$80
08F0 9102        LABA   STA ($02),Y
08F2 B104        LBLZ   LDA ($04),Y
08F4 F00A               BEQ LABB
08F6 29F7               AND #$F7
08F8 6A                 ROR
08F9 9002               BCC LABC
08FB 0904               ORA #$04
08FD 2A          LABC   ROL
08FE 9104               STA ($04),Y
0900 88          LABB   DEY
0901 F0D9               BEQ LABD
0903 10DF               BPL LBLO
                       END
```

```
           **************************
           *                        *
           * SYMBOL TABLE -- V 1.5 *
           *                        *
           **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:


** ABSOLUTE VARABLES/LABELS

LBLI   0800  LBLA   0821  LBLR   0824  LBLB   0825  LBLX   082B  LBLC   0831
LBLH   0835  LBLD   084D  LBLE   0850  LBLF   0859  LBLG   0863  LBLW   087A
LBLJ   0890  LBLK   0898  LBLM   089C  LBLL   089E  LBLN   08B4  LBLP   08BC
LBLV   08C0  LBLT   08C2  LBLU   08C7  LBLS   08CA  LABD   08DC  LBLY   08E2
LBLO   08E4  LABA   08F0  LBLZ   08F2  LABC   08FD  LABB   0900

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:00FA
```

```
  1 REM   ************************
  2 REM   *                      *
  3 REM   *   APPLE LIFE-SAVER   *
  4 REM   *   GREGORY TIBBETTS   *
  5 REM   *                      *
  6 REM   *      LIFESAVER       *
  7 REM   *                      *
  8 REM   *  COPYRIGHT (C) 1981  *
  9 REM   *    MICRO INK, INC.   *
 10 REM   * CHELMSFORD, MA 01824 *
 11 REM   *  ALL RIGHTS RESERVED *
 12 REM   *                      *
 13 REM   ************************
 14 REM
 15 LOMEM:2500
 16 DIM HEX$(30)
 30 PRINT "BLOAD LIFE"
 50 GOTO 800
 60 POKE -16302,0: COLOR=0: FOR K=40 TO 47
 70 HLIN 0,39 AT K: NEXT K
 80 KX= PDL (0)-10: IF KX>240 THEN KX=KX1: IF KX<0 THEN KX=0
 90 K1=KX*6:K2=KX*2:K3=500/(K1+50)+1
100 FOR I=1 TO K3
110 CALL GEN
120 FOR K=1 TO K2: NEXT K
130 CALL MOP
140 FOR K=1 TO SIZE: COLOR=11
150 NEXT I
160 GOTO 80
170 FOR I=1 TO SIZE: COLOR=11
180 X=XCTR+X(I):Y=YCTR+Y(I)
190 IF X<0 OR X>39 OR Y<0 OR Y>39 THEN 1210
200 PLOT X,Y: NEXT I
210 RETURN
220 FOR I=I1 TO I2: FOR J=J1 TO J2
230 COLOR=11: IF RND (L) THEN COLOR=0
240 PLOT I,J
250 NEXT J: NEXT I
260 GOTO 60
270 FOR I=1 TO SIZE
280 X=Y(I):Y=X(I)
290 IF Y(I) THEN X=X*-1
```

```
 300 X(I)=X:Y(I)=Y
 310 NEXT I: RETURN
 320 FOR I=1 TO SIZE
 330 X(I)=X(I)*-1:Y(I)=Y(I)*-1
 340 NEXT I: RETURN
 350 FOR I=1 TO SIZE
 360 IF XAX THEN 380
 370 X=X(I):Y=Y(I)*-1: GOTO 390
 380 Y=Y(I):X=X(I)*-1
 390 X(I)=X:Y(I)=Y: NEXT I
 400 RETURN
 410 PRINT D$;"OPEN";A$
 420 PRINT D$;"READ";A$
 430 FOR I=1 TO 255
 440 INPUT X(I),Y(I)
 450 IF X(I)=99 OR Y(I)=99 THEN 470
 460 NEXT I
 470 SIZE=I-1
 480 PRINT D$;"CLOSE";A$
 490 IF ROT THEN GOSUB 270
 500 IF HALF THEN GOSUB 320
 510 IF REV THEN GOSUB 350
 520 GOSUB 170
 530 HALF=0:ROT=0:REV=0:XAX=0:SIZE=0
 540 RETURN
 550 PRINT D$;"OPEN";A$
 560 PRINT D$;"DELETE";A$
 570 PRINT D$;"OPEN";A$
 580 PRINT D$;"WRITE";A$
 590 FOR I=1 TO SIZE
 600 PRINT X(I)
 610 PRINT Y(I)
 620 NEXT I
 630 PRINT D$;"CLOSE";A$
 640 RETURN
 650 FOR I=1 TO 255
 660 INPUT X,Y
 670 IF X=99 OR Y=99 THEN 720
 680 IF X<0 OR X>39 OR Y<0 OR Y>39 THEN 700
 690 X(I)=X-XCTR:Y(I)=Y-YCTR: GOTO 710
 700 PRINT "INPUT X,Y",X,Y
 710 NEXT I
 720 X(I)=99:Y(I)=99
 730 SIZE=I
 740 RETURN
 750 INPUT "INPUT X,Y",X,Y
 760 IF X=99 OR Y=99 THEN 60
 770 IF X<0 OR X>39 OR Y<0 OR Y>39 THEN 790
 780 COLOR=11: PLOT X,Y: GOTO 750
 790 PRINT "OUT OF RANGE!": GOTO 750
 800 TEXT
 810 DIM X(255),Y(255),A$(50),B$(2),D$(1)
 820 GEN=2088:MOP=2265:K1=1:K2=1:D$="": REM     D$=CNTRL D
 830 CALL -936: VTAB 5: TAB 9: PRINT "CONWAY'S GAME OF LIFE": FOR I=1 TO
     700: NEXT I
 840 GR
 850 PRINT "DO YOU WISH TO: 1.PLAY OR 2.CREATE"
 860 INPUT "A NEW PATTERN FILE (1/2).",C1
 870 IF C1=2 THEN 1140
 880 INPUT "SPEED=PDL(0):SET DEFAULT (0-255)",KX1
 890 PRINT "DO YOU WISH: 1.RANDOM PATTERN 2.PATTERN"
 900 INPUT "FROM DISK OR 3.STANDARD: (1/2/3)",C1
 910 IF C1=3 THEN 990
 920 IF C1=2 THEN 1010
 930 INPUT "X DIRECTION LIMITS ",I1,I2
 940 IF I1<0 OR I2>39 OR I1>I2 THEN 930
 950 INPUT "Y DIRECTION LIMITS ",J1,J2
 960 IF J1<0 OR J2>39 OR J1>J2 THEN 950
 970 INPUT "ONE IN 'N' CELLS WILL LIVE: ENTER N",L
 980 GOTO 220
 990 PRINT "ENTER YOUR PATTERN (X,Y):99,99 EXITS"
1000 GOTO 750
1010 INPUT "WHAT FIGURE NAME",A$
1020 INPUT "ENTER CENTER COORD'S (X,Y)",XCTR,YCTR
```

```
1030 INPUT "ENTER ROTATION (0/90/180/270)",ROT
1040 IF ROT=180 OR ROT=270 THEN HALF=1
1050 IF ROT=90 OR ROT=270 THEN ROT=1
1060 IF ROT<>1 THEN ROT=0
1070 INPUT "ENTER 1.REVERSED OR 2.STANDARD (1/2)",REV
1080 IF REV>1 THEN REV=0: IF NOT REV THEN 1110
1090 INPUT "REVERSE ON 1.X-AXIS OR 2.Y-AXIS (1/2)",XAX
1100 IF XAX>1 THEN XAX=0
1110 GOSUB 410
1120 INPUT "ANOTHER FIGURE (Y/N)",B$: IF B$="N" THEN 60
1130 PRINT "CAUTION: FIGURES MAY OVERWRITE!": GOTO 1010
1140 INPUT "ENTER CENTER COORD'S (X,Y)",XCTR,YCTR
1150 PRINT "ENTER ALL LIVE CELLS (X,Y):99,99 EXITS"
1160 GOSUB 650
1170 INPUT "ENTER NAME FOR THIS FIGURE",A$
1180 GOSUB 550
1190 PRINT "TESTING": GOSUB 410
1200 GOTO 60
1210 PRINT "PLOT ABORTED/FIGURE WENT OFF SCREEN"
1220 PRINT "MOVE CENTERPOINT:X AND Y WHEN ABORTED"
1230 PRINT "WERE ";X;",";Y: POP : POP
1240 IF I=1 THEN 1020:IE=I-1: COLOR=0: FOR I=1 TO IE
1250 PLOT X(I)+XCTR,Y(I)+YCTR: NEXT I: GOTO 1020
1260 REM   ADAPTATION BY GREG TIBBETTS OF RICHARD SUITOR'S PROGRAM IN
1265 REM   "BEST OF MICRO" VOLUME II 1979
1270 REM   LINES 0-50 PROGRAM SET-UP
1280 REM   60-160 SPEED AND GENERATION
1290 REM   170-210 GENERAL PLOT SUBR.
1300 REM   220-260 RANDOM PLOT SUBR.
1310 REM   270-340 ROTATION SUBR'S.
1320 REM   350-400 REVERSAL SUBR.
1330 REM   410-540 DISK READ SUBR.
1340 REM   550-640 DISK WRITE SUBR.
1345 REM   650-740 DISK INPUT SUBR.
1350 REM   750-790 STANDARD INPUT SUBR.
1360 REM   800-840 INITIALIZATION
1370 REM   850-920 MODE SELECTION
1380 REM   930-1200 USER INPUT/SELECT
1390 REM  1210-1250 PLOT ABORT SUBR.
10000 END
```

Applayer Music Interpreter

by Richard F. Suitor

The Apple's built-in ability to generate sound is well known. Yet oftentimes this powerful capability is underutilized by Apple users, due to the difficulty involved in programming meaningful tones. The Applayer music interpreter eliminates most of these complications, and provides a straightforward method to produce real music on your Apple.

This music program is more than a tone-making routine, it is a music interpreter. It enables you to generate a table of bytes that specify precisely the half-tone and duration of a note with a simple coding. Its virtue over the simpler routines is similar to that of any interpreter (such as Sweet 16, or, more tenuously, BASIC) over an assembler or hand coding—it is easier to achieve your goal and easier to decipher the coding six months later.

The immediate motivation for this interpreter was Martin Gardner's Mathematical Games Column in the April 1978 Scientific American. Several types of algorithmically generated music are discussed in that column; this program provides a means of experimenting with them as well as a convenient method of generating familiar tunes.

The program is written in 6502 assembly language. It would be usable on a system other than the Apple if a speaker was interfaced in a similar way. Accessing a particular address (C030) changes the current through the Apple speaker from on to off or from off to on; it acts like a push button on/off switch (or, of course, a flip-flop). Thus this program makes sound by accessing this address periodically with an LDA C030. Any interface that could likewise be activated with a similar (4 clock cycles) instruction could be easily used. A different interfacing software procedure would change the timing and require more extensive modification.

The tone is generated with a timing loop that counts for a certain number of clock cycles, N (all of the cycles in a period including the toggling of the speaker are counted). Every N cycles a 24 bit pattern is rotated and the speaker is toggled if the high order bit is set. Four cycles are wasted (to keep time) if the bit is not set.

There is a severe limit to the versatility of a waveshape made from on/off transitions, but tones resembling a variety of (cheap) woodwinds and pipes are possible, with fundamentals ranging from about 20 Hz to 8 KHz.

Applayer interprets bytes to produce different effects. There are two types of bytes:

Note bytes — Bit 7 Not Set
Control bytes — Bit 7 Set to 1

A note byte enables you to choose a note from one of 16 half tones, and from one to eight eighth notes in duration. The low order nibble is the half-tone; the high order nibble is the duration (in eighth notes) minus one.

Bit	7 6 5 4	3 2 1 0
Note Byte	0 (Duration)	(Half-Tone)

The control bytes enable you to change the tempo, the tonal range which the 16 half-tones cover, rests, the waveshape of the tone and to jump from one portion of the table to another.

Control Byte Table

HEX	DECIMAL	FUNCTION
81	129	The next three bytes are the new waveshape pattern.
82	130	JMP—New table address follows. Low order byte first, then page byte.
83	131	JSR—New table address follows. When finished, continuing this table at byte after address byte.
9N	144 + N	N is the number of 16th notes to be silent at the tail of a note. Controls rests and note definition.
AN	160 + N < 32	Selects the tonal range. Half-tone #0 is set to one of 32 half-tones giving a basic range of four octaves.
CN	192 + N < 62	Controls the tempo. Length of a note is proportional to N. Largest value gives a whole note lasting about 3.5 sec.
FF	255	RETURN. Stop interpreting this table. Acts as return for 83 JSR instruction or causes return from Applayer.

To use Applayer with sheet music, you must first decide on the range of the half tones. This must sometimes be changed in the middle of the song. For example, the music for "Turkey in the Straw", which appears later, was in the key of C; for the first part of the song I used the following table:

NOTE	C	D	E	F	G	A	B	C	D
TONE #	0	2	4	5	7	9	B	C	E

The tonal range was set with a control byte, B0. In the chorus, the range of the melody shifts up; there the tonal range is set with a B7 and the table is

```
NOTE    G A B C D E F G A
TONE #  0 2 4 5 7 9 A C E
```

(The actual key is determined by the waveshape pattern as well as the tonal range control byte. For the pattern used, 05 05 05, the fundamental for the note written as C would be about 346Hz, which is closer to F.)

Rests can be accomplished with a 9N control byte and a note byte. For example, 94 10 is a quarter rest, 98 30 is a half rest, etc. This control is normally set at 91 for notes distinctly separated, or to 90 for notes that should run together.

Let's try to construct a table that Applayer can use to play a tune. We can start simply with "Twinkle, Twinkle Little Star." That tune has four lines; the first and fourth are identical, as are the second and third. Our table will be constructed to:

1. Set up the tonal range, tone pattern and tempo that we want
2. JSR to a table for the first line
3. JSR to a table for the second line
4. Repeat #3
5. Repeat #2
6. Return
7. First line table and return
8. Second line table and return

Since Applayer is not symbolic, it will be easier to construct the tables in reverse, so that we can know where to go in steps 2-6. The note table for the first line can go at 0B00 and looks like:

```
0B00- 10 10 17 17 19 19 37 15
0B08- 15 14 14 12 12 30 FF FF
```

The second line can follow at 0B10:

```
0B10- 17 17 15 15 14 14 32 FF
```

Now we can start on step 1. I'll suggest the following to start; you'll want to make changes:

```
0B20- B0 81 05 05 05 E0 91
```

The above determines the tonal range, the tone waveshape, the tempo, and a sixteenth note rest out of every note to keep the notes distinct. To run them together, use 90 instead of 91. Steps 2 - 6 can follow immediately:

```
0B20-                      83
0B28- 00 0B 83 10 0B 83 10 0B
0B30- 83 00 0B FF
```

That completes the table for "Twinkle, Twinkle." We now have to tell Applayer where it is and turn it on. From BASIC we must set up some zero page locations first and then JSR to Applayer: (Don't forget to set LOMEM before running; 2900 will do for this table.)

```
100 POKE 19, 32   (low order byte of the table address, 0B20)
110 POKE 20, 11   (high order byte of the table address, 0B20)
120 POKE 1, 8     (high order byte of 1st page of Applayer program)
130 POKE 17, 8    (16 & 17 contain the tone table address)
140 POKE 16, 0
120 CALL 2346     (jump subroutine to 092A)
```

We can also make a short program in assembly language to set up the zero page locations. See routine ZERO, location 09C0 in the listing.

This initialization can be used most easily by reserving the A00 page, or much of it, as a "Table of Contents" for the various note tables elsewhere in memory. To do this with "Twinkle, Twinkle" we add the following table:

0A20- 82 20 0B

This jumps immediately to the table at 0B20. With this convention, we can move from table to table by changing only the byte at 9D0 (2512 decimal).

We can use this initialization from BASIC, too, by changing the last instruction to RTS:

```
100 POKE 2512,32 (low order table byte)
110 POKE 2538,96 (change inst. at 09EA to RTS)
120 CALL 2496    (jump subroutine to 9C0)
```

From the monitor: *9D0:20
*9C0G

will do.

If you quickly tire of "Twinkle, Twinkle," you may wish to play with "Turkey in the Straw." The table follows; its structure will be left as an exercise.

From the monitor: *9D0:0
*9C0G

will play it.

(Editor's Note: An Integer BASIC driver routine for APPLAYER, called APPLAYER MENU, is included on the disk. This driver program automatically loads and executes the music interpreter, allowing playback of either of the two example tunes discussed (these tunes are included in the APPLAYER binary file). Users without Integer BASIC in their systems may still load and execute APPLAYER directly from the monitor, as described in the article.)

(Editor's Note: Glitches in "Turkey in the Straw" were deliberately included. It is left as an exercise to the reader to correct them!)

Note Table for "Turkey in the Straw"

0A00:	83	90	0F	83	90	0F	FF	
0F00:	90	1C	1A	92	38	90	18	1A
0F08:	18	13	10	11	91	13	13	33
0F10:	33	90	18	1A	92	3C	3C	90
0F18:	1C	1A	18	1A	91	1C	38	18
0F20:	38	90	1C	1A	92	38	90	18
0F28:	1A	18	13	91	10	11	13	53
0F30:	33	90	18	1A	91	3C	3F	90
0F38:	1F	1C	18	1A	1C	18	92	3A
0F40:	94	78	91	FF				
0F50:	81	55	55	55	FF			
0F58:	81	05	05	05	FF			
0F60:	15	18	18	15	78	FF		
0F68:	16	1A	1A	16	7A	FF		
0F70:	1D	1D	1D	1D	18	18	18	18
0F78:	35	15	15	33	90	11	13	91
0F80:	15	18	18	18	90	18	15	11
0F88:	13	91	15	15	13	13	71	FF
0F90:	83	58	0F	D4	B0	83	50	0F
0F98:	B7	83	60	0F	83	50	0F	83
0FA0:	60	0F	83	50	0F	83	68	0F
0FA8:	83	50	0F	83	68	0F	83	50
0FB0:	0F	83	70	0F	FF			

Tone Table

0800:	A0	03	68	03	38	03	08	03
0808:	E0	02	B8	02	90	02	68	02
0810:	48	02	28	02	08	02	E8	01
0818:	D0	01	B4	01	9C	01	84	01
0820:	70	01	5C	01	48	01	34	01
0828:	24	01	14	01	04	01	F4	00
0830:	E8	00	DA	00	CE	00	C2	00
0838:	B8	00	AE	00	A4	00	9A	00
0840:	92	00	8A	00	82	00	7A	00
0848:	74	00	6D	00	67	00	61	00
0850:	5C	00	57	00	52	00	4D	00
0858:	49	00	45	00	41	00	3D	00

```
0800             ;************************
0800             ;*                      *
0800             ;*    APPLAYER MUSIC    *
0800             ;*     INTERPRETER      *
0800             ;*   RICHARD F. SUITOR  *
0800             ;*                      *
0800             ;*       APPLAYER       *
0800             ;*                      *
0800             ;*   COPYRIGHT (C) 1981 *
0800             ;*    MICRO INK, INC.   *
0800             ;* CHELMSFORD, MA 01824 *
0800             ;*  ALL RIGHTS RESERVED *
0800             ;*                      *
0800             ;************************
0800             ;
0800             ;
0800             ;
0800             ;
0860                    ORG $0860
0860                    OBJ $0860
0860             ;
0860             ;
0860 EA          TIME   NOP
0861 EA                 NOP
0862 EA                 NOP
0863 88          TIMEA  DEY
0864 8545               STA $0045     ;ANY INNOCUOUS 3 CYCLE INSTRUCTION
0866 D0FB               BNE TIMEA     ;BASIC 8 CYCLE LOOP
0868 F005               BEQ TIMEC
086A 88          TIMEB  DEY
086B EA                 NOP
086C EA                 NOP
086D D0F4               BNE TIMEA
086F 2404        TIMEC  BIT $0004     ;START CHECK OF BIT PATTERN
0871 38                 SEC           ;IN 2, 3, AND 4
0872 3002               BMI TIMED
0874 EA                 NOP
0875 18                 CLC
0876 2603        TIMED  ROL $0003
0878 2602               ROL $0002
087A 2604               ROL $0004
087C 9003               BCC TIMEE
087E AD30C0             LDA $C030     ;TOGGLE SPEAKER
0881 C606        TIMEE  DEC $0006     ;DURATION OF NOTE IN
0883 D005               BNE TIMEF     ;NO. OF CYCLES IN LOCATIONS
0885 C607               DEC $0007     ;6 AND 7
0887 D005               BNE TIMEG
0889 60                 RTS
088A EA          TIMEF  NOP           ;TIMING EQUALIZATION
088B EA                 NOP
088C D000               BNE TIMEG
088E A405        TIMEG  LDY $0005
0890 6C0000             JMP ($0000)
0893             ;
0893             ;SCALING ROUTINE FOR CYCLE DURATION
0893             ;CALCULATION LOC 6,7 = A REG *LOC
0893             ;50, 51
0893             ;
0893 8545        SCALE  STA $0045
0895 A900               LDA #$00
0897 8506               STA $0006
0899 8507               STA $0007
089B A205               LDX #$05
089D 18                 CLC
089E 6607        SCALEX ROR $0007
08A0 6606               ROR $0006
08A2 4645               LSR $0045
08A4 900C               BCC SCALEA
08A6 A506               LDA $0006
```

```
08A8 6550            ADC $0050
08AA 8506            STA $0006
08AC A507            LDA $0007
08AE 6551            ADC $0051
08B0 8507            STA $0007
08B2 CA      SCALEA  DEX
08B3 10E9            BPL SCALEX
08B5 E607            INC $0007       ; SIMPLE LOGIC IN TIMING ROUTI
08B7 60              RTS
08BE                 ORG $08BE
08BE         ;
08BE         ;NOTE PLAYING ROUTINE Y REG
08BE         ;HAS HALF-TONE INDEX
08BE         ;
08BE A512    NOTE    LDA $0012       ;NOTE LENGTH
08C0 8552            STA $0052
08C2 A50F            LDA $000F       ;NOTE TABLE OFFSET
08C4 8510            STA $0010
08C6 B110            LDA ($0010),Y ;LOW ORDER BYTE OF
08C8 38              SEC             ;MACHINE CYCLES PER PERIOD
08C9 8554            STA $0054
08CB E935            SBC #$35        ;CYCLES USED UP TIMING OVERHEAD
08CD 8508            STA $0008
08CF C8              INY
08D0 B110            LDA ($0010),Y ;HIGH ORDER BYTE  OF MACHINE
08D2 8555            STA $0055       ;CYCLES PER PERIOD
08D4 E900            SBC #$00
08D6 8509            STA $0009
08D8 A900            LDA #$00
08DA 8550            STA $0050
08DC 8551            STA $0051
08DE 8553            STA $0053
08E0 A010            LDY #$10
08E2 202403          JSR $0324
08E5         ;
08E5         ;
08E5         ;THE ROUTINE AT $324 EMULATES THE OLD
08E5         ;MONITOR DIVIDE ROUTINE, WHICH DIVIDES
08E5         ;LOCS 54,55 BY 52,53 AND LEAVES THE
08E5         ;RESULT IN 50,51 FOR THE SCALING
08E5         ;ROUTINE. THIS DIVIDE ROUTINE IS LISTED
08E5         ;IN THE REFERENCE MANUAL ON P.162 ($FB81)
08E5         ;
08E5 A508            LDA $0008
08E7 48              PHA
08E8 4609            LSR $0009
08EA 6A              ROR
08EB 4609            LSR $0009
08ED 6A              ROR
08EE 4609            LSR $0009
08F0 6A              ROR
08F1 8505            STA $0005       ;NO. OF 8 CYCLE LOOPS
08F3 68              PLA
08F4 2907            AND #$07        ;LEFT OVER CYCLES DETERMINE
08F6 AA              TAX             ;ENTRY POINT
08F7 BDF809          LDA TTABLE,X    ;TABLE OF ENTRY POINTS
08FA 8500            STA $0000       ; FOR TIMING LOOP
08FC A50E            LDA $000E       ;NOTE DURATION, QUARTER,
08FE 38              SEC             ;HALF
08FF E50D            SBC $000D       ;REST PART OF NOTE
0901 F00F            BEQ NOTEB       ;IF NOTHING TO DO
0903 209308          JSR SCALE       ;SCALING ROUTINE
0906 A202            LDX #$02        :START PATTERN LOAD
0908 B50A    NOTEA   LDA $0A,X
090A 9502            STA $02,X
090C CA              DEX
090D 10F9            BPL NOTEA
090F 206F08          JSR TIMEC      ;TIMING ROUTINE
```

```
0912 A50D      NOTEB  LDA $000D       ;REST PART OF NOTE
0914 F00E             BEQ MAIN        ;IF NOTHING TO DO
0916 209308           JSR SCALE       ;SCALING ROUTINE
0919 A900             LDA #$00
091B 8502             STA $0002       ;ZERO OUT PATTERN FOR
091D 8503             STA $0003       ;REST PART
091F 8504             STA $0004
0921 206F08           JSR TIMEC       ;TIMING
0924                  ORG $0924
0924           ;
0924           ;MAIN PART OF INTERPRETER
0924           ;ENTRY AT "ENTRY"
0924           ;
0924 E613      MAIN   INC $0013       ;TABLE ADDRESS
0926 D002             BNE ENTRY
0928 E614             INC $0014
092A A000      ENTRY  LDY #$00
092C B113             LDA ($0013),Y   ;NEXT TABLE BYTE
092E 3012             BMI MAINA       ;TO CONTROL SECTION
0930 48               PHA
0931 290F             AND #$0F        ;TONE
0933 0A               ASL
0934 A8               TAY
0935 68               PLA
0936 2970             AND #$70        ;DURATION
0938 4A               LSR
0939 4A               LSR
093A 4A               LSR
093B 6902             ADC #$02        ;TOTAL DURATION IN 16THS
093D 850E             STA $000E
093F 4CBE08           JMP NOTE        ;PLAY NOTE
0942 C9FD      MAINA  CMP #$FD        ;C0 + 3D IS LONGEST NOTE FOR
0944 9001             BCC MAINB       ;SCALING REASONS
0946 60               RTS
0947 48        MAINB  PHA
0948 0A               ASL
0949 1007             BPL MAINC
094B 68               PLA
094C 293F             AND #$3F        ;NOTE LENGTH
094E 8512             STA $0012
0950 B0D2             BCS MAIN        ;UNCONDITIONAL BRANCH
0952 0A        MAINC  ASL
0953 1008             BPL MAIND
0955 68               PLA
0956 291F             AND #$1F        ;TONAL RANGE INDEX
0958 0A               ASL
0959 850F             STA $000F
095B 90C7             BCC MAIN        ;UNCONDITIONAL BRANCH
095D 0A        MAIND  ASL
095E 1007             BPL MAINE
0960 68               PLA
0961 290F             AND #$0F        ;REST FRACTION
0963 850D             STA $000D
0965 90BD             BCC MAIN        ;UNCONDITIONAL BRANCH
0967 0A        MAINE  ASL
0968 1003             BPL MAING
096A 68        MAINF  PLA
096B 90B7             BCC MAIN        ;DUMMY, CONTROLS NOT INTERPRETED
096D 0A        MAING  ASL
096E 30FA             BMI MAINF
0970 0A               ASL
0971 102B             BPL MAINI
0973 68               PLA
0974 AA               TAX             ;JSR AND JMP SECTION
0975 4A               LSR
0976 900A             BCC MAINH
0978 A513             LDA $0013       ;JSR SECTION, PUSH RETURN TABLE
097A 6901             ADC #$01        ;ADDRESS ON TO STACK
097C 48               PHA
```

```
097D A514                LDA $0014
097F 6900                ADC #$00
0981 48                  PHA
0982 C8         MAINH    INY
0983 B113                LDA ($0013),Y    ;GET NEW ADDRESS
0985 48                  PHA
0986 C8                  INY
0987 B113                LDA ($0013),Y
0989 8514                STA $0014
098B 68                  PLA
098C 8513                STA $0013
098E 8A                  TXA              ;AND STORE IT FROM BEGINNING
098F 4A                  LSR              ;OF SELECTION
0990 9098                BCC ENTRY        ;JMP
0992 202A09              JSR ENTRY        ;JSR
0995 68                  PLA
0996 8514                STA $0014        ;PULL ADDRESS AND STORE IT
0998 68                  PLA
0999 8513                STA $0013
099B 18                  CLC
099C 9086                BCC MAIN         ;UNCONDITIONAL BRANCH
099E 68         MAINI    PLA
099F A003                LDY #$03         ;GET NEW PATTERN AND
09A1 B113       MAINJ    LDA ($0013),Y    ;STORE IT
09A3 990900              STA $0009,Y
09A6 88                  DEY
09A7 D0F8                BNE MAINJ
09A9 A513                LDA $0013
09AB 6903                ADC #$03         ;JUMP OVER PATTERN
09AD 8513                STA $0013
09AF 9002                BCC MAINK
09B1 E614                INC $0014
09B3 4C2409     MAINK    JMP MAIN
09C0                     ORG $09C0
09C0            ;
09C0            ;INITIALIZATION FOR ZERO PAGE
09C0            ;
09C0 D8         ZERO     CLD              ;JUST IN CASE
09C1 A900                LDA #$00
09C3 8510                STA $0010
09C5 A908                LDA #$08
09C7 8511                STA $0011
09C9 8501                STA $0001
09CB A90A                LDA #$0A
09CD 8514                STA $0014        ;NOTE TABLE PAGE
09CF A920                LDA #$20
09D1 8513                STA $0013        ;NOTE TABLE BYTE
09D3 A901                LDA #$01
09D5 850D                STA $000D        ;REST 16THS
09D7 A920                LDA #$20
09D9 8512                STA $0012        ;NOTE LENGTH, CONTROLS TEMPO
09DB A920                LDA #$20
09DD 850F                STA $000F        ;TONAL RANGE INDEX
09DF A905                LDA #$05
09E1 850A                STA $000A        ;WAVE SHAPE PATTERN
09E3 850B                STA $000B
09E5 850C                STA $000C
09E7 202A09              JSR ENTRY        ;TO APPLAYER
09EA 4C69FF              JMP $FF69        ;TO MONITOR, AFTER THE BEEP
09F8                     ORG $09F8
09F8            ;
09F8            ;TABLE OF ENTRY POINTS FOR TIMING ROUTINE
09F8            ;
09F8 636A62     TTABLE HEX 636A626D616C606B
09FB 6D616C
09FE 606B
                         END
```

```
          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERC PAGE VARIABLES:


** ABSOLUTE VARABLES/LABELS

TIME    0860  TIMEA  0863  TIMEB  086A TIMEC  086F TIMED  0876 TIMEE 0881
TIMEF   088A  TIMEG  088E  SCALE  0893 SCALEX 089E SCALEA 08B2 NCTE  08BE
NCTEA   0908  NOTEB  0912  MAIN   0924 ENTRY  092A MAINA  0942 MAINB 0947
MAINC   0952  MAIND  095D  MAINE  0967 MAINF  096A MAING  096D MAINH 0982
MAINI   099E  MAINJ  09A1  MAINK  09B3 ZERO   09C0 TTABLE 09F8

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:00FA
```

```
  1 REM  ************************
  2 REM  *                      *
  3 REM  *    APPLAYER MUSIC    *
  4 REM  *     INTERPRETER      *
  5 REM  *                      *
  6 REM  *   BY RICHARD SUITOR  *
  7 REM  *                      *
  8 REM  *     APPLAYER MENU    *
  9 REM  *                      *
 10 REM  *  COPYRIGHT (C) 1981  *
 11 REM  *    MICRO INK, INC.   *
 12 REM  * CHELMSFORD, MA 01824 *
 13 REM  *  ALL RIGHTS RESERVED *
 14 REM  *                      *
 15 REM  ************************
 16 REM
 17 REM
 18 PRINT "BLOAD APPLAYER"
 19 LOMEM:4095
 20 START=2496:LOBYTE=2512
 30 IMAX=2
100 CALL -936
110 TAB 13
120 PRINT "APPLAYER MENU"
130 VTAB 4
140 PRINT "1  -  TWINKLE, TWINKLE"
150 PRINT "2  -  TURKEY IN THE STRAW"
195 VTAB 19
200 INPUT "WHICH NUMBER",I
220 IF I<0 OR I>IMAX THEN 100
230 IF I=0 THEN END
240 IF I=1 THEN J=32
250 IF I=2 THEN J=0
300 POKE LOBYTE,J
320 CALL START
350 GOTO 100
```

Improved Star Battle Sound Effects

by William M. Shryock, Jr.

A long, long time ago... in a motion picture studio far, far away... there was a special effects team working on a science fiction epic. And they asked... "What would a star-battle sound like?"... and the Apple II answered......

```
 1 REM    ******************************
 2 REM    * STAR BATTLE SOUND EFFECTS *
 3 REM    *            BY              *
 4 REM    *    WILLIAM SHRYOCK, JR.    *
 5 REM    *    COPYRIGHT (C) 1981      *
 6 REM    *      MICRO INK, INC.       *
 7 REM    *    ALL RIGHTS RESERVED     *
 8 REM    ******************************
10 POKE 0,160: POKE 1,1: POKE 2,162: POKE 3,0: POKE 4,138: POKE 5,24: POKE
   6,233: POKE 7,1: POKE 8,208: POKE 9,252: POKE 10,141
20 POKE 11,48: POKE 12,192: POKE 13,232: POKE 14,224: POKE 15,150: POKE
   16,208: POKE 17,242: POKE 18,136: POKE 19,208: POKE 20,237: POKE 21
   ,96
30 CALL -936: VTAB 12: TAB 9: PRINT "STAR BATTLE SOUND EFFECTS"
40 SHOTS= RND (15)+1
50 LENGTH= RND (11)*10+120
60 POKE 1,SHOTS: POKE 15,LENGTH: CALL 0
70 FOR DELAY=1 TO RND (1000): NEXT DELAY
80 GOTO 40
```

This version can be used in Lo-Res programs without having to reset HIMEM. Also it can be loaded from BASIC.

Galacti-Cube

by Bob Bishop

You are the Captain of a starship exploring the outer limits of our universe. You have discovered a gigantic cube floating in space. Through the only opening you have flown your ship inside, but now you can't find your way back out!

GALACTI-CUBE is a simple maze game in three dimensions. You are in a 3 × 3 × 3 array of cubical compartments and must find your way out in no more than 40 moves, or else you lose. Moves are made by hitting the keys N, S, E, W, U, or D to move north, south, east, west, up or down, respectively. Although it appears small, a 3 × 3 × 3 cubical maze actually has 27 rooms in it, which can make the task of finding your way through deceptively non-trivial.

The program is written entirely in Apple II Integer BASIC and requires at least 8K bytes of memory. In fact, since the program uses no machine language, graphics, or special sound effects, it could probably be converted over to other CRT-type computers (such as the PET, TRS-80, etc.) without too much difficulty.

```
 10 REM  *************************
 12 REM  *                       *
 14 REM  *      GALACTI-CUBE     *
 16 REM  *      R.J.  BISHOP     *
 18 REM  *                       *
 20 REM  *   COPYRIGHT (C) 1981  *
 22 REM  *     MICRO INK, INC.   *
 24 REM  * CHELMSFORD, MA 01824  *
 26 REM  *  ALL RIGHTS RESERVED  *
 28 REM  *                       *
 29 REM  *************************
 30 DIM BOX(27),QUE(27),NODE(6),BIT(6),A$(5)
 40 GOSUB 9000
 50 GOSUB 1000
 60 VTAB 23: TAB 5: PRINT "(HIT ANY KEY TO START THE GAME) ";
 70 GOSUB 4000: GOSUB 5000
 90 LOC=14:OLD=LOC:FUEL=40
100 REM MAIN LOOP
110 GOSUB 2000
150 CALL -936: PRINT : PRINT : PRINT "   COMMAND:"
160 PRINT : TAB 7: GOSUB 4000: CALL -936
165 IF A$="" THEN 150
170 IF A$(1,1)#"F" THEN 250
180 CALL -936: PRINT : PRINT "  YOU HAVE ";FUEL
```

```
190 PRINT : PRINT "  FUEL UNITS"
210 FOR K=1 TO 1000: NEXT K: GOTO 150
250 Z=(OLD-1)/9+1
260 Y=(((OLD-1)/3) MOD 3)+1
270 X=((OLD-1) MOD 3)+1
300 IF A$="E" THEN X=X+1
310 IF A$="W" THEN X=X-1
320 IF A$="N" THEN Y=Y+1
330 IF A$="S" THEN Y=Y-1
340 IF A$="U" THEN Z=Z+1
350 IF A$="D" THEN Z=Z-1
360 LOC=X+3*(Y-1)+9*(Z-1)
370 IF LOC<>OLD THEN 390
380 PRINT "": GOTO 150
390 IF X<1 OR X>3 OR Y<1 OR Y>3 THEN 700
400 IF BOX(OLD)>=32 AND Z=0 THEN 800
410 VAL=BOX(OLD): IF VAL>=32 THEN VAL=VAL-32
420 IF VAL>=16 AND Z=4 THEN 800
430 IF Z<1 OR Z>3 THEN 700
450 BITS=BOX(OLD)
460 WAY=BITS-2*(BITS/2):BITS=BITS/2
470 IF WAY=0 AND A$="E" THEN 700
480 WAY=BITS-2*(BITS/2):BITS=BITS/2
490 IF WAY=0 AND A$="W" THEN 700
500 WAY=BITS-2*(BITS/2):BITS=BITS/2
505 IF WAY=0 AND A$="N" THEN 700
510 WAY=BITS-2*(BITS/2):BITS=BITS/2
515 IF WAY=0 AND A$="S" THEN 700
520 WAY=BITS-2*(BITS/2):BITS=BITS/2
525 IF WAY=0 AND A$="U" THEN 700
530 WAY=BITS-2*(BITS/2):BITS=BITS/2
535 IF WAY=0 AND A$="D" THEN 700
540 WAY=BITS-2*(BITS/2):BITS=BITS/2
550 FUEL=FUEL-1: IF FUEL>0 THEN 100
560 CALL -936: PRINT "   YOU ARE"
565 PRINT
570 PRINT "   OUT OF"
575 PRINT
580 PRINT "    FUEL!";
590 GOTO 830
700 CALL -936: PRINT " THAT DIREC-"
710 PRINT : PRINT " TION HAS AN"
720 PRINT : PRINT " OBSTRUCTION";
730 FOR K=1 TO 1000: NEXT K: GOTO 150
800 CALL -936: PRINT "YOU FOUND THE"
810 PRINT : PRINT " EXIT IN ONLY"
820 PRINT : PRINT "  ";41-FUEL;" MOVES!";
830 GOSUB 2700
840 FOR K=1 TO 2500: NEXT K
850 CALL -936: END
900 END
1000 REM GENERATE THE MAZE
1010 FOR K=1 TO 27
1020 BOX(K)=128
1030 NEXT K
1040 BOX(14)=0
1050 QUE(1)=14:QBIG=1
1060 XQBIG=1
1100 FOR K=1 TO QBIG
1110 IND=QUE(K)
1140 KNT=0:ROAD=1:DEL=1
1150 FOR J=0 TO 2
1160 SET=3*DEL
1170 FOR L=0 TO 1
1180 NDX=IND+DEL
1190 IF NDX<1 THEN 1400
1200 IF (NDX-1)/SET<>(IND-1)/SET THEN 1400
1250 IF BOX(NDX)<128 THEN 1400
1300 KNT=KNT+1:NODE(KNT)=NDX:BIT(KNT)=ROAD
```

```
1400 DEL=-DEL:ROAD=ROAD+ROAD
1450 NEXT L
1460 DEL=SET
1470 NEXT J
1500 IF KNT=0 THEN 1600
1510 NDX= RND (KNT)+1:XQBIG=XQBIG+1
1520 QUE(XQBIG)=NODE(NDX)
1530 BOX(IND)=BOX(IND)+BIT(NDX)
1540 TIB=2*BIT(NDX)
1550 IF TIB=4 OR TIB=16 OR TIB=64 THEN TIB=TIB/4
1590 BOX(NODE(NDX))=BOX(NODE(NDX))+TIB-128
1600 NEXT K
1610 QBIG=XQBIG: IF QBIG<27 THEN 1100
1700 HOLE=2* RND (2)+6* RND (2)+18* RND (2)+1
1710 OPEN=16: IF HOLE<14 THEN OPEN=32
1720 BOX(HOLE)=BOX(HOLE)+OPEN
1800 RETURN
2000 REM UPDATE THE DISPLAY
2005 GOSUB 2700
2010 Z=(OLD-1)/9+1
2020 Y=(((OLD-1)/3) MOD 3)+1
2030 X=((OLD-1) MOD 3)+1
2040 VTAB 13-Y-Y
2050 TAB 8*Z+X+X-7
2060 PRINT "-"
2110 Z=(LOC-1)/9+1
2120 Y=(((LOC-1)/3) MOD 3)+1
2130 X=((LOC-1) MOD 3)+1
2140 VTAB 13-Y-Y
2150 TAB 8*Z+X+X-7
2170 POKE PEEK (36)+ PEEK (40)+256* PEEK (41),109
2200 BITS=BOX(LOC)
2210 VT=20:T=34:A$="EAST": GOSUB 2500
2220 VT=22:T=34:A$="WEST": GOSUB 2500
2230 VT=20:T=28:A$="NORTH": GOSUB 2500
2240 VT=22:T=28:A$="SOUTH": GOSUB 2500
2250 VT=20:T=24:A$="UP": GOSUB 2500
2260 VT=22:T=23:A$="DOWN": GOSUB 2500
2300 GOSUB 2600
2400 OLD=LOC
2450 RETURN
2500 WAY=BITS-2*(BITS/2):BITS=BITS/2
2510 MODE=127: IF WAY THEN MODE=255
2520 POKE 50,MODE: VTAB VT: TAB T: PRINT A$: POKE 50,255
2550 RETURN
2600 VTAB 19: TAB 5
2610 POKE 32,2
2630 POKE 33,14
2660 POKE 34,17
2680 POKE 35,22
2690 RETURN
2700 POKE 32,0
2710 POKE 33,40
2720 POKE 34,0
2730 POKE 35,24
2750 RETURN
4000 REM 'GET' FROM THE KEYBOARD
4010 POKE -16368,0
4020 CHAR= PEEK (-16384): IF CHAR<128 THEN 4020
4030 POKE -16368,0:A$="?"
4080 IF CHAR=141 THEN A$=""
4090 IF CHAR=196 THEN A$="D"
4100 IF CHAR=197 THEN A$="E"
4110 IF CHAR=198 THEN A$="F"
4120 IF CHAR=206 THEN A$="N"
4130 IF CHAR=211 THEN A$="S"
4140 IF CHAR=213 THEN A$="U"
4150 IF CHAR=215 THEN A$="W"
4200 RETURN
```

```
5000 REM DRAW DISPLAY
5010 CALL -936: PRINT "      YOUR LOCATION           COMPASS"
5020 PRINT : PRINT "  (BOT)   (MID)   (TOP)      REFERENCE"
5030 PRINT : TAB 34: PRINT "N"
5040 PRINT : TAB 34: PRINT "!"
5050 TAB 34: PRINT "!"
5060 TAB 29: PRINT "W <--*--> E"
5070 TAB 34: PRINT "!"
5080 TAB 34: PRINT "!"
5090 PRINT : TAB 34: PRINT "S"
5100 VTAB 6
5110 FOR K=1 TO 3
5120 PRINT : PRINT "  - - -   - - -   - - -"
5130 NEXT K
5140 VTAB 16: TAB 21: PRINT "OBSTRUCTION SENSORS"
5200 POKE 50,63
5210 VTAB 5: PRINT "                               "
5220 FOR K=1 TO 7
5230 PRINT " ";: TAB 9: PRINT " ";: TAB 17: PRINT " ";: TAB 25:PRINT " "

5240 NEXT K
5250 PRINT "                               "
5300 VTAB 18: TAB 21: PRINT "                   "
5310 FOR K=1 TO 5
5320 TAB 21: PRINT " ";: TAB 39: PRINT " "
5330 NEXT K
5340 TAB 21: PRINT "                   ";
5400 VTAB 15: PRINT
5410 PRINT "                  "
5420 FOR K=1 TO 7
5430 PRINT " ";: TAB 18: PRINT " "
5440 NEXT K
5450 PRINT "                  ";
5500 POKE 50,255
5900 RETURN
9000 CALL -936: VTAB 10
9010 TAB 10: PRINT "*** GALACTI-CUBE ***"
9020 PRINT : TAB 19: PRINT "BY"
9030 PRINT : TAB 14: PRINT "ROBERT BISHOP"
9040 FOR K=1 TO 1500: NEXT K
9050 CALL -936
9110 PRINT "     YOU ARE THE CAPTAIN OF A STAR-SHIP"
9120 PRINT "EXPLORING THE OUTER LIMITS OF OUR UNI-"
9130 PRINT "VERSE.  YOU HAVE DISCOVERED A GIGANTIC"
9140 PRINT "CUBE FLOATING IN SPACE.  THROUGH THE"
9150 PRINT "ONLY OPENING YOU HAVE FLOWN YOUR SHIP"
9160 PRINT "INSIDE, BUT NOW YOU CAN'T FIND YOUR WAY"
9170 PRINT "BACK OUT!"
9190 PRINT "     FROM YOUR EXPLORATIONS YOU HAVE"
9200 PRINT "LEARNED THAT THE CUBE IS DIVIDED INTO"
9210 PRINT "AN ARRAY OF 3X3X3 CUBICAL COMPARTMENTS"
9220 PRINT "AND YOU ARE CURRENTLY IN THE CENTER-"
9230 PRINT "MOST ONE."
9250 PRINT "     YOUR SHIP IS EQUIPPED WITH A DIS-"
9260 PRINT "PLAY INDICATING YOUR LOCATION.  THE"
9270 PRINT "OBSTRUCTION SENSORS INDICATE WHICH DI-"
9280 PRINT "RECTIONS (FLASHING) ARE BLOCKED.  YOU"
9310 PRINT "MOVE YOUR SHIP BY HITTING THE FIRST"
9320 PRINT "LETTER OF THE DIRECTION YOU WANT TO GO."
9330 PRINT "YOUR FUEL SUPPLY (WHICH IS DISPLAYED BY"
9340 PRINT "HITTING THE LETTER, F) WILL ONLY LET"
9350 PRINT "YOU MAKE UP TO 40 MOVES.  GOOD LUCK!"
9999 RETURN
```

5
HARDWARE

INTRODUCTION

On a rainy weekend day, when there is nothing to do around the house, what better project could there possibly be than to interface some external hardware to your Apple. The Apple computer is equipped with several easy-to-use input and output ports. The articles in this section describe how to use them, and provide some interesting construction projects as well.

''The Color Gun for the Apple II,'' by Neil Lipson, describes how to build and interface a simple photocell array to the Apple. When used with the described software, this array can discern color. Robert Stein's ''A Cassette Operating System for the Apple II'' provides a means to file and store named programs on cassette tape. ''BASIC and Machine Language Transfers with the Micromodem II,'' by George Dombrowski, discusses techniques for program transfers using a popular communications interface.

''A Digital Thermometer for the Apple II,'' by Carl Kershner, discusses how to interface a thermistor to the Apple so that the Apple can provide a temperature display. Finally, ''KIM and SYM Format Casssette Tapes on the Apple II,'' by Steven Welch, provides a KIM-1 format tape dump capability for the Apple, using a special routine which outputs to the cassette port.

The Color Gun for the Apple II

by Neil D. Lipson

The Apple produces many colors—but what about recognizing them? With some quite inexpensive hardware, you can turn your Apple II into a color detector—a device which will automatically determine the colors of any object. So who says the Apple is color blind?

Shortly after I developed my light pen for the Apple back in May, 1978, I began thinking about other devices that could be hooked up to the paddle inputs. One idea was making a "color gun" which when pointed at an object would tell you the color. The idea is similar to that of the operation of a television transmitter. Color is broken down into three main colors, which are red, blue, and yellow. Therefore by having three inputs into the Apple, into paddle 0, paddle 1, and paddle 2, we could in effect have a device that would "see" the three color breakdown ratios of any object. By further analyzing this ratio, we could see different shades of color and with high quality color filters, we could make an extremely accurate device which could even give the exact color temperature of the object. One of the interesting aspects of this device that sets it apart from any other color temperature meter, is that you can calibrate it by pointing it at a piece of white paper to adjust for differences in the light source. Therefore, the color gun will work in any type of artificial lighting within certain parameters. (You could not use it under a red light for example.)

Building the Color Gun

To start off, buy three sensitive cadium sulfide photo cells (physically between ¼ to ½ inch in diameter). If the cells are not equal in sensitivity, they can be equalized easily in software. (This is illustrated in the listing.) Merely point the gun at a white piece of paper (or at the light source itself if it's not too bright) during the calibration procedure.

The construction of the gun is very simple. Mount the three cells in a triangle about 2″ for each side on a piece of wood or other material. Then place three filters over the cells, with red on paddle 0 cell, blue on paddle 1 cell, and yellow on paddle 2 cell. The purer the filter, the better. Photographic filters are the best, and will give the best results. However, red, blue or yellow clear plastic will work satisfactorily in most situations. Note the use of the REM statements in the program. These are for slowing down the paddle readings just a hair in order to avoid having the readings "overlap". The wiring diagram is shown in figure 1.

Mount the entire setup in some type of barrel or cylinder about 4 inches long, with the inside of the barrel painted white. Glue everything together and seal against light leaks. Plug it into the game paddle after the wiring is complete and you are ready to go. For the pin numbers of the paddles, consult your reference manual.

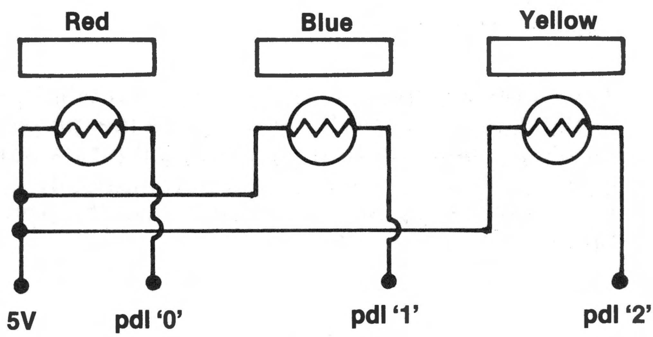


Figure 1

The Color Gun Program

Enter the Applesoft program, and run it. The gun will only recognize 6 colors, and when it isn't sure what the color is, it will give you two colors (one primary color and one secondary). This should not happen if the colors are absolutely pure, but most colors are not, so expect this situation often.

Notice the correction algorithm in statement 70 in the program to correct for the blue cell. The cells that I used were somewhat more sensitive to blue than the other colors (which is common of cadium sulfide). This was noticed when the color gun kept saying "orange" (the compliment of blue). The correction algorithm eliminates most of this problem. If the gun acts strangely, run it again until it gets a good calibration. It sometimes takes more than one run to get it working properly (usually because it is confused by a bright color nearby).

By fine tuning the software, and using more exact ratios, you can determine many other colors. Given enough ratios to choose from, you can give the color temperature of the object (with high quality cells and filters). The typical photographic filters you can use are the yellow (K2), the red (25 or 25A) and the blue (47). These may be varied if desired to meet the spectral response of the particular cell you buy. You could even use different colors in the filters as long as you adjust the software accordingly. Buy the smallest filter you can (it only has to cover about ½ inch diameter), but make sure there is no light leak from the sides of the cells. If you follow these instructions the gun will work perfectly the first time around. Have fun!

```
1  REM  **************************
2  REM  *                        *
3  REM  *       COLOR GUN        *
4  REM  *      NEIL LIPSON       *
5  REM  *                        *
6  REM  *  COPYRIGHT (C) 1981    *
7  REM  *     MICRO INK, INC.    *
8  REM  * CHELMSFORD, MA 01824   *
9  REM  *  ALL RIGHTS RESERVED   *
10  REM *                        *
11  REM **************************
14  REM
15  CALL  - 936: VTAB 10: HTAB 10: PRINT "COLOR GUN BY NEIL D. LIPSON":
    FOR I = 1 TO 2000: NEXT I
17  REM  YELLOW-2
18  REM  BLUE  -1
19  REM  RED   -0
20  REM
22  CALL  - 936: PRINT : PRINT : PRINT : PRINT
25  GOSUB 1000
30  CALL  - 936: PRINT : PRINT
32 A =  PDL (0)
35  REM
40 B =  PDL (1)
45  REM
50 C =  PDL (2)
55  REM
60 A = A * A1
61 B = B * B1
62 C = C * C1
70 B = B / 1.5
100  PRINT "RED CELL=";A
110  PRINT "BLUE CELL=";B
115  PRINT "YELLOW CELL=";C
116  PRINT : PRINT
117  PRINT "THE COLOR IS:": PRINT
118  PRINT "**********************"
121  IF C < B AND C < (A) THEN  PRINT "YELLOW"
123  IF A < B AND A < C THEN  PRINT "RED"
124  IF A > B AND A > C THEN  PRINT "GREEN"
125  IF B > A AND B > C THEN  PRINT "ORANGE"
126  IF C < AC > B THEN  PRINT "PURPLE"
129  IF B < C AND B < (A) THEN  PRINT "BLUE"
130  PRINT "**********************"
131  FOR X = 1 TO 2300: NEXT X
140  GOTO 30
200  END
1000  CALL  - 936: PRINT
1010  PRINT "POINT GUN AT A WHITE SHEET OF PAPER"
1020  FOR I = 1 TO 1500: NEXT I
1030 A1 =  PDL (0)
1035  REM
1040 B1 =  PDL (1)
1045  REM
1050 C1 =  PDL (2)
1055  PRINT "A1=";A1
1056  PRINT "B1=";B1
1057  PRINT "C1=";C1
1060 D1 = A1 * B1 * C1
1070 A1 = D1 / A1
1080 B1 = D1 / B1
1090 C1 = D1 / C1
1100  PRINT "CORRECTION FACTOR FOR RED = ";A1
1110  PRINT "CORRECTION FACTOR FOR BLUE = ";B1
1120  PRINT "CORRECTION FACTOR FOR YELLOW = ";C1
1125  FOR I = 1 TO 2000: NEXT I
1130  RETURN
10000  END
```

A Cassette Operating System for the Apple II

by Robert A. Stein, Jr.

Have you ever wished that, as great as the Apple II computer system is, you were able to load programs by name from a library cassette? Well, with this mini-sized cassette operating system you can stack many programs on one cassette and load the one you want by typing in its name. Great for showing off your system without juggling a dozen or so cassette tapes.

The Cassette Operating System [CASSOS] resides in memory at locations 02C0 to 03FF, where it won't get clobbered by BASIC programs or initialization. Add the optional cassette control circuit, or purchase one of the commercially available ones (Candex Pacific, 693 Veterans BLVD, Redwood City, CA 94063), and you never need envy the PET for its loading technique again.

Operation

First, load CASSOS into memory. To load a program using CASSOS, depress CTRL-Y and RETURN. "PROG?" will be displayed, enter a 1-10 character program name. The cassette tape will be searched and the program loaded if found. "XXXXXXXXXX LOADED" will be output, where XXXXXXXXXX is the program now in memory. If the cassette control circuit (described later) is present the tape will also be stopped. A line of question marks (?????????) is displayed if the request program was not found. To write a program to the library cassette enter Yc (CTRL-Y), "WRITE", and RETURN. Program will be saved under the name requested at PROG?. "XXXXXXXXXX OUT" will be displayed at completion and the recorder stopped. To end a cassette program file enter: Yc, "EOF", RETURN; a special record header will be written. Note that to conserve limited memory space the EOF routine utilizes the program write subroutine so the "XXXXXXXXXX OUT" message should be ignored.

The program is structured such that the last 63 locations of the input buffer are used for display messages, so if more than 191 characters are entered at one time the program will still function, but without messages. The listing as presented was for a 48K system with DOS; change location 0358 as follows for a different configuration:

Without DOS		With DOS
1F— 8K	5F—24K	35—24K
2F—12K	7F—32K	55—32K
3F—16K	8F—36K	65—36K
4F—20K	BF—48K	95—48K

Program Design

The method by which CASSOS functions is to write a program header block consisting of header ID, program name, and start of the BASIC load. This is followed by the program data itself, utilizing the Apple monitor routines.

A Cassette On/Off Circuit

The following diagram describes a simple circuit for stopping and starting a cassette recorder which has a ''remote'' plug from the Apple II under program control. The theory involves activating or deactivating the AN3 signal on the Apple game connector. A store to location CO5F turns the recorder on and location CO5E turns it off. The strobe triggers a transistor which in turn opens a relay and closes the connection to the remote plug, starting the recorder. If your recorder requires an open connection to start tape movement wire the relay normally closed instead of open. It is also possible to add a relay that would interrupt power to the recorder for control if you have no remote capability on your recorder.

Parts List

All parts were purchased at a local electronics store
- 6VDC Relay (275-004)
- NPN Transistor (2N3568 or equivalent)
- 1000 Ohm Resistor
- 250 Ohm Resistor
- Mini-Plug

All connections were made to a DIP Header which was modified by soldering a 16-pin IC to it so that the game paddles could be used without modification when the cassette ON/Off circuit was in use. The common 6VDC relay was modified to be triggered by the game connector signals by wiring a 2500 ohm resistance (utilizing a series of resistors connected in series so that the sum is 2500 Ohms) in parallel with the relay coil. If your recorder's rewind controls are disabled by the remote jack, wire a switch to bypass the transistor between chasis ground and the relay, which will allow the rewind to operate when depressed. If all this is beyond your scope simply stop, then start the recorder manually.

```
  1 REM  ************************
  2 REM  *                      *
  3 REM  *    CASSETTE  O.S.    *
  4 REM  *   BY ROBERT STEIN    *
  5 REM  *                      *
  6 REM  *       DIRECTORY      *
  7 REM  *                      *
  8 REM  *  COPYRIGHT (C) 1981  *
  9 REM  *   MICRO INK, INC.    *
 10 REM  * CHELMSFORD, MA 01824 *
 11 REM  *  ALL RIGHTS RESERVED *
 12 REM  *                      *
 13 REM  ************************
 14 REM
 15 REM
 16 REM
 20 N=1: CALL -936: VTAB (10): DIM X$(1)
 25 INPUT "INSERT LIBRARY TAPE AND DEPRESS 'RETURN'",X$
 30 POKE -16289,0: CALL -936: GOSUB 300
 40 PRINT "FILE #  PROGRAM NAME  BYTES"
 50 PRINT "------  ------------  -----"
 60 CALL 840: CALL -259
 70 IF PEEK (688)= ASC("E") THEN 210
 80 IF PEEK (688)= ASC("S") THEN 200
100 REM  LOAD PROGRAM INTO MEMORY BELOW THE DIRECTORY PROGRAM.
105 D= PEEK (856)-3
110 POKE 60, PEEK (700): POKE 61,( PEEK (701)-3)
120 POKE 62,255: POKE 63,D: CALL -259
130 PRINT N,: POKE 789,2: POKE 788,177: CALL 785
140 M=( PEEK (700)/2)+ PEEK (701)*128
150 L=2*(( PEEK (856)*128+128)-M):N=N+1
160 PRINT "    ";L: GOTO 60
200 GOSUB 300: PRINT "NO EOF MARK"
210 POKE -16290,0: GOSUB 300
230 PRINT : PRINT "***END OF FILE***"
240 CALL -155
300 FOR I=1 TO 30
305 L= PEEK (-16336)+ PEEK (-16336): NEXT I
310 CALL -1059: RETURN
```

```
0800        ;************************
0800        ;*                      *
0800        ;*    CASSETTE  O.S.    *
0800        ;*   BY ROBERT STEIN    *
0800        ;*                      *
0800        ;*        CASSOS        *
0800        ;*                      *
0800        ;*  COPYRIGHT (C) 1981  *
0800        ;*   MICRO INK, INC.    *
0800        ;* CHELMSFORD, MA 01824 *
0800        ;*  ALL RIGHTS RESERVED *
0800        ;*                      *
0800        ;************************
0800        ;
0800        ;
0800        SLO    EPZ $3C           ;TAPE BUFFER START/END
0800        SHI    EPZ $3D
0800        ELO    EPZ $3E
0800        EHI    EPZ $3F
0800        OFFSET EPZ $50           ;OFFSET STORAGE
0800        SAVEY  EPZ $60           ;SAVE Y-REG
0800        IN     EPZ $60           ;INPUT PARAMETERS
0800        INLO   EPZ $60
0800        INHI   EPZ $61
0800        PPL    EPZ $CA           ;INTEGER BASIC PROGRAM
0800        PPH    EPZ $CB           ; POINTER
0800        ;
```

```
0800               FCHAR  EQU $0201          ;1ST CHARACTER IN BUFFER
0800               WBUF   EQU $0200          ;WORK BUFFER
0800               IN1    EQU $02A3          ;PROG. NAME INPUT BUFFER
0800               ID     EQU $02B0          ;HEADER ID, 'S' OR '-'
0800               NAME   EQU $02B1          ;PROGRAM NAME
0800               PEND   EQU $02BB          ;END SENTINAL (FF)
0800               PHL    EQU $02BC          ;BASIC TOP
0800               PHH    EQU $02BD
0800               ;
0800               CLRAN3 EQU $C05F          ;CLEARS GAME I/O AN3
0800               SETAN3 EQU $C05E          ;SETS GAME I/O AN3
0800               BASIC2 EQU $E003          ;INTEGER BASIC WARM START
0800               BELL   EQU $FBDD          ;MONITOR BEEP ROUTINE
0800               CR     EQU $FC62          ;MONITOR CARRIAGE RETURN
0800               GETLN2 EQU $FD6C          ;MONITOR INPUT ROUTINE
0800               COUT   EQU $FDED          ;MONITOR OUTPUT ROUTINE
0800               WRITE  EQU $FECD          ;MONITOR TAPE WRITE
0800               READ   EQU $FEFD          ;MONITOR TAPE READ
0800               ;
02C0                      ORG $2C0
02C0                      OBJ $800
02C0               ;
02C0 A9D3          PWRITE LDA #$D3           ;SET LABEL ID TO 'S'
02C2 8DB002               STA ID
02C5 A9B1                 LDA #NAME          ;OFFSET TO BUFFER
02C7 206703               JSR INIT
02CA A9FF          WEOF   LDA #$FF           ;LABEL SENTINAL
02CC 8DBB02               STA PEND
02CF A5CA                 LDA PPL            ;STORE TOP OF PROGRAMADDRESS
02D1 8DBC02               STA PHL
02D4 A5CB                 LDA PPH
02D6 8DBD02               STA PHH
02D9 20CDFE               JSR WRITE          ;WRITE LABEL
02DC A4CA                 LDY PPL
02DE A5CB                 LDA PPH
02E0 206003               JSR SETS           ;SET TOP WRITE/HIMEM BOTTOM
02E3 20CDFE               JSR WRITE          ;WRITE PROGRAM
02E6 A9EB                 LDA #OUT           ;SET TO WRITTEN MESSAGE
02E8 207E03               JSR ECHO           ;PRINT XXXXXXXXXX OUT
02EB               ;
02EB 87A0CF        OUT    HEX 87A0CFD5D4FF   ;" OUT" MESSAGE
02EE D5D4FF
02F1 87A0CC        LOADED HEX 87A0CCCFC1C4C5C4FF ;" LOADED" MESSAGE
02F4 CFC1C4
02F7 C5C4FF
02FA D0D2CF        PROG?  HEX D0D2CFC7BFFF   ;" PROG?" MESSAGE
02FD C7BFFF
0300               ;
0300 A202          TYPE3  LDX #$02           ;SET HI ADDRESS TO 02
0302 D007                 BNE TYPE           ;BRANCH TO MAIN ROUTINE
0304 8460          NLTYPE STY SAVEY
0306 2062FC               JSR CR             ;OUTPUT CR/LF
0309 A460                 LDY SAVEY          ;RESTORE Y
030B 8E1503        TYPE   STX CONT+2         ;MODIFY LOAD INSTRUCTION
030E 8C1403               STY CONT+1
0311 A000                 LDY #$00           ;SET I-VALUE
0313 B9FA02        CONT   LDA PROG?,Y        ;GET CHARACTER
0316 C9FF                 CMP #$FF           ;DELIMETER?
0318 F02D                 BEQ TDONE          ;YES- RETURN
031A 20EDFD               JSR COUT           ;OUTPUT
031D C8                   INY                ;INCREMENT INDEX
031E D0F3                 BNE CONT           ;CONTINUE (JMP)
0320               ;
0320 48            INPUT  PHA                ;SAVE INPUT COUNT
0321 A902                 LDA /IN1           ;SET HI INPUT ADDRESS
0323 8660                 STX INLO           ;STORE ADDRESS
0325 8561                 STA INHI           ;(PHA & LDA TO CHG HI)
0327 A9A0                 LDA #$A0           ;SET PROMPT TO " "
0329 206CFD               JSR GETLN2         ;INPUT TO COMMON BUFFER
```

```
032C 68              PLA              ;RESTORE COUNT
032D AA              TAX              ;SET TO X
032E A000            LDY #$00         ;SET Y-INDEX
0330 B90002  MOVE    LDA WBUF,Y       ;LOAD FROM WORK BUFFER
0333 C98D            CMP #$8D         ;LAST INPUT?
0335 F008            BEQ CR1          ;YES
0337 9160            STA (IN),Y       ;STORE IN USER AREA
0339 C8              INY              ;INCREMENT POINTER
033A CA              DEX              ;DECREMENT COUNTER
033B F00A            BEQ TDONE        ;RETURN IF DONE
033D D0F1            BNE MOVE         ;ELSE BRANCH TO LOOP
033F A9A0    CR1     LDA #$A0
0341 9160            STA (IN),Y       ;SPACE FILL
0343 C8              INY
0344 CA              DEX
0345 D0F8            BNE CR1          ;LOOP TILL MAXIMUM
0347 60      TDONE   RTS              ;RETURN
0348         ;
0348 A0B0    SLBL    LDY #ID          ;SET ID LABEL ADDRESS
034A A200            LDX #$00         ;SET START FLAG
034C 205103          JSR SEC          ;SET-UP TO SET END TOO
034F A0BD            LDY #PHH         ;SET END OF LABEL
0351 A902    SEC     LDA #$02
0353 D004            BNE SET          ;BRANCH TO SET START
0355 A0FF    SHIM    LDY #$FF         ;SET HIMEM:
0357 A995            LDA #$95         ;(CHANGE FOR MORE MEMORY)
0359 953D    SET     STA SHI,X        ;SET START
035B 943C            STY SLO,X        ; OR END
035D E8              INX              ;BUMP END BY 2 FOR
035E E8              INX              ;END PAIR
035F 60              RTS
0360 A200    SETS    LDX #$00
0362 205903          JSR SET          ;SET BASIC TOP & BOTTOM
0365 D0EE            BNE SHIM
0367         ;
0367 8550    INIT    STA OFFSET       ;STORE INBUF OFFSET
0369 A202            LDX /PROG?       ;SET " PROG?" ADDRESS
036B A0FA            LDY #PROG?
036D 200403          JSR NLTYPE       ;OUPUT WITH NL
0370 204803          JSR SLBL         ;SET LABEL PARAMETERS
0373 A90A            LDA #$0A         ;INPUT = 10 CHARACTERS
0375 A650            LDX OFFSET       ;USER INPUT OFFSET
0377 202003          JSR INPUT        ;INPUT PROGRAM NAME
037A 8D5FC0          STA CLRAN3       ;TURN ON CASSETTE
037D 60              RTS
037E         ;
037E 48      ECHO    PHA              ;STORE OFFSET
037F 8D5EC0          STA SETAN3       ;TURN OFF CASSETTE
0382 A202            LDX /NAME        ;SET TO OUTPUT LABEL NAME
0384 A0B1            LDY #NAME
0386 200403          JSR NLTYPE
0389 68              PLA              ;GET MESSAGE
038A A8              TAY              ;PUT IN Y FOR TYPE
038B 200003          JSR TYPE3        ;OUTPUT " OUT" OR " LOADED"
038E 4C03E0          JMP BASIC2
0391         ;
0391 A9A3    PLOAD   LDA #IN1         ;INPUT PROGRAM NAME
0393 206703          JSR INIT         ; TO IN1 ($2A3)
0396 204803  TRYAGN  JSR SLBL         ;SET LABEL PARAMS.
0399 20FDFE          JSR READ         ;READ LABEL
039C ADB002          LDA ID           ;GET ID
039F C9D3            CMP #"S"
03A1 D029            BNE NFOUND       ;EOF OR NOT ON TAPE
03A3 ACBC02          LDY PHL
03A6 ADBD02          LDA PHH
03A9 206003          JSR SETS         ;READ PROGRAM PARAMETERS
03AC 20FDFE          JSR READ         ;READ PROGRAM
03AF A200            LDX #$00         ;SET INDEX
03B1 BDB102  TEST    LDA NAME,X       ;COMPARE FOUND NAME
```

```
03B4 DDA302          CMP IN1,X      ; WITH INPUT NAME
03B7 D0DD            BNE TRYAGN
03B9 E8              INX
03BA E00A            CPX #$0A       ;CHECK ALL LOOKED AT
03BC D0F3            BNE TEST
03BE ADBC02          LDA PHL        ;SET TOP OF BASIC ADDRESS
03C1 85CA            STA PPL
03C3 ADBD02          LDA PHH
03C6 85CB            STA PPH
03C8 A9F1            LDA #LOADED    ;SET TO " LOADED"
03CA D0B2            BNE ECHO       ;OUTPUT WITH VERIFY NAME
03CC 8D5EC0   NFOUND STA SETAN3     ;TURN OFF CASSETTE
03CF A220            LDX #$20
03D1 A9BF     NC     LDA #$BF       ;PRINT ??????????
03D3 20EDFD          JSR COUT
03D6 CA              DEX
03D7 D0F8            BNE NC         ;LOOP
03D9 20DDFB          JSR BELL       ;SOUND TONE
03DC F0B3            BEQ PLOAD      ;RETURN FOR NEW NAME
03DE          ;
03DE AD0102   WHICH  LDA FCHAR      ;FIRST CHAR OF FUNCTON (E,R,W)
03E1 C9D7            CMP #"W"       ;"WRITE"
03E3 F010            BEQ SAVE
03E5 C9C5            CMP #"E"       ;"EOF"
03E7 D0A8            BNE PLOAD      ;"READ"
03E9 8DB002          STA ID         ;STORE E AS ID IN LABEL
03EC 204803          JSR SLBL       ;SET LABEL PARAMETERS
03EF 8D5FC0          STA CLRAN3     ;TURN ON CASSETTE
03F2 4CCA02          JMP WEOF       ;BRANCH TO WRITE EOF
03F5 4CC002   SAVE   JMP PWRITE     ;BRANCH TO WRITE PROGRAM
03F8          ;
03F8 4CDE03   CTRLY  JMP WHICH      ;CONTROL-Y TRANSFER TO CHECK FN
03FB 0000     NMI    HEX 0000       ;NMI VECTOR
03FD 0000     IRQ    HEX 0000       ;IRQ VECTOR
03FF 00              HEX 00
                     END

***** END OF ASSEMBLY
```

```
          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5  *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

SLO     003C  SHI     003D  ELO     003E  EHI     003F  OFFSET  0050  SAVEY 0060
IN      0060  INLO    0060  INHI    0061  PPL     00CA  PPH     00CB

** ABSOLUTE VARABLES/LABELS

FCHAR   0201
WBUF    0200  IN1     02A3  ID      02B0  NAME    02B1  PEND    02BB  PHL   02BC
PHH     02BD  CLRAN3  C05F  SETAN3  C05E  BASIC2  E003  BELL    FBDD  CR    FC62
GETLN2  FD6C  COUT    FDED  WRITE   FECD  READ    FEFD  PWRITE  02C0  WEOF  02CA
OUT     02EB  LOADED  02F1  PROG?   02FA  TYPE3   0300  NLTYPE  0304  TYPE  030B
CONT    0313  INPUT   0320  MOVE    0330  CR1     033F  TDONE   0347  SLBL  0348
SEC     0351  SHIM    0355  SET     0359  SETS    0360  INIT    0367  ECHO  037E
PLOAD   0391  TRYAGN  0396  TEST    03B1  NFOUND  03CC  NC      03D1  WHICH 03DE
SAVE    03F5  CTRLY   03F8  NMI     03FB  IRQ     03FD

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:01E2
```

BASIC and Machine Language Transfers with the Micromodem II

by George J. Dombrowski, Jr.

The D.C. Hayes Micromodem is one of the most popular communications interfaces available for the Apple. With such an interface, it becomes possible to transfer programs between your Apple and remote computers. Here are a couple of routines which facilitate transfers of BASIC and machine language programs between two Apples.

There is no doubt that the Micromodem II, produced by D.C. Hayes Associates for the Apple II, is a very sophisticated telecommunications device. I purchased a Micromodem several months ago and have been pleased with its performance ever since. This device couples directly with Ma Bell and can be easily programmed to automatically answer your phone or even to transmit short messages to other machines.

One of the best features provided by D.C. Hayes Associates is the well-documented 85 page manual, complete with example programs. However, despite the quality of this manual, there is a glaring omission. I originally purchased the Micromodem II with the notion of easily transferring machine language and BASIC programs to other Apple owners. Although the manual details a procedure for adapting Apple Computer's Datamover program to the Micromodem firmware, easier more direct methods of sending BASIC programs to another computer were not described. This article describes an immediate mode procedure for transferring BASIC programs and also provides an Applesoft routine for sending machine language programs or binary data to another Apple II.

Sending a BASIC program in immediate mode is a simple matter using the Micromodem II. Once the phone connection has been established, the receiving computer must be placed in remote mode by sending a CTRL R followed by PR #S where S = modem slot #. When the BASIC prompt appears, remote control of the Apple at the other end has been achieved. The receiving computer is now waiting input. It will accept commands and input from its own keyboard, your keyboard or those issued automatically by your computer during program execution. In other words, the receiving computer will accept a LISTing of a program sent from another computer and interpret each line as a command. Before LISTing the program, however, a few additonal steps must be taken to set up both computers for the transfer.

Once remote control of the receiving machine has been established, the appropriate BASIC must be initialized by typing either the INT or FP DOS command. At this point output from the remote computer should be directed to the video port by executing a PR#0. This is a precautionary step to prevent the accidental transmission of messages generated by the receiving machine's command interpreter. These messages could be received by the sending computer and interfere with the program transfer. The operator of the sending computer will not see the BASIC prompt return after this command. In order to LIST the program on your computer, terminal mode must be exited by typing CTRL-A/CTRL-X. The receiving Apple is left in remote mode waiting for input, while the sending computer is set up to LIST the program.

Although this procedure seems complicated, after using it a few times it is easy to remember. For those of you who like to sit back and watch your machine do the work, the following program will create an EXEC file for this purpose.

From now on the commands typed at the local keyboard will not be sent to the remote machine. First, the firmware carriage-return-delay for out-going data must be set by typing POKE 1912 + S,18 followed by POKE 1528 + S,80. The pause after each carriage return allows sufficient time for the receiving machine to interpret and execute each line before another is sent. Register 1528 + *S* normally contains decimal 3 in terminal mode, which corresponds to a delay of 30 msec. Second, the program to be sent is loaded and the LIST formatting routine disabled by typing POKE 33,30. Finally, a PR #2 is issued and after the cursor returns (0.8 sec), the LIST command given.

Apple is left in remote mode waiting for input, while the sending computer is set up to LIST the program.

Run this program to create the EXEC file, and then LOAD the program you want to send. Finally, EXEC BASIC PROGRAM TRANSFER. This EXEC file will work with either BASIC. The user's machine will be placed in terminal mode when the transfer is finished. PR #2 must then be issued to the remote computer to receive its output.

Binary data or machine language programs can be transmitted in a similar fashion by employing a modified version of the monitor hexadecimal dump routine. Ordinarily upon hitting RETURN this routine displays a hexadecimal address followed by a hyphen following the address. The substitution is necessary because the monitor interpreter requires a colon to immediately follow the address when binary data is input. The change was accomplished by relocating a small portion of the F8 ROM chip ($FD92-$FDC5) to RAM memory at $1000-$1033. Address $100D was altered from $A0 ("-") to $BA (":"). In addition, the address for the JSR instruction at $1021-1023 was changed from $FD92 to

$1000. This HEX dump routine has been incorporated into an Applesoft BASIC program which takes care of the housekeeping chores described above for transferring BASIC programs plus a few more.

Applesoft Binary Transfer with the Micromodem II

Although these methods require little software and are easy to implement, they do have a disadvantage. The time required to send BASIC and machine language programs using these techniques is greater (approximately 20% and 130%, respectively) than would be expected from the time calculated based upon program length. This is because both Integer BASIC and Applesoft programs are stored in memory with reserved words tokenized. Tokenized words such as PRINT, POKE, or NEXT require only one byte of memory. Sending a byte at 300 baud takes about 1/30 second; however, with the LISTing procedure described here, transmitting a reserved word such as PRINT requires approximately 5/30's of a second.

Similarly, with machine language programs, for every 8 bytes of data transferred, a 4 digit hexadecimal address, colon, 8 pairs of hexadecimal data, and 8 spaces must be sent. A total of 29 characters are sent for every 8 bytes of memory.

In spite of this disadvantage, these techniques are handy for sending medium sized programs over short distances where time is not a costly factor.

NOTE: These programs were designed for the Micromodem to reside in slot 2. If another slot is chosen, registers 1530 and 1914 in the page listings must be changed to 1528 + S and 1912 + S, respectively where S = the Modem Slot Number.

```
1  REM *************************
2  REM *                       *
3  REM * MICROMODEM TRANSFERS *
4  REM *   GEORGE DOMBROWSKI   *
5  REM *                       *
6  REM *    BINARY TRANSFER    *
7  REM *                       *
8  REM *  COPYRIGHT (C) 1981   *
9  REM *    MICRO INK, INC.    *
10  REM * CHELMSFORD, MA 01824 *
11  REM *  ALL RIGHTS RESERVED *
12  REM *                      *
13  REM *************************
14  REM
15  REM
19  REM  BINARY TRANSFER/MICROMODEM II
20 D$ =  CHR$ (4)
30  PRINT D$"NOMON C,I,O"
40  GOSUB 420
50  INPUT "IS RECEIVING COMPUTER IN REMOTE MODE WITH EITHER BASIC INITIAL
     IZED?";ANS$
60  PRINT
70  IF  LEFT$ (ANS$,1) <  > "Y" THEN  PRINT "TRANSFER ADANDONED": END
80  POKE 1530,60: POKE 1914,18: REM  600 MSEC WAIT AFTER CARRIAGE RETURN.
     AUTO LINE FEED IS ACIVATED AND THE WAIT FUNCTION + LOCAL DISPLAY ENA
     BLED.
90  PRINT "STARTING ADDRESS-": INPUT "(MUST END WITH 0 OR 8)";ST$
100  REM  LINES 110/170 - HEXIDECIMAL TO DECIMAL CONVERSION.
110 Z$ = "0123456789ABCDEF"
120  FOR I =  LEN (ST$) TO 1 STEP  - 1
130  FOR J = 1 TO  LEN (Z$)
140  IF  MID$ (Z$,J,1) <  >  MID$ (ST$,I,1) THEN  NEXT J
150 DEC = DEC + (J - 1) * (16 ^ X)
160 X = X + 1: NEXT I
170 HB =  INT (DEC / 256):LB = DEC - (HB * 256)
180  REM  LINE 190 PLACES THE DECIMAL EQUIVALENTS OF THE HIGH & LOW BYTE
     ADDRESS INTO THE PAGE 0 LOCATIONS USED BY THE MEMORY DUMP ROUTINE.
190  POKE 61,HB: POKE 60,LB
200  INPUT "NUMBER OF BYTES (DECIMAL) ";NB
210  PRINT : INVERSE : HTAB 6: PRINT "HITTING ANY KEY ABORTS TRANSFER":
     NORMAL
220  PRINT D$"IN #0"
230  PRINT D$"PR #2"
240  PRINT "CALL-151"
250  PRINT : REM  SENDS CARRIAGE RETURN.
260  FOR I = 1 TO  INT (NB / 8) + 1
270  IF  PEEK ( - 16384) > 127 THEN  POKE  - 16368,0: GOTO 300
280  CALL 4113: REM  CALLS MACHINE LANGUAGE ROUTINE BELOW.
290  NEXT I
300  PRINT
310  PRINT "3DOG"
320  PRINT D$"PR #0"
330  PRINT
340  POKE 1530,3: REM  NORMAL 30 MSEC WAIT
350  PRINT "               *** ALL DONE ***"
360  PRINT : PRINT "THE SENDING COMPUTERIS NOW IN TERMINAL MODE & THE REC
     EIVING COMPUTER HAS BEEN RETURNED WITH BASIC UP IN REMOTE MODE."
370  PRINT : INVERSE : HTAB 15: PRINT "HIT RETURN": NORMAL
380  PRINT D$"IN #2"
390  POKE 1914,138: REM  INITIATE TEMINAL MODE/FULL-DUPLEX (USE 10 FOR
     HALF-DUPLEX).
400  END
410  REM  LINES 420/450 LOAD RELOCATED MEMORY DUMP ROUTINE AT $1000.
420  FOR M = 4096 TO 4147: READ D: POKE M,D: NEXT M
430  RETURN
440  DATA  164,61,166,60,32,142,253,32,64,249,160,0,169,186,76,237,253,16
     5,60,9,7,133,62,165,61,133,63,165,60,41,7,208,3,32,0,16
450  DATA  169,160,32,237,253,177,60,32,218,253,32,186,252,144,232,96
460  REM   THE BASIC PRGM + DUMP ROUTINE OCCUPY $800-$1040. IF THE BINARY
     DATA TO BE SENT RESIDES IN THIS RANGE, IT MUST FIRST BE RELOCATED  W
     ITH THE MONITOR MOVE COMMAND.
```

```
1  REM  *************************
2  REM  *                       *
3  REM  * MICROMODEM TRANSFERS  *
4  REM  *   GEORGE DOMBROWSKI   *
5  REM  *                       *
6  REM  *    BASIC TRANSFER     *
7  REM  *                       *
8  REM  *  COPYRIGHT (C) 1981   *
9  REM  *    MICRO INK, INC.    *
10  REM * CHELMSFORD, MA 01824  *
11  REM *  ALL RIGHTS RESERVED  *
12  REM *                       *
13  REM *************************
14  REM
15  REM
16  REM  BASIC TRANSFER/MICROMODEM II
20  REM  FIRST RUN THIS PROGRAM AND THEN
30  REM  ESTABLISH REMOTE CONTROL OF RECEIVING MACHINE
40  REM  LEAVE TERMINAL MODE BY TYPING CTRL-A/CTRL-X
50  REM  THEN TYPE <EXEC BASIC PROGRAM TRANSFER>
60 D$ =  CHR$ (4)
70  PRINT D$"OPEN BASIC PROGRAM TRANSFER"
80  PRINT D$"WRITE BASIC PROGRAM TRANSFER"
90  PRINT "POKE 1530,80:REM FOR LONG FLOATING POINT PROGRAMS A GREATER DE
     LAY MAY BE REQUIRED."
100  PRINT "POKE 1914,18"
110  PRINT "POKE 33,30"
120  PRINT "IN#0"
130  PRINT "PR#2"
140  PRINT "LIST"
150  PRINT "PR#0"
160  PRINT "IN#2"
170  PRINT "TEXT"
180  PRINT "POKE 1530,3"
190  PRINT "POKE 1914,138"
200  PRINT D$"CLOSE"
210  END
```

A Digital Thermometer for the Apple II

by Carl J. Kershner

Can the Apple II tell the temperature? Thermistor probes can be connected directly to the Apple II Game I/O Connector and their output signals processed via a linearizing algorithm to produce a digital display in both degrees Celsius and Fahrenheit. This article explains how.

A thermistor temperature measuring probe can be directly connected to the Apple II computer via its built-in Game I/O Connector. This is possible since thermistors are "thermal resistors" which exhibit large resistance changes in response to a change in temperature. Paddle input ports, PDL(0,1,2,&3), on the Apple are essentially eight bit A/D converters for such variable resistance sources.

The Apple and the thermistor are quite suited for one another since the inherent nonlinearity of the thermistor can be easily handled with a simple algorithm in software. In addition, the small current drain during the sampling cycle of the RC network on the Apple's 553 timer closely approaches the ideal zero-power operating condition for a thermistor. Both the nonlinearity and the induced temperature due to the probing current have been particularly troublesome characteristics which engineers have had to find ways of working around when applying thermistors.

The program written in Applesoft consists of an input section, a data reduction section and a display section. The input section calls for the selection of a paddle input and two thermistor specifications used by most manufacturers: the room temperature resistance designated as RO and a value representing the ratio of the resistance at 25°C to that at 50°C designated as RA. The selected paddle input is then read and scaled to represent the resistance value at the input port. The corresponding temperature in both degrees Celsius and Fahrenheit are calculated from the resistance via a temperature-resistance relationship:

$$R_1/R_2 = e^{\beta(1/T_1 - 1/T_2)}$$

where R_1 and R_2 are the resistances at the absolute temperature T_1 and T_2 respectively, and β is a constant for the particular thermistor material. The results are rounded to the nearest integer and displayed in a three-digit format with the blanking of leading zeros and a negative sign for temperatures below zero.

A thermistor probe can be connected to the Apple II by merely attaching one of its leads to the +5 volt supply, pin 1, and the other to one of the PDL ports, pins 6,7, 10, or 11 on the Game I/O connector J 14. No other components or modifications are required so long as a thermistor is chosen with a room temperature resistance and ratio which suits the temperature range and sensitivity desired for application. A 40,000 ohm thermistor with a ratio of 9 or 10 will provide at least one degree Fahrenheit sensitivity and a working range suitable for an indoor thermometer application. The best way to choose a thermistor for your particular application is to run the program using a game paddle as input, enter values for RO and RA from a manufacturer's specification sheet, and observe the useful operating range and sensitivity of the selected thermistor. This latter procedure demonstrates the additional usefulness of the program as an engineering design aid in selecting a thermistor for other applications.

Thermistors suitable for this application can be purchased for less than five dollars from most supply houses or directly from a manufacturer. A Fenwal GA44P2 glass probe type thermistor with a room temperature resistance of 40,000 ohms and a ratio of 9.53 is a good choice for an indoor thermometer application, whereas a Fenwal GA42P2 with a room temperature resistance of 15,000 ohms and a ratio of 9.1 is a good compromise for indoor-outdoor use. It is best to house the thermistor probe in a small metal tube to protect it from mechanical damage and to provide thermal inertia to minimize effects of short-term temperature transients. It is also advisable to calibrate the thermistor probes against a laboratory type thermometer, if high accuracy is desired, because the manufacturing tolerances on RO and RA values for the inexpensive probes described here are generally no better than ± 10%.

Because thermistors can be used that have relatively high resistances, transmission line and contact temperature effects can be neglected and the probes can be situated far from the computer console. Thus the Apple II digital thermometer can perform many useful temperature monitoring tasks in and around the house.

The Fenwal products mentioned in this article can be purchased from Fenwal Electronics, 63 Fountain St., PO Box 585, Framingham, MA 01701.

```
10  REM  ************************
15  REM  *                      *
20  REM  *  DIGITAL THERMOMETER *
25  REM  *     CARL KERSHNER    *
30  REM  *                      *
35  REM  *      THERMOMETER     *
40  REM  *                      *
45  REM  *  COPYRIGHT (C) 1981  *
50  REM  *    MICRO INK, INC.   *
55  REM  * CHELMSFORD, MA 01824 *
60  REM  *  ALL RIGHTS RESERVED *
65  REM  *                      *
70  REM  ************************
80  REM
90  REM
100 REM   DIGITAL THERMOMETER FOR THERMISTOR PROBE(DISPLAYS BOTH CELCIUS
     &FAHRENHEIT)
110 PRINT "WHICH INPUT DO YOU WANT(0,1,2,3)": INPUT NUMBER
120 PRINT "WHAT THERMISTOR CONSTANTS DO YOU WANT(RO,RATIO)": INPUT RO,RA

125 BETA = 1.7636E3 *  LOG (RA)
130  HOME : REM   CLEAR SCREEN
140  REM  PRINT TEMPERATURE SCALE CHARACTERS
150  GR : COLOR= 15
160 HLIN 26,27 AT 6: HLIN 26,27 AT 7: HLIN 26,27 AT 9: HLIN 26,27 AT 10:
     VLIN 7,9 AT 25: VLIN 7,9 AT 28
170 HLIN 34,38 AT 9: HLIN 34,38 AT 10: HLIN 34,36 AT 14: HLIN 34,36 AT 1
    5: VLIN 9,20 AT 33
180 HLIN 26,27 AT 23: HLIN 26,27 AT 24: HLIN 26,27 AT 26: HLIN 26,27 AT
    27: VLIN 24,26 AT 25: VLIN 24,26 AT 28
190 VLIN 28,29 AT 38: VLIN 27,28 AT 37: VLIN 26,27 AT 36: VLIN 26,27 AT
    35: VLIN 27,28 AT 34
200 VLIN 28,35 AT 33: VLIN 35,36 AT 34: VLIN 36,37 AT 35: VLIN 36,37 AT
    36: VLIN 35,36 AT 37: VLIN 34,35 AT 38
210 T = 298: REM   SET T(0) AT 298 DEGREES ABSOLUTE
220 RI = 589.94 *  PDL (NUMBER): REM   READ INPUT & SCALE TO OHMS
230  IF RI = 0 THEN RI = 1: REM   PREVENT DIVISION BY ZERO
240 TC =  INT (1 / (1 / T -  LOG (RO / RI) / BETA) - 272.5): REM CALCUL
     ATE TEMPERATURE IN DEGREES CELCIUS AND ROUND TO NEAREST INTEGER
245  IF  ABS (TC) > 999 THEN  GOTO 220: REM   LIMIT OVERFLOWING DISPLAY
250 SIGN = 0
260  IF TC < 0 THEN SIGN = 15
270  COLOR= SIGN
280  HLIN 3,5 AT 29: HLIN 3,5 AT 30: REM   DISPLAY NEGATIVE SIGN
290 TC =  ABS (TC)
300 J =  INT (TC / 100):I = J: REM   SEPARATE HUNDRED'S DIGIT
310  IF J = 0 THEN J = 10: REM   BLANK LEADING ZERO
320 X = 1:Y = 26: GOSUB 1000: REM   DISPLAY CELCIUS HUNDRED'S
330 J =  INT ((TC - J * 100) / 10): REM   SEPARATE TEN'S DIGIT
340  IF I = 0 AND J = 0 THEN J = 10: REM BLANK BOTH HUNDRED'S AND TEN'S
      LEADING ZEROS IF J&I ARE BOTH ZERO
350 X = 9:Y = 26: GOSUB 1000: REM   DISPLAY CELCIUS TEN'S DIGIT
360 J = TC - I * 100 - J * 10: REM   SEPARATE ONE'S DIGIT
370 X = 17:Y = 26: GOSUB 1000: REM   DISPLAY CELCIUS ONE'S DIGIT
380 TF =  INT (9 * (1 / (1 / T -  LOG (RO / RI) / BETA) - 273) /5 + 32.5
     ): REM   CALCULATE FAHRENHEIT & ROUND TO NEAREST INTEGER
390 SIGN = 0
400  IF TF < 0 THEN SIGN = 15
410  COLOR= SIGN
420  HLIN 3,5 AT 12: HLIN 3,5 AT 13: REM   DISPLAY NEGATIVE SIGN
430 TF =  ABS (TF)
440 J =  INT (TF / 100):I = J: REM  SEPARATE HUNDRED'S DIGIT
450  IF J = 0 THEN J = 10: REM   BLANK LEADING ZERO
460 X = 1:Y = 9: GOSUB 1000: REM   DISPLAY FAHRENHEIT HUNDRED'S DIGIT
470 J =  INT ((TF - J * 100) / 10): REM   SEPARATE TEN'S DIGIT
480  IF I = 0 AND J = 0 THEN J = 10: REM BLANK BOTH HUNDRED'S AND TEN'S
      LEADING ZEROS
490 X = 9:Y = 9: GOSUB 1000: REM   DISPLAY FAHRENHEIT TEN'S DIGIT
500 J = TF - I * 100 - J * 10: REM   SEPARATE ONE'S DIGIT
```

```
510 X = 17:Y = 9: GOSUB 1000: REM   DISPLAY FAHRENHEIT ONE'S DIGIT
520  GOTO 220
1000  REM   SEVEN SEGMENT ENCODER
1010  ON J GOTO 1110,1120,1130,1140,1150,1160,1170,1180,1190,1200
1100 A = 15:B = 15:C = 15:D = 15:E = 15:F = 15:G = 0: GOTO 2000
1110 A = 0:B = 15:C = 15:D = 0:E = 0:F = 0:G = 0: GOTO 2000
1120 A = 15:B = 15:C = 0:D = 15:E = 15:F = 0:G = 15: GOTO 2000
1130 A = 15:B = 15:C = 15:D = 15:E = 0:F = 0:G = 15: GOTO 2000
1140 A = 0:B = 15:C = 15:D = 0:E = 0:F = 15:G = 15: GOTO 2000
1150 A = 15:B = 0:C = 15:D = 15:E = 0:F = 15:G = 15: GOTO 2000
1160 A = 15:B = 0:C = 15:D = 15:E = 15:F = 15:G = 15: GOTO 2000
1170 A = 15:B = 15:C = 15:D = 0:E = 0:F = 0:G = 0: GOTO 2000
1180 A = 15:B = 15:C = 15:D = 15:E = 15:F = 15:G = 15: GOTO 2000
1190 A = 15:B = 15:C = 15:D = 15:E = 0:F = 15:G = 15: GOTO 2000
1200 A = 0:B = 0:C = 0:D = 0:E = 0:F = 0:G = 0:J = 0: GOTO 2000
2000  REM  SEVEN SEGMENT DISPLAY
2010  COLOR= A
2020  HLIN X + 1,X + 4 AT Y
2030  HLIN X + 1,X + 4 AT Y + 1
2040  COLOR= G
2050  HLIN X + 1,X + 4 AT Y + 5
2060  HLIN X + 1,X + 4 AT Y + 6
2070  COLOR= D
2080  HLIN X + 1,X + 4 AT Y + 10
2090  HLIN X + 1,X + 4 AT Y + 11
2100  COLOR= F
2110  VLIN Y + 1,Y + 5 AT X
2120  COLOR= B
2130  VLIN Y + 1,Y + 5 AT X + 5
2140  COLOR= E
2150  VLIN Y + 6,Y + 10 AT X
2160  COLOR= C
2170  VLIN Y + 6,Y + 10 AT X + 5
2180  RETURN
```

KIM and SYM Format Cassette Tapes on the Apple II

by Steven M. Welch

Now you can swap programs and data between your Apple and any AIM, SYM or KIM via cassette I/O.

Many KIM and SYM owners have graduated to bigger and better 6502 systems as their needs and financial situations changed. If you are one of these people, and find that your KIM is sitting in the corner gathering dust because your Apple is so much easier to work with, read on. With this program, you can use your Apple as a "host computer" for assembly language program development and then "down load" the finished program into your single board computer (SBC). Just like the big boys! Not only will you make better use of your several hundred dollar investment, but you will also have the bonus of a new set of computer jargon to bore your friends. The value of developing assembly language programs in this fashion cannot be fully appreciated until you use the Apple to develop a sizeable program for the SYM or KIM. The many miseries of hand assembling magically disappear. The constant verbal self-abuse which generally accompanies calculator keyboard entry and debugging quickly becomes a fading memory. Have you ever forgotten to initialize a loop counter only to realize it 300 bytes of hand assembly later?

The program listed here was produced to fill a need: to develop a large program on a SYM. I estimate that we have saved an absolute minimum of 2 man-months in the development of a 1500-byte program by using the Apple for entry, debugging and assembling. Also, having a real assembler easily available to us, we have written better code and have not needed the numerous patches and kludges which inevitably crop up when one writes large programs in machine code. At the University of Colorado at Boulder, where I am employed, we are developing a microprocessor-controlled Charge Coupled Photo Diode (CCPD) spectrographic detector for the Sommers-Bausch Observatory using a SYM-1 computer. Although this is a very nice SBC, the basic version lacks certain features which are highly desirable in a computer that will be used for program development; e.g., fast mass storage, an assembler, text editor, ASCII keyboard, and display device. It seemed to us that the controlling program was going to take a great deal of time to devise without these several conveniences.

The "big boys" get around the lack of these features by purchasing (usually for $10-20,000), a Microprocessor Development System. While our observatory didn't have the ten or twenty thousand dollars to throw away, we did have access to an Apple II computer belonging to my boss, Dr. Bruce Bohannan. The Apple has almost all of the features of the typical Microprocessor Development System except, perhaps, a means of communicating with the SBC in question. How can an Apple talk to a SYM? Fortunately, both computers use the 6502 micro-processor chip, so programs assembled for the Apple have little or no trouble running on the SYM or KIM. Also, fortunately, all of these machines have a means of reading and writing programs on audio cassettes. It goes without saying, of course, that the tape formats of these machines are totally incompatible. We had to do some translating; either convince the SYM to speak Apple, or convince the Apple to speak SYM. Since it's easier to develop programs on the Apple (that's why I did all this in the first place), I decided to teach my Apple to speak SYM.

It turns out that there is another good reason to teach the Apple SYMese. The SYNERTEK people who make the SYM, have been so kind as to publish listings of the SYM monitor in the back of their manual. This monitor listing has routines in it which produce SYM or KIM cassette tapes. The result is that the program is very easily modified to run on the Apple. No timers are used (the Apple has none), and the serial data is sent out through a single bit of a 6522 output port. Although the Apple doesn't have any 6522s, it does have several single bit outputs, and in particular, it has a single bit output with the level adjusted to be used as a cassette recorder interface. Even though this is not a 6522 output, under certain conditions it can be *thought* of as one. The way that the Apple works, any time the address of the cassette output port appears on the address bus, the cassette output flip-flop changes state. On the other hand, in the SYM we send a particular bit pattern to an address and these bits appear on the output latch.

Basically, what this means is that we can *pretend* that the Apple cassette is the SYM cassette output if we write only to this output when we want to *change* the level of the cassette port. With the Apple, it should be noted, there is no control over the phase of the output signal, but all of the cassette-read routines in question are not sensitive to phase. Fortunately, through good luck or the good planning of the programmers at SYNERTEK, 90% of the cassette output code was written in just this way. This feature makes the program a snap to adapt to the Apple. Once I had picked out the proper pieces of the SYNERTEK code and figured out what they had done, I had only to change a few lines to obtain the results listed here. Since I did not write the program, I won't explain how it works, but I have heavily commented the listing for those readers who are interested.

Using the Program

It is a good idea to make a SYNC tape first. The Apple output level is about ½ of the SYM's output level which may require changing the volume on playback from the usual value. Also, the Apple does not have a high-frequency roll-off capacitor which the SYM uses, and as a result, the tone controls may need adjustment. The SYNC tape enables you to set the controls properly on your tape recorder (as outlined in the SYM manual, Appendix F). To make a SYNC tape, load the SYMOUT program into your Apple, set the mode by setting the parameter, MODE (location $11E0), to $80 for SYM format or to $00 for KIM format and begin the program at SYNC: ($1000). This is an endless loop, so record a few minutes of the output before you hit RESET and use the resultant tape to set the level and tone on the tape recorder when reading it into the SYM (see Appendix F in SYM manual).

Once you have the proper level and tone settings, down-loading your program is fairly easy. First, load the SYMOUT program. Then, load your executable program into RAM. Next, put in the parameters: Starting Address ($11DB-C), Ending Address ($11DD-E), Tape I.D. Number ($11DF), and the MODE ($11E0) and start the program at SYMOUT: ($1080). Record the program, play it into your SYM, and there you have it!

Direct Computer to Computer Communication

A discovery by Dr. Bohannan: If your tape recorder has a monitor hookup, through which you can listen to whatever is being recorded, you can hook up the Apple directly to the SYM and reduce the error rate astronomically! On our SYM we have about a 70% chance of a successful load of our 1500 byte program with our tape recorder, a Sony. The level and tone control settings are extremely critical as well. When the machines are hooked up directly through the monitor jack of our tape recorder, we have success *every* time and the level and tone settings are unimportant. I've also found that several of my tape recorders work very well this way and have the monitor feature through the earphone jack even though it is not marked.

```
0800    ;*************************
0800    ;*                       *
0800    ;*    SYM-KIM FORMAT     *
0800    ;*   CASSETTE  OUTPUT    *
0800    ;*       S.WELCH         *
0800    ;*                       *
0800    ;*       SYM-KIM         *
0800    ;*                       *
0800    ;*  COPYRIGHT (C) 1981   *
0800    ;*    MICRO INK, INC.    *
0800    ;* CHELMSFORD, MA 01824  *
0800    ;*  ALL RIGHTS RESERVED  *
0800    ;*                       *
0800    ;*************************
0800    ;
0800    ;
0800    ;LARGELY COPIED FROM THE
0800    ;SYNERTEX MANUAL, AND RE-
0800    ;PRODUCED HERE WITH THE
0800    ;PERMISSION OF SYNERTEX
0800    ;SYSTEMS CORP.
0800    ;
0800    ;
0800    TAPOUT EQU $C020
0800    ;
0800    ;USE APPLE GAME PADDLE ANNUNCIATOR #0 FOR TAPE RECORDER
0800    ;ON-OFF CONTROL. RECORDER ON IS LOW.
0800    ;
0800    TAPEON EQU $C059              ;PUT 0 HERE TO TURN ON
0800    TAPEOF EQU $C058              ;PUT 1 HERE TO TURN OFF
0800    TM1500 EPZ $47                ;PROB SHOULD BE TWEAKED
0800    TIME99 EPZ $1A                ;FOR DELAY ROUTINE
0800    EOT    EPZ $04
0800    SYN    EPZ $16
0800    BUFADL EPZ $E7                ;ARBITRARY PLACE ON ZERO PAGE
0800    BUFADH EPZ $E8
0800    CHAR   EPZ $EA
```

```
0800              ;
0800              ;
0800              ;---PROGRAM STARTS HERE, LINE 390 OF SYM CODE LOC 8E87
0800              ;
0800              BEGIN  EQU $1080           ;MUST START IN MIDDLE OF PAGE
1080                     ORG BEGIN           ;OUT OF WAY OF MOST SYM PROGRAMS
1080                     OBJ $880
1080              ;
1080              ;--INITIALIZE--
1080 20BB11       SYMOUT JSR START           ;ENTRY-PARAMETERS SET BEFORE CALL
1083 A080                LDY #$80            ;INCASE WE TAKE KIM BRANCH
1085 2CE011              BIT MODE            ;TEST BIT 7 OF MODE (1=SYM,0=KIM)
1088 100D                BPL DUMPT1          ;KIM-DO 128 SYNS
108A              ;
108A              ;--WRITE 8 SECOND MARK--
108A A208                LDX #$8             ;8 TIMES...
108C A015         MARK8A LDY #$15            ;ONE SEC (21 DELAYSPER SEC)
108E 209511       MARK8B JSR DELAY           ;BENIGN PAUSE, SYM USES KIM CHAR
1091 88                  DEY
1092 D0FA                BNE MARK8B
1094 CA                  DEX
1095 D0F5                BNE MARK8A
1097              ;--WRITE 256 SYNS FOR SYNC--
1097 A916         DUMPT1 LDA #SYN
1099 200711              JSR OUTCTX
109C 88                  DEY
109D D0F8                BNE DUMPT1
109F              ;--WRITE START CHARACTER--
109F A92A                LDA #'*'
10A1 200711              JSR OUTCTX
10A4              ;--WRITE ID--
10A4 ADDF11              LDA ID
10A7 203B11              JSR OUTBTX
10AA              ;---WRITE STARTING ADDRESS---
10AA ADDB11              LDA SAL
10AD 203811              JSR OUTBCX
10B0 ADDC11              LDA SAH
10B3 203811              JSR OUTBCX
10B6 2CE011              BIT MODE            ;KIM OR HS?
10B9 100C                BPL DUMPT2
10BB              ;---WRITE ENDING ADDRESS---
10BB ADDD11              LDA EAL
10BE 203811              JSR OUTBCX
10C1 ADDE11              LDA EAH
10C4 203811              JSR OUTBCX
10C7              ;---START OF MEMORY DUMP---
10C7              ;--FIRST CHECK IF THIS IS THE LAST BYTE OUT---
10C7 A5E7         DUMPT2 LDA BUFADL          ;LOAD ADDRESS OF CURRENT BYTE
10C9 CDDD11              CMP EAL
10CC D029                BNE DUMPT4          ;COMPARE TO ENDING ADDRESS
10CE A5E8                LDA BUFADH
10D0 CDDE11              CMP EAH
10D3 D022                BNE DUMPT4          ;BRANCH IF MORE TO OUTPUT
10D5 A92F                LDA #'/'            ;YUP, LAST BYTE: WRITE '/'
10D7 200711              JSR OUTCTX
10DA              ;---WRITE CHECKSUM---
10DA ADE111              LDA CHKL
10DD 203B11              JSR OUTBTX
10E0 ADE211              LDA CHKH
10E3 203B11              JSR OUTBTX
10E6              ;---WRITE TWO EOT'S---
10E6 A904                LDA #EOT
10E8 203B11              JSR OUTBTX
10EB A904                LDA #EOT
10ED 203B11              JSR OUTBTX
10F0              ;---OK, NOW WE'RE ALL DONE, SO CLEAN UP AND EXIT---
10F0 18                  CLC                 ;INDICATE SUCCESS
10F1              ;---SKIPPED LOTS OF STUFF, MOSTLY SYM SPECIFIC---
10F1 A201                LDX #$01            ;SHUT OFF TAPE RECORDER
10F3 8E58C0              STX TAPEOF
10F6 60                  RTS                 ;AND WE'RE ALL DONE
10F7              ;NEXT IS THE CODE WHICH OUTPUTS THENEXT MEM LOCATION
10F7 A000         DUMPT4 LDY #$0             ;FIND THE NEXT BYTE
10F9 B1E7                LDA (BUFADL),Y
10FB 203811              JSR OUTBCX          ;WRITE IT AND UPDATE CHCKSUM
```

```
10FE E6E7           INC BUFADL          ;BUMP BUFFER ADDR
1100 D0C5           BNE DUMPT2
1102 E6E8           INC BUFADH          ;CARRY
1104 4CC710         JMP DUMPT2          ;GO BACK AND SEE IF WE'RE DONE
1107         ;
1107         ;
1107         ;START OF VARIOUS CHARACTER OUT ROUTINES
1107         ;
1107 2CE011  OUTCTX BIT MODE            ;HS OR KIM?
110A 1047           BPL OUTCHT          ;KIM TAKES BRANCH
110C         ;OUTBTH - NO CLOCK A,X DESTROYED
110C         ;MUST RESIDE ON ONE PAGE - TIMING CRITICAL
110C         ;
110C A209    OUTBTH LDX #$9             ;8 BITS+START BIT
110E 8CE411         STY TEMP2
1111 85EA           STA CHAR
1113         ;CAN'T READ LEVEL ON APPLE, SO NEXT INSTRUCTION IS DUMMY
1113 ADE311         LDA TEMP1           ;FOR TIMING
1116 46EA    GETBIT LSR CHAR
1118 49E5           EOR #TPBIT
111A 8D20C0         STA TAPOUT          ;INVERT LEVEL
111D         ;HERE STARTS FIRST 416 USEC PERIOD
111D A047           LDY #TM1500
111F 88      A416   DEY                 ;TIME FOR THIS LOOP IS 5Y-1
1120 D0FD           BNE A416
1122 9011           BCC NOFLIP          ;NOFLIP IF BIT 0
1124 49E5           EOR #TPBIT          ;BIT IS 1 - INVERT OUTPUT
1126 8D20C0         STA TAPOUT          ;END OF FIRST 416 USEC PERIOD
1129 A046    B416   LDY #TM1500-1
112B 88      B416B  DEY                 ;LENGTH OF LOOP IS 5Y-1
112C D0FD           BNE B416B
112E CA             DEX
112F D0E5           BNE GETBIT          ;GET NEX BIT (LAST IS OSTART BIT)
1131 ACE411         LDY TEMP2           ;(BY 9 BIT LSR)
1134 60             RTS
1135 EA      NOFLIP NOP                 ;TIMING
1136 90F1           BCC B416            ;(ALWAYS)
1138         ;
1138 20AC11  OUTBCX JSR CHKT            ;GO UPDATE CHECKSUM
113B 2CE011  OUTBTX BIT MODE
113E 08CC           BM1 OUTBTH          ;HS
1140         ;OUTBTC - OUTPUT ONE KIM BYTE
1140 A8      OUTBTC TAY                 ;SAVE DATA BYTE
1141 4A             LSR
1142 4A             LSR
1143 4A             LSR
1144 4A             LSR                 ;SHIFT HI NIBBLE INTO PLACE
1145 204811         JSR HEXOUT          ;AND OUTPUT HI NIBBLE FIRST
1148 290F    HEXOUT AND #$0F            ;CONVERT LO NIBBLE TO ASCII
114A C90A           CMP #$0A
114C 18             CLC
114D 3002           BMI HEX1
114F 6907           ADC #$07
1151 6930    HEX1   ADC #$30
1153         ;
1153         ;OUTCHT OUTPUTS AND ASCII CHAR IN KIM FORMAT
1153         ; (MUST RESIDE ON ONE PAGE, FOR TIMING)
1153         ;
1153 8EE311  OUTCHT STX TEMP1           ;SAVE X & Y
1156 8CE411         STY TEMP2
1159 85EA           STA CHAR
115B A9FF           LDA #$FF            ; USE FF W/SHIFTS TO COUNT BITS
115D 48      KIMBIT PHA                 ;SAVE BIT COUNTER
115E ADE411         LDA TEMP2           ;DUMMY FOR TIMING
1161 46EA           LSR CHAR            ;GET DATA BIT IN CARRY
1163 A212           LDX #$12            ; ASSUME ONE
1165 B002           BCS HF
1167 A224           LDX #$24            ;BIT IS ZERO
1169 A019    HF     LDY #$19
116B 49E5           EOR #TPBIT          ;DUMMY, REALLY
116D 8D20C0         STA TAPOUT          ;INVERT OUTPUT BIT
1170 88      HFP1   DEY                 ;PAUSE FOR 138 USEC
1171 D0FD           BNE HFP1
1173 CA             DEX
1174 D0F3           BNE HF
```

```
1176 A218        LF      LDX #$18            ;ASSUME BIT IS ONE
1178 B002                BCS LF20
117A A20C                LDX #$0C            ;BIT IS ZERO
117C A027        LF20    LDY #$27
117E 49E5                EOR #TPBIT          ;DUMMY
1180 8D20C0              STA TAPOUT          ;INVERT OUTPUT
1183 88          LFP1    DEY                 ;PAUSE FOR 208 USEC
1184 D0FD                BNE LFP1
1186 CA                  DEX
1187 D0F3                BNE LF20
1189 68                  PLA                 ;RESTORE BIT CTR
118A 0A                  ASL                 ;DECREMENT IT
118B D0D0                BNE KIMBIT          ;FF SHIFTED 8X-00
118D AEE311              LDX TEMP1
1190 ACE411              LDY TEMP2
1193 98                  TYA                 ;RESTORE X,Y, DATA BYTE
1194 60                  RTS
1195             ;WE NEED A DELAY FUNCTION, BECAUSE THE SYM PROG
1195             ;USES THE KIM CHARGOUT ROUTINE WITH OUT PUT DISABLED
1195             ;TO DELAY (AND WE CAN'T)
1195             ;
1195             ;THIS ONE SHOULD BE 1/21 SECOND, SINCE IT EMULATES
1195             ;THE KIM CHAR OUT ROUTINE, WHICH THE SYM PROGRAM USES
1195 8EE311      DELAY   STX TEMP1           ;PRESERVE X
1198 8CE411              STY TEMP2           ;AND Y
119B A200                LDX #$00            ;DO OUTER LOOP 256 TIMES
119D A01A        LOOP0   LDY #TIME99         ;LOOP
119F 88          LOOP1   DEY
11A0 D0FD                BNE LOOP1
11A2 CA                  DEX
11A3 D0F8                BNE LOOP0
11A5 AEE311              LDX TEMP1           ;RESTORE X
11A8 ACE411              LDY TEMP2           ;AND Y
11AB 60                  RTS
11AC             ;
11AC             ;CHKT...UPDATE CHECKSUM FROM BYTE IN ACC
11AC A8          CHKT    TAY                 ;SAVE ACC
11AD 18                  CLC
11AE 6DE111              ADC CHKL
11B1 8DE111              STA CHKL
11B4 9003                BCC CHKT10
11B6 EEE211              INC CHKH            ;BUMP HI BYTE
11B9 98          CHKT10  TYA                 ;RESTORE ACC
11BA 60                  RTS
11BB             ;START---LEAVING OUT SOME UNECESSARY JUNK
11BB 20C711      START   JSR ZERCK           ;ZERO CHECKSUM
11BE 20D011              JSR P2SCR           ;THATS WHAT THEY NAMED IT
11C1 A900                LDA #$00            ;TURN ON TAPE RECORDER
11C3 8D59C0              STA TAPEON
11C6 60                  RTS
11C7 A900        ZERCK   LDA #$00            ;ZERO CHECKSUM
11C9 8DE111              STA CHKL
11CC 8DE211              STA CHKH
11CF 60                  RTS
11D0             ;--P2SCR-- THIS MOVES THE STARTING ADDRESS
11D0             ;          TO THE RUNNING BUFFER ADDRESS.
11D0             ;THE WEIRD NAME IS DUE TO THE NAMES
11D0             ;OF THE LOCATIONS WHICH WE ARE MOVING IN THE SYM BOOK
11D0 ADDC11      P2SCR   LDA SAH             ;STARTING ADD HI
11D3 85E8                STA BUFADH
11D5 ADDB11              LDA SAL             ;STARTING ADD LO
11D8 85E7                STA BUFADL
11DA 60                  RTS
11DB             ;PAGE PARAMETERS, ETC.
11DB             ;THESE NEXT SIX LOCATIONS SHOULD BE
11DB             ;FILLED WITH THE CALLING PARAMETERS
11DB             ;BEFORE CALLING THE SYMOUT ROUTINE
11DB             ;
11DB             ;
11DB 00          SAL     HEX 00              ;STARTING ADDRESS, LO BYTE
11DC 00          SAH     HEX 00              ;STARTING ADDRESS, HI BYTE
11DD 00          EAL     HEX 00              ;ENDING ADDRESS+1, LO BYTE
11DE 00          EAH     HEX 00              ;ENDING ADDRESS+1, HI BYTE
11DF 00          ID      HEX 00              ;TAPE ID NUMBER
11E0 00          MODE    HEX 00              ;SYM=$80, KIM=$00
```

```
11E1 00        CHKL   HEX 00                ;VARIABLES
11E2 00        CHKH   HEX 00
11E3 00        TEMP1  HEX 00
11E4 00        TEMP2  HEX 00
11E5 00        TPBIT  HEX 00
11E6           ;
11E6           ;--- SHORT ROUTINE TO MAKE SYNC TAPES
11E6           ;    (APPLE PRODUCED TAPE WILL USUALLY NEED
11E6           ;     DIFFERENT VOLUME AND TONE SETTINGS
11E6           ;     THAN KIM OR SYM TAPES)
11E6           ;
1000                  ORG $1000
1000                  OBJ $800
1000           ;
1000 20BB11    SYNC   JSR START             ;MAKE A SYNC TAPE
1003 A916      SYNMOR LDA #SYN              ;LOAD SYNC CHARACTER
1005 200711           JSR OUTCTX            ;SEND IT
1008 4C0310           JMP SYNMOR            ;DO IT FOREVER
100B           ;
                     END
```

```
          *************************
          *                       *
          * SYMBOL TABLE -- V 1.5 *
          *                       *
          *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

TM1500 0047  TIME99 001A  EOT    0004  SYN    0016  BUFADL 00E7  BUFADH 00E8
CHAR   00EA

** ABSOLUTE VARABLES/LABELS

TAPOUT C020  TAPEON C059  TAPEOF C058  BEGIN  1080  SYMOUT 1080
MARK8A 108C  MARK8B 108E  DUMPT1 1097  DUMPT2 10C7  DUMPT4 10F7  OUTCTX 1107
OUTBTH 110C  GETBIT 1116  A416   111F  B416   1129  B416B  112B  NOFLIP 1135
OUTBCX 1138  OUTBTX 113B  OUTBTC 1140  HEXOUT 1148  HEX1   1151  OUTCHT 1153
KIMBIT 115D  HF     1169  HFP1   1170  LF     1176  LF20   117C  LFP1   1183
DELAY  1195  LOOP0  119D  LOOP1  119F  CHKT   11AC  CHKT10 11B9  START  11BB
ZERCK  11C7  P2SCR  11D0  SAL    11DB  SAH    11DC  EAL    11DD  EAH    11DE
ID     11DF  MODE   11E0  CHKL   11E1  CHKH   11E2  TEMP1  11E3  TEMP2  11E4
TPBIT  11E5  SYNC   1000  SYNMOR 1003

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:01DA
```

6
REFERENCE

INTRODUCTION

This chapter provides some assorted reference material which should be of great interest to any serious Apple user who wants to know more about the firmware and hardware features locked within the machine. Each of these articles explores a different feature of the Apple.

''Intercepting DOS Errors from Integer BASIC,'' by Andy Hertzfeld, presents a quick overview of the DOS error codes, where they are stored, and how to intercept them from within an Integer BASIC program. ''Applesoft Floating Point Routines,'' by R.M. Mottola, discusses the powerful floating point routines which are locked inside the Applesoft firmware. Incidentally, these are the routines used by the MEAN-14 system (see chapter 1). Richard Williams' ''How to Use Hooks'' explains the use of vectors, or hooks by the monitor, and how to use them to intercept program control. Two example programs are provided. Finally, Dick Suitor's ''Brown and White and Colored All Over'' discusses some of the theory behind the Apple's color graphics, and provides an example program.

All these programs should further your understanding of your Apple and what's in it. The article on hooks is especially recommended to the novice to aid understanding of the routines in chapter 1.

Intercepting DOS Errors from Integer BASIC

by Andy Hertzfeld

Implement true turnkey applications on the Apple with this DOS error handling interface. Now Integer BASIC programs can trap errors from DOS, diagnose problems, and take remedial action with no intervention from the operator.

When a DOS error such as FILE NOT FOUND occurs during execution of a BASIC program, execution is suspended and an error message is printed. Unfortunately, this is often not what we want to happen. We would prefer the program to be notified of the error and allowed to continue execution, dealing with the error in any fashion it desires.

This is fairly easy to achieve under Applesoft because it includes an ONERR error intercepting facility. It is much harder to intercept erors from Integer BASIC; this article describes one method for doing so.

Unlike Integer BASIC, the DOS resides in normal RAM. This means it can be patched to make it do almost anything we wish. It turns out that location 9D5A (for 48K systems) holds the address of the BASIC error-handling routine that DOS vectors to whenever an error arises. It usually contains E3E3, for Integer BASIC, and D865 for ROM Applesoft. However, we can store our own address into 9D5A (5D5A for 32K systems) and thereby gain control whenever a DOS error occurs.

The following 24-byte, relocatable routine will intercept errors from BASIC. When a DOS error arises, it will store the error number at location 2; the line number of the statement that caused the error in locations 3 and 4; and, finally, it will transfer control to the BASIC statement whose line number is found in locations 0 and 1. Since the routine is relocatable, you can position it anywhere you wish. Location 300 appears to be a pretty good place, unless you are keeping your printer driver there.

To activate the error intercept facility, perform the following two POKEs which store the address of the intercept routine in $9D5A:

```
POKE-25254,0: POKE-25253,3 (for 48K systems) or
POKE-23898,0: POKE-23899,3 (for 32K systems)
```

The error intercept routine itself can be POKEd into page 3 or BLOADed off disk, whichever you prefer. If you locate it somewhere other than $300, make sure to alter the above POKEs accordingly.

After the routine is loaded into memory, it is very easy to use. If LINE is the line number of the statement where the error handling portion of your program begins, you should "POKE 0, LINE mod 256" and "POKE 1, LINE/256" to inform the interceptor where you want it to branch to. Your BASIC error-handling can figure out which statement caused the error by PEEKing at locations 3 and 4.

PEEK(3) + 256*PEEK(4) is the line number. It can determine which type of DOS error occured by PEEKing at location $2. Table 1 gives the numbers for the various different classes of error.

Unfortunately, there is still one minor problem. Even though you regain control when a DOS error occurs, DOS still rings the bell and prints out any error message. One simple POKE will inhibit DOS from doing this, but since the POKE will suppress all DOS error messages, including immediate execution errors, it is a little bit dangerous. Also, the POKE is different for different memory size systems and for different versions of DOS.

```
48K with DOS V3.1:       POKE-22978,20
48K with DOS V3.2/3.3:   POKE-22820,18
32K with DOS V3.1:       POKE 26174,20
32K with DOS V3.2/3.3:   POKE 26332,18
```

Table 1 — Error Numbers and Messages

Number	Message
1	Language Not Available
2	Range Error
3	Range Error
4	Write Protection Error
5	End of Data Error
6	File Not Found Error
7	Volume Mismatch Error
8	Disk I/O Error
9	Disk Full Error
10	File Locked Error
11	Syntax Error
12	No Buffers Left Error
13	File Type Mismatch
14	Program Too Large Error
15	Not Direct Command

Note that these are error messages for DOS V3.2 or V3.3; the V3.1 messages are slightly different.

On all systems, you can restore error messages by POKEing 4 into the system-dependent address cited above.

The ability to capture DOS errors is very important, especially for turn-key systems where it is a disaster if a program crashes for any reason at all. Perhaps this little routine will allow more people to program in faster, more elegant Integer BASIC rather than choosing the Applesoft language.

```
0800          ;***********************
0800          ;*                     *
0800          ;*    INTERCEPTING     *
0800          ;*     DOS ERRORS      *
0800          ;*  BY ANDY HERTZFELD  *
0800          ;*                     *
0800          ;*        ERROR        *
0800          ;*                     *
0800          ;*  COPYRIGHT (C) 1981 *
0800          ;*   MICRO INK, INC.   *
0800          ;* CHEMSFORD, MA 01824 *
0800          ;* ALL RIGHTS RESERVED *
0800          ;*                     *
0800          ;***********************
0800          ;
0800          ;
0800          ERNUM  EPZ $02           ;ERROR NUMBER
0800          ERRLIN EPZ $03           ;LINE OF ERROR
0800          ONERR  EPZ $00           ;CONTROL TRANSFER LINE
0800          ;
0800          PR     EPZ $DC           ;BASIC LINE POINTER
0800          ACL    EPZ $CE           ;BASIC ACCUMULATOR
0800          ACH    EPZ $CF
0800          ;
0800          GOTO   EQU $E85E         ;BASIC 'GOTO' ROUTINE
0800          ;
0300                 ORG $300
0300                 OBJ $800
0300          ;
0300          ;
0300 8602     ERROR  STX ERNUM         ;SAVE ERROR NUMBER
0302 A001            LDY #01
0304 B1DC            LDA (PR),Y        ;GET LOW BYTE OF ERRING
0306 8503            STA ERRLIN        ;LINE NUMBER AND SAVE
0308 C8              INY
0309 B1DC            LDA (PR),Y        ;DITTO FOR HIGH BYTE
030B 8504            STA ERRLIN+1
030D A500            LDA ONERR         ;GET LOW BYTE OF LINE NUMBER
030F 85CE            STA ACL           ;OF ERROR HANDLING STATEMENT
0311 A501            LDA ONERR+1       ;DITTO FOR HIGH BYTE
0313 85CF            STA ACH           ;SET THINGS UP FOR BASIC AND
0315 4C5EE8          JMP GOTO          ;LET THE FIRMWARE TAKE OVER
                     END

         *************************
         *                       *
         * SYMBOL TABLE -- V 1.5 *
         *                       *
         *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

ERNUM  0002  ERRLIN 0003  ONERR  0000  PR     00DC  ACL    00CE  ACH    00CF


** ABSOLUTE VARABLES/LABELS

GOTO   E85E  ERROR  0300

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0052
```

Applesoft Floating Point Routines

by R.M. Mottola

Applesoft BASIC is a complete and easy-to-use language—but sometimes it can be annoyingly slow. To decrease execution time, many programmers code some routines in machine language. Yet it seems wasteful to re-code routines which already exist in the Applesoft interpreter. The solution to the dilemma: Use the floating point routines directly! Here is a discussion of where floating point routines are located, what they do, and an example of their direct use.

Part of a recent project required me to write a routine that would calculate various statistical data reductions on a series of data points. The initial result, written in Applesoft floating point BASIC, worked well enough but took a healthy amount of time to execute. Upon doing some timing experiments, it became apparent that a good deal of the time required to perform the task was eaten up by BASIC overhead conversion of types, floating point ''FOR-NEXT'' loops, and general interpereter related functions.

What I really wanted was to write all of the routine in machine language. To do this, there were two options available. The first was to write some floating point routines which maintained the Applesoft five byte variable format. This proved to be impractical due to the amount of memory required for these routines.

The second and much more memory efficient solution was to locate the floating point routines already in my machine in Applesoft. This proved to be reasonably difficult for a number of reasons but after much head-scratching I've managed to unearth the following routines. Before using them, its probably a good idea to familiarize yourself with the format of both the Applesoft variables and the Applesoft floating point accumulators.

The format of Applesoft variables is a standard five byte floating point representation, with the highest order byte containing the exponent and the lower four bytes containing a signed mantissa. (See page 137 of the Applesoft manual for more on this.) The format of the Applesoft accumulators is a little different. You will notice from various Applesoft zero page usage tables that seven bytes have been allocated for each of the two floating point accumulators. The format of these accumulators is as follows: The highest order byte contains the exponent. The next four bytes contain the negative absolute value of the mantissa, as represented in Applesoft variable format. The sixth byte contains the original high-order byte of the mantissa if a value has just been converted from variable format to accumulator format. In any case, this byte is used to represent the sign of the mantissa. The seventh and last byte of the accumulator is a ''function'' byte used in arithmetic operations. It is not initially assigned a value on conversion of a value from variable format to accumulator format.

To use the following floating point routines is a reasonably straight-forward process. For the sake of simplicity, you may find it easier to forget the accumulator formatting of values, and load all values into the accumulator using the ''FPLOAD'' subroutine listed. This routine performs the conversion while doing the load. You should also be careful to represent all values in normalized form. If you plan to use only values that have been previously specified by Applesoft, you will not have to do this as Applesoft normalizes all variables as they are specified. To use your own values, you may find the accompanying utility program useful.

Another thing to be careful about is floating point errors (Division by zero, Overflow). Since these floating point routines were not meant to be used outside of Applesoft, the entry points to the error handling routines are in ROM. Unfortunately, the vectors to these routines are cast in stone (or Silicone, anyway) and cannot be changed. There are two ways to deal with these errors:

1. Test your routines for ''worst case'' operation. If you can make sure that errors will never occur, you've got it made.

2. Applesoft has the ability to vector errors to a specified BASIC line number with the ONERR... GOTO statement to direct errors to a specified line number. On this line number, you can make a call to your own machine language error handling routines.

The following routines constitute the major arithmetic routines available in Applesoft. There are, of course, other functions buried in BASIC which have not been identified here.

Name: FPLOAD
Address: $EAF9
Symbolic: M→FPAC1

Loads variable into primary floating point accumulator. Converts to FPAC format. A and Y registers must point at variable in memory (ADL, ADH). Clears $AC.

Name: FPSTR
Address: $EB2B
Symbolic: FPAC1→M

Stores value in primary floating point accumulator in memory. Converts from FPAC format to Applesoft variable format. X and Y registers must point at first byte in memory in which value is to be stored (ADL, ADH). Clears $AC.

Name: TR1 > 2
Address: $EB63
Symbolic: FPAC1

Transfers the value contained in the primary floating point accumulator to the secondary floating point accumulator. Clears $AC.

Name: FPDIV2
Address: $EA60
Symbolic: FPAC2/M→FPAC1

Divides the value contained in the secondary floating point accumulator by the value pointed at by the A and Y registers (ADL, ADH) and stores the result in the primary floating point accumulator.

Name: TR2 > 1
Address: $EB53
Symbolic: FPAC2→FPAC1

Transfers the value contained in the primary floating point accumulator to the secondary floating point accumulator. Clears $AC.

Name: FPSQR
Address: $EE8D
Symbolic: FPAC1→FPAC1

Returns the positive square root of the value contained in the primary floating point accumulator in the primary floating point accumulator.

Name: FPEXP
Address: $EE94
Symbolic: FPAC2 M→FPAC1

Raises the value contained in the secondary floating point accumulator to the value pointed at by the A and Y registers. The result is stored in the primary floating point accumulator.

Name: FPINT
Address: $EC23
Symbolic: INT (FPAC1)→FPAC1

Returns the integer value of the value contained in the primary floating point accumulator to the primary floating point accumulator.

Name: FPABS
Address: $EBAF
Symbolic: ABS (FPAC1)→FPAC1

Returns the absolute value of the value contained in the primary floating point accumulator to the primary floating point accumulator.

Name: FPADD
Address: $E7BE
Symbolic: M + FPAC1→FPAC1

Adds the value of the variable pointed to by the A and Y registers (ADL, ADH) to the value contained in the primary floating point accumulator and stores the result in the primary floating point accumulator.

Name: FPADD2
Address: $E7A0
Symbolic: 0.5 + FPAC1→FPAC1

Similar to previous routine, but adds the value (0.5) to the primary floating point accumulator.

Name: FPMUL
Address: $E97F
Symbolic: M*FPAC1→FPAC1

Multiplies the value pointed at by the A and Y registers (ADL, ADH) by the value contained in the primary floating point accumulator and stores the result in the primary floating point accumulator.

Name: FPSUB
Address: $E7A7
Symbolic: M – FPAC1→FPAC1

Subtracts the value contained in the primary floating point accumulator from the value pointed at by the A and Y registers (ADL, ADH) and stores the result in the primary floating point accumulator.

Name: FPDIV
Address: $EA66
Symbolic: M / FPAC1→FPAC1

Divides the value pointed to by the A and Y registers (ADL, ADH) by the value contained in the primary floating point accumulator and stores the result in the primary floating point accumulator.

Name: FPSGN
Address: $EB90
Symbolic: SGN (FPAC1)→FPAC1

Returns the sign of the value contained in the primary floating point accumulator. A negative value will return (−1). A positive value will return a (1). A value of zero will return a (0).

Name: FPLOG
Address: $E941
Symbolic: LOG (FPAC1)→FPAC1

Returns the natural log of the value obtained in the primary floating point accumulator to the primary floating point accumulator.

Name: COMP2
Address: $E89E
Symbolic: TWO'S COMPLEMENT OF FPAC1→FPAC1

Returns the Two's Complement of the value contained in the primary floating point accumulator to the primary floating point accumulator.

Name: INT > FP
Address: $E2F2
Symbolic: (Y,A)→FPAC1

Converts a two byte integer to its floating point equivalent (FPAC format) and stores it in the primary floating point accumulator. The integer must be represented with the high-order byte stored in the A register, and the low-order byte stored in the Y register.

Name: FP > INT
Address: $E10C
Symbolic: FPAC1→($A0, $A1)

Converts the floating point contained in the primary floating point accumulator to a two byte integer, which is stored in the fourth and fifth bytes of the primary floating point accumulator ($A0, $A1). $A0 contains the high-order byte and $A1 contains the low-order byte.

```
1  REM  ***********************
2  REM  *                     *
3  REM  *    FLOATING POINT   *
4  REM  *       ROUTINES      *
5  REM  *     R.M. MOTTOLA    *
6  REM  *                     *
7  REM  *  COPYRIGHT (C) 1981 *
8  REM  *    MICRO INK, INC.  *
9  REM  * CHELMSFORD, MA 01824 *
10  REM *  ALL RIGHTS RESERVED *
11  REM *                     *
12  REM ***********************
13  REM
14  REM
80 :
90 X = 0:D$ =  CHR$ (4)
100  FOR N = 768 TO 792
110  READ A: POKE N,A
120  NEXT
130  REM  ESTABLISH CONVERSION ROUTINE AT $300
140  DATA  165,105,24,105,2
150  DATA  164,106,144,1,200
160  DATA  32,249,234,160,6
170  DATA  185,157,0,153,25
180  DATA  3,136,16,247,96
190  HOME : PRINT : PRINT  TAB( 7)"FLOATING POINT CONVERSIONS"
200  PRINT : PRINT : PRINT "INSTRUCTIONS-"
210  PRINT : PRINT "ENTER VALUE YOU WISH CONVERTED TO FLOATING POINT
    REPR  ESENTATION.  IF YOU WISH TO PRINT THE CONVERSIONS ON THE"
220  PRINT "PRINTER, FOLLOW THE VALUE WITH A 'P'.  TO RETURN TO BASIC,
     HI   T (RETURN) KEY."
230  VTAB 14: CALL  - 868
240  INPUT "ENTER VALUE: ";A$
250  IF A$ = "" THEN  VTAB 23: END : REM  ""=NULL$
260  IF  RIGHT$ (A$,1) >  < "P" THEN 300
270  PRINT D$;"PR#1"
280  REM  PRINTER IN SLOT #1
290  PRINT : PRINT
300 X =  VAL (A$): CALL 768
310  VTAB 18: CALL  - 958: PRINT "VALUE= "X
320  PRINT : PRINT "ACCUMULATOR: $";
330  FOR N = 793 TO 799
340 A =  PEEK (N): GOSUB 450
350  NEXT : PRINT : PRINT
360  PRINT "VARIABLE:    $";
370 B =  PEEK (105) +  PEEK (106) * 256 + 2
380  FOR N = B TO B + 4
390 A =  PEEK (N): GOSUB 450
400  NEXT : PRINT
410  PRINT D$;"PR#0"
420  GOTO 230
430 :
440  REM  DECIMAL TO HEX SUB
450 A = A / 16:B =  INT (A)
460 A = (A - B) * 16
470 B = B + 48: IF B > 57 THEN B = B + 7
480  PRINT  CHR$ (B);
490 A = A + 48: IF A > 57 THEN A = A + 7
500  PRINT  CHR$ (A)" ";
510  RETURN
```

How to Use the Hooks

by Richard Williams

There are a lot of great things you can do with your Apple, once you know how to use the available hooks.

The Apple II allows you easily to substitute your own input and output routines for the standard routines. Figure 1 shows the basic flow of control when a character is output by the Apple II. Figure 2 shows how the control path changes when you substitute your own output routine for the standard monitor path. By using what are known as "hooks," you can break the normal flow of control and redirect it to your own routine.

An example of how this method can be used is shown in figure 3. Control characters normally do not show on the screen. However, by inserting a routine to change control characters into inverse video when printed, the characters will show on the screen. This procedure is very useful for listing programs containing control characters.

How It Works

Before doing the actual input or output, the system does an indirect jump, via the zero page, to the actual input or output routine. By changing the jump address, you can substitute your own routine for the standard zone. For input, at location $FD18 in the monitor, there is a JMP (KSWL) instruction. KSWL (at $38) and KSWH (at $39) contain the address of the input routine with the low byte specified first. Similarly, at address $FDED, there is a JSR (CSWL) instruction which is the jump to the output routine. CSWL, address $36, and CSWH, at $37, contain the address of the output routine. This code can be seen on pages 166 and 167 of the Apple II reference manual.

How to Insert an Input Routine

The normal input routine is KEYIN at address $FD1B. To replace it with your routine, store its address in KSWL and KSWH. Your input routine needs to do the following.

1. Upon entry to your routine, the accumulator will contain the character that was replaced by the flashing prompt. You must restore this character on the screen by doing a STA (BASL), Y where BASL = $28. Do this before altering the A or Y registers.

2. Clear the keyboard strobe, if the character came from the keyboard.

3. Return the character, with the high bit set, in the accumulator.

4. The normal input routine increments the random number seed while it waits for input. You should do this also.

If you wish to get your input from the keyboard, you can do all of these by doing a call to KEYIN (JSR $FD1B). You can then do whatever processing you want on the character, which is in the accumulator, and then return with an RTS. If you write your own routine to replace KEYIN, you should first carefully study KEYIN.

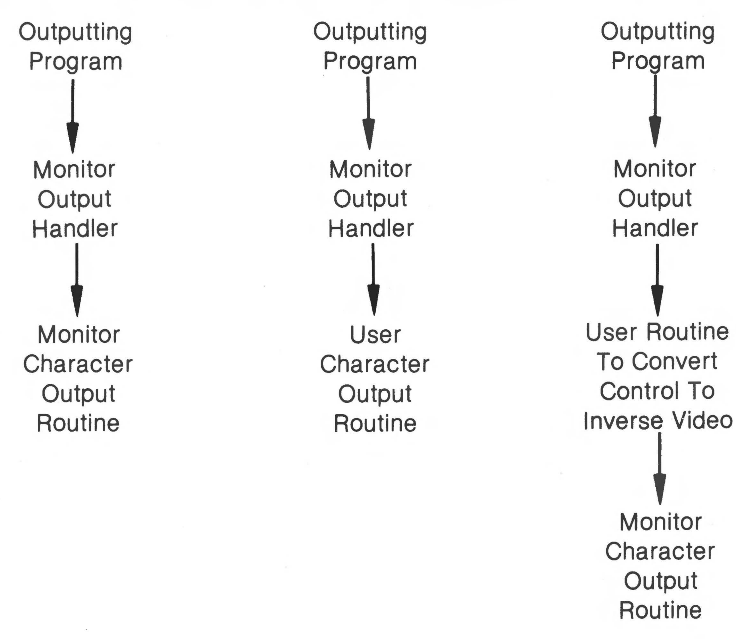


Figure 1 **Figure 2** **Figure 3**

How to Insert an Output Routine

The normal output routine is COUT1 (address $FDF0). To insert your routine, store its address in CSWL and CSWH (addresses $36 and $37) with the low byte first. The character to be output will be placed in the accumulator before your routine is called. If you wish the character in the accumulator to be printed on the screen after you are done, exit your routine by doing a JMP COUT1. A routine to convert control characters to inverse video is an example of this procedure.

How to Remove the Routines

The input and output routines can be removed from the hooks by typing IN#0 or PR#0 respectively. Or, if done in a program, a JSR SETKBD (address $FE89) simulates a IN#0, and a JSR SETVID (address $FE93) simulates a PR#0.

Special Notes for DOS Users

If you are using the disk operating system (DOS), you must follow some special rules when attaching or removing your routines. DOS normally sits in both the input and output hooks itself. Consequently, when you alter the hooks, you must call a DOS routine which informs DOS that the hooks have been changed. DOS will then reconnect itself to the hooks, but it will use your routines instead of the standard I/O routines. The routine to do this is at $3EA.

Example

The sample program in figure 4 inserts or removes a routine from the input hook.

To connect your routine do a 300G from the monitor. To remove your routine from the hook, do a 30CG.

```
300:  LDA   #low address of routine
302:  STA   $38   ;Store it in KSWL
304:  LDA   #high address byte of routine
306:  STA   $39   ;Store it in KSWH
308:  JSR   $3EA  ;Reconnect DOS
30B:  RTS
30C:  JSR   $FE89;JSR SETKBD to simulate IN#0
30F:  JSR   $3EA  ;Reconnect DOS
312:  RTS
```

Figure 4

A Sample Program Using the Input Hook

There are three characters that the Apple II can understand, but that cannot be typed in from the standard keyboard. They are the backslash (/), the left bracket ([ ), and the underscore (__). One way to type in these characters is to make a hardware modification to the keyboard. Another way is to attach a routine to the input hook that will convert unused control characters to these characters. The first program converts the following characters:

Control K to a left bracket ([ )

Control L to a backslash (/)

Control O to an underscore (_)

Here's how you use this program:

Type or BLOAD the program at $300. Note that this program is written for DOS users. If you aren't using DOS, then replace the JMP $3EA with RTS instructions.

To connect the routine, do a 303G from the moniter or a CALL 771 from BASIC.

To disconnect the routine, do a 300G from the monitor or a CALL 768 from BASIC.

The second sample program uses the output hook to convert control characters into inverse video characters. All control characters except contol M, which is the carriage return, are converted.

Summary of Important Addresses for Using the Hooks

Name	Address	Comment
COUT1	$FDF0	Monitor character output routine.
CSWL	$36	Low address byte of output routine.
CSWH	$37	High address byte of output routine.
KEYIN	$FD1B	Monitor keyboard input routine.
KSWL	$38	Low address byte of input routine.
KSWH	$39	High address byte of input routine.
MVSW	$3EA	Routine to reconnect DOS
SETKBD	$FE89	Simulates a IN#0
SETVID	$FE93	Simulates a PR#0

```
0800         ;*************************
0800         ;*                       *
0800         ;*   HOW TO USE HOOKS    *
0800         ;*   RICHARD WILLIAMS    *
0800         ;*                       *
0800         ;*       NEWKEYS         *
0800         ;*                       *
0800         ;*  COPYRIGHT (C) 1981   *
0800         ;*    MICRO INK, INC.    *
0800         ;* CHELMSFORD, MA 01824  *
0800         ;*  ALL RIGHTS RESERVED  *
0800         ;*                       *
0800         ;*************************
0800         ;
0800         ;
0800         ;
0800         BKSLSH EPZ 220            ;ASCII BACKLASH
0800         CTRLK  EPZ 139            ;ASCII CONTROL K
0800         CTRLL  EPZ 140            ;ASCII CONTROL L
0800         CTRLO  EPZ 143            ;ASCII CONTROL O
0800         KSWL   EPZ $38            ;INPUT HOOK ADDRESS
0800         KSWH   EPZ $39
0800         RTBRKT EPZ 219            ;ASCII RIGHT BRACKET
0800         UNDSCR EPZ 223            ;ASCII UNDERSCORE
0800         ;
0800         ;
0800         KEYIN  EQU $FD1B          ;MONITOR'S INPUT HANDLER
0800         MVSW   EQU $3EA           ;ROUTINE TO RECONNECT DOS
0800         SETKBD EQU $FE89          ;SIMULATES IN#0
0800         ;
0800         ;
0800         ;------NEXT OBJECT FILE NAME IS NEWKEYS.OBJ0
0800         ;
0800         ;
0800         ;
0300                ORG $300
0300                OBJ $800
0300         ;
0300         ;
0300         ;
0300 4C0F03         JMP UNHOOK         ;JUMP TO DISCONNECT ROUTINE
0303         ;
0303         ;*** THIS PART ATTACHES OUR ROUTINE INTO THE INPUT HOOK
0303         ;
0303 A916    ATTACH LDA #KEYCHK        ;A=LOW BYTE OF ADDRESS
0305 8538           STA KSWL
0307 A903           LDA /KEYCHK        ;GET HI BYTE
0309 8539           STA KSWH
030B 20EA03         JSR MVSW           ;GO TO IT
030E 60             RTS
030F         ;
030F         ;*** THIS PART UNHOOKS THE ROUTINE
030F         ;
030F 2089FE  UNHOOK JSR SETKBD         ;DO A IN#0
0312 20EA03         JSR MVSW
0315 60             RTS
0316         ;
0316         ;*** THIS IS THE ROUTINE
0316         ;
0316 201BFD  KEYCHK JSR KEYIN          ;GET THE KEY
0319 C98B           CMP #CTRLK         ;CONTROL K?
031B D003           BNE NOTK
031D A9DB           LDA #RTBRKT        ;MAKE IT A BRACKET
031F 60             RTS
0320 C98C    NOTK   CMP #CTRLL         ;CONTROL L?
0322 D003           BNE NOTL
0324 A9DC           LDA #BKSLSH        ;MAKE IT A BACKLASH
0326 60             RTS
0327 C98F    NOTL   CMP #CTRLO         ;CONTROL O?
0329 D002           BNE CHKDNE
032B A9DF           LDA #UNDSCR
032D 60      CHKDNE RTS
                    END
```

```
          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5 *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

BKSLSH 00DC  CTRLK  008B  CTRLL  008C  CTRLO  008F  KSWL   0038  KSWH   0039
RTBRKT 00DB  UNDSCR 00DF

** ABSOLUTE VARABLES/LABELS

KEYIN  FD1B  MVSW   03EA  SETKBD FE89  ATTACH 0303
UNHOOK 030F  KEYCHK 0316  NOTK   0320  NOTL   0327  CHKDNE 032D

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:009A
```

```
0800           ;*************************
0800           ;*                       *
0800           ;*    HOW TO USE HOOKS   *
0800           ;*    RICHARD WILLIAMS   *
0800           ;*                       *
0800           ;*        CONVERT        *
0800           ;*                       *
0800           ;*  COPYRIGHT(C) 1981    *
0800           ;*    MICRO INK, INC.    *
0800           ;* CHELSMFORD, MA 01824 *
0800           ;*  ALL RIGHTS RESERVED *
0800           ;*                       *
0800           ;*************************
0800           ;
0800           ;
0800           ;
0800           CSWH   EPZ $37            ;OUTPUT HOOK HIGH BYTE
0800           CSWL   EPZ $36            ;OUTPUT HOOK LOW ORDER BYTE
0800           CTRLM  EPZ $8D            ;CONTROL M
0800           MASK   EPZ $3F            ;MASK TO CONVERT TO INVERSE
0800           NULL   EPZ $80            ;NULL CHARACTER
0800           SPACE  EPZ $A0            ;SPACE CHARACTER
0800           ;
0800           ;
0800           ;
0800           COUT1  EQU $FDF0          ;CHARACTER OUTPUT ROUTINE
0800           MVSW   EQU $3EA           ;RECONNECTS DOS
0800           SETVID EQU $FE93          ;PERFORMS PR#0
0800           ;
0800           ;
0300                  ORG $300
0300                  OBJ $800
0300           ;
0300           ;
0300 4C0F03           JMP UNHOOK
0303           ;
0303           ;*** ROUTINE TO CONNECT ROUTINE INTO HOOK
0303           ;
0303 A916             LDA #CONVRT        ;GET LOW BYTE OF ADDRESS
0305 8536             STA CSWL
0307 A903             LDA /CONVRT        ;GET HIGH BYTE
0309 8537             STA CSWH
030B 20EA03           JSR MVSW
030E 60               RTS
030F           ;
```

```
030F             ;*** THIS UNHOOKS THE ROUTINE
030F             ;
030F 2093FE      UNHOOK JSR SETVID           ;SIMULATE PR#0
0312 20EA03             JSR MVSW             ;RECONNECT DOS
0315 60                 RTS
0316             ;
0316             ;*** THIS IS THE CONVERSION ROUTINE
0316             ;
0316 C980        CONVRT CMP #NULL            ;<NULL CHARACTER
0318 900A               BCC GOOUT
031A C9A0               CMP #SPACE           ;>=SPACE CHARACTER
031C B006               BCS GOOUT
031E C98D               CMP #CTRLM           ;RETURN CHAR?
0320 F002               BEQ GOOUT
0322 293F               AND #MASK            ;CONVERT TO INVERSE
0324 4CF0FD      GOOUT  JMP COUT1
                      END

***** END OF ASSEMBLY


          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5  *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

CSWH   0037  CSWL   0036  CTRLM  008D  MASK   003F  NULL   0080  SPACE  00A0


** ABSOLUTE VARABLES/LABELS

COUT1  FDF0  MVSW   03EA  SETVID FE93  UNHOOK 030F  CONVRT 0316  GOOUT  0324


SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0072
```

Brown and White and Colored All Over

by Richard F. Suitor

The video graphics memory buffers are the backbone of the Apple II's impressive color capabilities. This article discusses the Apple's color video output, emphasizing color generation theory and covering relationships between colors and screen memory locations. The information explored in this article is then used to generate several random color displays, which can be used to further explore Apple graphics.

The Color of Your Apple

The colors on your screen come from your color TV and are controlled in part by the video signal. Most of the signal carries the brightness information of the picture—a black and white set uses this part of the signal to generate its picture. Superimposed on this signal is the color carrier, a 3.58 MHz signal that carries the color information. The larger this signal, the more colorful that region of the picture. The hue (blue, green, orange, etc.) is determined by the phase of the color signal. Reference timing signals at the beginning of each scan line synchronize a "standard" color signal. The time during a 3.58 MHz period that the picture color signal goes high compared to when the standard goes high determines the hue. A color signal that goes high when the standard does, gives orange. One signal that goes low at that time gives blue. Signals that are high while the standard goes from high to low or from low to high give violet and green. (This, at least, was the intention. Studio difficulties, transmission paths and the viewer's antenna and set affect these relations, so the viewer is usually given final say with a hue or tint control.)

The time relation of the color signal to the standard signal is expressed as a "phase angle". It is measured in angular measures such as degrees or radians and can run from 0 to 360 degrees. This phase angle corresponds to position on a color circle, with orange at the top and blue at the bottom, as shown in figure 1.

The perimeter of the circle represents different colors or hues. The radial distance from the center represents amount of color, or saturation. The former is usually adjusted by the tint control, the latter by the color control. A color that

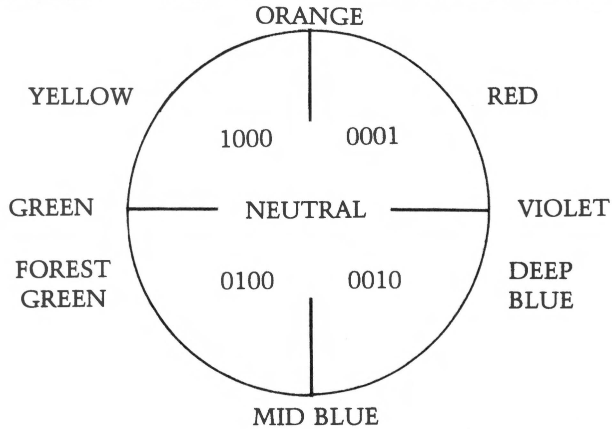


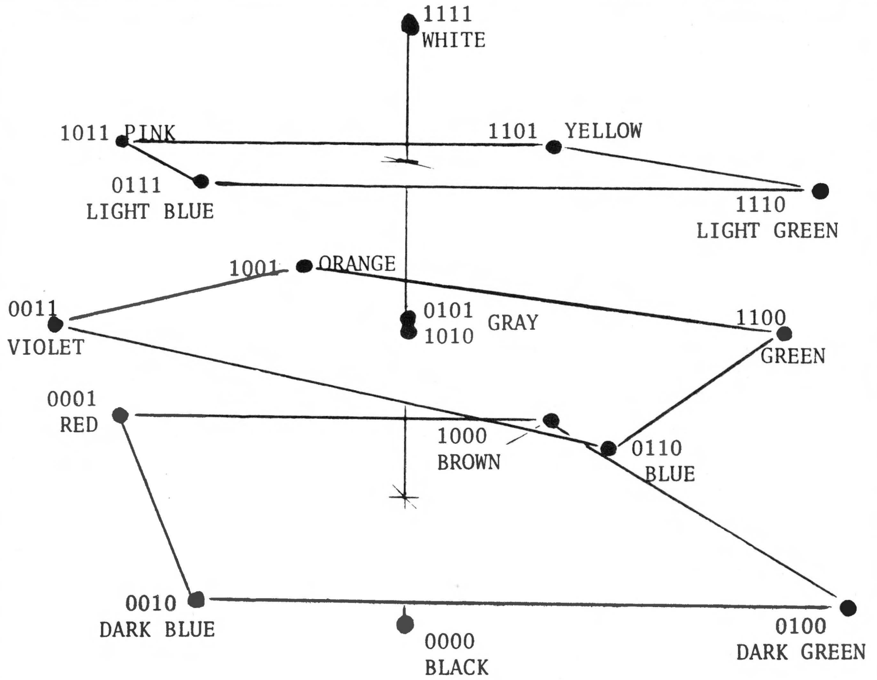
Figure 1 Color circle shows relations of color to color number bit position.

can be reproduced by a color TV can be related to a point in this circle. The angular position is coded in the phase of the 3.58 MHz color carrier signal; the radial distance from the center is given by the amplitude of the color carrier.

The numerical coding of the Apple colors can be appreciated using this circle and binary representation of the color numbers. The low order bit corresponds to red (#1). The second bit corresponds to dark blue (#2), the third to dark green (#4) and the high order bit to brown (dark yellow, #8). To find the color for any color number, represent each 1 bit as a quarter-pie piece centered over its respective color, as indicated in figure 1. The brightness or lightness of the color corresponds to the number of pie pieces and the color corresponds to the point where the whole collection balances. Black, #0, has no bits set, no pie and no brightness. White, #15, has four bits set, the whole pie, and is of maximum brightness and balances in the center of the circle at neutral. Orange, #9 or 1001 in binary, has pie over the top hemisphere and balances on a point between neutral and orange. The #5, binary 0101, has two separate wedges, one over red and one over green. Since it is symmetric, it balances at the center. It represents a neutral gray of intermediate brightness as does #10. The #14 has pie over every sector except the red one. It is bright and balances on a line toward forest green. It gives a bluish green light.

A diagram representing the relations of all the colors is given in figure 2. Each of the one, two and three bit numbers form planes, each corresponding to a color circle. You can think of these positions as points in space, with brightness increasing with vertical position and horizontal planes representing color circles of differing brightness.

The colors of the Apple are thus coded by the bit patterns of the numbers representing them. You can think of them as additive combinations of red, dark blue, dark green and brown, where adding two colors is represented by ORing the two numbers representing them. Subtractive combination can be represented by ANDing the light colors, pink, yellow, light green and light blue. The more bits set in a number, the brighter; the fewer, the darker. The bit patterns for 5 and 10 have no 3.58 MHz component and so generate a neutral tone. At a boundary



Color space locations of the Apple II colors.
Figure 2 Each horizontal plane forms a color circle of different brightness.

between 5 and 10 however, this pattern is disturbed and two bits or spaces adjoin. Try the following program which has only grays displayed:

```
10 GR
20 FOR I=0 TO 9
30 COLOR =5
40 HLIN 0,39 AT 2*I
50 VLIN 20,39 AT 2*I
60 VLIN 20,39 AT 2*I+21
70 COLOR=10
80 HLIN 0,39 AT 2*I+1
90 VLIN 20,39 AT 2*I+1
100 VLIN 20,39 AT 2*I+20
110 NEXT I
120 RETURN
```

The top half of the display has HLINs alternating 5 and 10. The botton half has VLINs, alternating 5 and 10. What do you see? The bit pattern for a number is placed directly on the video signal, with the four bits occupying one color carrier period. When two bits adjoin at a 5,10 boundary, a light band is formed. When two spaces adjoin, a dark band is formed. The slight tints are due to the boundaries having some color component. Changing the 5,10 order reverses this tint.

Now is a good time to consider just how large a 3.58 MHz period is. The Apple text is generated with a 5 × 7 dot matrix, a common method of character generation. These same dots correspond to individual bits in the high resolution display memory. One dot is one-half of a 3.58 MHz period and corresponds to a violet (#3) or green (#12) color signal. This is why the text is slightly colored on a color TV and the high resolution display has two colors (other than black and white), green and violet. (But you can make others, due to effects similar to those seen in the BASIC program above.)

[Note: The Apple II now has orange (#9) and blue (#6) as high resolution colors as well as green and violet. A circuit change interprets bit 7 of each word in the high resolution display (this bit is not displayed) and shifts the displayed dots for the other bits by a ¼ period or dot. This choice affects 7 consecutive bits or displayed dots. You cannot switch from orange to green with these seven. Thus in high resolution pictures, boundaries between orange and green, orange and violet, blue and green, or blue and violet can have a low resolution, "staircase" appearance.

Also note that not every high resolution point can be plotted in a particular color. Only half, for instance, can be plotted in green. The other half can be plotted in violet. That is why a high resolution plot of a colored point or vertical line sometimes seems to produce nothing. Plotting twice at two consecutive horizontal points solves this problem.]

The design of color TV has further implications for the display. The video black and white signal is limited to about 4 MHz, and many sets drop the display frequency response so that the color signal will not be obtrusive. A set so designed will not resolve the dots very well and will produce blurry text. Some color sets have adjustments that make the set ignore the color signal. Since the color signal processing involves subtracting and adding portions of the signal, avoiding this can sometimes improve the text resolution. Also, reducing the contrast and the brightness somewhat can help with text material.

The color TV design attempts to remove the color carrier from the picture (after duly providing the proper color), but you may be able to see the signal as 3 or 4 fine vertical lines per color block. They should not be apparent at all in the white, black or the gray (except on a high resolution monitor).

Tan is Between Brown and White

This section presents a brief application of the concepts of the relationships in color space of the Apple colors. Many of you, I suspect, are regular readers of Martin Gardner's "Mathematical Games" column in Scientific American. I strongly recommend it.

One column discussed the aesthetic properties of random variations of different kinds. To summarize briefly, three kinds are:

WHITE Each separate element is chosen randomly and is independent of every other element. It is called ''white'' because a frequency spectrum of the result shows all frequencies occur equally, a qualitative description of white light.

BROWN Each separate element is the previous element plus a randomly chosen deviation. It is called ''brown'' because Brownian motion is an example.

1/F Its frequency spectrum is intermediate between ''white'' and ''brown''.

The column presented arguments, attributed to Richard Voss, that 1/f variations are prevalent and aesthetically more satisfying than ''white'' (not enough coherence) or ''brown'' (not enough variation). An algorithm was given for generating elements with 1/f random variations. Briefly, each element is the sum of N terms (three, say). One term is chosen randomly for each element. The next is chosen randomly for every other element. The next is chosen randomly for every fourth element, and so forth.

With the Apple, you can experiment with these concepts aurally (hence Applayer) and visually with the graphic displays. Color is a dimension that was not discussed much in the column. This section presents an attempt to apply these concepts to the Apple display.

Most of us know what ''white'' noise is like on the Apple display. An exercise that many try is to choose a random point, a random color, plot and repeat. For example:

```
10  GR
20  X=RND(40)
30  Y=RND(40)
40  COLOR=RND(16)
50  PLOT X,Y
60  GOTO 20
```

Despite the garish display that results, this is a ''white'' type of random display. Except for all being within certain limits, the color of one square has no relationship to that of its neighbors and the plotting of one square tells nothing about which square is to be plotted next.

To implement the concept of ''1/f'', I used the following:

1. X and Y are each the sum of three numbers, one chosen randomly from each plot, one every 20 plots and the third every 200.

2. A table of color numbers was made (DIM(16) in the program) so that color numbers near each other would correspond to colors that are near each other. The choice given in the program satisfies the following restrictions:

a. Adjacent numbers are from adjacent planes in figure 2.

b. No angular change (in the color planes) is greater than 45 degrees between adjacent numbers.

3. The color number is the same for 20 plots and then is changed by an amount chosen randomly from −2 to +2. This is a "brown" noise generation concept. However, most of the display normally has color patches that have been generated long before and hence are less correlated with those currently being plotted. I'll claim credit for good intentions and let someone else calculate the power spectrum.

4. Each "plot" is actually eight symmetric plots about the various major axes. I can't even claim good intentions here; it has nothing to do with 1/f and was put in for a kaleidoscope effect. Those who are offended and/or curious can alter statement 100. They may wish then to make X and Y the sum of more than three terms, with the fourth and fifth chosen at even larger intervals.

A paddle and push buttons are used to control the tempo and reset the display. If your paddle is not connected, substitute 0 for PDL(0).

```
   1 REM  ************************
   2 REM  *                      *
   3 REM  * BROWN,WHITE,COLOURED *
   4 REM  *     RICHARD SUITOR   *
   5 REM  *                      *
   6 REM  *      BROWN/WHITE     *
   7 REM  *                      *
   8 REM  *                      *
   9 REM  *   COPYRIGHT (C) 1981 *
  10 REM  *     MICRO INK, INC.  *
  11 REM  * CHELMSFORD, MA 01824 *
  12 REM  *  ALL RIGHTS RESERVED *
  13 REM  *                      *
  14 REM  ************************
  20 DIM A(16):A(1)=0:A(2)=2:A(3)=6:A(4)=7:A(5)=3:A(6)=1:A(7)=5:A(8)=11
  22 A(9)=9:A(10)=8:A(11)=10:A(12)=13:A(13)=15:A(14)=14:A(15)=12:A(16)=4

  40 GOTO 3000
 100 PLOT X,Y: PLOT 38-X,Y: PLOT X,38-Y: PLOT 38-X,38-Y: PLOT Y,X: PLOT
     38-Y,38-X: PLOT Y,38-X: PLOT 38-Y,X
 110 RETURN
 120 Z=16
 125 L= RND (5)-2
 130 U= RND (9):V= RND (9)
 147 FOR B=1 TO 10
 150 R=U+ RND (9):S=V+ RND (9)
 155 IF PEEK (-16286)>127 THEN GR
 160 K=K+L: IF K>16 THEN K=K-Z
 165 IF K<0 THEN K=K+Z
 170 COLOR=A(K)
 180 Q=( PDL (0)/2) ^ 2
 190 FOR I=-Q TO Q: IF PEEK (-16287)>127 THEN 200: NEXT I
 200 FOR I=1 TO 20
 210 X=R+ RND (6):Y=S+ RND (6): GOSUB 100: NEXT I
 220 NEXT B
 230 GOTO 120
1010 K=1:L=5
1020 Z=16
2000 GOTO 120
3000 GR : CALL -936
3010 PRINT "PADDLE 0 CONTROLS PATTERN SPEED"
3020 PRINT "USE BUTTON 0 TO GO AT ONCE TO HI SPEED"
3030 PRINT "HOLD BUTTON 1 TO CLEAR SCREEN"
3040 GOTO 1010
9000 END
```

Language Index

Author Index
(Biographies included)

DISK VOLUME 002

```
*A 005 MICRO ON THE APPLE 2
*B 004 BREAKER
*B 002 STEP-TRACE
*B 002 STEP-TRACE.800
*B 002 TRACER
*B 003 PACK-LOAD
*B 002 MEAN-14
*B 002 SCREEN WRITE
*I 007 SCROLLER
*B 002 SCROLL
*B 005 PAGE LIST
*B 002 PAGE
*B 002 HEX PRINTER
*B 002 COM-VAR-I
*B 002 COM-VAR-A
*B 002 PRINT USING
*A 007 SEARCH
*B 002 STRING SEARCH
*A 008 MATRIX DEMO
*B 008 MATRICES
*A 009 AMPERSORT DEMO
*B 005 AMPERSORT
*I 009 TRACE LIST
*B 003 TRACE INTERRUPT
*I 003 TRACE TEST
*A 015 FNPLOTTER
*I 011 COMPRESS
*B 010 PICT COMP
*B 034 LADY BE GOOD
*I 016 LIFESAVER
*B 003 LIFE
*I 004 APPLAYER MENU
*B 010 APPLAYER
*I 004 BATTLE SOUNDS
*I 022 GALACTI-CUBE
*A 007 COLOR GUN
*B 003 CASSCS
*I 005 DIRECTORY
*A 005 BASIC TRANSFER
*A 010 BINARY TRANSFER
*A 014 THERMOMETER
*B 003 SYM-KIM
*B 002 ERROR
*A 007 FLOATING POINT
*B 002 NEWKEYS
*B 002 CONVERT
*I 006 BROWN/WHITE
```

Notice to Purchaser

When this book is purchased, this pocket should contain

A. One floppy disk entitled *MICRO on the Apple, Volume 2.*
B. A warranty card pertaining to the disk.

If either is missing, make sure you ask the seller for a copy.

The publisher hereby grants the retail purchaser the right to make one copy of the disk for back-up purposes only. Any other copying of the disk violates the copyright laws and is expressly forbidden.

MICRO on the Apple, Volume 2

Edited by Ford Cavallari

More Than 30 Programs on Diskette!

MICRO INK, Inc., publisher of *MICRO, The 6502/6809 Journal,* now brings you *MICRO on the Apple, Volume 2,* the second in a series of books containing applications for the Apple.

This volume, produced for the intermediate-to-advanced-level user, provides you with reference material, advanced machine language routines, programming techniques, graphics applications, and entertainment.

Chapter titles include Machine Language Aids, I/O Enhancements, Runtime Utilities, Graphics and Games, Hardware, and Reference. These articles have been updated by the MICRO staff, and authors when possible. The programs were tested and entered on the diskette, which comes with the book (13-sector DOS 3.2 format).

About the Editor

Ford Cavallari received a degree in mathematics from Dartmouth. While there, he made extensive use of the college's time-sharing and microcomputer facilities and helped convert several important BASIC academic programs to run on Apple II systems. His work with the Apple has ranged from large-scale computer architecture projects to tiny, recreational graphics programs. He is a founding member of the Computer Literacy Institute. As Apple Specialist on the staff of *MICRO, The 6502/6809 Journal,* he serves as Editor of the *MICRO on the Apple* book series.

$24.95 in U.S./Canada
(Including floppy disk)

ISSN 0275-3537
ISBN 0-938222-06-6

MICRO INK, Inc.
P.O. Box 6502
Chelmsford, Massachusetts 01824

Warranty

MICRO on the Apple

Although we've worked to create as perfect a diskette as possible, including hiring a reputable, reliable disk manufacturer to copy the diskettes, there is no guarantee that this diskette is error-free.

To cover the few instances of defective diskettes, we are providing the following warranty (this card must be filled out and returned to MICRO INK, Inc., immediately after purchase):

If within one month of purchase you find your diskette is defective, return the diskette to MICRO, along with $1.00 to cover shipping and handling charges.

If after one month of purchase, but within no time limit, this diskette proves defective, return it to MICRO with $6.00 to cover replacement cost, shipping and handling.

Your date of purchase must be validated by your dealer; if purchased directly from MICRO, the valid date appears on this card.

Defective diskettes must be returned to MICRO to enable our quality assurance personnel to test and check the diskette. We need to know what caused the defect to avoid similar problems in the future.

We recommend that you try LOADing or BLOADing each program on the diskette immediately after purchase to ensure that the diskette is not defective.

Signature   Date of purchase   (Volume 2)

Address (please print):

Name

Street

City   State/Province/Country   Code

Other Products from MICRO

In addition to the MICRO on the Apple series, MICRO INK, Inc., produces several other products, including MICRO magazine, a monthly journal which reports on new 6502/6809 microprocessor family applications, systems, and developments. Other books published include the Best of MICRO series (anthologies of some of the best general-interest articles from MICRO), and *What's Where in the Apple*, (a detailed Atlas and memory map for the Apple II computer).

Ask your dealer for MICRO, or subscribe by completing this form:

	Yearly Rates	**(U.S Dollars)**
	Surface	**Air Mail**
United States	$24.00	n/a
Canada	27.00	n/a
Europe	27.00	$42.00
Mexico, Central America Middle East, North Africa Central Africa	27.00	48.00
South America, South Africa Far East, Australasia	27.00	72.00

MICRO Books

	At Your Dealer	**Ordered by Mail**	
		Surface	Air Mail (Not U.S./Canada)
Best of MICRO, Vol. 1	$ 6.00	$ 8.00	$12.00
Best of MICRO, Vol. 2	8.00	10.00	15.00
Best of MICRO, Vol. 3	10.00	12.00	18.00
What's Where in the Apple	14.95	16.95	19.95
MICRO on the Apple (each volume)	24.00		

Note: Circle desired item.

Subscription rates are subject to change without notice. These prices are current as of January 1982.

☐ Check enclosed for $ ____________
☐ Bill VISA
☐ Bill MasterCard

____
Signature Card Number Expires

Please print

____
Name

____
Street

____
City State/Province/Country Code

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 60 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

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

This is a gummed flap. Moisten and fold down to seal automatic envelope.

- All orders must be prepaid in U.S. dollars or charged to your Master Charge or Visa.
- Make checks and international money orders payable to MICRO.

Call us today.

Our toll free number is:

1-800-227-1617 • Ext.564

SUBSCRIBER ORDER FORM

Yearly Subscription (ISSN 027-9002) Save 20% off single issue price.

U.S. DOLLARS

	SURFACE	AIR MAIL	
*United States	$24.00	n/a	United States
Canada	27.00	n/a	Canada
Europe	27.00	$42.00	Europe
Mexico, Central America, Mid East, No. & Central Africa	27.00	48.00	Mexico, Central America, Mid East, No. & Central Africa
So. America, Far East, So. Africa, Australasia	27.00	72.00	So. America, Far East, So. Africa, Australasia

*SPECIAL OFFER - save even more - 30% off single issue price - U.S. 2 yrs. $42.00.

Circle desired item.

Total for Service Selected $_____

OKAY! I'm an intelligent MICROcomputer user: Send me a subscription to MICRO.

Name: ______________ Occupation: ______________

Address: ______________

City: ______________ State: ______________ Zip: ______________

Country (if not U.S.): ______________ M.C.# ______________ Visa# ______________

Help **MICRO** bring you the kind of information you want by completing this short questionnaire.

Microcomputers Owned/Planning to Buy: AIM APPLE ATARI KIM OSI PET SYM Other: ______________

Peripherals Owned/Planning to Buy: Memory Disk Video Printer Other: ______________

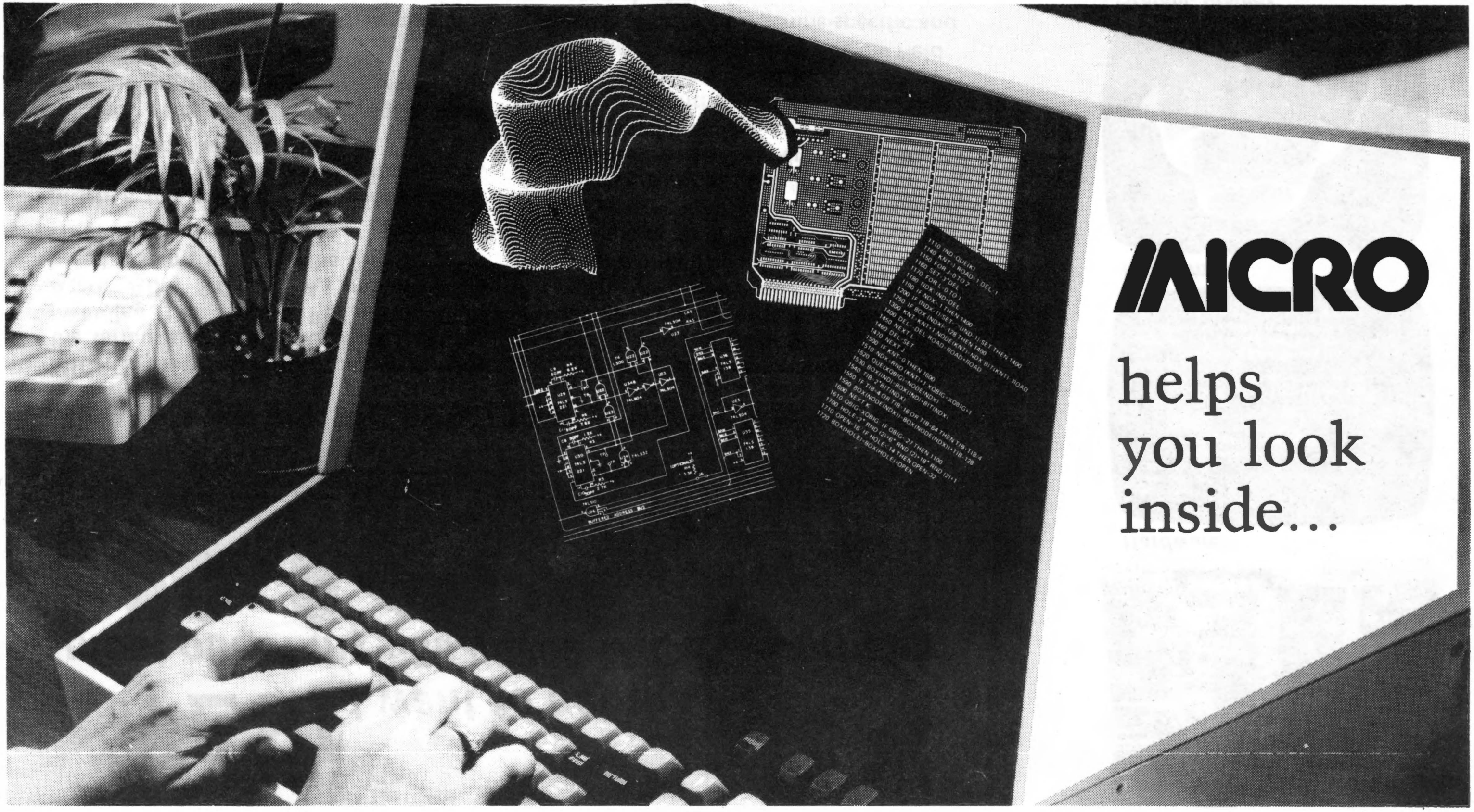
Microcomputer Usage: Educational Business Personal Control Games Other: ______________

Languages Used: Assembler Basic Forth Pascal Other: ______________

Your comments and/or suggestions on **MICRO**: ______________

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

Bulk Rate
U.S. Postage
PAID
Permit
No. 64
Chelmsford, MA
01824



MICRO/ APPLE

Volume 2

Recorded in 16 sector format (DOS 3.3)

198a