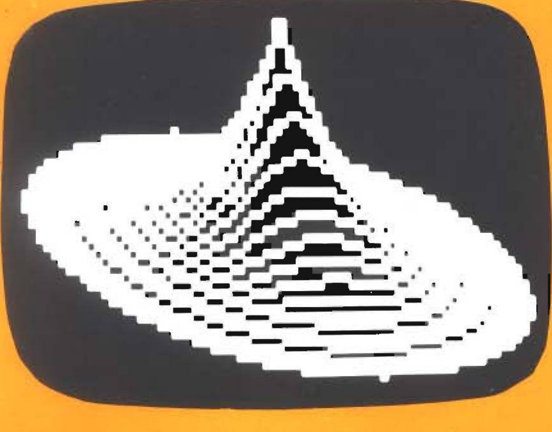
MICRO™

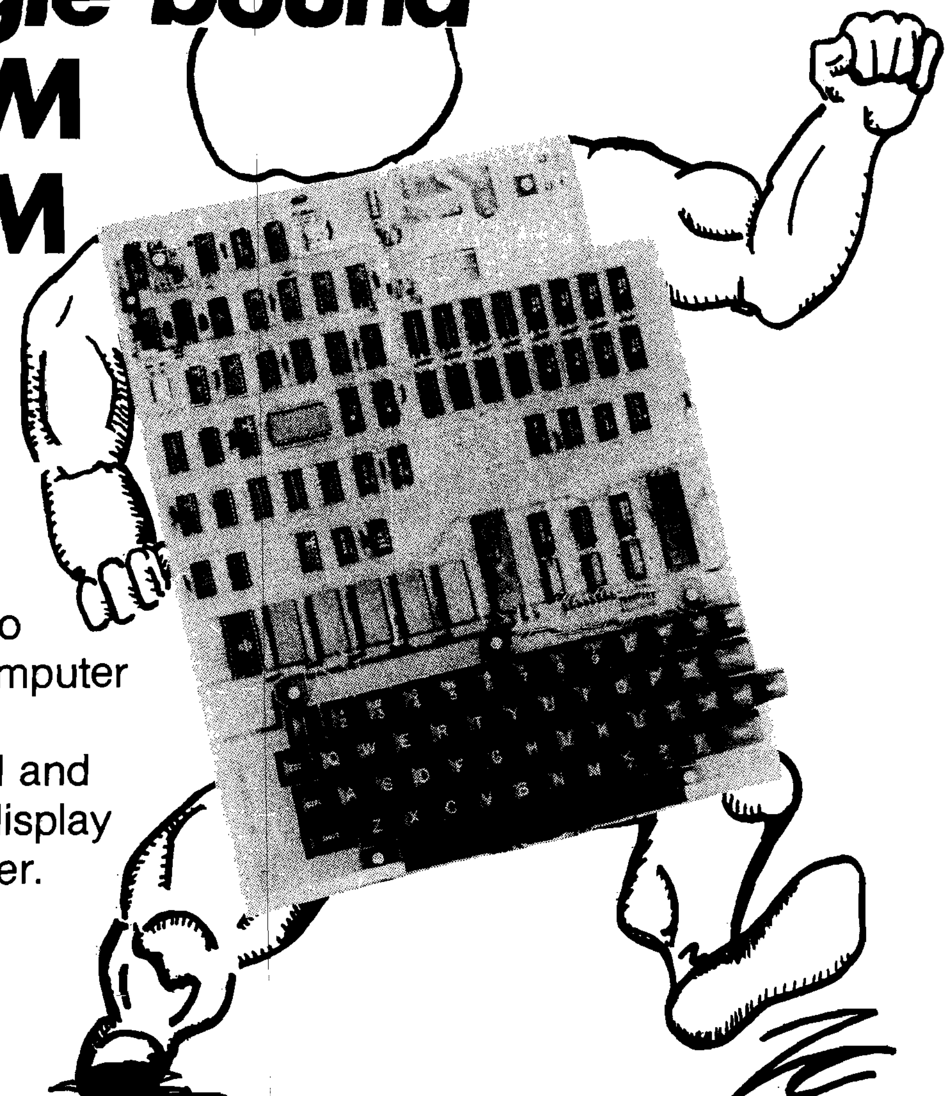
The Magazine of the APPLE, KIM, PET and Other 6502 Systems



```
1F08                  ORG    $1F08
1F08 A2 00            LDXIM  $00    BEGIN PLOTTING ROUTINE
1F0A A0 02            LDYIM  $02
1F0C B1 7C     LOOP   LDAIY  $7C    TRANSFER VALUES FOR X%
1F0E 10 10            BPL    POSI   Y%, Z% AND P%, Q%,
1F10 C8               INY           R% TO EX, WY, ZE,
1F11 C9 FF            CMPIM  $FF    AND PE, QU, AR,
1F13 F0 05            BEQ    NEGI   RESPECTIVELY.  REDUCE
1F15 A9 80     BITR   LDAIM  $80    VALUES FROM TWO
1F17 4C 2A 1F         JMP    STOR   BYTES TO ONE
1F1A B1 7C     NEGI   LDAIY  $7C
1F1C 10 F7            BPL    BITR
1F1E 30 0A            BMI    STOR
1F20 D0 05     POSI   BNE    TMCH
1F22 C8               INY
1F23 B1 7C            LDAIY  $7C
1F25 10 03            BPL    STOR
1F27 A9 7F     TMCH   LDAIM  $7F
1F29 C8               INY
1F2A 95 23     STOR   STAX   PE
1F2C E8               INX
1F2D 98               TYA
1F2E 18               CLC
1F2F 69 06            ADCIM  $06
1F31 A8               TAY
1F32 C9 2C            CMPIM  $2C    CHECK FOR END OF TRANSFER
1F34 D0 D6            BNE    LOOP
1F36 A5 23            LDA    PE     COMPUTE EXP = PE + EX
1F38 18               CLC
```

Plotting a Revolution

NO 16 **September 1979** $2.00



MICRO™

September 1979
Issue Number Sixteen

Table of Contents

Plotting a Revolution — 5
by John Sherburne

An AIM 65 Notepad — 11
by Dr. Marvin L. De Jong

Applesoft Renumbering — 15
by J.D. Childress

Move It: Relocating PET Source Programs and Object Code — 17
by Professor Harvey B. Herman

Life in the Fast Lane — 21
by Richard R. Auricchio

SYM-1 Event Timer — 26
Stephen J. Faris

AIM 65 in the Ham Shack — 29
by Dr. Marvin L. De Jong

MICROBES — 34

Speech Processor for the PET — 35
by Charles R. Husbands

Tiny Pilot: An Educational Language for the 6502 — 41
by Nicholas Vrtis

The MICRO Software Catalogue: XII — 51
by Mike Rowe

8080 Simulation with a 6502 — 53
by Dann McCreary

Writing for MICRO — 59
by Shawn Spilman

Staff

Publisher
Robert M. Tripp

Editor
Shawn Spilman

Assistant Editor
Mary Ann Curtis

Business Manager
Maggie E. Fisher

Circulation Manager
Carol Ann Stark

Intern
L. Catherine Bland

MICRO™ is published monthly by
MICRO Ink, Inc.
34 Chelmsford Street
Chelmsford, Massachusetts
617/256-5515
Second-class postage paid at
Chelmsford, MA 01824
POSTMASTER: Send address changes to
MICRO
P.O. Box 6502
Chelmsford, MA 01824
Publication Number: COTR 395770
Subscription in United States:
$15.00 per year/12 issues

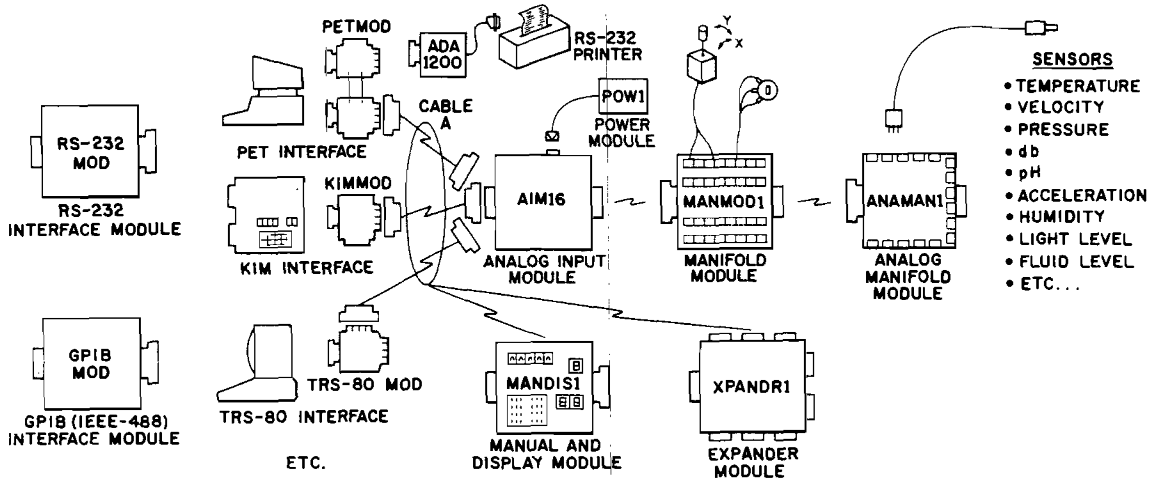
Advertiser's Index

AB Computers	25	Progressive Software	64
Beta Computer Devices	25	P.S. Software House	58
COMPAS Microsystems	50	Pygmy Programming	25
Computer Components	32, 33	Rainbow Computing, Inc.	IFC
The Computerist, Inc.	22, 23, 47	RNB Enterprises	14
Computer Shop	2	SKYLES Electronic Works	38, 39, 40
Connecticut microComputers	4	Small System Services, Inc.	63
Electronic Specialists, Inc.	57	Softouch, Inc.	57
EXCERT, Inc.	1	Softside Software	IBC
H. Geller Computer Systems	19	SYBEX	57
Hudson Digital Electronics	28	Synergistic Software	58
Optimal Technology, Inc.	42	Textcast	58
Powersoft, Inc.	20	Weldon Electronics	49
Programma International	BC	West Side Electronics	58

CONNECTICUT microCOMPUTER, Inc.

150 POCONO ROAD - BROOKFIELD, CONNECTICUT 06804

TEL: (203) 775-9659 *TWX: 710-456-0052*



DAM SYSTEMS by CmC
A complete system of modules to let your computer listen
to the real world.

DAM SYSTEMS PRICE LIST

DAM SYSTEMS components

AIM161 - Analog Input Module $179.00
16 8-bit analog inputs - 100 microsecond conversion time - 3 state output - requires one 8-bit computer output port for control and one 8-bit computer input port for data.

POW1 - Power Module $14.95
Supplies power for one AIM16 module.

ICON - Input Connector $9.95
For connecting analog inputs to the AIM16 - 20 pin card edge connector - solder eyelets.

OCON - Output Connector $9.95
For connecting the AIM16 to a computer - 20 pin card edge connector - solder eyelets.

MANMOD1 - Manifold Module $59.95
Use in place of ICON. Screw terminal barrier strips for connecting joysticks, potentiometers, voltage sources, etc. Eliminates the need for soldering. Plugs into the AIM16.

ANAMAN1 - Analog Manifold Module TBA
Use in place of ICON. Connects DAM SYSTEMS SENSORS to the AIM16 without soldering - sensor cables just plug in. Plugs into the AIM16 or the MANMOD1.

SENSORS TBA
Sensors for temperature, pressure, flow, humidity, level, pH, motion, etc.

COMPUTER INTERFACES TBA
For the PET, KIM, TRS-80, etc. Use in place of OCON. Eliminates the need for soldering or special construction.

PETMOD - PET Interface Module $49.95
Gives two IEEE ports, one user port and one DAM SYSTEMS interface port. Saves wear and tear on the PET's printed circuit board. Also called the PETSAVR.

KIMMOD - KIM Interface Module $39.95
Gives one application connector port and one DAM SYSTEMS interface port.

CABLE "A" - Interconnect Cables TBA
Connects computer interface to AIM16, MANDIS1, XPANDR1, etc.

CABLE A24 - Interconnect Cable $19.95
24 inch cable with interface connector on one end and an OCON equivalent on the other.

MANDIS1 - Manual and Display Module TBA
Connects between the AIM16 and the computer interface. Allows manual or computer control of the AIM16. Displays channel number and data.

GPIB MOD - GPIB (IEEE-488) Interface TBA
Allows the DAM SYSTEMS MODULES to be used with the GPIB bus instead of a computer's other I/O ports.

RS232 MOD - RS232 Interface Module TBA
Allows the DAM SYSTEMS MODULES to be used with an RS-232 port or terminal.

XPANDR1 - Expander Module TBA
Allows up to 128 8-bit analog inputs (8 AIM16 Modules) to be connected to one system.

DAM SYSTEMS sets

AIM161 Starter Set 1 $189.00
Includes one AIM161, one POW1, one ICON and one OCON.

AIM161 Starter Set 2 $239.00
Includes one AIM161, one POW1, one MANMOD1 and one OCON.

PETSET1a $295.00
Includes one PETMOD, one CABLE A24, one AIM161, one POW1 and one MANMOD1.

KIMSET1a $285.00
Includes one KIMMOD, one CABLE A24, one AIM161, one POW1 and one MANMOD1.

Plotting a Revolution

John Sherburne
206 Goddard
White Sands Missile Range, NM 88002

An assembly language plotting routine that is callable from BASIC will simplify and speed up the high resolution plotting process.

What does fomenting rebellion have to do with microcomputing? Plotting a revolution refers to the creation of three dimensional figures, called solids of revolution, that are formed by rotating a two dimensional figure about an axis to form a solid.

Solids of revolution can be generated and displayed, under BASIC, by using a fast, general purpose assembly language plotting routine and a technique that allows the assembly language routine to access BASIC variables. The plotting routine and the BASIC language interface are building blocks used to construct a generalized program to display solids of revolution.

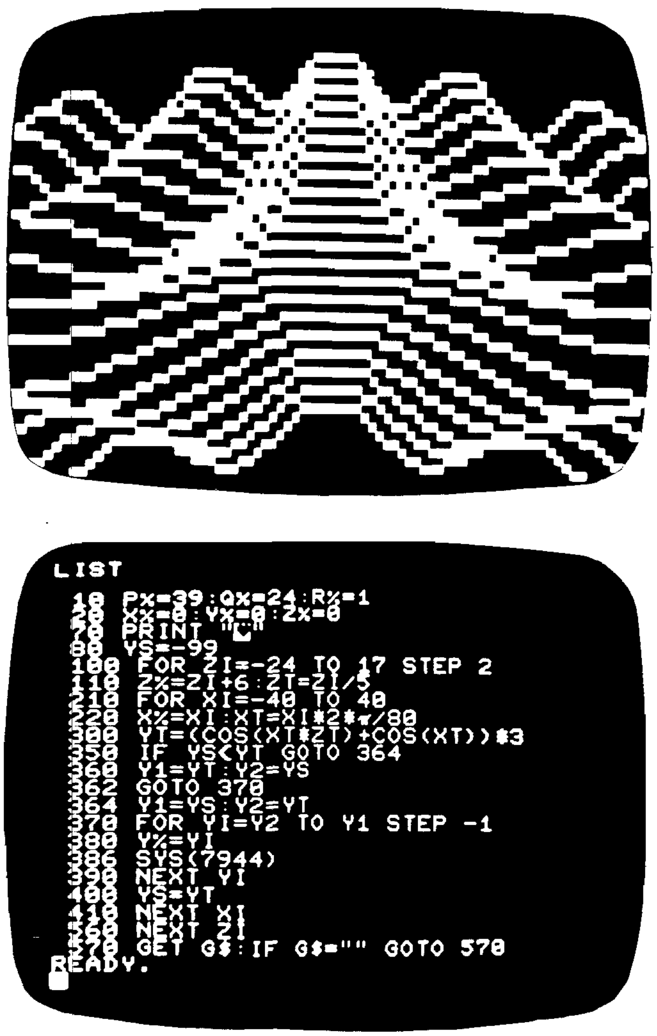


Figure 1: Two sine waves of different frequencies.

Plotting Routine

The purpose of the assembly language plotting routine is to simplify and speed up the high resolution plotting process. It also allows the operator to choose any point as the center and plot coordinates relative to that center, and it allows the option of plotting with a 45 degree perspective. To accomplish all this, six parameters must be passed from BASIC: P%, Q%, R%, X%, Y% and Z%.

P% and Q% are the screen location for the center of the plot. The screen contains 80 x 50 plot positions so P% = 40 and Q% = 25 would plot relative to the center of the screen.

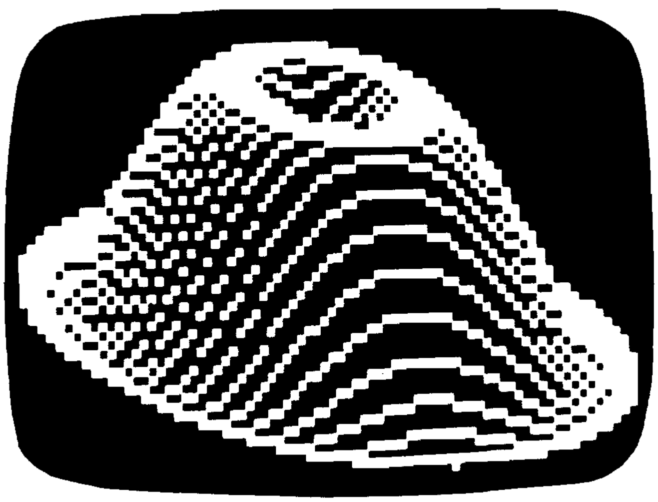
R% specifies the type of plot. If the zero bit of R% is set (R% is odd), the plot is displayed as though viewed straight on. If R% is even, the plot is at a 45 degree perspective to the viewer's right.

If the one bit of R% is not set, the plotting routine will plot over any non-plot characters on the screen and erase them. If the one bit is set, non-plot characters on the screen will not be erased. The other bits of R% are ignored.

X%, Y% and Z% are the coordinates of the point to be plotted. The X axis is horizontal, the Y axis is vertical and the Z axis is either vertical or at a 45 degree angle, depending on R%.

The most complex problem in making three dimensional plots is to draw only lines which are visible and to eliminate lines hidden to the viewer. The plotting routine can perform hidden line elimination automatically for one type of figure; a figure which can be imagined as an object covered by a very large, tight fitting sheet. More precisely, the figure must have a single Y value for each (X,Z) value pair, and the bottom of the figure will be hidden from view.

If such a figure is plotted, starting with the lowest value of Z and progressing in order to the highest value of Z, the



```
10 P%=39:Q%=19:R%=0
20 X%=0:Y%=0:Z%=0
30 ZP=16
40 XR=1.5*π:XP=35
50 YR=1:YP=14
60 ZF=XR/ZP:XF=XR/XP:YF=YP/YR
70 PRINT "▚"
100 FOR ZI=-Q% TO (49-Q%)
110 IF ZI<-ZP OR ZI>ZP GOTO 560
130 ZT=ZI*XP/ZP
140 Z%=ZI
150 YS=-20
200 XL=INT(.5+SQR(XP*XP-ZT*ZT))
210 FOR XI=-XL TO XL
220 XT=SQR(XI*XI+ZT*ZT)
230 XT=XT*XF
240 X%=XI
300 YT=SIN(XT)*YF
380 Y%=YT:SYS(7944)
400 YS=YT
410 NEXT XI
560 NEXT ZI
570 GET G$:IF G$="" GOTO 570
```

Figure 2: Sine wave rotated about the Y axis.

```
                MICRO-WARE ASSEMBLER 65XX-1.0

                            *
                            * PLOTTING A REVOLUTION
                            * MODIFIED 7-17-79 BY MICRO STAFF
                            *
                            * PAGE ZERO VARIABLES FOR VERIFICATION ROUTINE:
1F08                        STAT   *     $0023
1F08                        VFLAG  *     $0024
1F08                        VARY   *     $0025

                            * PAGE ZERO VARIABLES FOR PLOTTING ROUTINE:
1F08                        PE     *     $0023
1F08                        QU     *     $0024
1F08                        AR     *     $0025
1F08                        EX     *     $0026
1F08                        WY     *     $0027
1F08                        ZE     *     $0028
1F08                        EXPP   *     $0051
1F08                        WYPP   *     $0052
1F08                        RHI    *     $0053
1F08                        EXP    *     $0054
1F08                        WYP    *     $0055
1F08                        CHAR   *     $0056
1F08                        FLAG   *     $0057

1F08                               ORG   $1F08
1F08 A2 00                         LDXIM $00    BEGIN PLOTTING ROUTINE
1F0A A0 02                         LDYIM $02
1F0C B1 7C                  LOOP   LDAIY $7C    TRANSFER VALUES FOR X%
1F0E 10 10                         BPL   POSI   Y%, Z% AND P%, Q%,
1F10 C8                            INY          R% TO EX, WY, ZE,
1F11 C9 FF                         CMPIM $FF    AND PE, QU, AR,
1F13 F0 05                         BEQ   NEGI   RESPECTIVELY.  REDUCE
1F15 A9 80                  BITR   LDAIM $80    VALUES FROM TWO
1F17 4C 2A 1F                      JMP   STOR   BYTES TO ONE
1F1A B1 7C                  NEGI   LDAIY $7C
1F1C 10 F7                         BPL   BITR
1F1E 30 0A                         BMI   STOR
1F20 D0 05                  POSI   BNE   TMCH
1F22 C8                            INY
1F23 B1 7C                         LDAIY $7C
1F25 10 03                         BPL   STOR
1F27 A9 7F                  TMCH   LDAIM $7F
1F29 C8                            INY
1F2A 95 23                  STOR   STAX  PE
1F2C E8                            INX
1F2D 98                            TYA
1F2E 18                            CLC
1F2F 69 06                         ADCIM $06
1F31 A8                            TAY
1F32 C9 2C                         CMPIM $2C    CHECK FOR END OF TRANSFER
1F34 D0 D6                         BNE   LOOP
1F36 A5 23                         LDA   PE     COMPUTE EXP = PE + EX
1F38 18                            CLC
1F39 65 26                         ADC   EX
1F3B 70 08                         BVS   OFLO
1F3D 46 25                         LSR   AR     CHECK AR, IF ODD 90 DEGREE
1F3F B0 06                         BCS   CLER   PLOT, IF EVEN 45 DEGREES.
1F41 65 28                         ADC   ZE     IF 45, ADD AR TO EXP
1F43 50 02                         BVC   CLER
1F45 A9 7F                  OFLO   LDAIM $7F    SET TO 7F ON OVERFLOW
1F47 85 54                  CLER   STA   EXP
1F49 18                            CLC
1F4A A5 24                         LDA   QU     COMPUTE WYP = WY + QU
1F4C 65 28                         ADC   ZE
1F4E 70 05                         BVS   OVRF
1F50 38                            SEC
1F51 E5 27                         SBC   WY
1F53 50 02                         BVC   OKEY
```

hidden line problem will be simplified greatly. In fact, it becomes only a matter of printing the value of Y for each (X,Z) and eliminating all previously plotted lower values of Y. The plotting routine accomplishes this process by simply erasing all points below the currently plotted point.

Besides having to plot all points in increasing order of Z, the procedure requires that, for a given value of Z, an (X,Y) be computed for each feasible value of X. Otherwise gaps in the plot might leave non-visible points unerased. This process can be imagined as cutting the figure into slices parallel to the XY plane and then stacking the slices up in Z value order to reconstitute the figure.

If the hidden line function is not wanted, it can be turned off with the statement "POKE 8181,96". This will cause the routine to plot a point at X%, Y%, Z% without erasing lower points. Of course if Z% is held constant at zero, the routine is equivalent to a two dimensional plotting function.

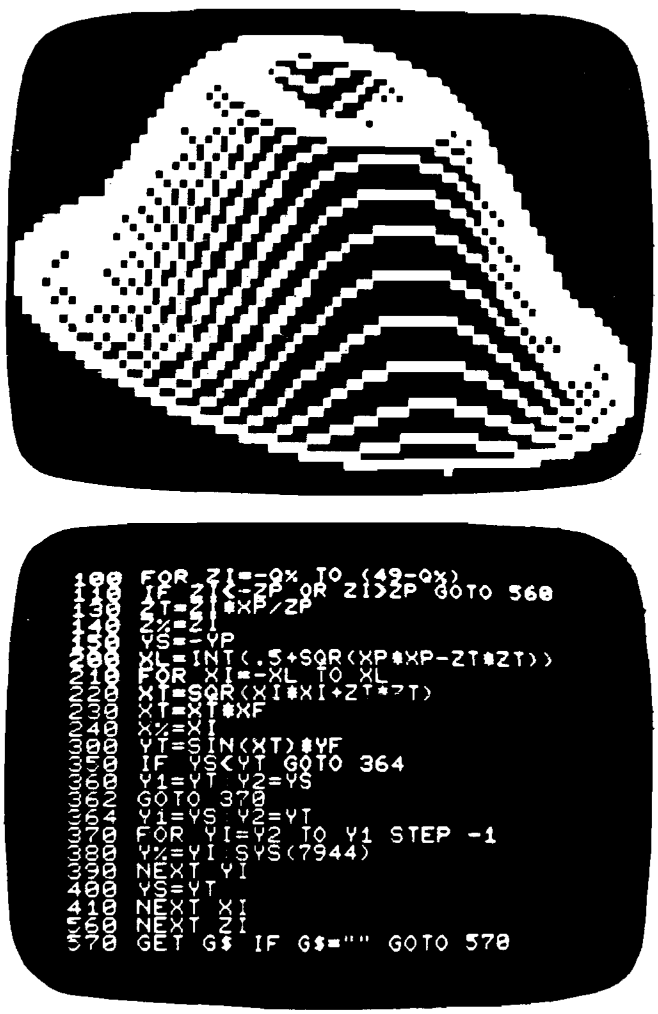


Figure 3: The problem has been corrected by drawing a line between plot points and eliminating the gaps.

```
1F55 A9 7F      OVRF   LDAIM $7F    SET TO 7F ON OVERFLOW
1F57 85 55      OKEY   STA   WYP
1F59 A9 00             LDAIM $00
1F5B 85 57             STA   FLAG   SET FLAG FOR FIRST ITERATION
1F5D 85 53      MAIN   STA   RHI    BEGIN PLOT
1F5F 85 56             STA   CHAR
1F61 A5 54             LDA   EXP
1F63 85 51             STA   EXPP
1F65 30 04             BMI   COUT   CHECK FOR EXP GREATER
1F67 C9 50             CMPIM $50    THAN ZERO, LESS THAN 80
1F69 90 01             BCC   YCHK
1F6B 60         COUT   RTS
1F6C A5 55      YCHK   LDA   WYP
1F6E 85 52             STA   WYPP
1F70 30 7B             BMI   RTRN   CHECK FOR WYP GREATER
1F72 C9 32             CMPIM $32    THAN ZERO, LESS THAN 50
1F74 B0 F5             BCS   COUT
1F76 46 51      GOON   LSR   EXPP   DIVIDE EXP AND WYP BY 2
1F78 90 02             BCC   XQUAD  COMPUTE QUADRANT OF
1F7A E6 56             INC   CHAR   PLOT POINT WITHIN THE
1F7C 46 52      XQUAD  LSR   WYPP   SCREEN POSITION
1F7E 90 04             BCC   YQUAD
1F80 E6 56             INC   CHAR
1F82 E6 56             INC   CHAR
1F84 A9 01      YQUAD  LDAIM $01    SET BIT IN CHAR CORRESPONDING
1F86 A4 56      ROND   LDY   CHAR
1F88 F0 06             BEQ   ROUT   TO QUADRANT
1F8A 0A                ASLA         OF PLOT POINT
1F8B C6 56             DEC   CHAR
1F8D 4C 86 1F          JMP   ROND
1F90 85 56      ROUT   STA   CHAR
1F92 06 52             ASL   WYPP   TRANSLATE RELATIVE COORD
1F94 06 52             ASL   WYPP   OF PLOT POINT TO
1F96 06 52             ASL   WYPP   SCREEN LOCATION FOR
1F98 A5 52             LDA   WYPP   POINT.  X + 40 * Y
1F9A 06 52             ASL   WYPP
1F9C 26 53             ROL   RHI
1F9E 06 52             ASL   WYPP
1FA0 26 53             ROL   RHI
1FA2 65 52             ADC   WYPP
1FA4 85 52             STA   WYPP
1FA6 A5 53             LDA   RHI
1FA8 69 00             ADCIM $00
1FAA 85 53             STA   RHI
1FAC A5 52             LDA   WYPP
1FAE 65 51             ADC   EXPP
1FB0 85 52             STA   WYPP
1FB2 90 02             BCC   PLUS
1FB4 E6 53             INC   RHI
1FB6 18         PLUS   CLC
1FB7 A9 80             LDAIM $80
1FB9 65 53             ADC   RHI
1FBB 85 53             STA   RHI
1FBD A0 10             LDYIM $10    FIND CHARACTER ALREADY
1FBF A2 00             LDXIM $00    AT SCREEN LOCATION
1FC1 A1 52             LDAIX WYPP
1FC3 88         AGIN   DEY
1FC4 D9 B3 1E          CMPY  TABLE
1FC7 F0 0B             BEQ   NOVR
1FC9 C0 00             CPYIM $00
1FCB D0 F6             BNE   AGIN
1FCD A6 25             LDX   AR     IF CHARACTER NOT IN TABLE,
1FCF F0 03             BEQ   NOVR   CHECK AR FOR OVERWRITE
1FD1 4C ED 1F          JMP   RTRN   INDICATOR (2ND BIT)
1FD4 A5 57      NOVR   LDA   FLAG   FIRST ITERATION?
1FD6 F0 0C             BEQ   SETR
1FD8 A5 56             LDA   CHAR   IF NOT FIRST ITERATION,
1FDA 49 0F             EORIM $0F    BLANK OUT CHARACTER ON SCREEN
1FDC 85 56             STA   CHAR
1FDE 98                TYA
1FDF 25 56             AND   CHAR
1FE1 4C E7 1F          JMP   PLOT
1FE4 98         SETR   TYA          IF FIRST ITERATION,
1FE5 05 56             ORA   CHAR   PRINT NEW CHARACTER
1FE7 A8         PLOT   TAY
```

```
1FE8 B9 B3 1E         LDAY  TABLE
1FEB 81 52            STAIX WYPP
1FED A9 FF      RTRN  LDAIM $FF
1FEF 85 57            STA   FLAG    SET FLAG TO NEXT ITERATION
1FF1 E6 55            INC   WYP     INCREMENT POINT FOR ERASING
1FF3 A9 00            LDAIM $00     AREA BELOW POINT
1FF5 4C 5D 1F         JMP   MAIN
1EB3                  ORG   $1EB3
1EB3 20         TABLE =     $20
1EB4 7E               =     $7E
1EB5 7C               =     $7C
1EB6 EC               =     $EC
1EB7 7B               =     $7B
1EB8 61               =     $61
1EB9 FF               =     $FF
1EBA EC               =     $EC
1EBB 6C               =     $6C
1EBC 7F               =     $7F
1EBD E1               =     $E1
1EBE FB               =     $FB
1EBF 62               =     $62
1EC0 FC               =     $FC
1EC1 FE               =     $FE
1EC2 A0               =     $A0
1EC3 A9 CF      IFACE LDAIM $CF     BEGIN VERIFICATION ROUTINE
1EC5 85 25            STA   VARY
1EC7 A0 01            LDYIM $01
1EC9 84 24            STY   VFLAG
1ECB 88               DEY
1ECC 84 23            STY   STAT
1ECE E6 25      INIT  INC   VARY    LOAD CONTENT OF WORKING
1ED0 18               CLC           STORAGE AREA
1ED1 26 24            ROL   VFLAG
1ED3 B1 7C            LDAIY $7C
1ED5 C5 25            CMP   VARY    COMPARE VARIABLE NAME
1ED7 F0 06            BEQ   CHEK    FIRST CHARACTER
1ED9 A5 24            LDA   VFLAG   IF CHARACTER INCORRECT
1EDB 05 23            ORA   STAT    UPDATE STAT
1EDD 85 23            STA   STAT
1EDF C8         CHEK  INY
1EE0 B1 7C            LDAIY $7C
1EE2 C9 80            CMPIM $80     SECOND CHARACTER
1EE4 F0 06            BEQ   RPET
1EE6 A5 24            LDA   VFLAG   IF CHARACTER INCORRECT
1EE8 05 23            ORA   STAT    UPDATE STAT
1EEA 85 23            STA   STAT
1EEC 98         RPET  TYA
1EED 18               CLC
1EEE 69 06            ADCIM $06     MORE TO NEXT VARIABLE
1EF0 A8               TAY
1EF1 C9 2A            CMPIM $2A
1EF3 F0 0D            BEQ   END     CHECK FOR END OF LOOP
1EF5 C9 15            CMPIM $15     BEGIN ROUTINE TO SKIP
1EF7 D0 D5            BNE   INIT    FROM R% TO X%
1EF9 A5 25            LDA   VARY
1EFB 69 04            ADCIM $04
1EFD 85 25            STA   VARY
1EFF 4C CE 1E         JMP   INIT
1F02 A5 23      END   LDA   STAT
1F04 8D 0C 02         STA   $020C   STORE STAT IN STATUS
1F07 60               RTS

SYMBOL TABLE 2000 2114
AGIN   1FC3    AR     0025    BITR   1F15    CHAR   0056
CHEK   1EDF    CLER   1F47    COUT   1F6B    END    1F02
EX     0026    EXPP   0051    EXP    0054    FLAG   0057
GOON   1F76    IFACE  1EC3    INIT   1ECE    LOOP   1F0C
MAIN   1F5D    NEGI   1F1A    NOVR   1FD4    OFLO   1F45
OKEY   1F57    OVRF   1F55    PE     0023    PLOT   1FE7
PLUS   1FB6    POSI   1F20    QU     0024    RHI    0053
ROND   1F86    ROUT   1F90    RPET   1EEC    RTRN   1FED
SETR   1FE4    STAT   0023    STOR   1F2A    TABLE  1EB3
TMCH   1F27    VARY   0025    VFLAG  0024    WY     0027
WYPP   0052    WYP    0055    XQUAD  1F7C    YCHK   1F6C
YQUAD  1F84    ZE     0028
```

The routine is written to reside in the upper portion of 8K RAM. Under most circumstances, this area is not used until the BASIC program employing the routine gets too large. If the BASIC program requires certain string manipulations, however, PET may use high RAM for string working area and clobber the plotting routine. For example, the sequence A$ = "123" : B% = "456": C$ = A% + B$ will cause C$ to be stored in high RAM and destroy the plotting routine.

The routine is designed to be saved using the PET Machine Language Monitor. The routine is first entered in memory using the monitor and then saved with "S,01,PLOTTER,1EB3,1FF8". Once saved, the routine can be loaded as would any other program, with LOAD "PLOTTER". The BASIC program using the routine can then be loaded in the normal manner. μ

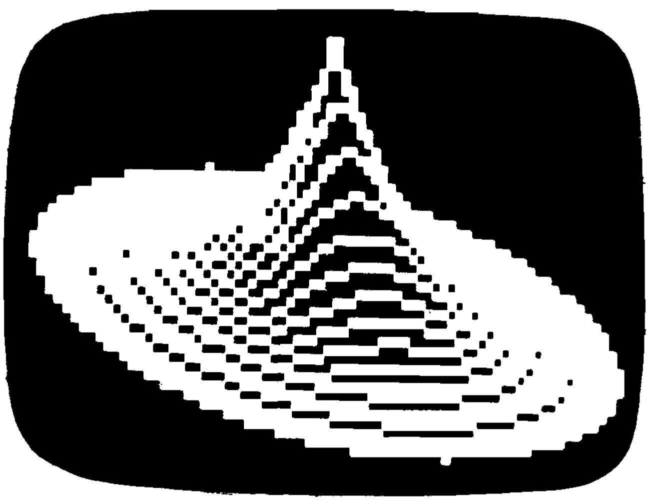


Figure 4: The solid formed by rotating Y = 15 exp (−X/3) about the Y axis.

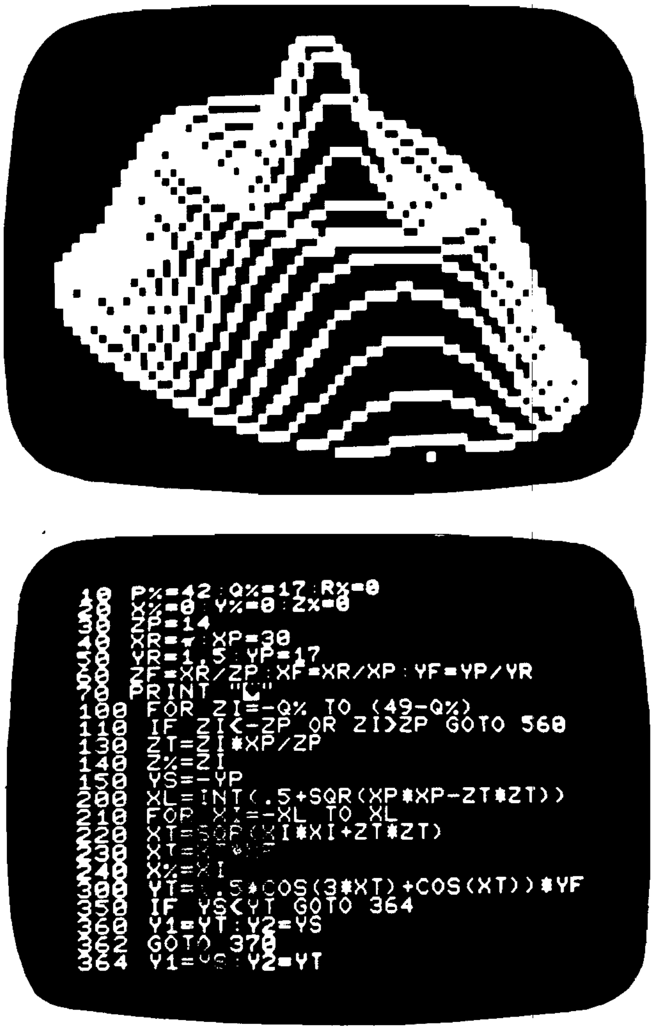


Figure 5: The solid formed by rotating Y = 1/2 cos (3X) + cos(X) about the Y axis.

BASIC Interface

Since the assembly language routine requires that the six parameters be passed from BASIC, the USR function with its single parameter argument cannot be used. POKE will not work, either, because it will not accept negative values. The method I used to overcome this problem was to have the assembly language routine access the BASIC working area to obtain the required parameters.

After the run command is given, PET BASIC takes each variable in the order it is encountered and creates a working storage area for it following the last BASIC statement. For non-subscripted variables the working storage is seven bytes long. The first two bytes are the variable name and the next five are the current value.

Floating point variables are stored in normalized form, while integer variables are stored as two-byte signed numbers. The address of the starting byte of the variable storage area is stored in location $7C. For simplicity, the plotting routine assumes that the six required parameters—P%, Q%, R%, X%, Y%, and Z%—are the first six variables in the program and are in that order.

To insure that all the required variables are in the proper place and in the proper sequence, the assembly language program includes a verification routine starting at location $1EC3. This routine is called with the statement "SYS (7875)" and checks for the presence and correct sequence of each parameter. The results of the checks are stored in the PET status word at location $020C.

If the value of the status word, ST, is zero, all variables were located. If one of the variables is not located, the corresponding bit of ST will be set. For example, ST = 6 would mean bits 1 and 2 are set and thus that P% and Q% were not found. Bit zero is not used. A typical sequence to establish and verify the BASIC routine would be:

The important point is that the six plotting variables must be the first six variables mentioned in the program, and they must be in the required sequence. Normally, the verification routine will only be used for diagnostic purposes. The plotting routine itself is entered with the statement "SYS (7944)".

BASIC Programs

The plotting routine described above can be used for any three dimensional plot that satisfies the requirements of being single valued in Y and having only the upper surface visible. For example, Figure 1 is a graph showing the effects of combining two sine waves of different frquencies. The difference in frequency is a function of Z; Y is amplitude and X is time.

Figure 2 is a solid of revolution formed by rotating a sine wave about the Y axis. The program is written a generalized format, and any function can be used in line 300 as the function generating the solid.

The scale and perspective of the figure are determined in lines 30 thru 50. XR is the actual maximum value that X can take, while −XR is the minimum. XP is the number of plot points that the distance XR will cover. For Figure 2, the X value runs from −1.5 pi to 1.5 pi and is plotted from −35 to 35. Changing XP changes the width of the plotted figure.

Similarly, the actual range of Y is YR, and YP is the plotted range of Y. Changing YP changes the heighth of the plotted figure. The XZ cross section of the figure is circular so the actual range of Z is the same as X − XR. However, the plotted range of Z − ZP depends upon perspective.

The larger ZP, the greater the apparent depth of the figure and the higher the apparent position of the viewer. The value YS in line 150 represents the lowest plotted value of Y or the base of the figure.

A potential problem with the program is that while each point in the X direction is plotted, not every point in the Y direction is. Thus for Z = 0, two consecutive plot points might be (X_1 = 3, Y_1 = 12) and (X_2 = 4, Y_2 = 9). While X_1 and X_2 are adjacent, Y_1 and Y_2 are not. The problem of such gaps is esthetically more severe with some figures than others.

In Figure 3, the problem has been corected by drawing a line between plot points and eliminating the gaps. The program for Figure 3 is the same as for Figure 2 except that the section between lines 300 and 400 has been modified.

Figure 4 is a plot of the solid formed by revolving $Y = 15e^{-x/3}$ about the Y axis. The program is the same as for Figure 3 except that line 300 contains the new function, line 150 is changed, and lines 10-50 contain the new center and scaling factors. Figure 5 is the solid of revolution of $Y = \frac{1}{2}\cos(3x) + \cos(x)$ about the Y axis. The program is the same as that of Figures 3 and 4 except for lines 10-50, 150, and 300.

The process of rotating the plane figure to obtain a solid of revolution is illustrated in Figures 5 and 6. As described before, the plot for a given value of Z is equivalent to a vertical slice through the solid parallel to the X axis. Figures 5 and 6 represent views of a solid form above and the dotted line is the path of a vertical slice.

The maximum radius of the solid is XT. The apparent distance of a point on the circle from the circle's center (view-

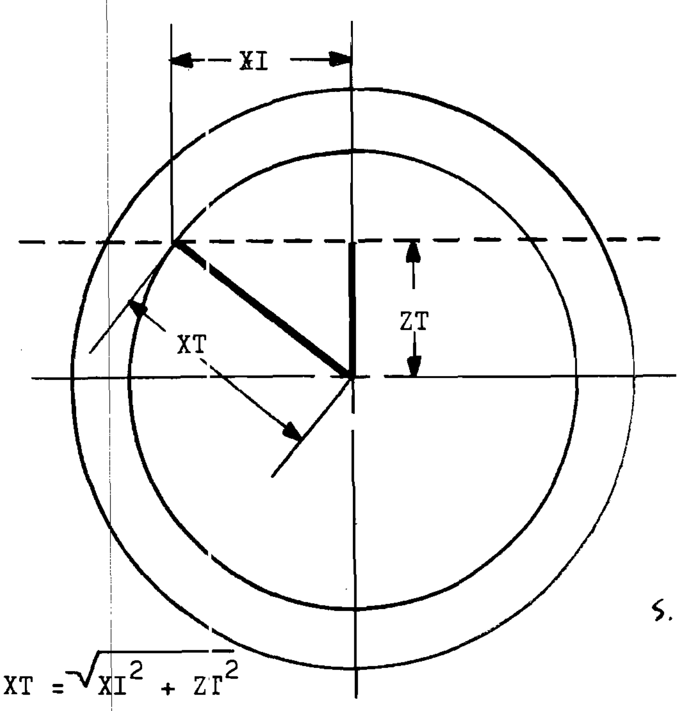


Figure 7: View of a solid form above illustrating the problem of computing the Y value for each X value.

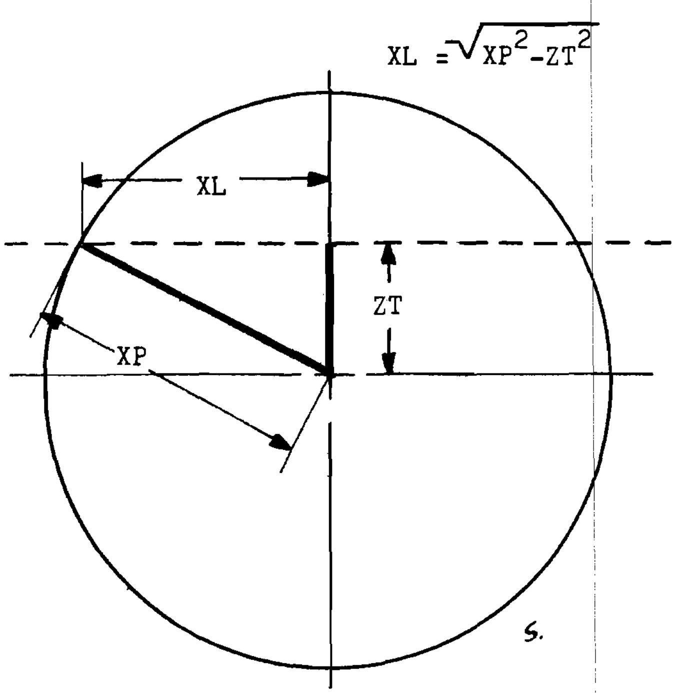


Figure 6: View of a solid from above. The dotted line is the path of a vertical slice.

ed straight on) is XL. XL is computed in line 200 of the programs for Figures 2 through 5.

The next step is to find the Y value for each point on the dotted line between −XL and XL. The FOR loop in line 210 insures that each possible X value is used. The process of computing the Y value for each X value is largely the reverse of the process described above and is illustrated in Figure 6.

Viewed from the top, the contours of the solid form concentric circles. That is, the Y value of every point at a given distance from the center is the same. For a point along the dotted line at an apparent distance XI from the center, the Y value will be the same as for a point where the inner circle crosses the Z = 0 line.

The distance of either point from the center is the square root of $(XI^2 + ZT^2)$. The calculation is performed in line 220 and the resultant distance is used to compute the Y value in line 300. The plotting function is called in line 380 and uses whatever values of X%, Y% and Z% are then current.

An AIM-65 Notepad

Dr. Marvin L. De Jong
Department of Mathematics and Physics
The School of the Ozarks
Point Lookout, MO 65726

A few short assembly language routines implement a notepad and provide the basis for versatile output to the AIM-65 display. These techniques overcome a variety of common output difficulties.

Do you want to learn how to use the 20-character AIM 65 display? This short article describes several assembly language subroutines that may be used to display input/output information. The entire program functions as a novel "notepad" that may be used to leave a message for someone else or for yourself. However, its primary utility will lie in the applications that you design which use the AIM 65 display. The program listing is given, and its description follows.

We will begin by describing some of the features of the notepad program, and then return to a description of some of the subroutines that you might want to duplicate in your assembly language programs. The notepad program allows the operator to enter a message containing from one to 256 ASCII characters (including spaces) into locations $0200 to $02FF of the AIM 65's memory space.

While entering the message, the characters typed on the keyboard are displayed on the 20-character display. The message enters the right-hand side of the display, and it is scrolled to the left. If an error is made, the DEL key allows the entire message to be backspaced, and a new character or set of characters may be entered.

Once the desired message is entered, the RETURN key starts the message circulating from right to left on the display. It circulates at a rate that makes it easy to read. If more than one space (ASCII value = $20) is encountered, the space is not displayed. Thus, a message that contains less than 256 characters does not take a noticeable amount of time to display "empty" locations.

You can leave a message to yourself such as "CALL SAM TONIGHT", or you can remind your wife to "BE SURE TO LET ROVER OUT WHEN YOU GET HOME." Of course there are much less expensive ways to do this than by purchasing AIM 65, and it is doubtful whether this notepad program will provide sufficient justification to convince your spouse that you ought to have a computer. The program is more of a novelty that might be useful as an advertising gimmick, if you are selling AIM 65s, or to impress your friends.

On the other hand, the subroutines could be useful in a large variety of programs. I use several of the subroutines in my Morse code program for the AIM 65 (available from me for $3.50). The subroutines might be useful in computer assisted instruction programs that require interaction of the computer with the operator. Or they might be useful in testing reading and comprehension speed in certain psychological tests of perception and cognition.

The read/write (RAM) memory locations from $A438 to $A43B, memory locations which are available on an off-the-shelf AIM 65, are used to store the ASCII characters to be displayed. We call these locations the display buffer. These 20 locations are filled with ASCII spaces by the subroutine CLEAR starting at address $03A0. Subroutine DISPLAY, starting at address $0360, transfers the ASCII characters in the display buffer to the AIM 65 display. It does this by making use of a subroutine in the AIM 65 monitor called OUTDD1 tht is located at $EF7B.

Subroutine OUTDD1 in the AIM 65 ROM is very useful in working with the 20-character display. The content of the X register addresses the display in the sense that X = $00 is the leftmost character on the display, and X = $13 (19) is the right-most character on the display.

The accumulator, A, must contain the ASCII representation of the character to be displayed before the jump to the OUTDD1 subroutine is made. The accumulator must also be ORAed with $80 before the subroutine call, or the cursor will be displayed. With the accumulator properly loaded and the appropriate "address' in the X register, a subroutine jump to OUTDD1 will display the character.

A jump to subroutine CLEAR, at $03A0, followed by a jump to subroutine DISPLAY will clear the display. To put some information in the display and scroll it to the left, subroutine MODIFY (starting at address $0372) is used. Subroutine MODIFY stores the contents of the accumulator in location $A44C. Then it proceeds to shift the contents of $A439 to $A438, $A43A to $A439, and so on until it finishes by shifting the contents of location $A44C to $A44B.

Once the display buffer is properly modified by subroutine MODIFY, then a subroutine call to DISPLAY will cause the down-shifted ASCII characters in the display buffer to appear as left-shifted characters on the AIM 65 display.

The sequence of events, starting at the beginning of the main program, is as follows: First, the display buffer is cleared by subroutine CLEAR. The message buffer from $0200 to $02FF is cleared (loaded with ASCII spaces).

Next, an AIM 65 monitor subroutine, READ, is called to get a character from the keyboard. As long as no key is depressed, the monitor stays in this subroutine. A key depression results in a return to the main program with the ASCII representation of the character in the accumulator. The contents of the accumulator are transferred to the message buffer, using Y as an index for the buffer's base address of $0200, unless it is the ASCII character for RETURN, DEL, or the F1 key.

The F1 key starts the entire program over. The DEL key removes the last character from the message buffer, and it backspaces (scrolls right) the display buffer and the display itself. The RETURN key starts the message, and this key should be pressed only when the desired message has been placed in the message buffer.

If a character is placed in the message buffer, then it is also displayed by calling subroutines MODIFY and DISPLAY in succession. If the message buffer is filled, or if the RETURN key is pressed, then the program will proceed to scroll the entire message across the display.

The message is displayed by getting characters from the message buffer, starting with location $0200, and then calling subroutines MODIFY and DISPLAY in succession. A time delay is in-

```
0010:
0020:                    * MAIN PROGRAM
0030:
0040: 0300           ORG    $0300
0050: 0300 20 A0 03  JSR    $03A0
0060: 0303 A0 00     LDYIM  $00
0070: 0305 84 00     STY    $00
0080: 0307 A9 20     LDAIM  $20
0090: 0309 99 00 02  STAY   $0200
0100: 030C C8        INY
0110: 030D D0 FA     BNE    $0309
0120: 030F 20 3C E9  JSR    $E93C
0130: 0312 C9 0D     CMPIM  $0D
0140: 0314 F0 1C     BEQ    $0332
0150: 0316 C9 5B     CMPIM  $5B
0160: 0318 F0 E6     BEQ    $0300
0170: 031A C9 7F     CMPIM  $7F
0180: 031C D0 06     BNE    $0324
0190: 031E A9 20     LDAIM  $20
0200: 0320 88        DEY
0210: 0321 20 85 03  JSR    $0385
0220: 0324 99 00 02  STAY   $0200
0230: 0327 B0 E6     BCS    $030F
0240: 0329 20 72 03  JSR    $0372
0250: 032C 20 60 03  JSR    $0360
0260: 032F C8        INY
0270: 0330 D0 DD     BNE    $030F
0280: 0332 A0 00     LDYIM  $00
0290: 0334 B9 00 02  LDAY   $0200
0300: 0337 C9 20     CMPIM  $20
0310: 0339 D0 08     BNE    $0343
0320: 033B A5 00     LDA    $00
0330: 033D D0 1B     BNE    $035A
0340: 033F E6 00     INC    $00
0350: 0341 D0 07     BNE    $034A
0360: 0343 A9 00     LDAIM  $00
0370: 0345 85 00     STA    $00
0380: 0347 B9 00 02  LDAY   $0200
0390: 034A 20 72 03  JSR    $0372
0400: 034D 20 60 03  JSR    $0360
0410: 0350 A9 FF     LDAIM  $FF
0420: 0352 8D 97 A4  STA    $A497
0430: 0355 2C 97 A4  BIT    $A497
0440: 0358 10 FB     BPL    $0355
0450: 035A C8        INY
0460: 035B 18        CLC
0470: 035C 90 D6     BCC    $0334
0480:
0490:                    * DISPLAY SUBROUTINE
0500:
0501: 0360           ORG    $0360
0510: 0360 A2 13     LDXIM  $13
0520: 0362 8A        TXA
0530: 0363 48        PHA
0540: 0364 BD 38 A4  LDAX   $A438
0550: 0367 09 80     ORAIM  $80
0560: 0369 20 7B EF  JSR    $EF7B
0570: 036C 68        PLA
0580: 036D AA        TAX
0590: 036E CA        DEX
0600: 036F 10 F1     BPL    $0362
0610: 0371 60        RTS
0620:
0630:                    * MODIFY SUBROUTINE
0640:
0650: 0372 8D 4C A4  STA    $A44C
0660: 0375 A2 01     LDXIM  $01
0670: 0377 BD 38 A4  LDAX   $A438
0680: 037A CA        DEX
0690: 037B 9D 38 A4  STAX   $A438
0700: 037E E8        INX
0710: 037F E8        INX
0720: 0380 E0 55     CPXIM  $15
0730: 0382 90 F3     BCC    $0377
0740: 0384 60        RTS
0750:
0760:                    * BACKSPACE SUBR
0770:
0780: 0385 A2 12     LDXIM  $12
0790: 0387 BD 38 A4  LDAX   $A438
0800: 038A E8        INX
0810: 038B 9D 38 A4  STAX   $A438
0820: 038E CA        DEX
0830: 038F CA        DEX
0840: 0390 10 F5     BPL    $0387
0850: 0392 98        TYA
0860: 0393 E9 14     SBCIM  $14
0870: 0395 AA        TAX
0880: 0396 BD 00 02  LDAX   $0200
0890: 0399 8D 38 A4  STA    $A438
0900: 039C 20 60 03  JSR    $0360
0910: 039F 60        RTS
0920:
0930:                    * CLEAR SUBROUTINE
0940:
0950: 03A0 A2 13     LDXIM  $13
0960: 03A2 A9 20     LDAIM  $20
0970: 03A4 9D 38 A4  STAX   $A438
0980: 03A7 CA        DEX
0990: 03A8 10 FA     BPL    $03A4
1000: 03AA 60        RTS
ID=
```

serted ($FF is loated into the divide-by-1024 counter on the 6532 chip) unless more than one space occurs in succession. In that case, the subroutines and the time delay are not used at all, and the program keeps searching through the message buffer until it finds another non-space ASCII character, in which case subroutines MODIFY and DISPLAY are called again.

One subroutine that remains to be mentioned is BACKSPACE used by the DEL key. It starts at $0385 and its effect is to backspace the display buffer, replacing the leftmost character with the appropriate character from the message buffer. It then calls subroutine DISPLAY to show the typist that the character has, in fact, been deleted and the entire message has been backspaced.

Again, I think the subroutines MODIFY, DISPLAY, CLEAR, READ, and OUTDD1 will be of considerable use if you are writing programs that use the keyboard or the display on the AIM 65. All of them are quite short, and a little study will show how they work. Most involve only simple loops and nothing more complicated than indexed addressing. Mimic or echo your display on your computer storefront and you will have something that will really catch the eye, but don't ask me where to get the appropriate neon sign elements.

A summary of the subroutines follows:

DISPLAY	Takes the contents of locations $AS438 to $A44B and transfers them to the AIM 65 display. A is modified, and X = 0 on return.
MODIFY	Successively shifts the contents of locations $A439 to $A44C to locations in memory whose addresses are one less. The contents of the accumulator, when the subroutine is called, will be stored in location $A438. A and X are modified.
CLEAR	Loads $20 in the display buffer, locations $A438 to $A44B. A and X are modified.
BACKSPACE	Reverses the effects found in MODIFY and, in addition, loads location $A438 with the contents of the message buffer in $0200 + (Y − $13). Y points to the last entry made in the message buffer. X and A are modified.

club notes

The MicroComputer Investor's Association is a non-profit, professional organization which was founded three years ago to enable members to share data and information. For an information packet, send $1.00 to:
Jack Williams, MCIA,
902 Anderson Drive
Fredericksburg, VA 22401

The New England APPLE Tree (NEAT) is a group of APPLE owners and users who have come together to learn more about their APPLEs, trade programs and information and enjoy the world of personal computers as much as possible. NEAT needs your programs, your writing skills, and your participation..
Mitch Kapor
31 Birch Road
Watertown, MA 02172
(617) 926-3809

The NW PET Users' Group is attempting to locate persons in the Oregon/Washington area, interested in a local users' group. If interested, please write or call:
NW PET Users' Group
John F. Jones
2134 NE 45th Avenue
Potland, OR 97213
(503) 281-4908

The Honolulu APPLE Users' Society supports a newsletter containing the latest up to date information concerning the APPLE... program tips and techniques, listings, reviews, etc. The club is interested in exchanging information and software with other clubs. Contact:
Bill Mark
98-1451-A Kaahumanu St.
Aiea, HA 96701
(808) 488-2026

The APPLE II users in the South Florida, Miami area have formed the Miami APPLE Users' Group, president, Steve Pierce. The club was formed to share software and technical information and to help new APPLE users use their APPLES. They plan to establish a quarterly newsletter and anticipate installing an on line system where anyone can have access to club information. If you wish to correspond or join, contact:
David Hall, Sec.
2300 NW 135th St.
Miami, FL 33167

Einleitung: Die vorliegende Nr. 0 ist als Informationsbroschure uber den Verein gedacht. Sie stimmt in wesentlichen Teilen mit dem Entwurf uberein. Zweck dieser Zeitschrift ist es, als Informationsforum fur APPLE - Benutzer zu dienen. Der Name Apple-Com-Post wurde in abgewandelter Form von der Zeitschrift COMPPOST des Rechenzentrums der Ruhn-Universitat Bochum ubernommen. Das Kurzel "Com" soll andeuten, dass es sich beim APPLE um einen Computer handelt. Wunche sowie druckbare Artikel sind sehr erwunscht.
APPLE User Group Europe
Postfach 4068
D-4320 Hattingen
West Germany

The NYC Users' Group in New York City is still alive. Their new address is:
The Drysdale Security
55 Water St.
New York, NY 10004
(516) 579-4295

Washington Area KIM Enthusiasts (WAKE) meet each month at the McGraw-Hill Continuing Education Center in Washington, D.C. to study operation, expansion, and applications of KIM-1 microcomputers. For a copy of the current WAKE newsletter, send a stamped, self-addressed envelope to:
WAKE
c/o Ted Beach
5112 Williamsburg Blvd.
Arlington, VA 22207

Applesoft Renumbering

Here is a fast and reliable utility for APPLE programmers who do not have disks. It can be adapted to the PET and other Microsoft BASIC systems.

J.D. Childress
5108 Springlake Way
Baltimore, MD 21212

The need for a program written in Applesoft to renumber Applesoft programs is moot now that APPLE has made available the 3.2 version of its disk operating system, that is, if one has a disk system. I wrote the present renumbering program while my disk drive was out of action, before the release of the 3.2 version, and after reading Mr. Carpenter's program in MICRO 12:45 based in turn on a PET program by Jim Butterfield, MICRO 8:33. Since some people do not have disks and since Applesoft programs can be adapted to the PET and other systems using Microsoft BASIC, my renumbering program still may find users.

Comparison

This Applesoft renumbering program (hereafter called RENUMB) is dreadfully slow; it took 7.9 minutes to renumber a 8.5K program. Even at that, it's faster than Mr. Carpenter's program, which took 13.2 minutes to renumber the same 8.5K program (and also had a problem with one THEN). In comparison, the 3.2 disk renumber program did the job in 7.8 seconds.

Like Mr. Carpenter's program, RENUMB cannot change the line number after a GOTO, a GOSUB, or a THEN equivalent of a GOTO when the new line number has more digits than the old one. The program prints a list of these changes which must be made by hand. If there is not enough space, RENUMB inserts only the least significant digits. For example, the line

```
100   ON L GOTO 180, 190
```

with a line number shift upwards by 1005 would be given as

```
1105   ON L GOTO 185, 195
```

With the manual change instructions shown here:

LINE 1105: INSERT 1185 AFTER GOTO.

LINE 1105: INSERT 1195 AFTER COMMA.

If there is more space than needed, RENUMB inserts leading zeros. (Note that the Applesoft interpreter preserves such leading zeros whereas the 3.2 disk renumber program does not.)

RENUMB has one useful feature in common with the 3.2 disk renumber program, namely the capability of renumbering only a specified portion of a program. This feature must be used with care since one can renumber a part of a program with line numbers equal to or in between some of the line numbers of the remaining part of the program.

Unlike the 3.2 disk program, RE-NUMB does not order such lines into the proper sequence. If you really want that, you must run RENUMB first then use the screen/cursor editing controls to copy the out-of-sequence lines through the Applesoft interpreter. The reader is left with the nontrivial problem of getting rid of the still remaining out-of-sequence lines.

Operation

To use RENUMB, one needs to append RENUMB to the program to be renumbered. The machine language APPEND program and procedure given by Mr. Carpenter are recommended. After the two programs are properly loaded, renumbering is accomplished by a RUN 63000 command. Give the requested information, then be patient; remember that RENUMB is numbingly slow.

Copy carefully all the manual changes listed. If you want to see them again, you can do so by a GOTO 63360 command provided you have done nothing to clear the variables, i.e., have not given any RUN commands or changed any line of the program.

You may use the SPEED command to slow up the display and the CTRL-C command to interrupt the display without clearing the variables. Once the variables have been cleared, there is nothing you can do except start from the beginning, that is, load the programs again.

At the beginning of the program run, you are asked for a rough estimate of the number of program lines (numbered lines) to be renumbered. Be generous, within limits of available memory. If your estimate is too small, you will get a

?BAD SUBSCRIPT ERROR IN 630X0

where X = 6, 7, or 8 since your estimate is used for array dimensioning. Unless your program is especially rich in branches, an estimate, say, about 50% greater than the number of line numbers will suffice.

Program Design

The design of RENUMB is quite simple. First RENUMB searches the program being renumbered for line numbers (and their memory locations) and the line numbers (and memory locations) after GOTO's, GOSUB's, THEN's, and COMMA's in multiple branches. This search is done by lines 63040-63090 and for branches, the subroutine at 63250. Lines 63130 and 63140 make the changes at the branches and line 63180 at the labels. The routine beginning at 63350 prints out those changes that must be made by hand.

All else is bookkeeping. Note: In line 63030, START is the address in memory of the beginning of the program. This is probably the only thing that needs to be changed for RENUMB to run on the PET (try START@1025 per Butterfield) and possibly on other systems using Microsoft BASIC. Finally, if you write very GOTOy and GOSUBy programs, you may want to change the definition of DD in line 63030.

Applesoft

Butterfield gives considerable information about the insight into the structure of Microsoft BASIC. What is even handier is your own APPLE II. Let it be your textbook and teacher. For example, starting fresh with Applesoft in the computer, enter

```
1 PRINT: GOTO 521
521 PRINT "FREE": LIST 521
```

While this little program runs without error, that is not necessary. You can enter anything you want to see how Applesoft handles it.

Now go to the monitor and look at

```
801- 0C 08 01 00 BA 3A
     AB 35 32 31 00

80C- 10 08 09 02 BA 22 46 52 45 45 22
     3A
     BC 35 32 31 00

810- 00 00
```

for ROM Applesoft (1001 for RAM Applesoft). In the above lines, arranged here for clarify, 0C, 08, 10 08, and the final 00 00 point to the next instruction in memory, the 00 00 pointer labelling the end of the program. 01 00 and 09 02 are the line numbers, 1 and 521 respectively. BA is the token for PRINT; 3A is the ASCII code for the colon; AB is the token for GOTO; 35 32 31 gives the line number for the GOTO; and 00 indicates the line ending. 22 46 52 45 45 22 is a direct ASCII code rendition of "FREE". Finally BC is the token for LIST and 35 32 31 is the line number 521 after LIST.

Study of the above paragraph shows that Applesoft puts things into memory almost exactly the way you type them on the keyboard, except that the interpreter removes spaces, puts in instruction addresses, translates its command words into tokens, and uses ASCII code and hexidecimal, low-order bit first notation.

I think we can be confident that Microsoft has written most of their BASIC interpreters in as similar a fashion as possible. After all, why not exploit one's own good work. μ

```
LISTING--APPLESOFT RENUMBERING
        PROGRAM


62999  END
63000  HOME : VTAB (3): PRINT "
     RENUMBERING PROGRAM": PRINT

63010  PRINT "LINES TO BE RENUMBE
     RED:": INPUT "  BEGINING LI
     NE--";BGN: INPUT "  ENDING
     LINE--";TRM: INPUT "  TOTAL
      NUMBER OF LINES (ROUGHLY)--
     ";D: PRINT
63020  INPUT "RENUMBERED BEGINNIN
     G LINE--";SK: INPUT "INCREME
     NT--";ADD
63030 START = 256 *  PEEK (104) +
      PEEK (103):M = START + 2:DD
     =  INT (D / 4): DIM LS(D),L
     N(DD),LM(DD),LOC(DD),NA$(DD)
     ,ND(DD),INS(DD),IMS(DD)
63040 L = L + 1:LS(L) = M:LC = 25
     6 *  PEEK (M + 1) +  PEEK (M
     ): IF LC > 62900 THEN 63100
63050  FOR J = M + 2 TO M + 255:T
     ST =  PEEK (J): IF TST = 0 THEN
     M = J + 3: GOTO 63040
63060  IF TST = 171 THEN NA$(K +
     1) = "GOTO": GOSUB 63250
63070  IF TST = 176 THEN NA$(K +
     1) = "GOSUB": GOSUB 63250
63080  IF TST = 196 AND  PEEK (J +
     1) > 47 AND  PEEK (J + 1) <
     58 THEN NA$(K + 1) = "THEN":
      GOSUB 63250
63090  NEXT
63100  FOR J = 1 TO L:LNU = 256 *
      PEEK (LS(J) + 1) +  PEEK (L
     S(J)): IF LNU > TRM OR LNU >
     62900 THEN  PRINT : PRINT "R
     ENUMBERING COMPLETED THROUGH
      LINE ";LUN;".": GOTO 63350
63110  IF LNU < BGN THEN 63190
63120 SK$ = "0000" +  STR$ (SK):S
     K$ =  RIGHT$ (SK$,5)
63130  FOR I = 1 TO K: IF LNU <  >
     INS(I) THEN  NEXT : GOTO 631
     80
63140  FOR KA = 1 TO ND(I): POKE
     LOC(I) + 1 + ND(I) - KA, VAL
     ( MID$ (SK$,6 - KA,1)) + 48:
      NEXT
63150  IF LNU = INS(I) THEN IMS(I
     ) = SK
63160  IF  LEN ( STR$ (SK)) > ND(
     I) THEN PCR = 1
63170  NEXT
63180 S0 =  INT (SK / 256): POKE
     (LS(J) + 1),S0: POKE (LS(J))
     ,SK - 256 * S0
63190  FOR I = 1 TO K: IF LNU = L
     N(I) THEN LM(I) = SK: IF LNU
      < BGN THEN LM(I) = LNU
63200  NEXT
63210 SK = SK + ADD:LUN = LNU
63220  NEXT
63250 K = K + 1:LN(K) = LC:SU =  PEEK
     (J + 1) - 48
63260  FOR KA = J + 2 TO J + 6:CP
     R =  PEEK (KA): IF CPR = 0 OR
     CPR = 58 OR CPR = 44 THEN  GOTO
     63290
63270 SU = 10 * SU + CPR - 48
63280  NEXT
63290 LOC(K) = J:ND(K) = KA - 1 -
     J:INS(K) = SU:J = KA - 1: IF
     CPR = 44 THEN NA$(K + 1) = "
     COMMA":J = KA: GOTO 63250
63300  RETURN
63310  END
63350  IF PCR <  > 1 THEN  END
63360  PRINT : PRINT "NOTE: YOU M
     UST MAKE THE FOLLOWING CHAN-
     ": PRINT "GES MANUALLY:": PRINT

63370  FOR I = 1 TO K: IF  LEN ( STR$
     (IMS(I))) <  = ND(I) THEN  NEXT
     : END
63380  PRINT "LINE ";LM(I);": INS
     ERT ";IMS(I);" AFTER ";NA$(I
     );"."
63390  NEXT : END
```

MOVE IT: Relocating PET Source Programs and Object Code

A useful program need not perform the entire task. If ten percent of the total coding effort achieves ninety-nine percent of the desired result, perhaps manual intervention will be more efficient than additional programming.

Professor Harvey B. Herman
Department of Chemistry
The University of North Carolina
Greensboro, NC 27412

MICRO readers probably know that when a PET program is saved on cassette tape it normally loads back into the same area of memory. Several times recently I wished that was not the case because I found the need to relocate information already saved.

For example, I originally assigned source code for an assembler to what later turned out to be an *inconvenient* area of memory. Being naturally lazy, I had no desire to retype the long source program into the newly assigned memory region. Let the PET move it, I said—and it did. This article tells how.

Information one might wish to relocate falls into three categories. I have already mentioned ASCII source code which would require no modifications after being moved. The next category also requires no extra work. BASIC programs which you might want to append, relocate, or relocate, do indeed have address links which need to be modified. Fortunately for us the PET has routines which do this automatically.

Finally, machine language programs are not always located where they do the most good and it could become necessary to move them to more useful areas. In this case many changes probably will be necessary. Instructions which use absolute addressing modes and indexed pointers are the principle culprits. Finding the necessary changes can be difficult without a source listing of the original program.

The first method I considered to move programs (the first step in relocation) was a modified program from the *First Book of KIM* (MOVEIT, p. 127). I rejected this approach for a number of reasons, but mainly because I was convinced the PET had the routines already built in. An article by Jim Butterfield (bless his bones) in the *PET User's Notes* (Vol. 2, #1, p. 7) gave me the concept I needed to have the PET operating system help relocate code already saved on tape.

This method has the decided advantage of not requiring the old memory locations to be present. A program originally located at hex 6000 and saved on tape in another PET can be loaded into an 8K machine at hex 400 if desired.

After placing the machine language program in a new area of memory, it is necessary to make various address changes. These modifications can be made with a machine language program. See *The First Book of KIM*, p. 130, for an example. Since I feel more comfortable working in BASIC, I develcped a simple BASIC program to do most of the address modifications.

The program is not perfect, so any remaining changes or corrections need to be done with a monitor program. I deliberately used an easy-to-write (slightly flawed) program in combination with manual correction, instead of spending lots of time writing an elegant program which did everything. I felt this approach gave the best results because the total time to accomplish a task is what really counts.

In summary, the relocating method discussed here can be broken down into three essential steps:

1. Loading the information on the cassette tape into the new area of memory.
2. Running a BASIC program which makes most of the address changes.
3. Manually correcting errors, using a monitor program, and making other necessary changes missed by the simple minded BASIC program.

As an example, I have picked what I hope is a useful exercise: relocating Commodore's machine language monitor. It is important to have available monitors which are located in different areas of memory. When we want to modify low areas of memory, it is necessary to use a monitor in high memory, and vice versa.

Furthermore, the top of memory is consistently changing as PET owners add extra memory. It is a decided disadvantage to be stuck with only a low monitor, as supplied by Commodore. The latest PET's have a monitor in ROM,

```
1EA8 A9 23 85 F9 20 5E 1E 29
1EB0 20 CF FF C9 2C F0 55 C9
1EC0 0D F0 0B E0 10 F0 F1 95
1EC8 23 E6 EE E8 D0 EA A5 20
1ED0 C9 06 D0 C8 A2 00 8E 0B
1ED8 02 A5 F1 D0 03 4C 9B 1C
1EE0 C9 03 B0 F9 20 67 F6 20
1EE8 3B F8 20 FF F3 A5 EE F0
1EF0 08 20 95 F4 D0 08 4C 9B
1EF8 1C 20 AE F5 F0 F8 20 4D

1D00 0A 60 3A 3B 52 4D 47 58
1D08 4C 53 1D 1D 1D 1D 1D 1D
1D10 1E 1E C1 B1 2C 5E D7 FD
1D18 9E 9E 20 50 43 20 20 53
1D20 52 20 41 43 20 58 52 20
1D28 59 52 20 53 50 A5 0D D0
1D30 06 20 F2 1C 20 37 1E 20
1D38 37 1E A2 00 BD 1A 1D 20
1D40 D2 FF E8 E0 13 D0 F5 20
1D48 F2 1C A2 2E A9 3B 20 22
1D50 1E 20 37 1E 20 08 1E 20
1D58 E7 1C 20 BB 1C F0 4D 20
1D60 90 1E 20 4F 1E 90 48 20
1D68 3F 1E 20 90 1E 20 4F 1E
1D70 90 3D 20 3F 1E A0 00 B9
1D78 4A 1E 30 06 20 D2 FF C8
1D80 D0 F5 29 7F 20 D2 FF 20
1D88 2A F3 F0 20 A6 0A D0 1C
1D90 20 A3 1C 90 17 20 F2 1C
1D98 A2 2E A9 3A 20 22 1E 20
1DA0 37 1E 20 04 1E A9 08 20
1DA8 BB 1C F0 DB 4C 57 1C 4C
1DB0 9B 1C 20 5E 1E 20 4F 1E
1DB8 90 03 20 B2 1C 20 E7 1C
1DC0 D0 0A 20 5E 1E 20 4F 1E
1DC8 90 E5 A9 08 85 21 20 90
1DD0 1E 20 CF 1C D0 F8 F0 D4
1DD8 20 CF FF C9 0D F0 0C C9
1DE0 20 D0 CC 20 4F 1E 90 03
1DE8 20 B2 1C A6 1F 9A A5 1A
1DF0 48 A5 19 48 A5 1B 48 A5
1DF8 1C A6 1D A4 1E 40 A6 1F

1F00 F6 20 22 F4 20 8A F8 20
1F08 13 F9 AD 0C 02 29 10 D0
1F10 E5 4C 57 1C 20 4F 1E A5
1F18 11 85 F7 A5 12 85 F8 20
1F20 CF FF C9 20 F0 F9 C9 0D
1F28 F0 A4 C9 2C F0 03 4C 9C
1F30 1E 20 4F 1E A5 11 85 E5
1F38 A5 12 85 E6 A5 20 C9 06 *
1F40 F0 92 A2 00 20 B1 F6 4C
1F48 57 1C 0D 20 20 20 20 20
1F50 20 20 20 20 20 30 20 20
1F58 31 20 20 32 20 20 33 20
1F60 20 34 20 20 35 20 20 36
1F68 20 20 B7
```

```
1C00 00 0D 04 0A 00 9E 28 31
1C08 30 33 39 29 00 00 00 A9
1C10 27 8D 1B 0A A9 1C 8D 1C
1C18 02 A9 1E 85 7D A9 6B 85
1C20 7C A9 43 85 21 D0 12 A9
1C28 42 85 21 D8 4A 68 85 1E
1C30 68 85 1D 68 85 1C 68 85
1C38 1B 68 69 FF 85 19 68 69
1C40 FF 85 1A BA 86 1F 58 20
1C48 F2 1C A6 21 A9 2A 20 22
1C50 1E A9 52 85 0D D0 2B A9
1C58 00 85 CA 85 0D 85 0A 20
1C60 F2 1C A9 2E 20 D2 FF A6
1C68 20 E0 02 F0 04 E0 03 D0
1C70 06 20 3A 1E 20 37 1E 20
1C78 90 1E C9 2E F0 F9 C9 20
1C80 F0 F5 A2 07 DD 02 1D D0
1C88 0F A5 20 85 0E 86 20 BD
1C90 0A 1D 48 BD 12 1D 48 60
1C98 CA 10 E9 A9 3F 20 D2 FF
1CA0 4C 57 1C 38 A5 13 E5 11
1CA8 85 0B A5 14 E5 12 A8 05
1CB0 0B 60 A5 11 85 19 A5 12
1CB8 85 1A 60 85 21 A0 00 20
1CC0 3A 1E B1 11 20 13 1E 20
1CC8 F7 1C C6 21 D0 F1 60 20
1CD0 5E 1E 90 0D A2 00 81 11
1CD8 C1 11 F0 05 68 68 4C 9B
1CE0 1C 20 F7 1C C6 21 60 A9
1CE8 1B 85 11 A9 00 85 12 A9
1CF0 05 60 A9 0D 4C D2 FF E6
1CF8 11 D0 06 E6 12 D0 02 E6
```

Location		Object Code		Source Code and Notes
Original	Modified	Original	Modified	
0447	1C47	20F204	20F21C	JSR CRLF (1)
0484	1C84	DD0205	DD021D	CMP CMDS, X (2)
0414	1C14	A904	A91C	LDA # BRKE (3)
0511	1D11	06	1E	.BYT ZZ8 (4)
073E	1F3E	C906	C912	CMP #6 (5)

NOTES

(1) Absolute address identified and changed by BASIC MODIFY program.

(2) Address not changed by BASIC program. Changed manually with the monitor.

(3) High order byte of address of the break vector stored at $21C must be changed.

(4) Jump table value (address of command) has to be relocated.

(5) Code which was erroneously changed because of preceeding hex 20. Changed back manually with the monitor.

Typical Changes While Relocating Monitor Program

```
5 REM MODIFY PROGRAM (FLAWED)
6 REM 1K LOCATIONS SEARCHED
7 REM HARVEY B. HERMAN
8 N=0
10 FOR I=0 TO 1023
15 REM DEC 7168 IS HEX 1C00
20 L=7168+I
30 A=PEEK(L):B=PEEK(L+2)
35 REM DEC 32 & 76 ARE HEX 20 (JSR) & 4C (JMP)
36 REM PGS. DEC/HEX 4,5,6 & 7 SEARCHED
40 IF A=32 AND B=4 THEN GOSUB 1000
50 IF A=32 AND B=5 THEN GOSUB 1000
60 IF A=32 AND B=6 THEN GOSUB 1000
70 IF A=32 AND B=7 THEN GOSUB 1000
80 IF A=76 AND B=4 THEN GOSUB 1000
90 IF A=76 AND B=5 THEN GOSUB 1000
100 IF A=76 AND B=6 THEN GOSUB 1000
110 IF A=76 AND B=7 THEN GOSUB 1000
120 NEXT I
125 PRINT "LOCATIONS MODIFIED =";N
130 STOP
999 REM MODIFIED TO DEC 24/HEX 18 HIGHER
1000 POKE L+2,B+24
1005 N=N+1
1010 RETURN
```

but the general ideas presented here will still be useful to owners of those machines in other applications.

The article by Jim Butterfield showed a procedure which loaded the tape into the screen memory area. I wanted to move the monitor program to the top of memory, 8K in my PET. This procedure is shown in Figure 1. Step 2 loads the tape header. I used the monitor program, in low memory, to modify the tape address from the header in steps 4 and 5. Moving the program on tape to the new area of memory occurs in step 7. After protecting the program, step 8, it is necessary to make address modifications before the program can be run successfully.

Most of the address modifications can be made with the BASIC program. The program looks for JSR (hex 20/dec 32) and JMP (hex 4C/dec 76) values in the new memory locations. The majority of changes necessary were in those two instructions alone. When the program finds dec 32 or dec 76 followed by a location in pages 4 through 7 (where the original program was located), it modifies the page number to the relocated values, 28 through 31 respectively. This program is quite a bit slower than a machine language version, but it certainly runs faster than I could type in the changes.

Since the BASIC program has flaws, it is important to check for errors. The relocated monitor program contained two unnecessary changes which were easy to find and change back. I manually corrected these errors using a low monitor and looked for other locations which needed to be changed. All instructions besides JSR and JMP that have an absolute addressing mode referring to relocated addresses must be modified. Much harder to find are table values and page zero references. A source listing or disassembler output listing is essential.

I had the advantage of having the source code for the monitor (PET User Manual, p. 100) and changes were easier to identify. However, I have successfully relocated code with just a dissassembled listing and no comments or mnemonic variable names. A few examples of what to look for are shown in Figure 2.

The trickiest part of relocating involves indirect instructions. The instruction itself does not have to be changed, but the numbers stored as page 0 pointers may have to be. Somewhere in the code, there may be a combination like LDA $05 / STA $35, which would have to be changed to LDA #1D / STA

```
1E00 9A 4C 8B C3 A2 01 D0 02
1E08 A2 09 B5 10 48 B5 11 20
1E10 13 1E 68 48 4A 4A 4A 4A
1E18 20 2B 1E AA 68 29 0F 20
1E20 2B 1E 48 8A 20 D2 FF 68
1E28 4C D2 FF 18 69 06 69 F0
1E30 90 02 69 06 69 3A 60 20
1E38 3A 1E A9 20 4C D2 FF A2
1E40 02 B5 10 48 B5 12 95 10
1E48 68 95 12 CA D0 F3 60 20
1E50 5E 1E 90 02 85 12 20 5E
1E58 1E 90 02 85 11 60 A9 00
1E60 85 0F 20 90 1E C9 20 D0
1E68 09 20 90 1E C9 20 D0 0E
1E70 18 60 20 85 1E 0A 0A 0A
1E78 0A 85 0F 20 90 1E 20 85
1E80 1E 05 0F 38 60 C9 3A 08
1E88 29 0F 28 90 02 69 08 60
1E90 20 CF FF C9 0D D0 F8 68
1E98 68 4C 57 1C 4C 9B 1C 20
1EA0 90 1E A9 00 85 EE 85 FA
```

$35. The monitor program did not contain examples of these instructions.

A hex listing of the relocated monitor is shown in Figure 3. All functions have been checked and appear to be working.

The BASIC modify program changed 72 locations (2 of which were in error). I have underlined the correct changes and put an asterisk beside the corrected errors. Fourteen locations needed to be changed manually, and these have been boxed. By my count, more than 3/4 of the changes were made by the BASIC program. I was satisfied, but others may wish to write a more comprehensive utility.

Once properly moved and relocated, the monitor can be run (SYS 7183) and saved on tape (S 01, 1C00, 1F6B). The break vector is set automatically on entering the program. After the first run, the program can be restarted with SYS 1024 which is easier to remember. That trick takes advantage of the zero (BRK) first byte in every BASIC program.

Moving and relocating programs can be fun as well as useful. In some respects it's like a game or puzzle. I believe this is the aspect that appeals to me. I would enjoy hearing from other PETters about their success or failures in relocating programs (SASE for reply).

Life in the Fast Lane

Richard R. Auricchio
1596 Stapleton Court
San Jose, CA 95118

This high speed version of the game of LIFE uses lo-res graphics on the APPLE II. A clean assembly language implementation makes it easy to enhance or adapt.

What? Yet another game of LIFE? Yes, this one's for the APPLE II computer, and it's a fairly quick one. The game runs in Lo-Res graphics using a 32 x 32 array. The current version is black-and-white, but adding color should not prove too difficult a task. The assembly language module which actually computes the generations is capable of running off about three per second on the APPLE screen.

The program is designed to utilize both of the APPLE's graphic screen buffers to avoid the ripple effect which occurs when the display is updated. When both buffers are used, the new image is created in the buffer not currently being displayed on the screen; after the complete image is created, the buffers are swapped via the hardware controls in the APPLE.

In addition to the two screen buffer areas, the actual LIFE generations are performed in a 32 x 32 array (1K bytes). Separating the screen and LIFE array makes it easier to interface with the LIFE program from BASIC; in addition, a speed increase was realized because it was not necessary to "read" the screen points to compute the next generation. The code to perform an assembly language SCRN(X, Y) function, although short, requires computation of screen coordinates. This computation would cost valuable compute-time.

Program Organization

There are two entry points to the LIFE program. One, which performs initialization, is used to clear the LIFE array and set up the screen buffer pointers. The second screen buffer is then blanked out by copying the first into the second.

The second entry is the "run" entry to perform LIFE generations. This is the main part of the LIFE program. It runs until either the screen becomes completely blank, or the user hits any key on the APPLE keyboard. The program will not detect a stable LIFE pattern. It will keep running more generations even though the display does not appear to change.

The LIFE program makes two passes over the LIFE array to compute each generation. The first pass sets up pending births and deaths within the array. This is done by accessing cells (neighbors) which border the current cell being examined. The array is allowed to wrap around and going off one edge brings you back in on the other side. The second pass completes the birth/death process, displays the cells in the inactive screen buffer, and swaps the screens. This process continues until all cells die or a key is hit. In either event, a return is made to the BASIC program; the screen and LIFE array are not altered. This allows the BASIC program to actually edit the LIFE array, say, to add/delete cells or to center the image on the screen.

Driving the Program from BASIC

A simple Integer BASIC driver is included here: It allows one to type in points until (0.0) is entered, and then calls the LIFE program to display generations on the screen. No fancy editing facilities have been coded, but they're easy enough to add if you find them useful.

Structuring the Code

The LIFE program was coded using straightforward techniques. No tricks or shortcuts were used to save a byte here, a microsecond there. Comments have been sprinkled throughout the listing to enable changes or customization of the module, and coding tricks might have made that next to impossible.

Use with APPLE DOS

The LIFE program is completely compatible with APPLE DOS (both versions 3.1 and 3.2). There are no memory areas used which will conflict with DOS usage, and no DOS features are affected by running LIFE. Users with DOS systems should BSAVE the LIFE module and insert an appropriate BLOAD command at the beginning of the BASIC driver program.

```
************************
*                      *
* THE GAME OF LIFE FOR *
*  THE APPLE-II, ON A  *
*  32 X 32 ARRAY.      *
*                      *
* BY RICK AURICCHIO    *
*  10-30-78            *
* MODIFIED BY MICRO    *
*  STAFF 7-17-79       *
*                      *
************************

APTR   *     $0000  LIFE ARRAY POINTER (LO)
NPTR   *     $0002  NEIGHBOR CELL POINTER (LO)
NNUM   *     $0004  NUMBER NEIGHBOR CHECKS
NCNT   *     $0005  NEIGHBOR COUNT
NCELLS *     $0006  NUMBER LIVE CELLS
CRT    *     $0007  CRT OFFSET: 00=1ST, 04=2ND
COLOR  *     $0030  PLOT COLOR
GBASH  *     $0027  GRAPHIC BASE ADDRESS (HI)
A1L    *     $003C  MONITOR WORK BYTES
A1H    *     $003D
A2L    *     $003E
A2H    *     $003F
A4L    *     $0042
A4H    *     $0043

ASTART *     $000C  START PAGE FOR ARRAY
AEND   *     $0010  END PAGE FOR ARRAY

KB     *     $C000  KEYBOARD INPUT ADDRESS
KBS    *     $C010  KEYBOARD STROBE CLEAR
CRTFLI *     $C054  C054/C055 FLIPS CRT
GBASCA *     $F847  CALCULATE PLOT ADDRESS
MOVE   *     $F32C  BLOCK MOVE ROUTINE

* MEMORY LAYOUT:
*  PAGE(S)    CONTENT
*  04-07       CRT#1
*  08-0B       CRT#2
*  0C-0F       LIFE ARRAY
*  10-11       PROGRAM
*
       ORG   $1000
*
* CALL TO INIT WILL CLEAR THE LIFE ARRAY
*  AND SET UP THE APPROPRIATE CRT POINTER
                 *
1000 20 DA 10  INIT   JSR    ORIGIN SET ARRAY POINTER
1003 A9 04            LDAIM  $04    SET CRT TO SECOND
1005 85 07            STA    CRT    FOR THE FIRST GENERATION
1007 A0 00            LDYIM  $00    ZERO INDEX
1009 98        CLRA   TYA           ZERO THE AC
100A 91 00            STAIY  APTR   CLEAR ARRAY BYTE
100C 20 E5 10         JSR    BUMP   BUMP TO NEXT BYTE
100F 90 F8            BCC    CLRA   =>MORE TO DO
                 *
                 * CLEAR SECOND CRT BUFFER
                 *
1011 A2 00            LDXIM  $00
1013 86 3C            STX    A1L    SET UP ADDRESSES
1015 86 42            STX    A4L    TO COPY CRT DATA:
1017 CA               DEX           0400-07FF ==> 0800-0BFF
1018 86 3E            STX    A2L
101A A2 04            LDXIM  $04
101C 86 3D            STX    A1H
101E A2 08            LDXIM  $08
1020 86 43            STX    A4H
1022 CA               DEX
1023 86 3F            STX    A2H
1025 20 2C F3         JSR    MOVE   BLOCK MOVE
1028 60               RTS           DONE. BACK TO BASIC(S)
```

```
                  *
                  * ENTRY AT RUN WILL DO THE
                  *  PROCESSING UNTIL EITHER:
                  *    1) ALL CELLS DIE, OR
                  *    2) ANY KEY IS HIT.
                  *  IT WILL THEN RETURN
                  *
1029 20 DA 10     RUN    JSR    ORIGIN SET ARRAY POINTER
                  *
                  * PASS1 WILL SCAN THE ARRAY
                  *  AND SET UP BIRTHS/DEATHS
                  *  FOR ALL CELLS
                  *
102C A9 07        PASS1  LDAIM  $07    SET TO CHECK OUT
102E 85 04               STA    NNUM   SEVEN NEIGHBORS
1030 A9 00               LDAIM  $00    THERE ARE NO
1032 85 05               STA    NCNT   NEIGHBORS YET
1034 A5 04        NBCHK  LDA    NNUM   GET NEIGHBOR NUMBER
1036 0A                  ASLA          AND MAKE IT A
1037 AA                  TAX           2-BYTE INDEX
                  *
                  * SET NPTR BY ADJUSTING THE CURRENT
                  *  CELL POINTER BY THE FOLLOWING VALUES:
                  *  -33, -32, -31, -1, +1, +31, +32, +33
                  *
1038 18                  CLC
1039 A5 00               LDA    APTR   GET LO HALF
103B 7D 0A 11            ADCX   OFFSET ADD/SUBTRACT
103E 85 02               STA    NPTR   SET NPTR LO
1040 A5 01               LDA    APTR   +01 GET HI HALF
1042 7D 0B 11            ADCX   OFFSET +01 ADD/SUBTRACT
1045 C9 10               CMPIM  AEND   PAST THE END OF ARRAY
1047 B0 08               BCS    SUB04  =>YES: BACK IT UP!
1049 C9 0C               CMPIM  ASTART BELOW START OF ARRAY?
104B B0 06               BCS    STORE  =>NO: WITHIN BOUNDS
104D 69 04               ADCIM  $04    BUMP UP INTO ARRAY
104F D0 02               BNE    STORE  AND GO STUFF NPTR

1051 E9 04        SUB04  SBCIM  $04    BACK UP INTO ARRAY

1053 85 03        STORE  STA    NPTR   +01 NOW SET NPTR HI
                  *
                  * CHECK OUT THIS NEIGHBOR
                  *
1055 A0 00               LDYIM  $00    INDEX = 0
1057 B1 02               LDAIY  NPTR   GET NEIGHBOR
1059 10 02               BPL    NEXTNB =>NONE HERE
105B E6 05               INC    NCNT   =>ONE HERE: COUNT UP
                  *
                  * TRY NEXT NEIGHBOR
                  *
105D C6 04        NEXTNB DECZ   NNUM   MORE TO DO?
105F 10 D3               BPL    NBCHK  =>YES: DO ALL
                  *
                  * ALL NEIGHBORS COUNTED
                  * MAKE LIFE/DEATH DECISION
                  *
1061 A6 05               LDX    NCNT   GET CURRENT COUNT
1063 B1 00               LDAIY  APTR   GET CURRENT CELL
1065 10 0A               BPL    CHKBIR =>EMPTY: MAYBE BIRTH HERE?
1067 E0 02               CPXIM  $02    ALIVE: TWO NEIGHBORS?
1069 90 12               BCC    DIE    => 0 OR 1: DIE!
106B E0 04               CPXIM  $04    4-7 NEIGHBORS?
106D B0 0E               BCS    DIE    =>YUP: DIE OF OVERCROWDING!
106F 90 04               BCC    SURVIV =>2 OR 3: SURVIVE

1071 E0 03        CHKBIR CPXIM  $03    EXACTLY THREE NEIGHBORS?
1073 D0 08               BNE    DIE    =>NO: EMPTY CELL STAYS DEAD

1075 E6 06        SURVIV INC    NCELLS BUMP COUNT OF LIVE CELLS
1077 A9 40               LDAIM  $40    "OR" IN 40 BIT
1079 11 00               ORAIY  APTR   TO SURVIVE
107B 91 00               STAIY  APTR   THIS TIME
107D 20 E5 10     DIE    JSR    BUMP   SET NEXT CELL TO DO
1080 90 AA               BCC    PASS1  =>MORE TO DO
```

```
                    *
                    * PASS2 WILL DISPLAY THE
                    *  ARRAY BY PLOTTING POINTS
                    *  IN CRT #1 OR #2 AND
                    *  WILL THEN SWAP SCREENS
                    *
1082 B1 00          PASS2  LDAIY APTR   GET CURRENT CELL
1084 A2 00                 LDXIM $00    ASSUME DEAD: COLOR = 0
1086 0A                    ASLA         SHIFT ONE BIT LEFT
1087 91 00                 STAIY APTR   AND PUT BACK
1089 F0 02                 BEQ   SETCOL =>NOT ALIVE!
108B A2 FF                 LDXIM $FF    ALIVE: COLOR = 15
108D 86 30          SETCOL STX   COLOR
108F A9 1F                 LDAIM $1F    X CO-ORDINATE IS
1091 25 00                 AND   APTR   LOW 5 BITS
1093 18                    CLC          BUMP DOWN TO
1094 69 04                 ADCIM $04    CENTER OF CRT
1096 A8                    TAY          OF APTR
1097 A5 00                 LDA   APTR   Y CO-ORDINATE IS
1099 4A                    LSRA         HIGH 3 BITS
109A 4A                    LSRA
109B 4A                    LSRA         OF
109C 4A                    LSRA
109D 4A                    LSRA         APTR
109E 85 04                 STA   NNUM   HOLD TEMPORARILY
10A0 A9 03                 LDAIM $03    NOW MERGE IN
10A2 25 01                 AND   APTR   +01 2 LOW BITS OF
10A4 0A                    ASLA         APTR HI
10A5 0A                    ASLA         TO FORM
10A6 0A                    ASLA         FULL
10A7 05 04                 ORA   NNUM   Y CO-ORDINATE
10A9 18                    CLC          BUMP DOWN TO
10AA 69 04                 ADCIM $04    CENTER OF CRT
10AC 20 F9 10              JSR   PLOTX  PLOT THE POINT


10AF A0 00                 LDYIM $00    INDEX
10B1 20 E5 10              JSR   BUMP   BUMP TO NEXT POINT
10B4 90 CC                 BCC   PASS2  =>DO ALL CELLS
                    *
                    * SET HARDWARE TO DISPLAY THE CURRENT
                    * SCREEN AND SWAP OVER TO THE OTHER SIDE
                    *
10B6 A5 07                 LDA   CRT    GET CRT NUMBER
10B8 4A                    LSRA         SCALE DOWN
10B9 4A                    LSRA         TO 0 OR 1 RANGE
10BA AA                    TAX          TO INDEX REG
10BB 9D 54 C0              STAX  CRTFLI FLIP CRT DISPLAY
10BE BD 08 11              LDAX  CRTNUM GET CRT NUMBER
10C1 85 07                 STA   CRT    FOR NEW CRT
                    *
                    * CONTINUE RUNNING UNLESS
                    * ALL DEAD OR KEY HIT
                    *
10C3 AD 00 C0              LDA   KB     CHECK KEYS
10C6 10 07                 BPL   NOKEY  =>NO KEY HIT
10C8 8D 54 C0       RETURN STA   CRTFLI SET CRT #1 ALWAYS
10CB 8D 10 C0              STA   KBS    CLEAR KEYBOARD
10CE 60                    RTS
10CF A5 06          NOKEY  LDA   NCELLS GET COUNT OF CELLS
10D1 F0 F5                 BEQ   RETURN =>ALL DEAD
10D3 A9 00                 LDAIM $00    MORE LEFT
10D5 85 06                 STA   NCELLS CLEAR COUNT AND
10D7 4C 2C 10              JMP   PASS1  GO AROUND AGAIN
                    *
                    * SET UP ARRAY POINTERS
                    *
10DA A9 00          ORIGIN LDAIM $00    SET UP THE
10DC 85 00                 STA   APTR   LO BYTE OF APTR
10DE 85 06                 STA   NCELLS AND CLEAR CELL COUNT
10E0 A9 0C                 LDAIM ASTART SET UP START
10E2 85 01                 STA   APTR   +01 OF ARRAY
10E4 60                    RTS
                    *
                    * BUMP APTR.  RESETS IT IS
                    * WE GO PAST END OF ARRAY.
```

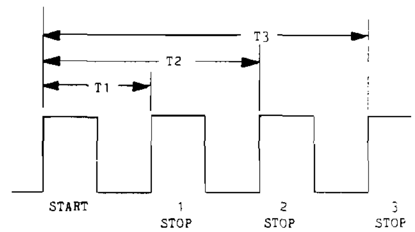
```
                    *
                    * CARRY TELLS IF WE HIT
                    *  THE END OF THE ARRAY:
                    *    0 8 NO, 1 = YES
                    *
10E5 E6 00          BUMP   INCZ  APTR   BUMP LO HALF
10E7 D0 0E                 BNE   BUMP0  =>NOT OFF END
10E9 E6 01                 INC   APTR   +01 BUMP HI HALF
10EB A5 01                 LDA   APTR   +01 GET IT
10ED C9 10                 CMPIM AEND   OFF THE END?
10EF D0 06                 BNE   BUMP0  =>NOT YET
10F1 A9 0C                 LDAIM ASTART =>YES: RESET AND
10F3 85 01                 STAZ  APTR   +0,1 TELL CALLER
10F5 38                    SEC
10F6 60                    RTS
10F7 18             BUMP0  CLC
10F8 60                    RTS
                    *
                    * SPECIAL FORM OF "PLOT"
                    *  ROUTINE: MONITOR'S ONE
                    *  DOESN'T ALLOW PLOTTING
                    *  IN SECOND CRT BUFFER,
                    *  SO WE ADD A HOOK FOR IT
                    *
10F9 4A             PLOTX  LSRA
10FA 08                    PHP
10FB 20 47 F8              JSR   GBASCA
                    *
                    * ABOVE INSTRUCTIONS ARE
                    *  TAKEN RIGHT OUT OF THE
                    *  MONITOR'S ROUTINE.  BUT
                    *  WE WILL NOW UPDATE THE
                    *  HI ADDRESS IN "GBASH"
                    *  TO FORCE PLOTTING IN THE
                    *  CORRECT SCREEN BUFFER
                    *
10FE A5 07                 LDA   CRT    GET CRT
1100 18                    CLC          ADD TO THE
1101 65 27                 ADC   GBASH  VALUE FOR
1103 85 27                 STA   GBASH  POSSIBLE SECOND CRT
1105 4C 05 F8              JMP   $F805  AND CONTINUE WITH
                    *
                    * DATA AREAS
                    *  (READ ONLY)
                    *
1108 04             CRTNUM =     $04
1109 00                    =     $00
110A DF             OFFSET =     $DF
110B FF                    =     $FF
110C E0                    =     $E0
110D FF                    =     $FF
110E E1                    =     $E1
110F FF                    =     $FF
1110 FF                    =     $FF
1111 FF                    =     $FF
1112 01                    =     $01
1113 00                    =     $00
1114 1F                    =     $1F
1115 00                    =     $00
1116 20                    =     $20
1117 00                    =     $00
1118 21                    =     $21
1119 00                    =     $00

SYMBOL TABLE 2000 20FC
AEND   0010   APTR   0000   AQH    003D   AQL    003C
ARH    003F   ARL    003E   ASTART 000C   ATH    0043
ATL    0042   BUMP   10E5   BUMPP  10F7   CHKBIR 1071
CLRA   1009   COLOR  0030   CRTFLI C054   CRTNUM 1108
CRT    0007   DIE    107D   GBASCA F847   GBASH  0027
INIT   1000   KB     C000   KBS    C010   MOVE   F32C
NBCHK  1034   NCELLS 0006   NCNT   0005   NEXTNB 105D
NNUM   0004   NOKEY  10CF   NPTR   0002   OFFSET 110A
ORIGIN 10DA   PASSQ  102C   PASSR  1082   PLOTX  10F9
RETURN 10C8   RUN    1029   SETCOL 108D   STORE  1053
SUBPT  1051   SURVIV 1075
```

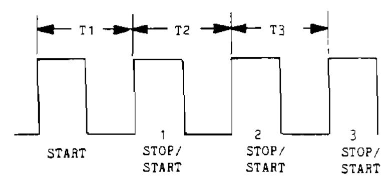
SYM-1 Event Timer

Help that onboard 6532 earn its keep with this 100 KHz event timer. It handles up to 50 elapsed time intervals or successive timed events.

Stephen J. Faris
Synertek Inc.
400 Humphrey Street
Swampscott, MA 01907

Very often it is desired to measure the time between two events, such as the start and end of a race or the time taken to respond to a given stimulus. The time between events can occur from a given start pulse to each succeeding pulse as follows:





This article will use the SYM-1 board's 6532 timer and keyboard display to create a device which can measure up to 50 time intervals, store them in memory and then read them out one at a time.

The first segment of the program is a loop loop to look for the start pulse, set up the 6532 timer and then look for the stop pulse. To measure the time between events, the 6532 generates an interrupt whenever it times out. The interrupt routine increments a 6-digit counter which counts the number of time intervals until the stop pulse is found. With minor modifications the program can accomplish both types of event measurement mentioned earlier.

For example, the program listing shown will measure time events per Figure 2. In order to measure as per Figure 1, change the BNE (24B) instruction to jump to STOP 1. The number of time events is fixed by N. The last segment of the program is the readout routine. This routine will read out each time interval from 1 to N, stopping so that the answer can be written down before going on automatically to the next. After completing the routine, the program jumps back to the beginning.

```
                    MICRO-WARE ASSEMBLER 65XX-1.0
                  *
                  * EVENT TIMER PROGRAM
                  *
                  * BY STEPHEN J. FARIS
                  * MODIFIED BY MICRO STAFF
                  *

02AF              GETKEY *     $88AF
02AF              ACCESS *     $8B86
02AF              OUTBYT *     $82FA
02AF              SCAND  *     $8906
02AF              D1     *     $0000
02AF              D2     *     $0001
02AF              D3     *     $0002
02AF              N      *     $0006
02AF              R1     *     $0006
02AF              R2     *     $0056
02AF              R3     *     $00A6
02AF              DB1    *     $A645
02AF              DB2    *     $A644
02AF              DB3    *     $A643
02AF              DB4    *     $A642
02AF              DB5    *     $A641
02AF              PAO    *     $A001
02AF              PBO    *     $A000

0200                     ORG   $0200
0200 78                  SEI                 DISABLE INTERRUPTS
0201 F8                  SED
0202 A2 00               LDXIM $00           SET DECIMAL MODE
0204 A9 50               LDAIM $50           AND SET X=0
0206 8D 7E A6            STA   $A67E
0209 A9 02               LDAIM $02
020B 8D 7F A6            STA   $A67F         THEN SET INTERRUPT VECTOR
020E A9 FF        INIT   LDAIM $FF
0210 85 00               STA   D1
0212 85 01               STA   D2
0214 85 02               STA   D3            INIT COUNTER
0216 AD 01 A0     START1 LDA   PAO
0219 29 01               ANDIM $01           LOOK FOR "1" LEVEL
021B F0 F9               BEQ   START1        AT PAO INPUT
021D AD 01 A0     START2 LDA   PAO
0220 29 01               ANDIM $01           LOOK FOR "0" LEVEL
0222 D0 F9               BNE   START2        AT PAO INPUT
0224 58                  CLI
0225 00                  BRK                 GENERATE INTERRUPT
0226 EA                  NOP
0227 A5 02        STOP1  LDA   D3
0229 C9 FF               CMPIM $FF           LOOK FOR "1" LEVEL AT PBO
022B F0 0E               BEQ   ESC           INPUT WITH TIMEOUT AFTER
022D AD 00 A0            LDA   PBO           99 IN D3
0230 29 01               ANDIM $01
0232 F0 F3               BEQ   STOP1
0234 AD 00 A0     STOP2  LDA   PBO           LOOK FOR "0" LEVEL
0237 29 01               ANDIM $01           AT PBO INPUT
0239 D0 F9               BNE   STOP2
023B E8           ESC    INX                 INCREMENT X FOR NEXT POINT
023C A5 00               LDA   D1
023E 95 06               STAZX R1
0240 A5 01               LDA   D2
0242 85 56               STA   R2
```

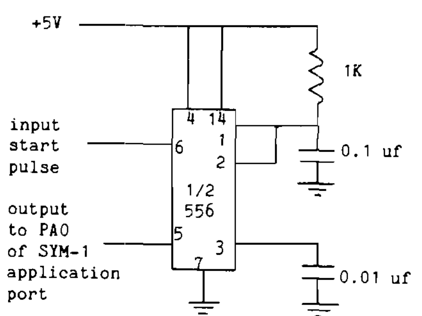
The time interval increments can be changed by accessing different dividers of the 6532 and changing the timer count

To operate the program the following steps are necessary:

1. Enter in location N the number of time intervals to be measured. (In HEX, less than 31.)
2. Decide what type of time intervals are to be measured (i.e. Figure 1 or Figure 2).
3. Decide time interval needed and enter VAL from Table I.
4. Start program at location 200.
5. Display results by hitting 1 on the keyboard.

The interface hardware to the event timer can be a 556 timer connected as shown in Figure 3.

The input signals can be derived from switches, light coupled devices or transducers. The output pulses are 50 microseconds wide and positive going. This conditions the input pulses so the software can look for a minimum width pulse. The only other hardware required is a SYM-1 board, cassette and power supply.



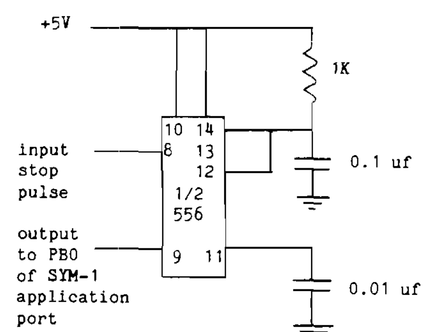


Figure 3: Connecting a 556 timer for use as an interface to the event timer.

```
0244 A5 02            LDA    D3          STORE RESULTS
0246 85 A6            STA    R3          IN MEMORY LOCATIONS
0248 78               SEI
0249 E4 06            CPX    N           IS NUMBER OF INTERVALS DONE
024B D0 C1            BNE    INIT
024D 4C 6A 02         JMP    RDOUT       JUMP TO DISPLAY
                *
                * 6-DIGIT COUNTER INTERRUPT ROUTINE
                *
0250 48         INTR  PHA                PUCH ACCUMULATOR
0251 A9 49            LDAIM  $49
0253 8D 1C A4         STA    $A41C       LOAD TIMER
0256 A9 01            LDAIM  $01
0258 65 06            ADC    R1          TWO LEAST SIGNIFICANT
025A 85 06            STA    R1          DIGITS
025C A5 00            LDA    $00
025E 65 56            ADC    R2          MIDDLE TWO DIGITS
0260 85 56            STA    R2
0262 A9 00            LDAIM  $00
0264 65 A6            ADC    R3
0266 85 A6            STA    R3
0268 68               PLA                PULL ACCUMULATOR
0269 40               RTI                RETURN FROM INTERRUPT
                *
                * READOUT ROUTINE
                *
026A 20 86 8B   RDOUT JSR    ACCESS      UN-WRITE PROTECT
026D 20 AF 88   KEY   JSR    GETKEY      SEARCH FOR "1" KEY
0270 C9 31            CMPIM  $31
0272 D0 F9            BNE    KEY
0274 A6 06            LDX    N
0276 A0 F0      L4    LDYIM  $F0         LOAD Y FOR DISPLAY TIME
0278 B5 A6            LDAZX  R3          ON EACH INTERVAL
027A 20 FA 82         JSR    OUTBYT
027D B5 56            LDAZX  R2
027F 20 FA 82         JSR    OUTBYT
0282 B5 06            LDAZX  R1
0284 20 FA 82         JSR    OUTBYT
0287 A9 FF      L3    LDAIM  $FF
0289 8D 1F A4         STA    $A41F       SET UP TIMER FOR
028C 8D 04 A4         STA    $A404       DISPLAY TIME
028F 8E 00 03         STX    $0300
0292 8C 01 03         STY    $0301
0295 20 06 89   L1    JSR    SCAND       SCAN DISPLAY UNTIL TIMEOUT
0298 AD 05 A4         LDA    $A405
029B 10 F8            BPL    L1
029D AE 00 03         LDX    $0300
02A0 AC 01 03         LDY    $0301       RESTORE X AND Y
02A3 88               DEY
02A4 F0 03            BEQ    L2
02A6 4C 87 02         JMP    L3
02A9 CA         L2    DEX
02AA D0 CA            BNE    L4
02AC 4C 0E 02         JMP    INIT

 ACCESS 8B86    DBQ    A645    DBR    A644    DBS    A643
 DBT    A642    DBU    A641    DQ     0000    DR     0001
 DS     0002    ESC    023B    GETKEY 88AF    INIT   020E
 INTR   0250    KEY    026D    LQ     0295    LR     02A9
 LS     0287    LT     0276    N      0006    OUTBYT 82FA
 PAP    A001    PBP    A000    RDOUT  026A    RQ     0006
 RR     0056    RS     00A6    SCAND  8906    STARTQ 0216
 STARTR 021D    STOPQ  0227    STOPR  0234
```

Table 1: Time Interval Data

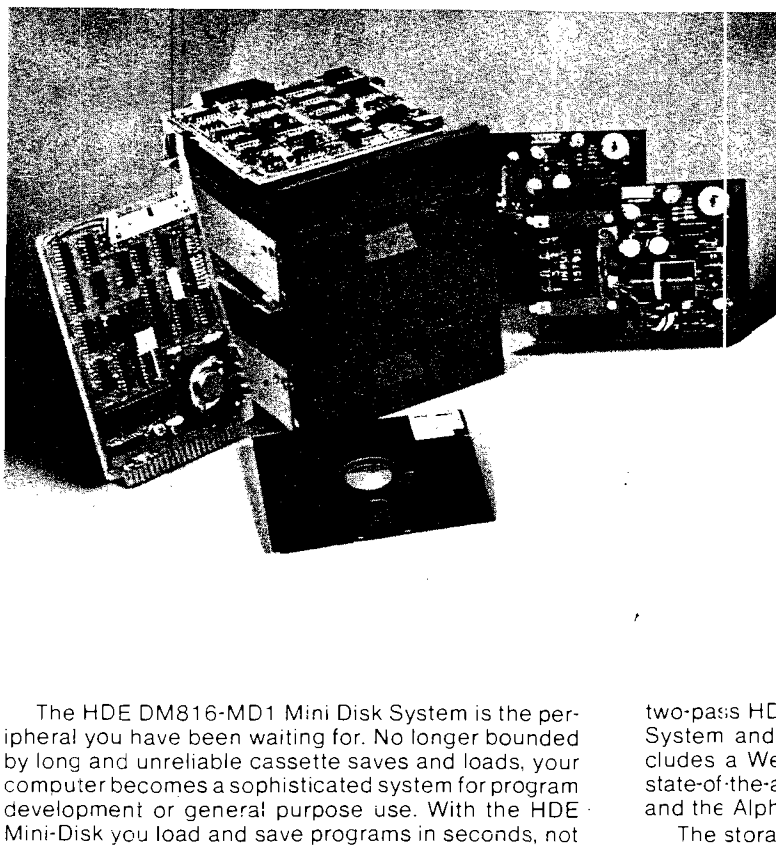
TIME INTERVAL	VALUE	ADDRESS (253 of Program)
100 MICROSECONDS	49	A41C
1 MILLISECOND	7A	A41D
10 MILLISECONDS	9C	A41E
100 MILLISECONDS	62	A41F

HDE inc.

BOX 120
ALLAMUCHY, N.J. 07820
201-362-6574

HUDSON DIGITAL ELECTRONICS INC.

THE HDE MINI-DISK SYSTEM



VERSIONS

KIM

TIM

AIM 65 – 4th Qtr. '79

SYM – 1st Qtr. '80

SINGLE DRIVE	$ 795.00
DUAL DRIVE	$1195.00

Complete with all hardware. Interconnecting cables, FODS, text editor and user and installation manuals.

The HDE DM816-MD1 Mini Disk System is the peripheral you have been waiting for. No longer bounded by long and unreliable cassette saves and loads, your computer becomes a sophisticated system for program development or general purpose use. With the HDE Mini-Disk you load and save programs in seconds, not minutes or hours. And, since all transfers to and from the Mini-Disk are verified for accuracy, the data will be there when you need it.

The HDE DM816-MD1 Mini-Disk has been "systems" engineered to provide a complete and integrated capability. Software and hardware have been built as a team using the most reliable components available. The systems software includes the acclaimed and proven HDE File Oriented Disk System and Text EDitor, requiring only 8K for the operating software and overlay area. Systems expanding programs available include the two-pass HDE assembler, the Text Output Processing System and Dynamic Debugging Tool. Hardware includes a Western Digital 1771 based controller in a state-of-the-art $4^{1/2} \times 6^{1/2}$" card size, Shugart SA 400 drive and the Alpha power supply.

The storage media for the DM816-MD1 is the standard, soft sectored $5^{1/4}$" mini diskette readily available at most computer stores, and HDE has designed the system so that the diskettes rotate only during disk transactions, favorably extending media life. A disk formatter routine included with the system, formats the diskettes, verifies media integrity by a comprehensive R/W test and checks drive RPM. Additional utilities provide ascending or descending alpha numeric sort, disk packing, text output formatting, file renaming, file addressing and other capabilities.

HDE PRODUCTS. BUILT TO BE USED WITH CONFIDENCE.

AVAILABLE DIRECT OR FROM THESE FINE DEALERS:

JOHNSON COMPUTER	PLAINSMAN MICROSYSTEMS	ARESCO	LONG ISLAND COMPUTER GENERAL STORE	LONE STAR ELECTRONICS
Box 523	Box 1712	P.O. Box 43	103 Atlantic Avenue	Box 488
Medina, Ohio 44256	Auburn, Ala. 36830	Audubon, Pa. 19407	Lynbrook, N.Y. 11563	Manchaca, Texas 78652
216-725-4560	800-633-8724	215-631-9052	516-887-1500	612-282-3570

AIM-65 in the Ham Shack

Dr. Marvin L. De Jong
Department of Mathematics
and Physics
The School of the Ozarks
Point Lookout, MO 65726

Have a field day with this message transmitter and keyer. It will accept and save messages to be broadcast automatically, as needed, in response to a single keystroke.

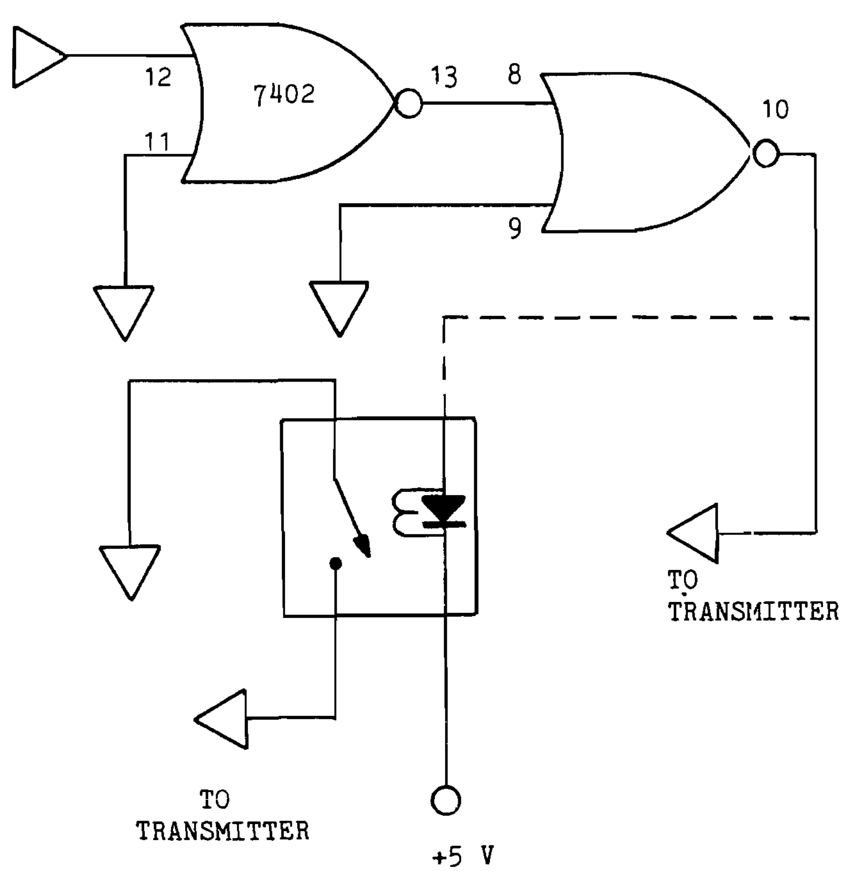


Figure 1: Buffer circuit to key the transmitter. Some transmitters may require the keying relay. Only two ICs, a 74LS00 and a 7402 are required to construct this interface and the keyer interface in Figure 2. Both ICs have pin 14 as +5V and pin 7 as GND.

Contest operating is a lot easier if standard messages such as "CQ CQ VQ DE K0EI K0EI K" can be sent automatically. The program listed in Table I in the AIM 65 disassembly format allows you to do just that. It has the following features:

1) Three different messages, called A, B, and C, may be stored in one page of memory. The total number of characters, including spaces, may not exceed 256 characters.

2) The messages are composed and edited using the AIM 65 keyboard. As the message is typed on the keyboard, it appears on the display and scrolls left. The delete key allows corrections to be made. The carriage return key signals that the message is complete, and the display blanks in preparation for the next message.

3) When all three messages are entered the display blanks again, and you enter your code speed in words per minute (in decimal). The code speed will remain displayed until new messages are entered by restarting the program.

4) With the messages and code speed loaded, a depression of the A key will result in message A being sent, the B key will cause the B message to be sent, and the C key will cause the C message to be sent.

5) The code speed can be changed at the end of any message by pressing the S key. This display will blank, and a new code speed can be entered.

6) A simple interface circuit and an interrupt routine allow the same program to be used as a keyer, operating at the same speed as the speed entered on the keyboard. You must provide paddle, or modify a bug to make the mechanical connections.

7) Code speeds from 5 wpm to 99 wpm are possible for the message sender and keyer, though it is

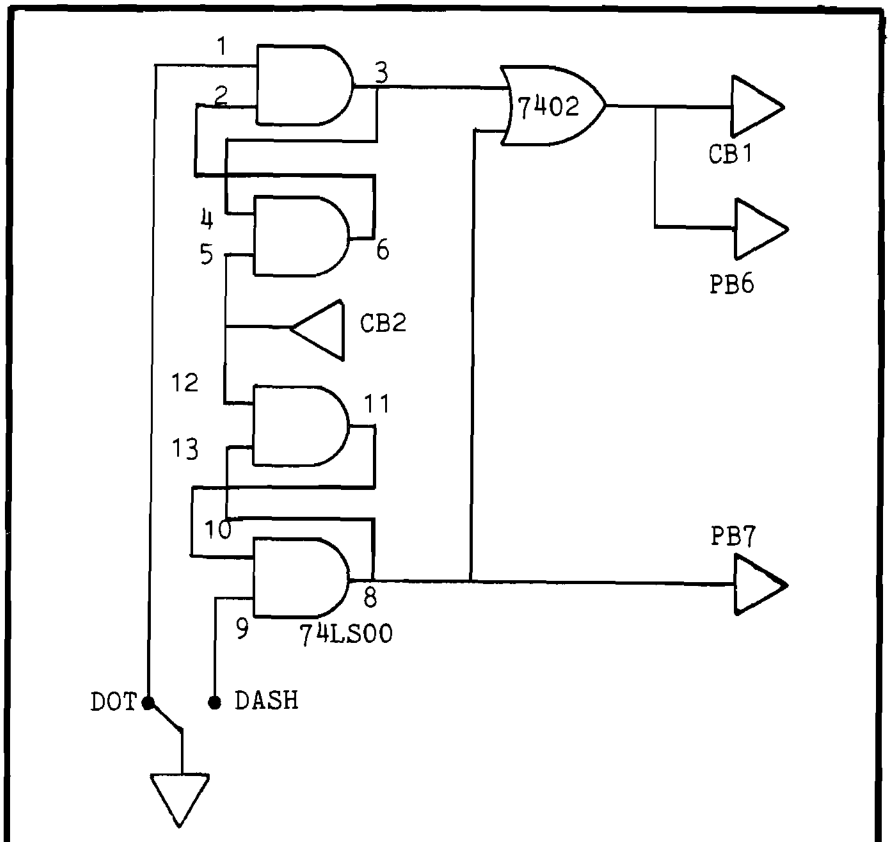


Figure 2: Interface circuit for the keyer. All connections are to the Applications Connector on the AIM 65.

unlikely that any of us will ever send 99 wpm with a keyer.

Before illustrating how the program might be used, let me point out that a similar program for the KIM-1 is scheduled to appear in the September or October issue of **73** Magazine, and the details of its *operation are* described there. Only a few features unique to the AIM 65 version will be given here. Also, most of the display routines have been described in a companion article in MICRO, and they will not be described again.

Let me describe how the keyer and message sender might be used for Field Day. You would start the program, then load message A as follows: "CQ CQ CQ FD DE K0EI/0 K0EI/0 K" Expecting someone to respond to your CQ, message B would be "DE K0EI UR MO MO BK" To use message B, you would first send the other guy's call with the keyer and then hit key B on the keyboard. The blanks, spaces inserted by pressing the keyboard, would be filled by you, again using the keyer, to give the proper signal report. Message C might read "QSL ES TU OM" and would be sent after you received his signal report and section correctly. It would not be difficult to modify the program for sweepstakes or

```
                     *
                     * MAIN PROGRAM
                     * MODIFIED 7-16-79
                     *   BY MICRO STAFF
                     *
0200                        ORG    $0200
0200 78                     SEI
0201 A9 A0                  LDAIM  $A0
0203 8D 0C A0               STA    $A00C
0206 A9 01                  LDAIM  $01
0208 8D 00 A0               STA    $A000
020B 8D 02 A0               STA    $A002
020E 20 9B 03               JSR    $039B
0211 A2 00                  LDXIM  $00
0213 A0 00                  LDYIM  $00
0215 94 01                  STYZX  $0001
0217 20 3C E9               JSR    $E93C
021A C9 7F                  CMPIM  $7F
021C D0 10                  BNE    $022E
021E A9 20                  LDAIM  $20
0220 88                     DEY
0221 99 00 01               STAY   $0100
0224 8A                     TXA
0225 48                     PHA
0226 20 85 03               JSR    $0385
0229 68                     PLA
022A AA                     TAX
022B 18                     CLC
022C 90 E9                  BCC    $0217
022E C9 5B                  CMPIM  $5B
0230 F0 2A                  BEQ    $025C
0232 C9 0D                  CMPIM  $0D
0234 F0 13                  BEQ    $0249
0236 99 00 01               STAY   $0100
0239 8A                     TXA
023A 48                     PHA
023B B9 00 01               LDAY   $0100
023E 20 72 03               JSR    $0372
0241 20 60 03               JSR    $0360
0244 68                     PLA
0245 AA                     TAX
0246 C8                     INY
0247 D0 CE                  BNE    $0217
0249 8A                     TXA
024A 48                     PHA
024B 20 9B 03               JSR    $039B
024E 20 60 03               JSR    $0360
0251 68                     PLA
0252 AA                     TAX
0253 88                     DEY
0254 94 04                  STYZX  $0004
0256 C8                     INY
0257 E8                     INX
0258 E0 03                  CPXIM  $03
025A 90 B9                  BCC    $0215
025C 20 9B 03               JSR    $039B
025F 20 60 03               JSR    $0360
0262 20 3C E9               JSR    $E93C
0265 48                     PHA
0266 20 72 03               JSR    $0372
0269 20 60 03               JSR    $0360
026C 68                     PLA
026D 38                     SEC
026E E9 30                  SBCIM  $30
0270 0A                     ASLA
0271 0A                     ASLA
0272 0A                     ASLA
0273 0A                     ASLA
0274 85 11                  STAZ   $0011
0276 20 3C E9               JSR    $E93C
0279 48                     PHA
027A 20 72 03               JSR    $0372
027D 20 60 03               JSR    $0360
0280 68                     PLA
0281 38                     SEC
0282 E9 30                  SBCIM  $30
0284 18                     CLC
0285 65 11                  ADCZ   $11
0287 48                     PHA
0288 29 F0                  ANDIM  $F0
028A 4A                     LSRA
028B 85 10                  STAZ   $0010
028D 4A                     LSRA
028E 4A                     LSRA
028F 18                     CLC
0290 65 10                  ADCZ   $0010
0292 85 10                  STAZ   $0010
0294 68                     PLA
0295 29 0F                  ANDIM  $0F
0297 65 10                  ADCZ   $0010
0299 85 10                  STAZ   $0010
029B 38                     SEC
029C A2 00                  LDXIM  $00
029E A9 94                  LDAIM  $94
02A0 85 08                  STAZ   $0008
02A2 A9 04                  LDAIM  $04
02A4 85 09                  STAZ   $0009
02A6 A5 08                  LDAZ   $0008
02A8 E5 10                  SBCZ   $0010
02AA 85 08                  STAZ   $0008
02AC A5 09                  LDAZ   $0009
02AE E9 00                  SBCIM  $00
02B0 85 09                  STAZ   $0009
02B2 E8                     INX
02B3 B0 F1                  BCS    $02A6
```

```
02B5 86 07          STXZ  $0007
02B7 A9 43          LDAIM $43
02B9 8D 04 A4       STA   $A404
02BC A9 03          LDAIM $03
02BE 8D 05 A4       STA   $A405
02C1 A9 90          LDAIM $90
02C3 8D 0E A0       STA   $A00E
02C6 20 3C E9       JSR   $E93C
02C9 58             CLI
02CA C9 53          CMPIM $53
02CC F0 8E          BEQ   $025C
02CE A0 00          LDYIM $00
02D0 C9 41          CMPIM $41
02D2 F0 0A          BEQ   $02DE
02D4 C9 42          CMPIM $42
02D6 F0 05          BEQ   $02DD
02D8 C9 43          CMPIM $43
02DA D0 EA          BNE   $02C6
02DC C8             INY
02DD C8             INY
02DE B6 01          LDXZY $01
02E0 20 ED 02       JSR   $02ED
02E3 8A             TXA
02E4 D9 04 00       CMPY  $0004
02E7 F0 DD          BEQ   $02C6
02E9 E8             INX
02EA 4C E0 02       JMP   $02E0
              *
              * SEND SUBROUTINE
              *
02ED 8A             TXA
02EE 48             PHA
02EF BD 00 01       LDAX  $0100
02F2 AA             TAX
02F3 B5 00          LDAX  $00
02F5 F0 1E          BEQ   $0315
02F7 0A             ASLA
02F8 F0 10          BEQ   $030A
02FA 48             PHA
02FB B0 06          BCS   $0303
02FD 20 1A 03       JSR   $031A
0300 4C 06 03       JMP   $0306
0303 20 33 03       JSR   $0333
0306 68             PLA
0307 4C F7 02       JMP   $02F7
030A A2 02          LDXIM $02
030C 20 38 03       JSR   $0338
030F CA             DEX
0310 D0 FA          BNE   $030C
0312 68             PLA
0313 AA             TAX
0314 60             RTS
              *
              * DIT AND DAH
              * SUBROUTINES
              *
0315 A2 04          LDXIM $04
0317 4C 0C 03       JMP   $030C
031A A2 01          LDXIM $01
031C CE 00 A0       DEC   $A000
031F 20 38 03       JSR   $0338
0322 CA             DEX
0323 D0 FA          BNE   $031F
0325 AD 00 A0       LDA   $A000
0328 4A             LSRA
0329 B0 07          BCS   $0332
032B EE 00 A0       INC   $A000
032E E8             INX
032F 4C 1F 03       JMP   $031F
0332 60             RTS
0333 A2 03          LDXIM $03
0335 4C 1C 03       JMP   $031C
              *
              * TIMER SUBROUTINE
              *
0338 A5 07          LDAZ  $0007
```

other contests. Any time you want to insert something in a message, be sure to leave enough time, in ASCII spaces, to key in the insert. You soon get the hang of working so smoothly that no one will recognize your insert for what it is.

Some notes about the program may be useful if you want to modify it for your own purposes. Instructions starting at $0200 and ending at $025F are used to load the three messages. The instructions starting at $0262 and ending at $0285 are used to enter the code speed in decimal, convert the ASCII representations to decimal numbers, and store the result in location $0011. The instructions starting at $0287 and ending at $0299 are used to convert this decimal number to a hexadecimal number and store it in location $0010. The instructions starting at $029B and ending at $02B5 are used to convert the speed to a number to be loaded into the divide-by-1024 timer. The remainder of the program tests for A, B, or C key depressions, and it calls on various subroutines to send the message If you do not want to use the AIM 65 as a keyer, then you may omit the interrupt routine.

Note that in my program listings I used page one, addresses $0100 and upward to store the message. I would not recommend this, but since I have only 1K of memory, I could not use $0400 to $04FF. If you have more than 1K of memory, I would urge you to change all the $0100 addresses in the program to $0400 or the page of your own choosing. These instructions are located at $0221, $0236, $023B, and $02EF.

The interface circuits are shown in Figures 1 and 2. Figure 1 gives a circuit that simply buffers the output of the PB0 pin to key my transmitter. A optional relay may be required for other transmitters. The NOR gates were used because I needed a NOR gate in the keyer circuit, and the NOR gates allow you to OR your own keyer to the message sender circuit. The keyer circuit is shown in Figure 2. Basically it debounces the dot and dash paddle connections, and it may be reset with a pulse from pin CB2.

When the key is put in either the dot or dash position, a negative going signal is produced at pine one of the 7402. A negative pulse on CB1 produces an interrupt, so the program jumps to process the interrupt routine. In the interrupt routine Port B is read (LDA A000), producing a negative pulse on CB2. This negative pulse will reset the cross-coupled NAND gates if the key is up, otherwise pin one of the 7402 will stay at logic zero. As long as it is at logic zero the program continues to send dots (or dashes). Reading Port B also clears the interrupt flag on the 6522. As soon as PB6 is set to logic one by the negative pulse from CB2, the interrupt routine is completed and execution continues in the main program. μ

```
033A 8D 97 A4       STA   $A497
033D 2C 97 A4       BIT   $A497
0340 10 FB          BPL   $033D
0342 60             RTS
              *
              * INTERRUPT ROUTINE
              *
0343 48             PHA
0344 8A             TXA
0345 48             PHA
0346 AD 00 A0       LDA   $A000
0349 30 06          BMI   $0351
034B 20 1A 03       JSR   $031A
034E 4C 54 03       JMP   $0354
0351 20 33 03       JSR   $0333
0354 AD 00 A0       LDA   $A000
0357 0A             ASLA
0358 10 EC          BPL   $0346
035A 68             PLA
035B AA             TAX
035C 68             PLA
035D 40             RTI
              *
              * DISPLAY
              * SUBROUTINU
              *
0360                ORG   $0360
0360 A2 13          LDXIM $13
0362 8A             TXA
0363 48             PHA
0364 BD 38 A4       LDAX  $A438
0367 09 80          ORAIM $80
0369 20 7B EF       JSR   $EF7B
036C 68             PLA
036D AA             TAX
036E CA             DEX
036F 10 F1          BPL   $0362
0371 60             RTS
              *
              * MODIFY SUBROUTINE
              *
0372 8D 4C A4       STA   $A44C
0375 A2 01          LDXIM $01
0377 BD 38 A4       LDAX  $A438
037A CA             DEX
037B 9D 38 A4       STAX  $A438
037E E8             INX
037F E8             INX
0380 E0 15          CPXIM $15
0382 90 F3          BCC   $0377
0384 60             RTS
              *
              * BACKSPACE
              * SUBROUTINE
              *
0385 A2 12          LDXIM $12
0387 BD 38 A4       LDAX  $A438
038A E8             INX
038B 9D 38 A4       STAX  $A438
038E CA             DEX
038F CA             DEX
0390 10 F5          BPL   $0387
0392 A9 20          LDAIM $20
0394 8D 38 A4       STA   $A438
0397 20 60 03       JSR   $0360
039A 60             RTS
              *
              * CLEAR SUBROUTINE
              *
039B A2 13          LDXIM $13
039D A9 20          LDAIM $20
039F 9D 38 A4       STAX  $A438
03A2 CA             DEX
03A3 10 FA          BPL   $039F
03A5 60             RTS
```

ATARI® PERSONAL COMPUTER
CCI COMPUTER SYSTEMS DELIVERS . . . NOW!!!

ATARI® 400™
PERSONAL COMPUTER SYSTEM

PRICE INCLUDES:
Computer Console
BASIC Language Cartridge
BASIC Language Programming Manual (Wiley)
400 Operator's Manual with Note Book
Power Supply
TV Switch Box

$549.99

ATARI® 800™
PERSONAL COMPUTER SYSTEM

PRICE INCLUDES:
Computer Console
BASIC Language Cartridge
Education System Master Cartridge
BASIC Language Programming Manual (Wiley)
800 Operator's Manual with Note Book
ATARI 410 Program Recorder
Guide to BASIC Programming Cassette
8K RAM Module • Power Supply • TV Switch Box

$999.99

WE PROMISE TO DELIVER!!!

- We GUARANTEE ship dates on prepaid Computer System orders. Send your orders in now!!!
- If for reasons beyond our control we miss a ship date, we will REFUND the shipping and handling charges and give you a 10% DISCOUNT on any future software purchases for your ATARI® System.*
- For prepaid system orders, get a FREE Accessory Controller of your choice.

*Order must be prepaid by Cashier's Check made out to Computer Components of Orange County.

ATARI USER'S GROUP INFORMATION . . . CALL (714) 891-2584

PERIPHERALS AND ACCESSORIES

ATARI® 810™ DISC DRIVE* $749.99

DISKETTES
CX8100 BLANK DISKETTES*
CX8101 DISK FILE MANAGER*
$5.00/ea.

ATARI® 820™ PRINTER* $599.99

ACCESSORY CONTROLLERS
CX2C-01 DRIVING CONTROLLER PAIR
CX30-04 PADDLE CONTROLLER PAIR
CX40-04 JOYSTICK CONTROLLER PAIR
$19.95/ea.

ATARI® 410™ PROGRAM RECORDER $89.99

ADD-ON MEMORY (800 ONLY)
CX852 8K RAM MEMORY MODULE $124.99
CX853 16K RAM MEMORY MODULE $249.99

SOFTWARE

ROM CARTRIDGES
CXL4001 EDUCATION SYSTEM MASTER CARTRIDGE $34.99
KEY: (j) = uses joystick controller
(p) = uses paddle controller
(d) = uses driving controller

GAMES . . . . $49.99/ea.
CXL4004 BASKETBALL . . . . . (j)
CXL4005 LIFE . . . . . (j)
CXL4006 SUPER BREAKOUT™ . . . . . (p)
CXL4008 SUPER BUG™** . . . . . (d)

APPLICATION $69.99
CXL4002 ATARI BASIC
CXL4003 ASSEMBLER DEBUG**
CXL4007 MUSIC COMPOSER
CXL4009 COMPUTER CHESS** . . . . . (j)

EDUCATION SYSTEM CASSETTE PROGRAMS $39.99/ea.
CX6001 U.S. HISTORY
CX6002 U.S. GOVERNMENT
CX6003 SUPERVISORY SKILLS
CX6004 WORLD HISTORY (WESTERN)
CX6005 BASIC SOCIOLOGY
CX6006 COUNSELING PROCEDURES
CX6007 PRINCIPLES OF ACCOUNTING
CX6008 PHYSICS
CX6009 GREAT CLASSICS (ENGLISH)
CX6010 BUSINESS COMMUNICATIONS
CX6011 BASIC PSYCHOLOGY
CX6012 EFFECTIVE WRITING
CX6013 AUTO MECHANICS
CX6014 PRINCIPLES OF ECONOMICS
CX6015 SPELLING
CX6016 BASIC ELECTRICITY
CX6017 BASIC ALGEBRA

BASIC GAME AND PROGRAM CASSETTES
CX4101 GUIDE TO BASIC PROGRAMMING*
CX4102 BASIC GAME PROGRAMS*
$29.95/ea.

*October Delivery **November Delivery

—Prices subject to change.—

COMPUTER COMPONENTS OF ORANGE COUNTY

6791 Westminster Ave., Westminster, CA 92683 714-891-2584 Telex 182274
Hours: Tues-Fri 11:00 AM to 8:00 PM—Sat 10:00 AM to 6:00 PM—Sun 12:00 to 4:00 PM (Closed Mon)
Master Charge, Visa, B of A are accepted. Allow 2 weeks for personal check to clear.
Add $2.00 for handling and postage. For computer systems please add $10.00 for shipping, handling and insurance. California residents add 6% Sales Tax.
• MAIL ORDER LINE (714) 891-2587 or TELEX 182274.

VAN NUYS	BURBANK	LAWNDALE	ORANGE COUNTY	IRVINE
(213) 786-7411	(213) 848-5521	(213) 370-4842	(714) 891-2584	(714) 891-2589

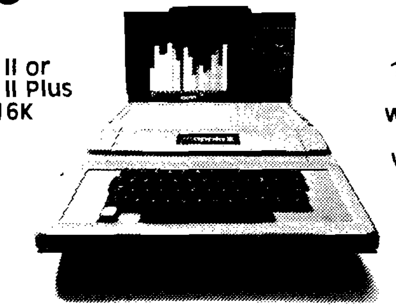
In Southern California We're Number One!!

COMPUTER COMPONENTS (CCI) OF ORANGE COUNTY, your full support computer store presents the finest in personal computers

Send for our 36 page fully illustrated catalog $3⁰⁰. Save 50¢ by sending the top one inch of this ad with your order.

AN INDUSTRY STANDARD

apple computer



Apple II or Apple II Plus with 16K $1195

SPECIAL

13" Color TV compatible with Apple II only 99⁰⁰ with system purchase

APPLE II will change the way you think about computers. That's because it is specifically designed to handle the day to day tasks of education, financial planning, building security, scientific calculation, and entertainment. **APPLE II** is appealing and comfortable (like other appliances that make your life easier); and it brings to personal computing a new level of simplicity through hardware and software sophistication.

INTERFACE CARDS

Prototyping/Hobby Card	$ 24.00
Parallel Printer Interface Card	180.00
Communications Card & DB25 Connector Cable	225.00
High-Speed Serial Interface Card	195.00
Language System with Pascal (48K RAM & Disk II Required)	495.00
Applesoft II Firmware Card	200.00
16 Input Analog Card	295.00

ACCESSORIES

Disk II—Drive Only	495.00
Disk II—Drive & Controller (32K Min. RAM Recommended)	595.00
Vinyl Carrying Case	30.00
Tape Recorder	40.00
Programmers Aid No. 1 Firmware (For Use with Integer BASIC)	50.00
Clock/Calendar Card	199.00
Auto—Start ROM Package (For Apple II Only	65.00
Digikitizer Pad by Talos (Kitform)	499.00
High Resolution Light Pen	199.00
Micromodem D.C. Hayes)	379.00
12" B/W Leedex Monitor	149 00
Cable from Monitor to Apple II	9.95
13" Color TV Compatible with Apple II	290.00
Sup-R-Modulator (RF)	25.00

SOFTWARE FOR APPLE II

PASCAL from Programma	49.95
FORTH	49.95
LISP—from Apple Software Bk No. 3	N/C
LISA—Interactive disk assembler	34.95
WHATSIT—Excellent conversational data base manager 32K 100.00	48K 125.00
SARGON-Champ of 2nd West Coast Computer Faire	19.95
APPLE PIE—Excellent text editor	24.95
FORTE—Music editor in hires	19.95
FASTGAMMON—Excellent backgammon game with graphics Tape 20.00	Disk 25.00
APPLE 21—Excellent blackjack game	9.95
BRIDGE CHALLENGER—Computer bridge	14.95

FINANCIAL MANAGEMENT SYSTEM

- Accounts Payable
- Accounts Receivable
- Inventory Control
- $200 Each Package
- Ledger Processing
- Payroll
- $800 Complete
- $10 for Manual

PRINTER SPECIALS FOR APPLE AND PET

TRENDCOM 100 with interface for Apple or PET	450.00
LITE PEN used with TV or monitor screen	34.95
ALF Music Synthesizer Boards	265.00
Supertalker	279.00
Anadex DP-8000 with tracter 8" paper width and interface to Apple	1050.00
Centronics 779-2 for Apple II with parallel interface	1245 00

SOFTWARE (Send for complete Software Catalog $1.00)

Dow Jones Portfolio Evaluator/ Stock Quote Reporter Disk	50.00
Microchess 2.0 Chess Disk	25.00
Disk Utility Pack with DOS 3.2	25.00
The Controller (General Business System)	625.00
Apple Post (Mailing List System)	49.95
Bowling Program Diskette	15.00
The Cashier (Retail Store Management	250.00
Checkbook Cassette	20.00
Applesoft II Language & Demo Cassette	20.00
RAM Test Tape with Manual	7.50
Finance 1-2 Cassette Package	25.00
Datamover/Telepong Cassette (Com. Card & Modem Req'd)	7 50
Microchess 2.0 Chess Tape	20 00
Bowling Program Tape	15 00
Pascal with Language System (48K RAM & Disk II Required)	495.00

MISCELLANEOUS

Vinyl Diskette Holder Pages (Pkg of 10)	8.50
Diskette Boxes (BA Plastic)	4 50
Diskettes (5")	
Apple (Box of 10)	50.00
Verbatim (Box of 10)	34 00
Dysan (Box of 5)	25.00
Decals (Rainbow Apple)	
Inside Window (2")	10¢
Outside Window (2")	10¢
Outside Window (1")	5¢
Apple Logo (Rainbow) T-Shirts Specify—Men, Women, or child (small, medium, or large)	6.00

Reference Books for Apple and PET Owners

Programming Manual (MOSTEK)	$12.00
Hardware Manual (MOSTEK)	12.00
Programming the 6502 (ZAKS)	9.95
6502 Application Book (ZAKS)	12.95
Programming a Micro Computer: 6502 (FUSTER)	9.95

PET Owners Only

PET User Manual	$9.95
Hands on BASIC with a PET	14.95
PET Machine Language Guide	9.95

Apple Owners Only

Apple II Reference Manual	$10.00
Apple Soft Manual	10.00
Programmer's Guide (Computer Station)	5.95
Apple II Monitor Peeled	9.95
Software Directory for Apple	
• Business, Finance & Utility	4.95
• Games, Demo, Utility	4.95
Best of Contact '78	2.50
Programming in PASCAL (Grogono)	9.90

A PROFESSIONAL BUSINESS SYSTEM

commodore CBM™

2001-16B
$995

Also
2001-8 ...... $795
2001-16N $995
2001-32N ... $1295
2001-32B ... $1295

341K DUAL DRIVE**

CBM 2040
$1295

Also
External Cassette $95
PET to IEEE Cable $39.95

80 COL. DOT MATRIX PRINTER

CMB 2022
$995

Also
CBM 2023 Printer $849
IEEE to IEEE Cable $49.95

**Retrofit kit required for operation with PET 2001-8.

ACCESSORIES FOR PET

Commodore PET Service Kit	$30.00
Beeper—Tells when tape is Loaded	24.95
Petunia—Play music with PET	29.95
Video Buffer—Attach another display	29.95
Combo—Petunia and Video Buffer	49.95

SOFTWARE FOR PET

Mirrors and Lenses	19.95
The States	14.95
Real Estate 1 & 2	59.95
Momentum and Energy	19.95
Projectile Motion	19.95
Mortgage	14.95
Dow Jones	7.95
Petunia Player Sftwr	14.95
Checkers and Baccarat	7.95
Chess	19.95
Series and Parallel Circuit Analysis	19.95
Home Accounting	9.95
BASIC Math	29.95
Game Playing with BASIC Vol. I, II, III	9.95 each

Plus many more. Send for Software Catalog $1.00.

—Prices subject to change—

COMPUTER COMPONENTS OF ORANGE COUNTY

6791 Westminster Ave., Westminster, CA 92683 714-891-2584 Telex 182274

Hours: Tues-Fri 11:00 AM to 8:00 PM—Sat 10:00 AM to 6:00 PM—Sun 12:00 to 4:00 PM (Closed Mon)

Master Charge, Visa, B of A are accepted. Allow 2 weeks for personal check to clear. Add $2.00 for handling and postage. For computer systems please add $10.00 for shipping, handling and insurance. California residents add 6% Sales Tax.

microbes

Stephen Bach of Rt 2, Box 50A1, Scottsville, VA reports that the 24 Hour AIM Clock program on MICRO 10:7 should contain F8 (SED) at location 0305, rather than 38 (SEC) as published.

Lt. Robert Carlson speculates that his article on A Baudot Teletype Driver for the APPLE II in MICRO 14:5 was mutilated by the editorial staff. It's true: location 037C should contain 68 (RORA), rather than 6E (ROR), and the spurious operand bytes at 037D and 037E should be removed to close up the code. In addition, the following lookup table should follow the program code, beginning at location 0381, immediately after the RTS which moved upward two bytes:

```
0381- 02 45 0A 41 20 53 49
0388- 55 0D 44 52 4A 4E 46 43
0390- 4B 54 5A 4C 57 48 59 50
0398- 51 4F 42 47 06 4D 58 56
03A0- 00 99 33 99 2D 99 07 38
03A8- 37 03 24 34 27 2C 21 3A
03B0- 28 35 22 29 32 23 36 30
03B8- 31 39 3F 26 99 2E 2F 3B
03C0- 99 00 00 C0 00
```

Roger Cohen, 100 Nimbus Road, Holbrook, NY 11741 reported the same editorial slip-up.

Several readers reported problems with the AMPERSORT article on MICRO 14:41. G. Lewis Scott of 6220 Colchester Place, Charlotte, NC 28210 sent in seven corrections to the assembly language source:

Line 0370 should be 5201- 20 E7 54
Line 0400 should be 5207- DD 2D 55
Line 1180 should be 5298- BD 33 55
Line 1240 should be 52A5- 20 0A 55
Line 2810 should be 53CA- 20 C2 54
Line 2870 should be 53D8- 20 c2 54
Line 3320 should be 5425- 20 0A 55

and Mr. Scott noted some additional problems once he got the program running. William G. Trawick of the Georgia State University Dept. of Chemistry in Atlanta, GA 30033 reported some of the same microbes. Mark Crosby of Washington APPLE Pie and 1373 E Street SE, Washington DC 20003 is also working on these difficulties. Alan G. Hill, the author of AMPERSORT, is collating corrections to these problems, most of which developed when last minute enhancements were integrated into the source. If you have keyed in AMPERSORT, save that tape! Any final patches will appear next month.

Peter J. Sleggs of 1208 Half Moon Lane, Oakville Ontario L6H 2E5 reports that the EKIM Extension to the KIM monitor in MICRO 11:20 will not perform as expected in the autoincrement and branch modes. He suggests changing 17D1 from B0 AD (BCS START) to B0 B4 (BCS GETK). Mr. Sleggs included an insightful enhancement to this routine; however, another very elegant enhancement arrived from

Gary A. Foote reports that his article on Sorting with the APPLE, in MICRO 13:22, should have line 80 reading:
80 I=J=K=L=M=X=T=Z=LL=II=LM=HM=W=N
whereas, in the original article, the "N" was inadvertently omitted. Also, for 48K system operation, line 90 should be changed so that it does not exceed the 32767 limit. It should be:
90 LM=PEEK(204)+PEEK(205)*256:HM=32767

Ralph W. Leiper of 18 Alberta Street, Windsor Locks, CT 06096 noticed a microbe in Harmonic Analysis for the APPLE, MICRO 13:5, which works perfectly unless one of the harmonics happens to be off scale. His fix is easy. Change line 1290 and 1300 to read:

```
1290 S=70: REM SETTING INITIAL SCALE
1300 PRINT: PRINT: PRINT "PLOT OF INPUT DATA
     CALCULATED TO FIFTH HARMONIC> Y AT 100
     = ";T: H=0: HGR
```

He also made the following changes to improve readability of the graphics output:
Add: 1325 HPLOT 0,79
Chg: 1360 HPLOT TO k, 79- Y
This plots the original curve as a solid line which will stand out from the harmonics.

William O. Taylor writes of an error in his article, The Basic Morse Keyboard, MICRO 13:13. The tone board schematic has output from the computer and +5 V power reversed! Exchange PB0 and +5 V to correct. Although the parts list shows a 50 yf cap while the schematic shows 35 yf for C2, either value will work. Finally, line 5 of the BASIC program should include keyword PRINT before the output string.

The article, APPLE II Serial Output Made Simple in MICRO 15:5 contains a full page of extraneous code on 15:7. All of page 15:7 should be removed from this article. This was another example of editorial staff confusion, hopefully exhausting our quota for many months.

You may write or telephone MICRO to obtain the current status of any published program.

Speech Processor for the PET

A speech processor unit samples audio waveforms and digitizes the input signal. Digitized speech can be stored, cataloged, processed as discrete data, and output through a D/A converter. The output speech quality rivals that of a CB radio.

Charles R. Husbands
24 Blackhorse Drive
Acton, MA 01720

Within the past year a low cost speech processor unit has appeared on the market. This device designed for the computer hobbyist can be used in a variety of applications from voice augmentation to computer games to direct computer-to-phone modem implementations.

This article will briefly describe the device and how the unit can be interfaced to a PET computer. A software driver program capable of storing the digitized sound, playing it back, saving the processed information on cassette files, and then reloading it will also be presented. The article will conclude with an illustration of how this device might be used with a home computer system.

The Speech Processor Unit

The speech procesor unit used in this article is the DATA-BOY™ Speech Processor developed by Mimic Electronics Company. This processor is an extremely low-cost audio signal processing system designed for the hobbyist. The speech processor is essentially a speech "digitizer" which uses a proprietary signal processing technique to convert the human voice into a single bit stream, and vice versa. "Digitized" speech is typically thought of as speech which has been sampled with an analog-to-digital form, and then reconverted to analog form by a digital-to analog (D/A) converter.

By using certain characteristics of the speech waveform, especially the fact that the amplitude components tend to decrease with increasing frequency, the resolution of the A/D and D/A converters required can be decreased from, say, 8 bits down to a single bit while maintaining intelligibility. When this bit stream is sampled at a rate of 8000 samples per second, highly intelligible speech can be obtained. The speech quality is close to that which is given by a CB radio.

Speech Processor Interface

Figure 1 describes the components necessary to support the speech processor and its interface with the PET Computer. To digitize and then reproduce speech the speech processor unit requires the use of an additional speaker, microphone and power supply. The speech processor unit is designed on a 3 inch by 5 inch printed circuit board. The author's unit was built into a 9 by 5½ inch box which also contained the power supply. The simple power supply design was taken directly from the users manual provided with the speech processing unit.

In addition to the interfaces shown in Figure 1, the author added two additional features to his unit. To determine when the squelch threshold level was exceeded one side of a light emitting diode was connected to the DATA READY line. The other side of the LED was connected to the +5 volt supply through a 300 ohm registor. When the squelch threshold level is exceeded the DATA READY line goes low and LED glows.

A computer bypass switch was also added to the author's unit. This switch allows the TO COMPUTER and FROM COMPUTER lines to be directly interconnected or interfaced to the computer. This feature allows the speech processor system to be tested independent of the computer. It also allows the user to demonstrate the difference in intelligibility produced by the computers quanitization effects.

The Speech Processor Unit is interconnected to the PET Computer by three lines. Each line is accompanied by a ground to provide some degree of shielding. The TO COMPUTER line is connected to PA0 (Pin C) on the USER Connector. This line will be sampled by the processor at the proper data rate to quanitize the input data stream. The digital output data stream will be returned to the speech processor unit on the line marked FROM COMPUTER. This line is attached to PA7 (Pin L) on the USER Connector. A third line termed the DATA READY line is used to indicate if the input signal level exceeds the threshold established in the speech processor unit. The DATA READY line is connected to PA1 (Pin D) on the PET USER Connector.

Software Description

The software used to drive the speech processor device is written in two parts: A User Interface Program written in BASIC and a pair of Speech Processor to Computer Interface Programs written in machine language. The User Interface Program is designed to allow the user to interact easily with the speech processor. This program provides four user options: RECORD, PLAYBACK, SAVE and LOAD.

The RECORD Program calls one of the machine language interface programs which samples the state of the speech processor bit stream and stores the sampled input data into sequential locations of buffer memory.

The PLAYBACK process is the direct counterpart of the RECORD process. In this mode each word in the buffer memory is unpacked and returned to the speech processor to reproduce the speech data examined during the RECORD sequence. The PLAYBACK process like the RECORD process calls a supporting machine language program.

The SAVE routine is a data file storage program which allows the user to save all or some portion of the recorded data on tape for later use.

The LOAD routine is a data file retrieval program which allows digital date files stored on tape by the SAVE program to be restored into the computers memory. Both the SAVE and LOAD routines allow the user to designate the beginning and ending address of the data to be manipulated. With this facility data words stored in memory can be saved and rearranged in order to build a data base where the beginning of each utterance or sound is uniquely defined.

An illustration of the memory map organization used to support the speech processor unit is shown in Figure 2. From this map it can be seen that the machine language programs required to support the BASIC programs are stored in tape buffer #2. In order to establish sufficient buffer memory to store the digitized speech information, a cap was placed on the BASIC program. By forcing the BASIC Interpreter into thinking it is operating with a 4K memory limitation, the upper 4K of memory can be used for storing the recorded digitized speech. A small number of bytes in zero page working storage are used for pass-

```
0010: REM ** SIMPLE VOICE PROCESSOR PROGRAM **
0020: REM
0030: REM BY CHARLIE R. HUSBANDS
0040: REM
0050: PRINT"
0060: PRINT"***VOICE PROCESSOR PROGRAM***
0070: PRINT
0080: POKE 135,12
0090: DATA 169,00,141,67,232,133,54,133
0100: DATA 56,168,169,08,133,55,169,12
0110: DATA 133,58,169,12,133,57,169,30
0120: DATA 133,59,78,79,232,38,54,198
0130: DATA 55,165,55,240,11,165,58,170
0140: DATA 202,138,208,252,234,76,84,03
0150: DATA 234,165,54,145,56,234,200,169
0160: DATA 08,133,55,152,208,220,230,57
0170: DATA 165,57,197,59,208,212,96,00
0180: DATA 169,255,141,67,232,169,00,133
0190: DATA 56,168,169,08,133,55,169,12
0200: DATA 133,58,169,12,133,57,169,30
0210: DATA 133,59,177,56,141,79,232,234
0220: DATA 165,58,170,202,138,208,252,234
0230: DATA 198,55,165,55,240,07,14,79
0240: DATA 232,234,76,162,03,200,177,56
0250: DATA 141,79,232,169,08,133,55,152
0260: DATA 208,222,230,57,165,57,197,59
0270: DATA 208,214,96,234,234,234,234,234
0280: DATA 169,00,141,67,232,173,79,232
0290: DATA 36,60,208,244,76,58,03,234
0300: A=826
0310: FOR I=1 TO 168
0320: READ D%
0330: POKE A,D%
0340: A=A+1
0350: NEXT I
0360: PRINT"******************************
0370: PRINT" PRESS R FOR RECORD"
0380: PRINT" PRESS P FOR PLAYBACK"
0390: PRINT" PRESS S FOR SAVE"
0400: PRINT" PRESS L FOR LOAD"
0410: PRINT"******************************
0420: GET C$:IF C$="" GOTO 420
0430: IF C$="R" GOTO 600
0440: IF C$="P" GOTO 700
0450: IF C$="S" GOTO 800
0460: IF C$="L" GOTO 900
0470: GOTO 420
```

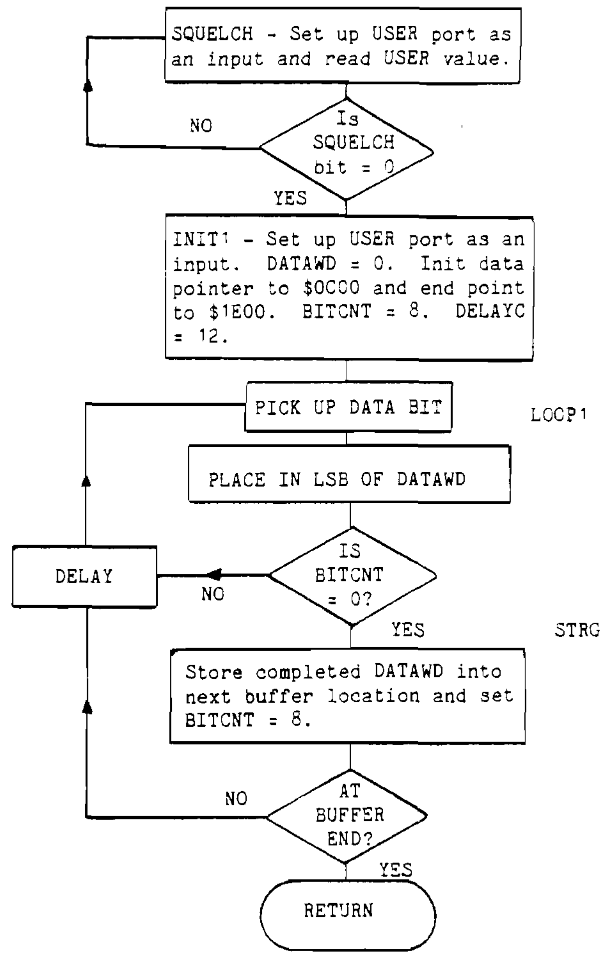


Figure 3: Speech processor flowchart

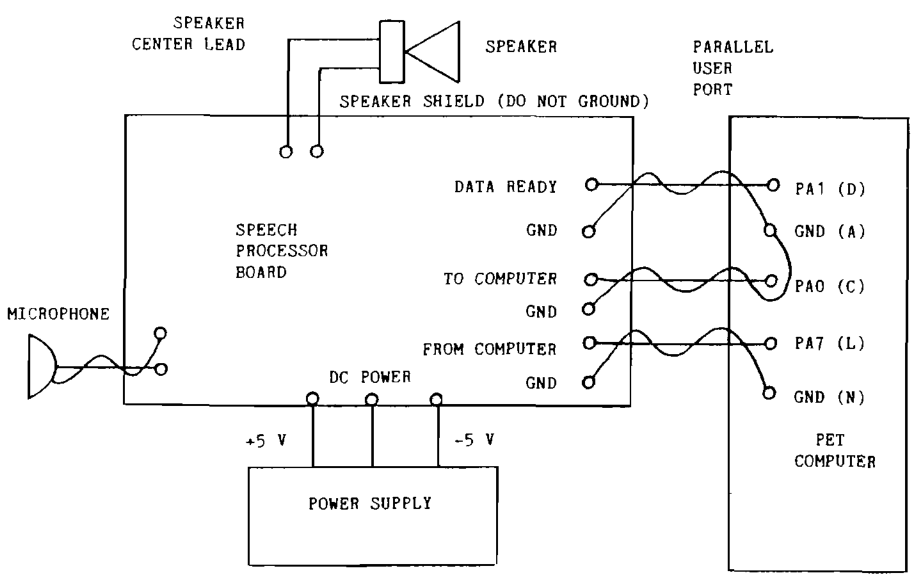


Figure 1: Speech processor components

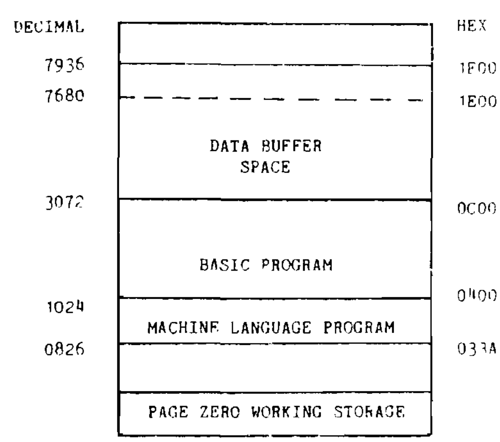


Figure 2: Speech processor memory map

ing variables between the machine language programs and their BASIC counterparts.

The Record Program

The RECORD process is entered by pressing "R". This action causes the pointing vector corresponding to the beginning address of the RECORD machine language program to be placed into locations 1 and 2. A value of 02 is also placed in decimal location 60, which is the squelch mask value to be used in the machine language routine. The machine language program is then entered by executing the USR instruction.

A flow diagram of the RECORD Program is shown in Figure 3 and a machine language listing of the process is also given. After initialization, the program loops waiting for the DATA READY line to go low. This action occurs when the amplitude of the voice level exceeds the squelch threshold set on the speech processor board. Once the squelch level is sensed, the program proceeds and the record program initialization commences.

A machine language listing of the SQUELCH Process is shown. If the user wishes not to implement the squelch test, the values in line 610 of the program can be changed to:

```
610   POKE 01, 58   : POKE 02,03
```

and the record program will be entered directly.

After initialization the record process beings. The value of TO COMPUTER line is sampled at PA0 of the user's port and stored into the LSB of the buffer location DATAWD. A counter (BITCNT) is then tested to see if 8 samples have been sensed. If 8 samples have not yet been sensed, DATAWD is shifted left one place and a delay loop is executed before the value of the next bit is sampled. When a full byte of data has been received, the byte is stored away in the next memory location. The values of DATAWD and BITCNT are reinitialized. A short delay loop is executed and the process is repeated.

At the time that DATAWD is stored away the location into which it is being stored is checked against an upper bound pointer in memory. When the two address correspond, the process has run out of available buffer space for the record process and the machine language routine returns control to the BASIC program. Completion of the RECORD process is indicated by the line "RECORD PROCESS COMPLETED" appearing on the display screen.

Playback Program

The PLAYBACK program is entered by pressing "P". This action forces the pointing vector corresponding to the beginning address of the playback machine language program to be placed into locations 1 and 2. The machine language program is then entered by the execution of the USR instruction. A flow diagram of the PLAYBACK program is shown in Figure 4. A machine language listing of the PLAYBACK program is also given.

The PLAYBACK program repeats the same basic process developed in the record process. As each new byte is retrieved from the buffer memory, the state of the most significant bit is outputted to the speech processor unit. After a finite delay, the DATAWD is shifted left one position and the state of the new MSB is sent to the speech processor. When all the bits have been examined, the next byte in buffer memory is retrieved. When all of the bytes in the data buffer memory have been examined, the PLAYBACK process is completed and the message "PLAYBACK PROCESS COMPLETED" appears on the monitor screen.

Save Process

The SAVE process is a BASIC program written to allow the user to dump portions of the buffer memory on tape for later use. The process is entered by pressing "S". A prompting message asks the user to enter the desired starting address and ending address in buffer memory to be saved. The contents of the memory locations between the two selected locations is then written on tape and upon completion of this operation the message "SAVE PROCESS COMPLETED" appears on the monitor.

Load Process

The LOAD program is also a BASIC routine designed to load into memory a tape prepared by the SAVE program. To enter the LOAD process the user presses the "L" key. A message will prompt the user to enter the starting and ending address of the data file to be stored. When the LOAD process is completed the message "LOAD PROCESS COMPLETED" will appear on the screen.

Typical Application

For an illustration of how these programs might be employed, let's assume the user wants to have his computer automatically dial up telephone numbers. Using the speech processor and the RECORD process, each dual tone multiple frequency (DTMF) output is recorded from a standard touch tone telephone. As each tone is recorded, a data file can be written using the SAVE program. The starting locating for each tone would be the beginning of buffer memory. The ending address could be set at the beginning of memory plus, say 200 bytes. After all ten tones have been recorded, the data files collected by the SAVE program can be stacked consecutively on

```
600   REM..RECORD MODE......
610   POKE 01,210: POKE 02,03
615   POKE 60,02
620   LET X=USR(R)
630   PRINT" RECORD PROCESS COMPLETED"
640   GOTO 420
700   REM..PLAYBACK MODE......
710   POKE 01,130: POKE 02,03
720   LET X=USR(R)
730   PRINT" PLAYBACK PROCESS COMPLETED"
740   GOTO 420

800   PRINT"**SAVE PROCESS INITATED"
805   INPUT" FILE NAME"; N$
810   INPUT" INPUT STARTING ADDRESS"; S
820   INPUT" INPUT ENDING ADDRESS"; E
825   POKE 243,122: POKE 244,02
830   OPEN 1,1,1,N$
840   FOR I=0 to (E-S)
850   PRINT #1,PEEK(S+I)
860   NEXT I
870   CLOSE 1
880   PRINT"**SAVE PROCESS COMPLETED**"
890   GOTO 420

900   PRINT"**LOAD PROCESS INITATED"
905   INPUT" FILE NAME"; N$
910   INPUT" INPUT STARTING ADDRESS"; S
920   INPUT" INPUT ENDING ADDRESS"; E
925   POKE 243,122:POKE 244,02
930   OPEN 1,1,0,N$
940   FOR I=0 to (E-S)
950   INPUT #1,A
960   POKE(S+I),A
970   NEXT I
980   CLOSE 1
990   PRINT"**LOAD PROCESS COMPLETED**"
995   GOTO 420
1000  END
```

200 byte boundaries using the LOAD program. We would now have a data base in buffer storage with each tone starting and ending on a known boundary.

In order to now dial any number, a small BASIC program would be required to call the PLAYBACK program with the appropriate starting boundary and ending boundary addresses in the required sequence. The resulting tones developed through the speech processor would then be acoustically coupled to the telephone to complete the process.

Conclusions

This paper was designed to illustrate how a low cost speech processor might be interfaced with a PET Computer. However, the same machine language software could be used to interface the device to any 6502 based processor with only slight modifications. The use of BASIC in this application provided an easy method of mechanizing the man-machine interface. The application of voice or sound feedback in computing is almost limitless and it is hoped that this article illustrates one method of achieving this goal. μ

```
03CD           DATAWD *     $0036
03CD           BITCNT *     $0037
03CD           WDB    *     $0038
03CD           DLYCNT *     $003A
03CD           ENDBUF *     $003B
03CD           MASK   *     $003C
03CD           PADD   *     $E843
03CD           PAD    *     $E84F
033A                  ORG   $033A
033A A9 00     INIT1  LDAIM $00    SET UP DIRECTION REG
033C 8D 43 E8         STA   PADD   AS AN INPUT
033F 85 36            STA   DATAWD DATAWD = 0
0341 85 38            STA   WDB
0343 A8               TAY
0344 A9 08            LDAIM $08
0346 85 37            STA   BITCNT BITCNT = 8
0348 A9 0C            LDAIM $0C
034A 85 3A            STA   DLYCNT DLYCNT = 12
034C A9 0C            LDAIM $0C
034E 85 39            STA   WDB    +01 WDB+1 = 12
0350 A9 1E            LDAIM $1E
0352 85 3B            STA   ENDBUF ENDBUF = $1E
0354 4E 4F E8  LOOP1  LSR   PAD    PICK UP DATA BIT
0357 26 36            ROL   DATAWD STORE IN LSB OF DATAWD
0359 C6 37            DEC   BITCNT
035B A5 37            LDA   BITCNT
035D F0 0C            BEQ   STRG1
035F A5 3A     DLY1   LDA   DLYCNT DELAY FOR 8KHZ RATE
0361 AA               TAX
0362 CA               DEX
0363 8A               TXA
0364 D0 FC            BNE   DLY1   +03
0366 EA               NOP
0367 4C 54 03         JMP   LOOP1
036A EA               NOP
036B A5 36     STRG1  LDA   DATAWD
036D 91 38            STAIY WDB
036F EA               NOP
0370 C8               INY
0371 A9 08            LDAIM $08
0373 85 37            STA   BITCNT
0375 98               TYA
0376 D0 DC            BNE   LOOP1
0378 E6 39            INC   WDB    +01
037A A5 39            LDA   WDB    +01
037C C5 3B            CMP   ENDBUF
037E D0 D4            BNE   LOOP1
0380 60               RTS
0381 EA               NOP

03D2                  ORG   $03D2
03D2 A9 00     SQUELC LDAIM $00
03D4 8D 43 E8         STA   PADD
03D7 AD 4F E8         LDA   PAD
```

```
03DA 24 3C              BIT   MASK
03DC D0 F4              BNE   SQUELC
03DE 4C 3A 03           JMP   INIT1
03E1 EA                 NOP

0382                    ORG   $0382
0382 A9 FF      INIT2   LDAIM $FF    SET UP DIRECTION REGISTER
0384 8D 43 E8           STA   PADD   AS OUTPUT
0387 A9 00              LDAIM $00
0389 85 38              STA   WDB
038B A8                 TAY
038C A9 08              LDAIM $08
038E 85 37              STA   BITCNT BITCNT = 8
0390 A9 0C              LDAIM $0C
0392 85 3A              STA   DLYCNT DLYCNT = 12
0394 A9 0C              LDAIM $0C
0396 85 39              STA   WDB    +01 WDB+1 = 10
0398 A9 1E              LDAIM $1E
039A 85 3B              STA   ENDBUF ENDBUF = 1E
039C B1 38              LDAIY WDB    PICK UP WORD FROM STORAGE
039E 8D 4F E8           STA   PAD    AND PLACE IN OUTPUT REG
03A1 EA                 NOP
03A2 A5 3A      DLY2    LDA   DLYCNT DELAY TO ESTABLISH
03A4 AA                 TAX          8KHZ RATE
03A5 CA                 DEX
03A6 8A                 TXA
03A7 D0 FC              BNE   DLY2   +03
03A9 EA                 NOP
03AA C6 37      LOOP2   DEC   BITCNT
03AC A5 37              LDA   BITCNT
03AE F0 07              BEQ   STRG2
03B0 0E 4F E8           ASL   PAD
03B3 EA                 NOP
03B4 4C A2 03           JMP   DLY2
03B7 C8         STRG2   INY
03B8 B1 38              LDAIY WDB
03BA 8D 4F E8           STA   PAD
03BD A9 08              LDAIM $08
03BF 85 37              STA   BITCNT
03C1 98                 TYA
03C2 D0 DE              BNE   DLY2
03C4 E6 39              INC   WDB    +01
03C6 A5 39              LDA   WDB    +01
03C8 C5 3B              CMP   ENDBUF
03CA D0 D6              BNE   DLY2
03CC 60                 RTS


SYMBOL TABLE 2000 2066
BITCNT 0037   DATAWD 0036   DLYCNT 003A   DLYQ   035F
DLYR   03A2   ENDBUF 003B   INITQ  033A   INITR  0382
LOOPQ  0354   LOOPR  03AA   MASK   003C   PADD   E843
PAD    E84F   SQUELC 03D2   STRGQ  036B   STRGR  03B7
WDB    0038
```

Tiny PILOT: An Educational Language for the 6502

PILOT is a higher level language used for computer aided instruction. This version includes an editor and an interpreter. It requires fewer than 800 bytes of memory.

Nicholas Vrtis
5863 Pinetree S.E.
Kentwood, MI 49508

```
**************************************************************************
* CHAR    * EDIT FUNCTION
**************************************************************************
*         * START EXECUTION OF THE PILOT PROGRAM
* UPARROW* MOVE EDIT POINTER TO START OF PROGRAM
* /       * DISPLAY NEXT LINE OF THE PROGRAM
* %       * PAD TO END OF LINE WITH DELETE CHARACTERS
* B/S     * BACKSPACE TO CORRECT TYPING ERROR
* C/R     * CARRIAGE RETURN - INDICATE END OF STATEMENT
* ANY     * CHARACTER IS STORED IN PROGRAM (MAX 127 PER LINE)
**************************************************************************
* FORMAT * STATEMENT                * WHAT IT DOES
**************************************************************************
* T:TEXT * TYPE                     * DISPLAY THE TEXT ON THE TERMINAL
*        *                          *
* A:     * ACCEPT                   * INPUT UP TO 40 CHARACTERS INTO
*        *                          * ANSWER FIELD
* ?:     * ACCEPT NAME              * INPUT UP TO 40 CHARACTERS INTO
*        *                          * NAME AND ANSWER FIELD.
* M:TEXT * MATCH                    * COMPARE TEXT TO LAST IMPUT FROM
*        *                          * TERMINAL AND SET MATCH FLAG TO
*        *                          * Y IF EQUAL, N IF NOT EQUAL.
* J:N    * JUMP                     * JUMP TO LABEL N FOR NEXT LINE.
*        *                          * J:A MEANS JUMP TO LAST ACCEPT.
*        *                          * J:* MEANS RESTART FROM BEGINNING.
* U:N    * USE SUBROUTINE N         * SAVE ADDRESS OF START OF NEXT
*        *                          * LINE AND THEN PERFORM AS IN JUMP.
* E:     * EXIT FROM SUBROUTINE     * RETURN TO ADDRESS SAVED BY PRIOR
*        *                          * USE STATEMENT.
* S:     * STOP                     * STOP PROGRAM AND RETURN TO EDITOR
*        *                          *
* C:     * COMPUTE                  * PERFORMS ARITHMETIC ON VARIABLES
*        *                          * NAMED A THROUGH Z. ALLOWED
*        *                          * OPERATIONS ARE =, +, AND -
*        *                          * RANGE IS + OR - 999
*        *                          * C:$= WILL PLACE RESULT IN ANSWER
*        *                          * FIELD INSTEAD OF A VARIABLE
* R:     * REMARKS                  * PROGRAM REMARKS - NOT EXECUTED
*        *                          *
*        * CONDITIONALS             * MAY PRECEED ANY STATEMENT.
* N      *                          * EXECUTE ONLY IF MATCH FLAG IS N
* Y      *                          * EXECUTE ONLY IF MATCH FLAG IS Y
*        *                          *
* *N     * LABEL                    * MAY PRECEED ANY STATEMENT OR
*        *                          * CONDITIONAL.  ACTS AS DESTINATION
*        *                          * FOR A JUMP OR USE STATEMENT
* $X     * VARIABLE ITEM            * AS PART OF TEXT CAUSES CONTENTS
*        *                          * OF VARIABLES TO BE DISPLAYED OR
*        *                          * MATCHED.
*        *                          * $? INDICATES NAME FIELDS.
**************************************************************************
```

Are you envious of the guys on your block who have big BASIC systems? Have you ever tried to teach machine language to someone who thinks HEX is an evil spell? I had the same problem until I discovered PILOT, and implemented a small version on my SYM-1. For those who haven't heard of PILOT yet, it is an educational, high level language intended for computer aided instruction. It is a very simple language, with only ten basic instructions, but it incorporates a number of features that make it easy enough to use as a method for introducing people to computers. I have written some math drill programs for my six- and eight-year olds, and in turn, my eight-year old loves to write programs for her little brother to run.

This implementation of PILOT is not a full "standard" version. After all, what do you expect from an interpeter and editor that run in less than 800 bytes? I also could not resist the temptation to change things a little here and there. It is close enough to give a flavor for what PILOT can do, and it makes a nice language to have fun with, even on a 2K system.

The editor performs only the most elementary functions required to get a program in and running. It accepts characters without checking syntax rules, the only limitation being that each line is a maximum of 127 characters long. I compromised at 127, instead of 80, because the sign of the index register changes at 128, and so I avoided a compare.

The program looks for the ASCII back-space character, hex 08, because my CRT actually backspaces. If your terminal doesn't, you might want to change this to a printable character such as the underscore used by many timesharing systems. A check is also made for the backspace in the code for the ACCEPT statement, so be sure to change it there as well.

The editor doesn't have a provision for inserting a line between existing lines, but it is possible to change a line, provided you replace it with one of the same length or shorter. The percent key fills from the current position to the next end of line with delete characters, hex FF. Since most terminals ignore these, it works effectively as a delete to the end of the line. The program has to check for these during MATCH and COMPUTE statement processing, since they represent the logical end of line.

The carriage return, entered as the end of line, is converted to a zero by the editor. This simplifies looking for the end of each line, later on, since the zero flag is set as the byte gets loaded. The SYM monitor routine CRLF outputs both the carriage return and the line feed, so one doesn't save anything by keeping the return in the line to output it.

The locations CURAD and CURAD + 1 address the start of each PILOT line. Initially, this is set to $500 by the routine SETBGN. The Y register is incremented o access the next character in the line. At the end of each line, subroutine SCURAD bumps Y one more time to get past the end of line character, and then adds the resulting Y value to the current address and resets Y to zero.

This sets things up for the start of the next line. Performing the line scan in this way saves two bytes each time I need to get to the next character because an INY is used instead of a JSR, and it also makes it easy to check for a line too


```
                   *
                   * PAGE ZERO DATA REFERENCES
                   *
                   LST    *      $0000  ADDRESS OF LAST ACCEPT COMMAND
                   FLG    *      $0002  CURRENT YES/NO FLAG
                   CHRS   *      $0003  ALLOW 40 BYTES OF INPUT
                   NAME   *      $002B  VARIABLE AREAS - 2 BYTES EACH
                   VARIBS *      $0053  VARIABLE AREAS - 2 BYTES EACH
                   IFLAG  *      $0087  SPECIAL INDICATOR FLAG AREA
                   HOLDY  *      $0088  HOLD AREA FOR Y VALUE
                   WORK   *      $0089  TEMP WORK VARIABLE
                   RESULT *      $008B  RESULT HOLD AREA FOR COMPUTATIONS
                   ANSX   *      $008D  HOLD AREA FOR ANSWER INDEX POINTER
                   SIGNIF *      $008E  SIGNIFICANCE INDICATOR
                   OPRATN *      $008F  LAST OPERATION IN COMPUTE STATEMENT
                   NUMDSP *      $0090  DISPLAY VARIABLE BUILD AREA
                   RETURN *      $0095  JUMP RETURN ADDRESS
                   CURAD  *      $0097  ADDRESS OF START OF CURRENT LINE

                   CR     *      $0D    CARRIAGE RETURN CODE
                   *
                   * EXTERNAL ADDRESS REFERENCES
                   *
                   CRLF   *      $834D  OUTPUT A CR AND LF
                   INCHR  *      $8A1B  INPUT ONE CHARACTER
                   OUTCHR *      $8A47  OUTPUT ONE CHARACTER

                          ORG    $0200
                   *
                   * START OF THE EDITOR PORTION
                   *
0200 A9 80         START  LDAIM  $80    SET MODE TO EDIT FOR "PRT" ROUTINE
0202 85 87                STA    IFLAG
0204 20 83 04             JSR    SETBGN SET UP STARTING DATA AREA ADDRESS
                   *
                   * HERE IS THE START OF EACH NEW LINE
                   *
0207 A9 3E         ELINE  LDAIM  $3E    OUTPUT A ">" PROMPT CHARACTER
0209 20 47 8A             JSR    OUTCHR
                   *
                   * HERE IS WHERE EACH INPUT CHARACTER IS OBTAINED
                   *
020C 20 1B 8A      EGET   JSR    INCHR
020F AA                   TAX           CHECK FOR NULLS AND IGNORE
0210 F0 FA                BEQ    EGET   SO THEY DON'T GET CONFUSED WITH EOL

0212 C9 5E                CMPIM  $5E    IS IT AN UPARROW?
0214 F0 EA                BEQ    START  YES - START AT BEGINNING AGAIN

0216 C9 40                CMPIM  $40    IS IT "AT" SYMBOL FOR EXECUTE REQUEST?
0218 F0 38                BEQ    EXEC   YES - GO START ON THAT

021A C9 08                CMPIM  $08    IS IT A BACKSPACE?
021C D0 06                BNE    TRYDSP NO - GO CHECK FOR DISPLAY REQUEST

021E 88                   DEY           YES - BACK UP ONE CHARACTER
021F 10 EB                BPL    EGET   BUT CHECK FOR PAST START OF LINE
0221 C8                   INY           HE BACKED UP TOO FAR - DISALLOW
0222 10 E8                BPL    EGET   UNCONDITIONAL

0224 C9 2F         TRYDSP CMPIM  $2F    IS IT "/" FOR DISPLAY LINE REQUEST?
0226 D0 05                BNE    TRYREP NO - CHECK FOR REPLACEMENT REQUEST
                   *
                   * DISPLAY TO THE NEXT CARRIAGE RETURN
                   *
0228 20 21 04             JSR    PRT    PRINT THE LINE
022B B0 DA                BCS    ELINE  UNCONDITIONAL

022D C9 25         TRYREP CMPIM  $25    IS IT "%" REQUEST TO PAD A LINE?
022F D0 0E                BNE    CHAR   NO - MUST BE DATA CHARACTER
                   *
                   * PAD THE LINE FROM CURRENT LOC TO EOL WITH DELETE CHAR
                   *
0231 B1 97         PADLOP LDAIY  CURAD  GET CURRENT CHARACTER
0233 F0 18                BEQ    SETNL  IF ZERO, WE ARE DONE
0235 A9 FF                LDAIM  $FF    ELSE MAKE IT A DELETE CHAR
0237 91 97                STAIY  CURAD
0239 C8                   INY           BUMP TO NEXT CHARACTER
023A 10 F5                BPL    PADLOP LOOP IF HAVEN'T DONE 128
023C 88                   DEY           LINE IS TOO LONG - BACK UP ONE
023D A9 0D                LDAIM  CR     FORCE IN AN EOL HERE
                   *
                   * IT WASN'T AN EDIT CHARACTER - MUST BE DATA TO SAVE
                   *
```

```
023F C9 0D     CHAR   CMPIM CR     IS IT CARRIAGE RETURN AS EOL?
0241 D0 02            BNE   CHAR1  SKIP AHEAD IF NOT
0243 A9 00            LDAIM $00    ELSE CONVERT CR TO ZERO AS EOL
0245 91 97     CHAR1  STAIY CURAD  PUT IT AWAY
0247 F0 04            BEQ   SETNL  BRANCH IF YES
0249 C8               INY          ELSE BUMP TO SET UP FOR NEXT ONE
024A 10 C0            BPL   EGET   AND GO GET IT IF STILL ROOM ON LINE
024C 88               DEY          ELSE POINT BACK TO LAST CHAR & FALL THRU

024D 20 57 04  SETNL  JSR   LINEND DO CR/LF AND FIX UP CURAD
0250 B0 B5            BCS   ELINE  GO START A NEW LINE
               *
               * EXECUTION PORTION BEGINS HERE
               *
0252 20 4D 83  EXEC   JSR   CRLF   EXTRA BLANK LINE AFTER EDITOR

0255 20 83 04  RESTRT JSR   SETBGN HERE IF FROM J:*
0258 A2 33            LDXIM $33    ZERO VARIABLE ZREAS
025A A9 00            LDAIM $00
025C 85 96            STA   RETURN +01
025E 95 53     RESTR1 STAX  VARIBS
0260 CA               DEX
0261 10 FB            BPL   RESTR1

0263 B1 97     LSTART LDAIY CURAD  GET CHARACTER FROM THE LINE
0265 C9 2A            CMPIM $2A    CHECK FOR "*" LABEL MARKER
0267 D0 04            BNE   CHKCON IF NOT - GO CHECK FOR CONDITIONAL
0269 C8               INY          OTHERWISE SKIP PAST THE "*"
026A C8        SKPNXT INY          SKIP PAST THE NEXT CHARACTER
026B D0 F6            BNE   LSTART UNCONDITIONAL
               *
               * FLAG DEPENDENT PROCESSING HERE
               *
026D C9 59     CHKCON CMPIM $59    CHECK FOR "Y" REQUEST
026F F0 04            BEQ   TFLAG  BRANCH IF YES
0271 C9 4E            CMPIM $4E    IF NOT - CHECK FOR "N" REQUEST
0273 D0 09            BNE   STRTST BRANCH IF NEITHER
               *
               * SEE IF CONDITIONAL MATCHES FLAG
               *
0275 C5 02     TFLAG  CMP   FLG    SEE IF THEY MATCH
0277 F0 F1            BEQ   SKPNXT SKIP TO NEXT CHAR & EXECUTE LINE
               *
               * NO MATCH - SKIP THIS STATEMENT
               *
0279 20 5A 04  FWD    JSR   FWD1   USE THIS SUBROUTINE
027C B0 E5            BCS   LSTART UNCONDITIONAL

027E 85 87     STRTST STA   IFLAG  THIS WILL CLEAR HIGH BIT FOR EDITOR
0280 C8               INY          POINT TO THE ":" CHAR
0281 C8               INY          AND TO THE FOLLOWING CHARACTER
               *
               * ENTER NAME STATEMENT
               *
0282 C9 3F     XQUEST CMPIM $3F    IS IT "?" FOR ENTER NAME?
0284 D0 05            BNE   XA     BRANCH IF NOT
0286 38               SEC          TURN HIGH ORDER BIT ON TO INDICATE
0287 66 87            ROR   IFLAG  PROCESSING NAME COMMAND
0289 D0 0C            BNE   TAKEIN NOW USE THE ACCEPT LOGIC
               *
               * ACCEPT STATEMENT
               *
028B C9 41     XA     CMPIM $41    SEE IF HAVE ACCEPT STATEMENT
028D D0 34            BNE   XC     BRANCH IF NOT
028F A5 97            LDA   CURAD  SAVE ADDRESS OF THE "A" STATEMENT
0291 85 00            STA   LST    NOTE: WILL INCLUDE CONDITIONALS
0293 A5 98            LDA   CURAD  +01
0295 85 01            STA   LST    +01


0297 A9 3F     TAKEIN LDAIM $3F    DISPLAY "?" PROMPTING CHARACTER
0299 20 47 8A         JSR   OUTCHR

029C A2 27            LDXIM $27    CHRS GETS STORED BACKWARDS
029E 20 1B 8A  ACHR   JSR   INCHR  GET AN INPUT CHARACTER
02A1 C9 08            CMPIM $08    IS IT A BACKSPACE?
02A3 D0 03            BNE   ACHR1  BRANCH IF NOT
02A5 E8               INX          ELSE FORGET ABOUT LAST CHARACTER IN
02A6 D0 F6            BNE   ACHR   UNCONDITIONAL
02A8 C9 0D     ACHR1  CMPIM CR     WAS IT A CARRIAGE RETURN?
02AA D0 02            BNE   ACHR2  NO - SKIP AHEAD
02AC A9 00            LDAIM $00    YES - CONVERT CR TO END OF LINE
02AE 95 03     ACHR2  STAX  CHRS   AND SAVE IT FOR MATCH STATEMENT
02B0 24 87            BIT   IFLAG  SEE IF GETTING NAME FIELD
```

long. If Y is minus after it has been incremented, more than 128 characters have gone by since the start of the line.

The editor inserts an end of line at this point and continues on. If this occurs during line print or scan for end of line, it probably means that the PILOT program has gone off the end, so these routines branch to SETBGN to start at the beginning again. This does not prevent the PILOT program from looping while looking for an undefined label, but it does prevent printing some garbage.

The first character on a line is not necessarily useful for executing a PILOT statement. There might be a line feed or some other control character present there. The asterisk and the label are not used except as a destination for a USE or JUMP statement. If we do find one of these, we not only need to skip it, but we must also skip the next character, since that is the label. The routine SKPJNK takes care of skipping over everything but the asterisk, since the same routine is used by both normal command start and by the label search routine.

Once the program has searched out the first probable command character on the line, the next thing it has to do is look for a conditional flag. This will determine whether it must examine the rest of the line. A "Y" or an "N" is a conditional, and if the character of one of these lines, it is checked against the current value in FLG. If they do match, the program simply increments Y to point to the following character, and also starts again, but this time Y is pointing to the operation code following the conditional.

Most of the other operations execute in a similar manner. They look at the current character in A, do their processing if it is their turn, or branch to the next routine if it isn't theirs. There are some exceptions to this (naturally). The TEXT command is last because, if the character isn't a valid statement, the whole line must be printed anyway. One of the other exceptions is the processing for ENTER NAME (?:) and ACCEPT statements, which share much of the same code. Another is the code for JUMP and USER statements, which also share common code.

Logically, the only difference between the "?:" statement and the "A:" statement is that the "?:" inputs characters into both CHRS and into NAME, while the "A:" saves the starting address of the line for use in "J:A" (jump to last accept) processing. In fact, the processing of the ENTER NAME statement merely involves setting the high order bit of IFLAG on and skipping the save of the line address that the ACCEPT statement performs. The high order bit of IFLAG is normally turned off by storing the ASCII command character in it. The code for the ACCEPT statement checks the high order bit of IFLAG and stores the input character in NAME if the bit is on.

One thing to note is that data saved in NAME and CHRS are stored backwards, with the first input character in CHRS + 39, the second in CHRS + 38, etc. Since I have to initialize the X register anyway, I could initialize it with zero and count up, or with 39 and count down. If I am counting up, though, I need to do a compare to see if I have reached the maximum value. If I am counting down, the minus flag will automatically set when I reach the end.

The COMPUTE statement uses decimal arithmetic. Each variable is two bytes long, with the high order first. The high order decimal digit (bits 0-3 of the first byte) are used to indicate the sign. A value of 8 or 9 indicates a negative number, while anything else is considered positive. It works out to be tens' complement arithmetic. To illustrate, assume I want to calculate 1 minus 2, which everybody knows is −1. The actual result from the decimal subtract is $9999, much as it would be $FFFF in binary.

In order to display this as −1, we have to subtract $9999 from zero to get $0001. Using decimal arithmetic does have some disadvantages, particularly the fact that the range of numbers is −2000 to +7999 ($8000 to $7999) for two bytes instead of −32768 to +32767 for binary. Another disadvantage is that INC is not a decimal instruction.

The primary advantage of using decimal mode is the ease of translating from ASCII to internal and back. The ASCII characters zero through nine are $30 through $39 in hex. Multiplying by 10 in order to accept the next digit into a number is also very easy, since it only requires a four bit shift left. Converting to display merely means shifting each digit to the low order four bits. ANDing off the high order part, and ORing in $30.

The MATCH statement is the most complicated statement apart from COMPUTE. In theory, all that has to be done is compare the characters in CHRS against those in the MATCH statement line, and then set FLG to Y if they match, and to N if they don't. This works fine if they match. The problems come when they are different. Before the flag gets set to N, we have to determine why they did not match.

For one thing, it might be the end of the MATCH statement line. Since all the characters up to that point have matched, the program treats this condition as a complete match. PILOT uses the comma as a seperator in the match statement to indicate alternate possible matches, so if the mismatch character is a comma, it is treated as the end of line, and FLG is set to Y.

```
02B2 10 02           BPL    ACHR3  BRANCH IF NOT
02B4 95 2B           STAX   NAME   ELSE SAVE IN NAME FIELD ALSO
02B6 C9 00    ACHR3  CMPIM  $00    IS IT DONE YET?
02B8 F0 03           BEQ    ADONE  BRANCH IF HE HAS SIGNALLED END
02BA CA              DEX           ELSE BUMP FOR NEXT INPUT
02BB 10 E1           BPL    ACHR   AND GO GET IT IF ROOM STILL LEFT
02BD 20 4D 83 ADONE  JSR    CRLF   DO CR/LF TO LET GUY KNOW
02C0 4C 79 02        JMP    FID
              *
              * COMPUTE STATEMENT
              *
02C3 C9 43    XC     CMPIM  $43    IS IT A "C" FOR COMPUTE?
02C5 F0 03           BEQ    XC1    BRANCH IF IT IS
02C7 4C 56 03        JMP    XM     ELSE LONG JUMP TO TEST FOR M
02CA 20 94 04 XC1    JSR    GETIDX GET INDEX POINTER TO RESULT
02CD 86 8D           STX    AIDX   SAVE IT FOR NOW
02CF A9 00           LDAIM  $00    CLEAR RESULT
02D1 85 8B           STA    RESULT
02D3 85 8C           STA    RESULT +01
02D5 C8              INY           POINT TO "="
02D6 A2 2B           LDXIM  $2B    SET 1ST OPERATION TO "+" FOR ADD
02D8 D0 4A           BNE    OPWRAP GO SAVE & SET UP WORK AREA
              *
              * LOOP FOR EACH NEW CHARACTER IN COMPUTE PROCESSING
              *
02DA C8       CMPLOP INY           BUMP TO NEXT CHARACTER
02DB B1 97           LDAIY  CHEAD  GET A CHARACTER
02DD 30 20           BMI    ISOPR  MINUS IS DELETE/ALSO LAST "OPERATOR"
02DF C9 2F           CMPIM  $2F    IS IT "/" FOR AN OPERATION SPECIFIED?
02E1 90 1C           BCC    ISOPR  BRANCH IF YES
02E3 C9 3A           CMPIM  $3A    IF NOT - IS IT ":" FOR A NUMBER?
02E5 B0 12           BCS    NOTNMB BRANCH IF NOT - MUST BE A VARIABLE

02E7 29 0F           ANDIM  $0F    CONVERT NUMBER TO BINARY
02E9 6A              RORA          SPIN TO HIGH ORDER PART OF A
02EA 6A              RORA
02EB 6A              RORA
02EC 6A              RORA          LEAVE BIT 3 IN CARRY
02ED A2 04           LDXIM  $04    4 BITS TO ROLL INTO WORK
02EF 26 8A    BITROL ROL    WORK   +01 RIPPLE CARRY INTO WORK
02F1 26 89           ROL    WORK   FOR 16 BITS
02F3 0A              ASLA          PUT NEXT BIT INTO CARRY
02F4 CA              DEX           COUNT ONE JUST DONE
02F5 D0 F8           BNE    BITROL CONTINUE IF MORE TO GO
02F7 F0 E1           BEQ    CMPLOP ELSE GET NEXT CHARACTER (DIGITS)

02F9 20 9C 04 NOTNMB JSR    VTRANS TRANSFER VARIABLE TO WORK AREA
02FC 4C DA 02        JMP    CMPLOP GO GET NEXT CHARACTER (OPERATION?)
              *
              * GOT AN OPERATION - FIRST PERFORM PREVIOUS REQUEST
              *
02FF F8       ISOPR  SED           SET TO DECIMAL MODE
0300 AA              TAX           SAVE NEW OPERATION IN X FOR NOW
0301 A5 8F           LDA    OPRATN GET PREVIOUS OPERATION
0303 C9 2D           CMPIM  $2D    WAS IT A "-" FOR SUBTRACT?
0305 F0 10           BEQ    OPMNUS BRANCH IF YES
0307 18              CLC           ALL OTHERS ASSUME IT IS ADD
0308 A5 8A           LDA    WORK   +01
030A 65 8C           ADC    RESULT +01
030C 85 8C           STA    RESULT +01
030E A5 89           LDA    WORK
0310 65 8B           ADC    RESULT
0312 85 8B           STA    RESULT
0314 4C 24 03        JMP    OPWRAP GO WRAP UP THE OPERATION

0317 38       OPMNUS SEC           SUBTRACTION
0318 A5 8C           LDA    RESULT +01
031A E5 8A           SBC    WORK   +01
031C 85 8C           STA    RESULT +01
031E A5 8B           LDA    RESULT
0320 E5 89           SBC    WORK
0322 85 8B           STA    RESULT

0324 D8       OPWRAP CLD           GET OUT OF DECIMAL MODE
0325 86 8F           STX    OPRATN SAVE NEW OPERATION
0327 8A              TXA           DO TRANSFER TO CHECK FOR "00"/"FF"
0328 F0 0A           BEQ    CMPDON DONE IF IT WAS ZERO (EOL)
032A 30 08           BMI    CMPDON OR DELETE CHARACTERS (FROM FILLING)

032C A9 00           LDAIM  $00    ELSE CLEAR WORK AREA FOR NEXT ONE
032E 85 89           STA    WORK
0330 85 8A           STA    WORK   +01
0332 F0 A6           BEQ    CMPLOP AND GO DO NEXT CHARACTER
```

```
0334 A6 8D     CMPDON LDX    ANSX   GET INDEX TO RESULT
0336 10 13            BPL    TOVRIB PLUS IS NORMAL INDEX TO A VARIABLE

0338 A2 38            LDXIM $38     ELSE FUDGE INDEX FOR "FROM" RESULT
                                         USING "RESULT - VARIBS"
033A 20 9F 04         JSR    VTRANS +03 MOVE RESULT TO WORK AREA
033D 20 AB 04         JSR    CNVDSP +03 CONVERT IT TO DISPLAY FORM
0340 A2 04            LDXIM $04     TRANSFER DISPLAY TO ANSWER AREA
0342 B5 90     TALOOP LDAX   NUMDSP
0344 95 26            STAX   CHRS   +23 NOTE OFFSET TO PUT IT AT THE END
0346 CA               DEX
0347 10 F9            BPL    TALOOP
0349 30 08            BMI    XFWD   UNCONDITIONAL
034B A5 8C     TOVRIB LDA    RESULT +01 DESIRED VARIABLE
034D 95 54            STAX   VARIBS +01
034F A5 8B            LDA    RESULT
0351 95 53            STAX   VARIBS
0353 4C 79 02  XFWD   JMP    FWD    AND GO DO NEXT ONE
               *
               * PROCESS MATCH STATEMENT
               *
0356 C9 4D     XM     CMPIM $4D     IS IT "M" FOR MATCH?
0358 D0 4F            BNE    XU     BRANCH IF NOT
035A 88               DEY           BACK UP ONE FOR WHAT FOLLOWS
035B C8        MCHKX  INY           POINT TO MATCH CHARACTER
035C A2 27            LDXIM $27     START AT FIRST ACCEPTED CHARACTER
035E B1 97     MCHK   LDAIY CURAD   GET THE MATCH CHARACTER
0360 F0 08            BEQ    MXY    THEY HAVE MATCHED TO END OF "M:" STMT
0362 D5 03            CMPX   CHRS   CHECK FOR MATCH
0364 D0 08            BNE    MXNMCH BRANCH IF MATCH FAILED
0366 C8               INY           ELSE BUMP TO NEXT PAIR OF CHARACTERS
0367 CA               DEX
0368 10 F4            BPL    MCHK   AND GO CHECK IF STILL DATA LEFT
036A A2 59     MXY    LDXIM $59     BOTH EQUAL - SET FLAG TO "Y"
036C D0 37            BNE    MX     UNCONDITIONAL
036E C9 24     MXNMCH CMPIM $24     IS IT "$" FOR VARIABLE REQUEST?
0370 F0 13            BEQ    MNUMB  YES - MATCH TO NUMERIC VARIABLE
0372 C9 2C            CMPIM $2C     IS IT A COMMA GROUP SEPARATOR?
0374 F0 F4            BEQ    MXY    YES - MATCHED SO FAR - SET IT AS YES

0376 C8        MCOMMA INY           NO - SO NEED TO SKIP AHEAD TO COMMA
0377 B1 97            LDAIY CURAD
0379 F0 28            BEQ    MXSETN IF TO EOL, THERE IS NO MORE TO CHECK
037B C9 2C            CMPIM $2C     CHECK FOR A COMMA CHARACTER
037D F0 DC            BEQ    MCHKX  RESTART COMPARE AT NEXT MATCH CHARACTER
037F D0 F5            BNE    MCOMMA LOOP IN SEARCH OF A COMMA
0381 A4 88     MCOMX  LDY    HOLDY  RESET Y TO CURRENT LINE POINTER
0383 D0 F1            BNE    MCOMMA AND GO LOOK FOR NEXT COMMA

0385 C8        MNUMB  INY           VARIABLE - BUMP TO VARIABLE ID
0386 86 8D            STX    ANSX   SAVE CURRENT X FOR NOW
0388 20 A8 04         JSR    CNVDSP CONVERT VARIABLE TO DISPLAY FORM
038B A6 8D            LDX    ANSX   GET POINTER TO INPUT BACK
038D 84 88            STY    HOLDY  SAVE CURRENT "Y" POINTER
038F A0 04            LDYIM $04     HAVE TO SEARCH UP TO 5 BYTES

0391 B9 90 00  MXNOLP LDAY   NUMDSP GET ONE NUMERIC CHARACTER
0394 F0 08            BEQ    MXDIFF BRANCH IF END - MIGHT BE MATCH
0396 D5 03            CMPX   CHRS   ELSE CHECK AGAINST INPUT
0398 D0 E7            BNE    MCOMX  BRANCH IF NO MATCH
039A CA               DEX           ELSE CONTINUE MATCHING
039B 88               DEY
039C 10 F3            BPL    MXNOLP UNCONDITIONAL

039E A4 88     MXDIFF LDY    HOLDY  RESET Y TO CURRENT LINE POINTER
03A0 C8               INY           BUMP TO CHARACTER AFTER VARIABLE
03A1 D0 BB            BNE    MCHK   UNCONDITIONAL CONTINUE CHECKING

03A3 A2 4E     MXSETN LDXIM $4E     GET "N" - MATCH WAS UNSUCCESSFUL
03A5 86 02     MX     STX    FLG    STORE IT
03A7 D0 AA            BNE    XFWD   UNCONDITIONAL FOWRARD TO NEXT LINE
               *
               * PROCESS USE SUBROUTINE STATEMENT
               *
03A9 C9 55     XU     CMPIM $55     IS IT A "U" FOR USE SUBROUTINE?
03AB D0 11            BNE    XJ     BRANCH IF NOT
03AD B1 97            LDAIY CURAD   GET DESTINATION
03AF 48               PHA           SAVE THE LABEL CHARACTER
03B0 20 5A 04         JSR    FWD1   MOVE TO START OF NEXT LINE
03B3 A5 97            LDA    CURAD
03B5 85 95            STA    RETURN SAVE FOR RETURN ADDRESS
03B7 A5 98            LDA    CURAD  +01
03B9 85 96            STA    RETURN +01
03BB 68               PLA           GET DESTINATION BACK
```

There is also the possibility it might be caused by a request to match against the current value of a variable. To perform variable matching, the program calls CNVDSP which converts the variable to display format with leading zeros suppressed. It then matches the display format against the characters in CHRS. If the variable value matches, the program continues checking the rest of the MATCH statement.

If, even after all this, we still have a no-match condition, all is not lost yet. We have to scan forward in the MATCH statement, to look for a comma or the end of line. If we find the end of line, then FLG gets set to N. If we find a comma, the program starts the whole match process over again, from the character after the comma in the MATCH statement and from the beginning of CHRS. All this sounds confusing but, for example, the statement "M:YE,OK,SUR" will provide a Y indication for most affirmative responses such as YES or YES SIR or YEP or SURE WILL or OK.

As I mentioned earlier, the USE subroutine statement shares much of its code with the JUMP statement. The main difference is that the USE statement must save the address of the start of the *next* statement, while the JUMP statement doesn't need to. Note that the USE statement does not nest levels (sorry about that).

There are two reserved labels in PILOT. The first is the asterisk, which is used to completely restart the PILOT program (including zeroing the variables). The second reserved label is "A". This label indicates a JUMP (or USE) to the last ACCEPT statement. If the label in the statement is not one of the reserved labels, the program sets CURAD back to the start of the PILOT program via a call to SETBGN + 3 and starts the search for that label.

The STOP statement is trivial. It merely requires a jump back to the start of the editor.

Processing of the EXIT from subroutine statement is slightly more complex. It involves a check of the high order byte of the address contained in RETURN. If it is zero, then there was no USE statement executed to get there, and the program merely advances to the next line. The high order byte can never be zero, since all the lines are stored above $500. After restoring the return address to CURAD, the program resets the high order byte to zero. This means that the PILOT program can either "fall through" a subroutine, or use it in a normal fashion.

The REMARKS processing rivals that of the STOP statement for complexity. It merely involves advancing to the next statement. One final PILOT statement is the TYPE statement. It is also the default statement if none of the above sections processed it. If the statement is not a true TYPE statement, Y is backed up twice, so the whole line will be printed. Otherwise, the line is printed following the "T:".

The remainder of the program consists of subroutines used by various PILOT statements. The routine PRT prints the current line to the end. It uses the high order bit of IFLAG to see if the program is in editor mode. If it is, then all characters are printed, instead of being checked for a "$" to indicate a variable. After the line has been printed, a carriage return and line feed are output. It then falls through to FWD1.

The purpose of this routine is to advance to the end of the current line, and set up CURAD for the next line. Since it checks for end of line first, before incrementing Y, the fall through from PRT will immediately exit this routine, thus saving a branch in PRT.

FWD1, in turn, exits to a routine called SCURAD. This adds one to Y, and adds the result to CURAD as the start of the next line. Finally, this routine falls through to SKPJNK, which skips over any unwanted junk at the start of the line and executes the return.

With the exception of CNVDSP, the remaining routines are short and pretty much to the point. The VTRANS routine must transfer the high order byte of the variable last, so it sets the sign flag for CNVDSP. The format of the NUMDSP array is set up in the same "backward" manner used for CHRS and NAME, and it is the output of CNVDSP. If the variable is negative, a " – " is inserted as the first character.

The high order bit of SIGNIF is used to keep track of whether a non-zero digit has been encountered in the number being converted. If the bit is off and the current digit is zero, the index is not decremented, but the zero is stored anyway. If the bit is on, the digit gets stored regardless of its value. Any non-zero digit turns on the high order bit, just to make sure. An end of line zero is inserted after the last digit.

There are three SYM monitor routines used in this program. If you plan to bring Tiny PILOT up on another system you will have to change the addresses for these routines. They are all fairly standard, so most systems should have equivalents. INCHR gets one ASCII character from the terminal into the A register, without parity; OUTCHR outputs one ASCII character from A; and CRLF outputs a carriage return then a line feed. Tiny PILOT assumes that all registers are preserved by these routines. μ

```
03BC D0 06             BNE    JDO    NO GO HANDLE AS JUMP STATEMENT
                *
                * PROCESS JUMP STATEMENT
                *
03BE C9 4A      XJ     CMPIM  $4A    IS IT "J" FOR JUMP STATEMENT?
03C0 D0 2E             BNE    XS     BRANCH IF NOT
03C2 B1 97             LDAIY  CURAD  GET DESTINATION

03C4 85 87      JDO    STA    IFLAG  SAVE LABEL CHARACTER
03C6 C9 2A             CMPIM  $2A    HAVE "*" TO REQUEST RETURN TO BEGINNING?
03C8 F0 23             BEQ    IREST  BRANCH IF SO
03CA C9 41             CMPIM  $41    SEE IF A LABELLED JUMP
03CC D0 0A             BNE    JF     IF NOT "A", IT'S A NORMAL JUMP


03CE A5 00             LDA    LST    ELSE SET TO START OF LAST ACCEPT
03D0 85 97             STA    CURAD
03D2 A5 01             LDA    LST    +01
03D4 85 98             STA    CURAD  +01
03D6 D0 43             BNE    ILNEXT UNCONDITIONAL

03D8 20 86 04   JF     JSR    SETBGN +03 AND GET BACK TO START OF PROGRAM


03DB B1 97      FNDMRK LDAIY  CURAD  GET FIRST CHARACTER
03DD C9 2A             CMPIM  $2A    IS IT "*" FOR A MARKER?
03DF D0 07             BNE    FMNEXT NOPE - GO AHEAD TO NEXT LINE
03E1 C8                INY           ELSE BUMP TO MARKER CHARACTER
03E2 B1 97             LDAIY  CURAD  GET LABEL
03E4 C5 87             CMP    IFLAG  SEE IF ITS THE ONE WE WANT
03E6 F0 33             BEQ    ILNEXT YES - GO EXECUTE IT
03E8 20 5A 04   FMNEXT JSR    FWD1   ELSE GO TO NEXT LINE
03EB B0 EE             BCS    FNDMRK AND CONTINUE LOOKING
03ED 4C 55 02   IREST  JMP    RESTRT INDIRECT TO RESTRT
                *
                * STOP STATEMENT
                *
03F0 C9 53      XS     CMPIM  $53    IS IT AN "S" FOR STOP STATEMENT?
03F2 D0 03             BNE    XE     BRANCH IF NOT
03F4 4C 00 02          JMP    START  ELSE RETURN TO EDITOR START
                *
                * EXIT FROM SUBROUTINE
                *
03F7 C9 45      XE     CMPIM  $45    IS IT AN "E"
03F9 D0 10             BNE    XR     BRANCH IF NOT
03FB A5 96             LDA    RETURN +01 MOVE RETURN ADDRESS TO CURAD
03FD F0 10             BEQ    XXFWD  SKIP LINE IF NOT SET
03FF 85 98             STA    CURAD  +01

0401 A5 95             LDA    RETURN
0403 85 97             STA    CURAD
0405 A9 00             LDAIM  $00    NOW SET TO NOT-USED AGAIN
0407 85 96             STA    RETURN +01
0409 F0 10             BEQ    ILNEXT UNCONDITIONAL
                *
                * REMARK STATEMENT
                *
040B C9 52      XR     CMPIM  $52    IS IT AN "R"
040D D0 03             BNE    XT     BRANCH IF NOT - ELSE SKIP THE LINE
040F 4C 79 02   XXFWD  JMP    FWD    CAN'T REACH THAT FAR ALONE
                *
                * TYPE STATEMENTS AND SYNTAX ERRORS
                *
0412 C9 54      XT     CMPIM  $54    IS IT A VALID "T" STATEMENT
0414 F0 02             BEQ    TE     BRANCH IF SO
0416 88                DEY           ELSE BACK UP TO ORIGINAL START
0417 88                DEY
0418 20 21 04   TE     JSR    PRT    NOW PRINT THE LINE
041B 20 6E 04   ILNEXT JSR    SKPJNK CURAD IS SET - SKIP OVER LEADING JUNK
041E 4C 63 02          JMP    LSTART AND GO START ON THE LINE
                *
                * PRINT A LINE FROM CURRENT LOCATION TO
                * NEXT EOL AND THEN SET UP FOR NEXT LINE
                *
0421 B1 97      PRT    LDAIY  CURAD  GET THE CURRENT CHARACTER
0423 F0 32             BEQ    LINEND BRANCH IF TO END OF LINE
0425 24 87             BIT    IFLAG  SEE IF IN EDITOR
0427 30 26             BMI    CHROUT IF SO, DON'T LOOK FOR "$"

0429 C9 24             CMPIM  $24    IS IT A SPECIAL ONE ("$")
042B D0 22             BNE    CHROUT BRANCH IF NOT
042D C8                INY           ELSE BUMP TO NEXT ONE
042E B1 97             LDAIY  CURAD  GET VARIABLE
0430 C9 3F             CMPIM  $3F    IS IT REQUEST FOR NAME ("$")?
0432 F0 0F             BEQ    NAMEO  BRANCH IF YES
```

```
0434 20 A8 04         JSR   CNVDSP CONVERT VARIABLE TO DISPLAY
0437 A2 04            LDXIM $04    GOT 5 BYTES POSSIBLE

0439 B5 90     VBDISP LDAX  NUMDSP GET A CHARACTER
043B F0 15            BEQ   CHROUT +03 BRANCH IF TO END OF VARIABLE
043D 20 47 8A         JSR   OUTCHR ELSE OUTPUT IT
0440 CA               DEX          AND COUNT IT
0441 10 F6            BPL   VBDISP UNCONDITIONAL LOOP

0443 A2 27     NAMEO  LDXIM $27    REMEMBER - IT CAME IN BACKWARDS
0445 B5 2B            LDAX  NAME
0447 F0 09            BEQ   CHROUT +03 BRANCH IF TO END OF NAME
0449 20 47 8A         JSR   OUTCHR
044C CA               DEX
044D 10 F6            BPL   NAMEO  +02 UNCONDITIONAL

044F 20 47 8A  CHROUT JSR   OUTCHR
0452 C8               INY
0453 10 CC            BPL   PRT    LOOP IF NOT TOO MANY
0455 30 2C            BMI   SETBGN RESET TO BEGINNING IF PAST THE END
0457 20 4D 83  LINEND JSR   CRLF   OUTPUT A CR AND THE LINE FEED
               *
               * ENTER HERE TO SKIP A LINE WITHOUT PRINT
               * AND INITIALIZE FOR THE NEXT LINE
               *
045A B1 97     FWD1   LDAIY CURAD  GET A CHARACTER
045C F0 05            BEQ   SCURAD BRANCH IF END OF LINE
045E C8               INY          ELSE BUMP TO NEXT ONE
045F 10 F9            BPL   FWD1   LOOP IF NOT TOO MANY
0461 30 20            BMI   SETBGN RESET TO BEGINNING IF PAST THE END
               *
               * HERE FIXES UP CURAD TO POINT TO BEGINNING OF A LINE
               * CURAD SHOULD INDEX END OF LINE (WITH Y) ON ENTRY
               *
0463 C8        SCURAD INY          BUMP PAST THE CR
0464 98               TYA          MOVE COUNT TO A
0465 18               CLC          CLEAR CARRY FOR ADD
0466 65 97            ADC   CURAD  ADD TO LOW ORDER FIRST
0468 85 97            STA   CURAD  AND SAVE RESULT
046A 90 02            BCC   SKPJNK SKIP IF NO CARRY FORWARD
046C E6 98            INC   CURAD  +01 ELSE BUMP HIGH ORDER
               *
               * HERE TO SKIP PAST LEADING JUNK ON A LINE
               *
046E A0 FF     SKPJNK LDYIM $FF    SET UP Y THIS WAY
0470 C8        SJLOOP INY          INCREMENT TO NEXT CHARACTER
0471 24 87            BIT   IFLAG  SEE IF IN EDIT MODE
0473 30 0C            BMI   SJRTS  DON'T TRY SKIPPING JUNK IF SO
0475 B1 97            LDAIY CURAD  GET CHARACTER TO LOOK AT
0477 30 F7            BMI   SJLOOP IGNORE DELETE CHARACTER ALSO
0479 C9 2A            CMPIM $2A    LOOK FOR "*" LABEL MARKER
047B F0 04            BEQ   SJRTS  RETURN IF FOUND
047D C9 3F            CMPIM $3F    LOOK FOR POSSIBLE OPERATION CHARACTER
047F 90 EF            BCC   SJLOOP CONTINUE SKIPPING IF TOO LOW
0481 38        SJRTS  SEC          SET CARRY FOR BRANCHES AFTER RETURN
0482 60               RTS          BEFORE RETURN
               *
               * SET UP BEGINNING ADDRESS OF USER AREA
               *
0483 20 4D 83  SETBGN JSR   CRLF   START ON A NEW LINE
0486 A0 00            LDYIM $00    EVEN PAGE BOUNDARY
0488 84 97            STY   CURAD
048A 84 00            STY   LST    ALSO SET UP THIS GUY AS DEFAULT
048C A9 05            LDAIM $05
048E 85 98            STA   CURAD  +01
0490 85 01            STA   LST    +01
0492 D0 DA            BNE   SKPJNK UNCONDITIONAL
               *
               * COMPUTE INDEX FOR A VARIABLE
               *
0494 B1 97     GETIDX LDAIY CURAD  GET VARIABLE LETTER
0496 38               SEC
0497 E9 41            SBCIM $41    SUBTRACT "A" TO MAKE RELATIVE TO ZERO
0499 0A               ASLA         TIMES TWO BYTES PER VARIABLE
049A AA               TAX          MOVE TO INDEX REGISTER
049B 60               RTS          AND RETURN
               *
               * TRANSFER A VARIABLE'S DATA TO WORK AREA
               *
049C 20 94 04  VTRANS JSR   GETIDX GET INDEX POINTER FIRST
049F B5 54            LDAX  VARIBS +01 NOW MOVE TO WORK AREA
04A1 85 8A            STA   WORK   +01
04A3 B5 53            LDAX  VARIBS
04A5 85 89            STA   WORK
04A7 60               RTS
```

```
                   *
                   * CONVERT A VARIABLE TO DISPLAY FORM
                   *
04A8 20 9C 04  CNVDSP JSR   VTRANS MOVE TO WORK AREA
04AB 10 17            BPL   ISPLUS BRANCH IF POSATIVE
04AD A9 2D            LDAIM $2D    ELSE PUT IN MINUS SIGN
04AF 85 94            STA   NUMDSP +04
04B1 F8               SED          SET DECIMAL MODE INDICATOR
04B2 38               SEC
04B3 A9 00            LDAIM $00    SUBTRACT FROM ZERO TO COMPLEMENT
04B5 E5 8A            SBC   WORK   +01
04B7 85 8A            STA   WORK   +01
04B9 A9 00            LDAIM $00
04BB E5 89            SBC   WORK
04BD 85 89            STA   WORK
04BF D8               CLD          CLEAR DECIMAL MODE
04C0 A2 03            LDXIM $03    ONLY 4 POSITIONS LEFT
04C2 D0 02            BNE   ISPL   SKIP INDEX SET

04C4 A2 04     ISPLUS LDXIM $04    PLUS HAS FIVE POSITIONS AVAILABLE
04C6 18        ISPL1  CLC          TURN OFF SIGNIFICANCE INDICATOR
04C7 66 8E            ROR   SIGNIF
04C9 A5 89            LDA   WORK   GET FIRST DIGIT
04CB 20 E6 04         JSR   TOOUT  PUT TO OUTPUT AREA
04CE A5 8A            LDA   WORK   +01 SECOND DIGIT IS HIGH ORDER OF THIS
04D0 4A               LSRA         MOVE TO LOW ORDER
04D1 4A               LSRA
04D2 4A               LSRA
04D3 4A               LSRA
04D4 20 E6 04         JSR   TOOUT
04D7 A5 8A            LDA   WORK   +01 LOW ORDER IS THIRD DIGIT
04D9 20 E6 04         JSR   TOOUT
04DC 24 8E            BIT   SIGNIF SEE IF HAD ANY SIGNIFICANT CHARS
04DE 30 01            BMI   ISPL2  SKIP NEXT IF YES
04E0 CA               DEX          ELSE KEEP THE LAST ZERO THERE
04E1 A9 00     ISPL2  LDAIM $00    INSERT END OF LINE MARKER
04E3 95 90            STAX  NUMDSP
04E5 60               RTS          AND RETURN
                   *
                   * CONVERT CURRENT VALUE TO ASCII AND PUT TO OUTPUT AREA
                   *
04E6 29 0F     TOOUT  ANDIM $0F    KEEP ONLY LOW ORDER
04E8 09 30            ORAIM $30    MAKE IT ASCII
04EA 95 90            STAX  NUMDSP SAVE REGARDLESS
04EC 24 8E            BIT   SIGNIF SEE IF SIGNIFICANCE STARTED
04EE 30 05            BMI   SETSIG YES - ALL ARE IMPORTANT NOW
04F0 C9 30            CMPIM $30    ELSE SEE IF SHOULD START NOW
04F2 D0 01            BNE   SETSIG IMPORTANT IF NOT ZERO
04F4 60               RTS          ELSE RETURN

04F5 38        SETSIG SEC          SET SIGNIFICANCE BIT ON
04F6 66 8E            ROR   SIGNIF ALWAYS
04F8 CA               DEX          AND POINT TO NEXT AVAILABLE POSITION
04F9 60        PGMEND RTS          AND THEN RETURN
```

```
SYMBOL TABLE 2000 226A
ACHR   029E   ACHRQ  02A8   ACHRR  02AE   ACHRS  02B6
ADONE  02BD   ANSX   008D   BITROL 02EF   CHAR   023F
CHARQ  0245   CHKCOM 026D   CHROUT 044F   CHRS   0003
CMPDON 0334   CMPLOP 02DA   CNVDSP 04A8   CR     0D0D
CRLF   834D   CURAD  0097   EGET   020C   ELINE  0207
EXEC   0252   FLG    0002   FMNEXT 03E8   FNDMRK 03DB
FWD    0279   FWDQ   045A   GETIDX 0494   HOLDY  0088
IFLAG  0087   ILNEXT 041B   INCHR  8A1B   IREST  03ED
ISOPR  02FF   ISPLQ  04C6   ISPLR  04E1   ISPLUS 04C4
JDO    03C4   JF     03D8   LINEND 0457   LSTART 0263
LST    0000   MCHK   035E   MCHKX  035B   MCOMMA 0376
MCOMX  0381   MNUMB  0385   MX     03A5   MXDIFF 039E
MXNMCH 036E   MXNOLP 0391   MXSETN 03A3   MXY    036A
NAME   002B   NAMEO  0443   NOTNMB 02F9   NUMDSP 0090
OPMNUS 0317   OPRATN 008F   OPWRAP 0324   OUTCHR 8A47
PADLOP 0231   PGMEND 04F9   PRT    0421   RESTRQ 025E
RESTRT 0255   RESULT 008B   RETURN 0095   SCURAD 0463
SETBGN 0483   SETNL  024D   SETSIG 04F5   SIGNIF 008E
SJLOOP 0470   SJRTS  0481   SKPJNK 046E   SKPNXT 026A
START  0200   STRTS1 027E   TAKEIN 0297   TALOOP 0342
TE     0418   TFLAG  0275   TOOUT  04E6   TOVRIB 034B
TRYDSP 0224   TRYREF 022D   VARIBS 0053   VBDISP 0439
VTRANS 049C   WORK   0089   XA     028B   XC     02C3
XCQ    02CA   XE     03F7   XFWD   0353   XJ     03BE
XM     0356   XQUEST 0282   XR     04CB   XS     03F0
XT     0412   XU     03A9   XXFWD  040F
```

THE MICRO SOFTWARE CATALOG: XII

Mike Rowe
P.O. Box 6502
Chelmsford, MA 01824

Name: **AIM 65 Morse Code Send Program**
System: **AIM 65**
Memory: **Less than 1K**
Language: **Assembly Language**

Hardware: **One IC** (inverter), **one relay** (no relays necessary for solid state transmitters)

Description: This program converts the AIM 65 keyboard input into Morse code characters that are then output to pin PA0 on the applications connector. A suitable buffer (two 7404 inverters) will key the transmitter or drive a relay. The following feature are provided:

1) The characters are displayed on the AIM 65 display as they are typed on the keyboard. Up to 20 characters may be displayed, with a 20 character overflow buffer, giving the ability to type 40 characters ahead.
2) The display is updated (scrolled to the left) as the characters are sent.
3) The display is updated as the characters are entered.
4) The DEL key allows characters to be deleted (backspace and delete).
5) Code speed is set and controlled digitally by entering the speed (in decimal) on the keyboard. Speeds from 05 to 99 wpm are possible.
6) The speed can be changed at any time by pressing the ESC key, followed by entering the speed in words per minute.
7) A message being sent may be halted at any time by means of the carriage return key.

Copies: **Just Released** (hopefully thousands)
Price: **$3.50**

Includes: Source listing in the AIM 65 dissasembly format interface description, and instructions for operation.

Author: **Marvin L. De Jong**
Available from: Marvin L. De Jong
S.R. 2, Box 364A
Branson, MO 65616

Name: **MONITOR-II**
System: **APPLE-II**
Memory: **3K + DOS 3.2 requirements**
Language: **Machine Language**
Hardware: **APPLE-II, DISK-II**
(Supported) High speed serial card, Programmer's Aid ROM Applesoft ROM

Description: MONITOR-II is an extension to Apple's ROM Monitor that adds an interactive command language. MONITOR-II provides the user: Extended cursor control-Named program load, initialize, and execute from tape or disk-Programmer's Aid ROM #1 interface and commands-Transient area management-Variable speed listings-Split screen display-Special I/O routine support-User extensible interactive commands-Integer BASIC Variables Utility-Resident supervisor-much more

Copies: **Just released**
Price: **$45.00** on disk (Introductory)

Includes: System disk, user's guide
Optional: Assembly listings, systems guide

Author: **W.C. Deegan**
Available from: W.C. Deegan
2 Fairfax Towne
Southfield, MI 48075

Name: **Single Drive Copy**
System: **APPLE**
Memory: **16K**
Language: **Integer BASIC**

Description: Allows you to copy a diskette using one drive. Automatically adjusts for available memory and comes with the option of not initializing disk.
Copies: **10**
Price: **$19.95 + $1.00** postage & handling (PA residents add 6% sales tax).
Includes: Cassette with instructions.

Author: **Vince Corsetti**
Available from: Progressive Software
P.O. Box 273
Ply. Mtg., PA 19462

Name: **CLASS ATTENDANCE**
CHURCH ATTENDANCE
System: **PET**
Memory: **8K** or more
Language: **BASIC**
Hardware: **PET, 8K** or larger

Description: Class Attendance & Church Attendance maintain attendance records for any group which meets regularly, using data tapes. The school version records 0 to 5 days' attendance for each of up to 39 weeks. The church version does the same for 0 to 5 times for each of 12 months.

Attendance automatically sorts entires alphabetically within & between data tapes. Though presently dimensioned for up to 10 tapes of 70-85 names each, there is no limit to the number of tapes that can be used.

Commands include: ADD, DELETE, LIST, UPDATE, END & CATEGORY. Within CATEGORY there are 8 subcommands for frequency of attendance, (perfect attendance this month, for example). There is also a Help command to escape to the main menu from anywhere in the program, and a Back-up command to correct mistakes in updating. With four SIMPLE line changes, PET's with the upcoming roms can use Attendance also.

Copies sold: **Just released.**
Price: **$12.95** (either version)
Includes: Program cassette with sample Directory & sample data, 2 blank C-10 tapes for data, and a 6 page instruction booklet with lists of variables used & location of major routines.

Author: **James Strasma**
Available from: Dr. Daley
425 Grove Av.
Berrien Springs, MI 49103
(616) 471-5514 (Sun-Thurs, Noon-9PM)
(Master Charge & Bank Americard OK)

Name: **Omni Plotting Package**
System: **APPLE**
Memory: **32K**
Language: **BASIC**
Hardware: **Disk and Applesoft on ROM**

Description: 7 data sets and 2 exec file examples 12 basic programs, employ the High Resolution Mode. No shape tables are used, data base is Cartesian coordinates which can be sent over the phone or to hardcopy plotters. Package allows ease of intricate graphics generation supported by easy manipulation of graphics after generation. Package is excellent for map making, drafting, artistry and any other purely visual application.

Price: **$19.00 + $5.00** for diskette or you send one. We pay postage.

Includes: 12 basic programs, 7 data sets, 2 examples, 19 page illustrated users manual with step by step instructions.

Author: **P.S. Truax**
Available from: Omni Plotting Package
c/o P.S. Truax
237 Star Rte.
Santa Barbara, CA 93105

Name: **Applesoft BASIC-Optimization Library**
System: **APPLE II**
Memory: **16K**
Language: **6502 Assembly Language**
Hardware: **Standard** (Applesoft ROM card optional)

Description: The Library consists of two 1.3K assembly-language programs (VAROPT & REMOUT) that will work in any APPLE II with APPLOSOFT IIa, VAROPT renames all variables to unique 1-2 character variable names and displays (prints) a cross-reference listing with new name, old name, and all line numbers where the variable was referenced. REMOUT removes remarks, removes extra colons, renumbers from 1 by 1, and concatenates short lines into a reduced number of long lines.

Together, these programs will convert a verbose, well-documented development version of an Applesoft program into an extremely memory-efficient, more-secure production version.

Copies: **Just Released**
Price: **$15**/Cassette, **$20**/Disk

Includes: Cassette/disk with hex listing & instructions
Available from: Sensible Software
P.O. Box 2395
Dearborn, MI 48123

Name: **APPLE XFR**
System: **APPLE II with disk**
Memory: **32K**
Language: **Machine Language**
Hardware: **Apple II, Disk II, D.C. Hayes Micromodem II**

Description: This program will establish a session between two Apples and allow transfer of any text file from one to another. All text file I/O is written in machine language. Programs are also included to facilitate the conversion of any Integer or Applesoft program to a text file.

Copies: **Just released**
Price: **$15.95** on diskette
Includes: Diskette and documentation

Author: **Travis Johns**
Available from: Travis Johns
1642 Heritage Cr.
Anaheim, CA 92804

Name: **OTHELLO**
System: **APPLE**
Memory: **16K**
Language: **Integer BASIC**

Description: A game played by one or two players. Once a piece is played the color may be reversed many times, with sudden reverses of luck. Can win with a single move. Computer keeps all details and flips the pieces.

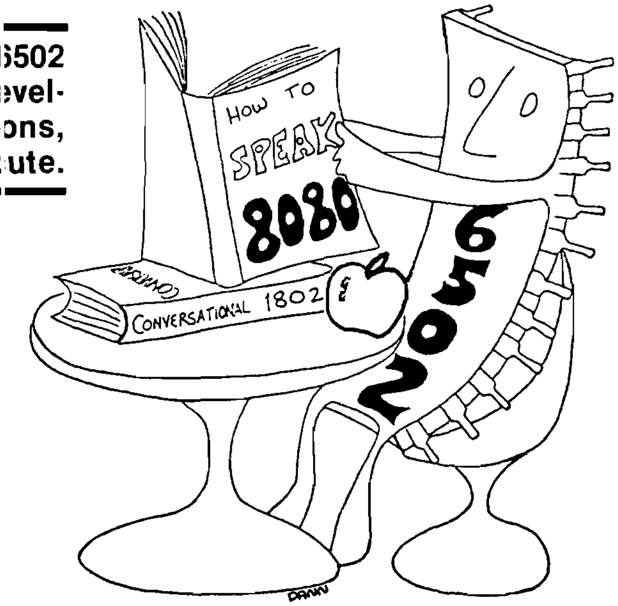
Copies: **10**
Price: **$9.95 + $1.00** postage & handling (PA residents add 6% sales tax).

Includes: Cassette with instructions.
Author: **Vince Corsetti**

Available from: Progressive Software
P.O. Box 273
Ply. Mtg., PA 19462

8080 Simulation with a 6502

The design for an 8080 simulator running on the 6502 illustrates how your micro can assist program development for other machines, master new applications, and double the quantity of software it will execute.



Dann McCreary
Box 16435-M
San Diego, CA 92116

Why Bother to Simulate?

While many advantages of simulating one microprocessor with another might be cited, there are several which I believe stand out above the rest.

Educators and students can use simulation software as an enhancement to introductory courses in microprocessing. Such courses often make use of single board microcomputers like the Commodore KIM-1. These computers provide invaluable hands-on experience. The addition of simulation software can multiply their effectiveness by enabling the study of alternate architectures and instruction sets without the expense of purchasing more hardware.

The entrepreneur and the hobbyist are typically owners of systems based on a single type of processor. Should a situation arise in which they would like to develop software for some other processor, they are faced with another significant capital investment. The availability of simulation packages which can run on their present hardware can make it economical to design and debug code for other processors.

Applications software fulfilling particular functions is sometimes hard to come by. Some might claim that the availability of a given application varies inversely with the need for a version written for the microprocessor available to run it on. Enter simulation software and your choice of applications can be easily doubled. One good example might be the use of an inexpensive 8080 assembler for a one-time task rather than going to the time or expense of producing a cross-assembler.

The experimenter, never quite satisfied with the status quo, can use simulation techniques to try out his theories about an optimized instruction set. He can, in software, model the processor of his design and by doing so he can gather actual data about the validity of his ideas.

The major and most obvious drawback to simulating one microprocessor with another is the large speed penalty. In the Cosmac 1802 Simulator which I have implemented on the 6502, about fifty 6502 instructions are executed in the course of executing one 1802 instruction. In my 8080 Simulator, twice as many or more are required for each 8080 instruction executed. High speed real-time code or applications requiring precise timing relationships derived from instruction cycle timing are clearly outside the scope of this technique.

A somewhat lesser problem involved is the space occupied by the simulator program, which must be co-resident in memory with the application program. Careful design here can make the simulator quite compact but it does take up a finite amount of space.

For a majority of applications, I feel that the advantages of using a simulator overshadow the drawbacks, making this type of modeling very worthwhile.

Optimizing the Approach

A simulation of sorts could be accomplished by compiling or translating the code of an 8080 into 6502 code. This approach would in fact be advantageous from an execution speed standpoint and would be a good choice if running application software were the only consideration. It would, however, generate large amounts of code and would not meet some of the other objectives I had for an 8080 simulator.

The interpretive approach seemed to best fulfill my self-imposed requirements. It would provide an accurate model of the 8080 processor, complete with all internal registers and duplicating all 8080 instructions. It would allow for single stepping or tracing through an 8080 program invaluable for debugging and for educational purposes. An interpreter could be very code-efficient, not only using little memory itself but also allowing 8080 object code to run unmodified in a 6502 environment.

I could have taken a "brute force" approach to interpretation, using perhaps a table lookup scheme and transfering to a separate routine for each 8080 op-code. This offered some advantages in simplicity and execution speed but it required far more memory than I cared to use.

A careful analysis of the 8080 instruction set suggested that the 256 table entries and routines required by a "brute force" technique could be reduced by 25 by grouping the 8080 op-codes into categories sharing common functions.

In addition, certain judicious tradeoffs could be made between simplicity and ideal features, taking best advantage of the addressing modes and features of the 6502. For instance, the 6502 stack resides in page one and many of the 6502's instructions and addressing modes make use of page zero. To avoid memory use conflicts it would have been nice to simulate 8080 memory starting at 0200 HEX, making that address equivalent internally to 0000 HEX. This would have required a great deal of overhead in the form of a special monitor to show addresses minus the 200 HEX offset.

The addresses being used by the 8080 program while running would have to be converted dynamically, and in order to use indirect addressing a special set of simulated registers would have to be maintained in page zero. Besides requiring much more code, this would slow execution speed down considerably. I decided instead to simply require the user to patch around the small areas in page zero and page one being used by the simulator.

Final Design Overview

Laying out the 8080 instruction set graphically on a hexadecimal grid, as illustrated, reveals some interesting features. Four major divisions are apparent, neatly dividing the instruction set into quadrants. The second quadrant is composed alomst entirely of MOV instruction op-codes. This MOV group most clearly illustrates the way that 8080 op-codes break down into source and destination fields, and suggests the best way to organize simulated 8080 registers in memory.

With simulated registers arranged properly in memory, source and destination field data can be extracted from the op-code and used as indexes to the registers involved. In every case where instructions act upon individual registers their order, as dtermined by this source/destination indexing scheme is B.C,D,E,H,L,M,A—where M is not an actual register but rather the content of the mrmory location pointed to by the HL register pair.

This order suggests a general method for accessing individual registers with some slight exceptional logic for the M "pseudo-register". By inverting the source and destination indexes and reversing the order of the 8080 registers in memory it becomes possible to use the HL register pair directly as an indirect pointer to memory. Adding the Stack Pointer and Program Counter to the register array in the same reversed order completes the simulated register set.

Looking again at the instruction set grid it can be seen that a symmetry exists based on the source field of the op-code. For instance, all INR instructions have source fields containing 04 HEX while all DCR instructions have 05 HEX as their source field. The fourth quadrant exhibits similar symmetry. The third quadrant is more logically defined by the destination field, but still divides into 8 groups of similar instructions as do the first and fourth quadrants. These, along with the entire MOV quadrant, total 25 groups of similar instructions. A major task, then, of the simulator mainline is to determine from the op-code which of the 25 groups it belongs in so tht control can be transfered to the proper routine to interpret it.

To keep the simulator as compact as possible it is advantageous to perform as many common operations as possible in the mainline. Fetching the op-code, extracting source and destination indexes from it and incrementing the Program Counter are fundamental. The mainline also fetches the content of memory pointed to by the HL register pair, clears a flag used by many simulator routines, saves data from the register pointed to by the destination index for later operations, tests for and handles interrupts and handles other "housekeeping" type functions.

At the end of the mainline the address of the selected interpreter routine is pushed onto the stack along with a preset status. A 6502 RTI instruction is executed, transferring control to the proper module entry.

It is the responsibility of each module to correctly interpret all op-codes which result in a call to that module. Each module is constructed as a subroutine, returning control to the mainline via an RTS. This also enables certain modules to be used as subroutines by other modules. A brief look at the modules and their support subroutines will help to illustrate their functions.

MOV. While encompassing the largest number of op-codes of any module, MOV has perhaps one of the simplest tasks. It merely takes the content of the register indicated by the source index and stores it in the register pointed to by the destination index. No condition flags in the PSW (Processor Status Word) are affected. The only slight complication whether the destination is memory, in which case the HL register pair is used as an indirect pointer to store the result in memory.

INX/DCX. This module must increment or decrement a selected register pair. The least significant bit of the destination index is tested to determine whether the instruction is an increment or a decrement instruction. The bit is then dropped and what remains is an index to the proper register pair—except for the cases of 33 HEX and 3B HEX when the Stack Pointer is the register pair of interest. In these cases, the proper index for the Stack Pointer is substituted.

With the proper index set, a call is made to INCDEC. INCDEC is a 16 bit adder designed to add two zero page 16 bit operands. With the 6502's X and Y indexes properly set at the entry to this support routine, the content of a double precision one (0001 HEX) or a double precision minus one (FFFF HEX) is added to the chosen register pair, performing the increment or decrement.

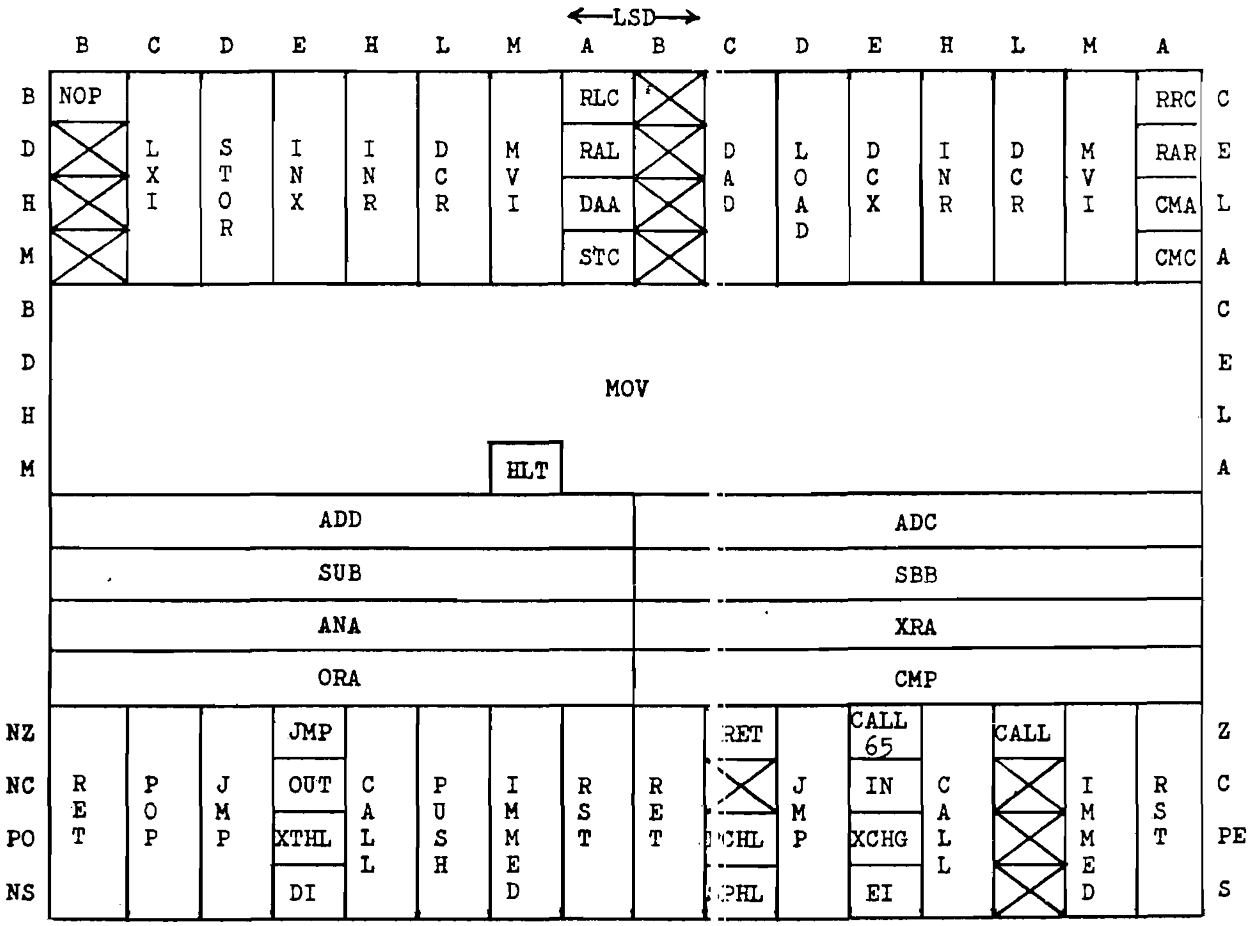
The proper register pair is selected in the same fashion for the DAD and LXI instructions also.

INR, DCR, MVI. These instructions are very consistent in their use of the destination index for determining which register (or memory as the case may be) they operate on. INR and DCR have the added complication of modifying the PSW condition bits, with the exception of the 8080 Carry.

Rotates. This is a mixture of quite different instructions lumped into one module. Proper execution depends on separating the Rotate instructions from the DAA, STC, CMC and CMA instructions, providing special logic for each and insuring the proper setting of PSW flags.

PUSH/POP. While handling register pairs somewhat like INX, DCX, DAD and LXI do, these instructions differ in substituting a register pair made up of the 8080 Accumulator and PSW for the Stack Pointer. The simulator handles this by looking for the special case and then decrementing the destination index to the proper position. The Stack Pointer is then incremented or decremented appropriately and the register pair data transferred to or from the stack as required.

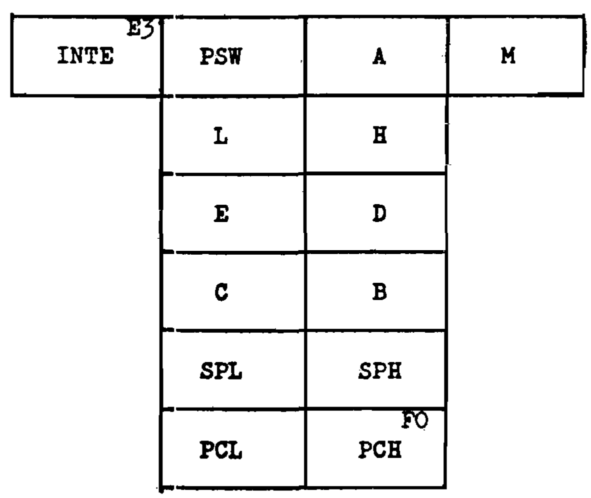
Several support routines come into play, including INCDEC and various routines for transferring the content of register pairs between each other and memory. An intermediate register pair (not illustrated) is utilized as a temporary storage location during the exchange of register pairs. I've labeled it simply "SCR", though I believe it bears an actual hardware analog in the 8080 in the form of a hidden register pair, temporary registers W and Z.



8080 Instruction Set Diagram
"X"s = Unimplemented

CALL and RETURN. These also manipulate the stack, using it as a storage location for the content of the Program Counter. The same set of support routines are used to get the transfer address from memory (for the CALL instruction) and to move data to and from the stack memory. RST is treated like a CALL instruction, except that the transfer address is computed from the destination field of the op-code rather than taken from memory. Conversely, JUMP gets its transfer address from memory, but does not save any return address on the stack.

Condition Codes. CALL, RET, and JMP all make use of a subroutine called CONDIT. CONDIT examines the destination index derived from the op-code and subdivides it into a condition index and a True/False indicator bit. The index is used to select a PSW bit mask from a table of masks. These masks align with the appropriate bit in the PSW. Based on the state of the selected PSW bit and the True/False indicator bit, CONDIT returns an indication of whether or not the JMP, CALL, or RET should take place.



8080 Simulator Register Map

Arithmetic and Logic. These instructions occupy the third quadrant of the instruction set. Rather than being grouped vertically by their source fields they are grouped horizontally by their destination fields. This is due to the fact that while they may have different sources of data, they all have one implied destination—the 8080 Accumulator. The CMP instruction is the only one of this group which does not place its results in the Accumulator. It merely discards the result, setting only the PSW flags accordingly. This is accomplished by forcing the destination index to point to a scratchpad location.

Probably one of the most difficult things to simulate successfully is the proper setting of the Processor Status Word. Different instruction groups affect different subsets of PSW flags but the Arithmetic and Logic group affect all the flags. Zero, Sign and Parity flags are always affected as a group. A routine called STATUS sets these three flags simultaneously when a result is passed to it. Carry and Auxilliary Carry are handled separately as they may be affected in isolation by some instructions and not affected at all by others.

Special Features. For the purpose of using the simulator as a debugging tool, I chose to trap unimplemented op-codes. When the 8080 Simulator determines that the current instruction is an illegal op-code it forces a jump to the system monitor. This can be used to advantage as a simple type of breakpoint. Alternately, a table of breakpoint addresses may be set up in memory. After each instruction, the 8080 Program Counter is compared to each address in the breakpoint table. If a match is found, a jump is forced to the system monitor. This makes it possible to step from breakpoint to breakpoint, seeing the result of groups of steps rather than only individual steps.

I/O Instructions. I/O is also handled via a table of addresses. Each entry in the table is the address of a port in the 6502 system. The entries in the table are associated with 8080 ports in sequential ascending order. Setting of the Data Direction Register, as in a 6530 PIO, is handled transparently to the user.

Call 65. I have "borrowed" one of the 8080's unimplemented op-codes for a special purpose function—calling 6502 subroutines from an 8080 program. This enables you to use existing system I/O routines and other utilities. All that is required is to add brief header and trailer routines to transfer the required parameters to and from simulator registers and 6502 registers used by the subroutine. The CALL 65 instruction may also be useful for handling time dependent code segments.

Summary

Modeling one microprocessor with another is a technique which provides many potential benefits. It has certain significant drawbacks, most notable of which is a large penalty in execution speed. These drawbacks, however, are not of paramount importance in a large number of applications in instructional, personal and experimental use.

Designing such a simulator involves tradeoffs between the complexity and quantity of the coding required for the task on one hand, and the features and execution speed of the final product on the other. I chose to minimize the quantity of code, emphasizing commonality of functions within the simulator.

Simulators for the 8080 and Cosmac 1802 microprocessors are available from the author in versions designed to run on the Commodore/MOS Technology KIM-1.

Thanks to Gary Davis for his generous support in the form of access to his 8080 system and his assistance in running comparison tests. μ

APVENTURE

GAMES OF HIGH ADVENTURE
FOR THE APPLE II FROM

SYNERGISTIC SOFTWARE

5221 120th Ave. S.E.
Bellevue, WA 98006

The Apventure games combine the exciting graphics and sound effects capabilities of the APPLE II with the fascinating complexity of a mythical adventure game. Monsters, hazards, obstacles, weapons, magical devices and evil powers confront the player at every turn as you gather treasure and try to reach your goal. Two adventures are now available:

DUNGEON CAMPAIGN - Full color graphics subterranean adventure. 16K required.
CASS. $12.50 DISK $15.00

WILDERNESS CAMPAIGN - HIRES graphics surface adventure. 48K required.
CASS. $15.00 DISK $17.50

GET BOTH ON ONE DISK FOR $30.00

(WA Res. add 5.3% sales tax)

MORE INNOVATIONS!

FROM

P.S. SOFTWARE HOUSE

FORMERLY PETSHACK

PET™ INTERFACES

NEW!

PET to CENTRONICS INTERFACE	$98.00
PET to 2nd CASSETTE INTERFACE	$49.95

PET™ SCHEMATICS

FOR ONLY $24.95 YOU GET:
24" X 30" schematic of the CPU board, plus oversized schematics of the Video Monitor and Tape Recorder, plus complete Parts layout - all accurately and painstakingly drawn to the minutest detail.

PET™ ROM ROUTINES

FOR ONLY $19.95 YOU GET:
Complete Disassembly listings of all 7 ROMS, plus identified subroutine entry points; Video Monitor, Keyboard routine, Tape Record and Playback routine, Real Time Clock, etc. To entice you we are also including our own Machine Language Monitor program for your PET using the keyboard and video display.
You can have the Monitor program on cassette for only $9.95 extra.

SOFTWARE

6502 DISASSEMBLER	$12.95
MAILING LIST - For personal or business applications.	$9.95
MACHINE LANGUAGE MONITOR - Write Machine Code. Save on tape	$9.95
BUDGET NEW - Keep track of Bills and Checks. Update as needed	$14.95
STARTREK - All-time favorite written for the PET's special Graphics	$7.95

Send for our free SOFTWARE BROCHURE. Dealer inquiries welcome.

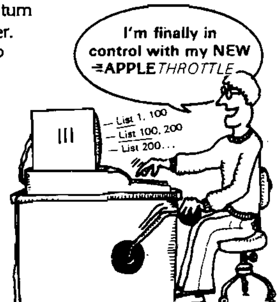
P.S. SOFTWARE HOUSE

P.O. Box 966 Mishawaka, IN 46544
Tel: (219) 255-3408

PET is a trademark of Commodore Business Machines

Put Yourself in Control with the APPLE*THROTTLE*

That's right! The APPLE*THROTTLE* will turn your game paddles into a speed controller. By simply pushing a button, you can stop your computer for as long as you want. Release the button, and your computer enters a slow-motion mode with one paddle controlling the speed. And if that isn't enough, look at these additional features:



- Plugs into any slot
- Works with machine language, Integer BASIC, and Applesoft
- Normal - slow - stop
- Use to LIST, TRACE, RUN, etc.
- NO SOFTWARE to load
- Unveil program secrets

APPLE*THROTTLE* **$89.95**

And there's more! No more multiple LIST commands to view small program sections. With the APPLE*THROTTLE*, you'll be able to list or trace long programs while watching your program flow in slow-motion. So get in control with the APPLE*THROTTLE* and order yours today!

APPLETIME, a Real Time Clock for the Apple II. Plugs directly into any slot and keeps time even when computer is off. Features 12/24 Hour, BCD/ASCII data format, and AC/Crystal time base selection. Includes software examples for machine language and BASIC programs. Completely assembled and tested.
APT-1 Real Time Clock **$79.95**

PROTOBOARD, with over 1300 holes on 0.1 centers for designing your own circuits.
APB-1 Protoboard **$17.95**

VERBATIM 5¼" DISKETTES
Soft-Sector Box of 10 ... **$34.50**
(plastic file case included)

west side electronics
P.O. Box 636, Chatsworth, CA 91311
We pay all shipping in Continental U.S.A.
Others add 10%. California residents add 6% tax

TEXTCAST™

Turn your PET into a WORD PROCESSOR comparable to large systems for a fraction of the cost!

CREATE-REVIEW-EDIT FILES ON TAPES OR DISKS

PRINT TEXT-LETTERS-FORMS-TABLES

TEXTCAST FITS YOUR PET SYSTEM, OLD AND NEW ROMS

CLEAR INSTRUCTIONS!

Prices: Tape plus manual, $60.
Diskette plus manual, $65.
Manual separately, $20.
Write: TEXTCAST M9
P.O. Box 2592
CHAPEL HILL, N.C. 27514

Writing for MICRO

Shawn Spilman
Box 6502
Chelmsford, MA 01824

Who writes for MICRO? Subscribers just like yourself! How does one go about it? Read on!

The kind of material published in MICRO can be broken down into three general categories: application notes describing hardware or software projects, tutorials conveying general information about specific subjects, and reviews presenting informed opinion. The division into categories is not hard and fast; one easy way to get published is to write a piece whose very novelty defies categorization. Yet most articles published in MICRO and other magazines describe a project, or a concept, or a product, and can be labelled accordingly.

The label serves a purpose by reminding the writer that abstract theory may be out of place in an application description; working laboratory projects may detract from a tutorial; and opinion, the mainstay of reviews, is alien to both other forms.

This article is a tutorial on the subject of writing technical articles. It illustrates how most anyone can write a piece that will be received gratefully by any number of magazines, including MICRO, and it explains exactly why one would want to do so. Because it is a tutorial, and not an application note, it will not present step by step instructions that could be executed in sequence, with the usual backtracking and microbe debugging, to produce a working (publishable) project. If it did, about ten thousand readers would promptly deluge MICRO with drafts of essentially identical manuscripts.

Instead, we will examine each of the three forms of technical prose, describe some easy ways to get started, mention a few techniques that may be applied along the way, and encourage you to rush the result directly to MICRO.

Application notes

Many personal computers are much like the H.O. railroad train toys of two decades ago: expensive indulgences that occupy a few delightful hours on Christmas day and spend the next six months gathering dust on a closet shelf. A decent model railroad used to run big bucks, what with engines and cars and lots of track segments, tunnels and crossings and enough plastic brakemen to simulate featherbedding.

Yet once assembled, it served only one useful purpose. Like model trains, the personal computer may provide little more than entertainment the day it is unpacked and assembled. It can play all kinds of games, and play most of them exceedingly well. It comes complete with a formidable library of recreational software. Whether any individual machine ever rises above recreational applications depends entirely upon the dilligence and ingenuity of its owner.

This explains why many programming buffs scorn "personal computers." Quite a few data processing professionals actually eschew the term itself, perhaps because there is little that is personal in computing and less that is computational in pure and simple gamesmanship. The measure of any computer lies in its ability to implement true data processing applications.

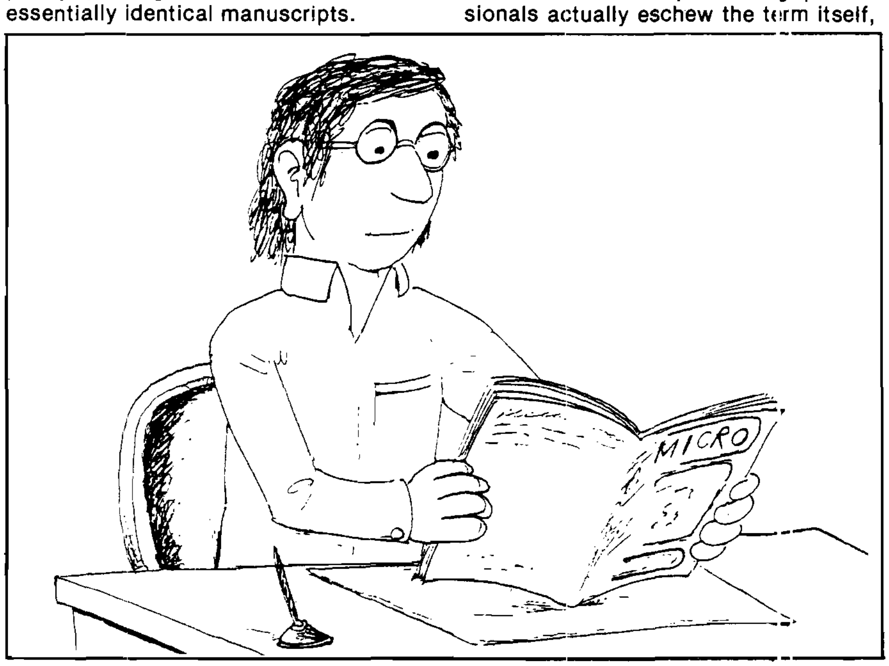
Recreation is a legitimate application, of course, but no one doubts that personal computers can be recreational. The question is, "Can they be applied to other problems?" Can they afford economical, effective solutions to the classical problems of data management? Can they admit to new applications reflecting the unsolved problems raised by recent technology?

We suspect that the answer is a resounding yes, and we can demonstrate this by describing applications that are served by personal computers. Each application implemented successfully enhances the versatility of the machine that was used to perform the task and, perhaps more important, each makes the next application all that much easier to implement.

A 6502 application note will benefit the entire 6502 community if it describes the solution to an open problem or an application never before implemented on a particular machine. Beneficial articles might also report new or unusual approaches to problems that have been solved using different methods. The novelty, general applicability, and overall ellegance of the solution are quite important because, after all, brand new applications that solve open problems are very rare.

It is easy to write an article that describes a computer application. This is fortunate because the application is not fully implemented until it has been described in writing. An application note should describe the problem that was solved, the method of solution, and the implications of the method. It should answer the questions: "What?" and "So what?"

It is impossible to overrate the value of a problem description. Serious computer scientists are constantly refining their ability to evaluate problems, assign them to categories or classes, distinguish those solved in the past from those that remain open, separate the easy from the difficult, generalize the solution to other applications, and extract specific techniques that might serve well in future projects.



This skill derives from exposure to problems, as well as solutions. So state the problem clearly. Describe the situation that caused the problem. Indicate its analogs in related situations. Outline previous attempts to solve the problem, and mention the measure of their success or the reason for their failure.

You will only have to deal with your solution once, now that it is fully implemented, yet whether you are describing a simple memory test routine or a mind boggling speech synthesizer, yet another fast Fourier transform or the first algorithm to play a competitive game of GO, you will undoubtedly encounter your problem over and over again. Few people understand the problem as well as you do, now that you have solved it once. Take the time to describe it well, so that you and everyone else will recognize all of its manifestations.

The problem solution is most often a program or a piece of logic. Software solutions and hardware solutions have much in common. Most interesting problems admit to solutions of either type.

The best presentation of a software solution reproduces a working source of the actual computer program; that is, the assembler or compiler output listing of a program that was loaded and tested thoroughly immediately after assembly or compilation.

Listings that were transcribed or manually corrected imply that the person who made the copy or revision is less prone to error than the computer. The entire article is likely to be viewed with the same scepticism anyone would accord this implication. As a rule, let the computer generate the program listing.

It goes without saying that any program worth coding is worth commenting. Or does it? The first self explanatory computer program remains to be written. If and when it finally appears, odds are that it will contain comments.

The program description is perhaps the least important part of an application note. The program is right there, after all, cleanly coded and with ample commentary. The big question, "What?", has been answered by the problem description and the source listing.

Authors stress program descriptions because they provide an effective mechanism for answering the question, "So what?" The description illustrates why an application is deserving of study. It points out noteworthy aspects of the designer's methodology, perspective or approach. It identifies techniques that may be generalized to solve other problems. In much the way a map enhances appreciation of unfamiliar terrain, it uncovers pitfalls, highlights the points of interest, and distinguishes one particular application from the variety of similar programs that almost always exist as equivalent solutions. Is your memory test routine any different from a thousand others written since Babbage named the game? If so, the program description is the place to point this out.

A hardware application note will require a logic schematic in lieu of a program listing, block diagrams in place of flowcharts, pin out lists instead of calling sequences, and perhaps a photograph. That photo might not provide much hard and fast information, but it relates your article to the real world.

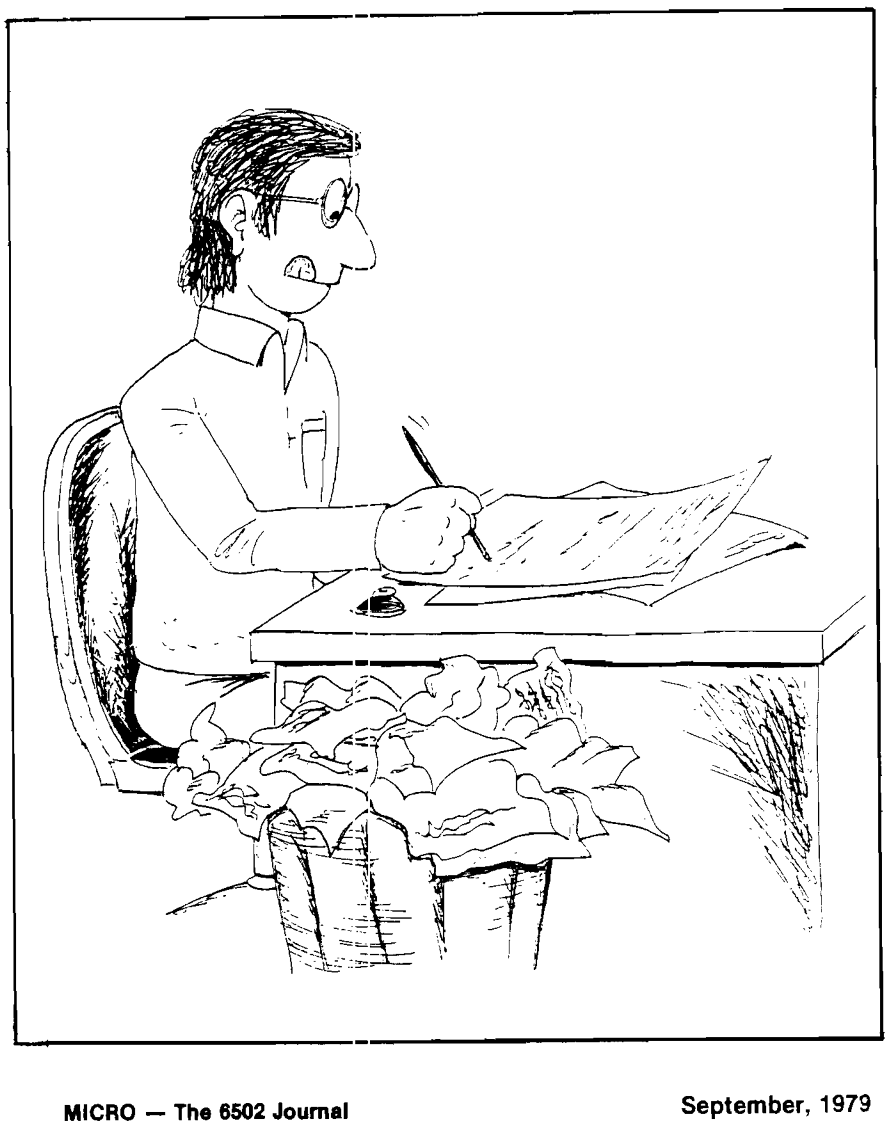
Schematics and block diagrams are almost always drawn by hand and, therefore, susceptible to errors no proofreader will ever catch. Unlike software designers, whose computer generated source listings instill a measure of reader confidence, the hardware designer must rely on manual reproduction techniques to express his implementation. That extraneous photo is one exception to this rule. Like computers, cameras might not always tell the truth, but they never make mistakes.

Tutorials

A tutorial is a short, complete and entertaining explanation of a technical subject. Unlike application notes, tutorials need not describe operational programs or projects that may be constructed to perform useful tasks. Although they may include program segments or circuit details, by way of illustration, tutorials present techniques for solving general problems, instead, and avoid specific problem solutions.

The subject of a tutorial must be selected carefully to resolve the conflicting demands of brevity and completeness. A single chip, such as the 6522 or 6532, might make a good subject. A major subsystem, such as a video driver or a tape I/O package, might be too complex to describe in sufficient detail. The general subject of subroutine calling sequences would make a good tutorial; however, the subject of floating point math packages is much too broad.

Because of its limited size, a tutorial may employ writing techniques that are not appropriate in other types of



technical prose. Use of the first person is common, for example, and casual or vernacular writing may be effective. These techniques help make the tutorial entertaining, fun to read; their use gives tutorials a big advantage over longer technical articles, which can tend to be rather dull.

Of course, a tutorial must present more than warmed over material from the manufacturer's documentation or a clever rehash of material excerpted from a textbook. Like any other form of writing, its impact depends upon the author's originality. Here again, careful selection of subject and perspective is the key to success. A fresh, innovative point of view applied to the right topic yields a tutorial that will practically write itself.

Analogy is an effective technique that sets tutorials apart from run of the mill technical documentation. Virtually all significant hardware and software problems have analogs outside of computer science. Textbooks cannot develop analogies for more than a few aspects of the material they treat without ranging far afield. Yet the tutorial, because of its limited scope, responds perfectly to the use of analogy.

Historical perspective is another useful trick. A general solution is only as interesting as the general problem it solves, and the tutorial provides an ideal format for discussion of problems, as well as solutions.

Innovative modelling can also pay off, to the extent that it is effective; but this little trick is fraught with risk. The author who devises his own paradigm has guaranteed originality and a fresh perspective right off the bat. If the model is effective, he might just become as famous as Hollerith with punched cards; Baudot with character codes; or Hamming, whose simple concept of "distance" sold thousands of books. However, if the model is not effective, the tutorial fails. It is as simple as that, and there is no middle ground, but perhaps someone will print that questionable model as a humerous bit of satire.

The writing style matters, in a tutorial, because virtually identical information is available from many other sources. Dr. De Jong's application note in this issue provides the only description of an AIM Notepad you will find anywhere; in contrast, there are countless articles on even such an unlikely tutorial subject as writing articles. If you survived that freshman English composition course, enough said. But if you opted instead for a tensor calculus elective, some common sense guidelines will make a whole lot of difference.

Short sentences win big. Use first person, present tense, active voice. Avoid any grammatical construction you can't identify by name. Resist all impulse to employ parentheses, quotation marks, footnotes or dashes. And when in doubt, triple space the manuscript, leaving your editor plenty of room to ply his trade.

If you ever wanted to write a book of nonfiction, a tutorial is really the ideal place to start. It only takes a few idle evenings, it requires a format in which it is difficult to bog down, and you can always use it as Chapter 27 of your hardcover best seller. Besides, what could provide more motivation than your first royalty check? That check is only five typewritten pages away.

Reviews

MICRO has published very few product reviews, in the past, largely because of uncertainty about how reviews should be solicited, prepared and presented. MICRO will publish many more software, hardware and book reviews in future issues. This is how we plan to go about it.

All product reviews must be solicited by the magazine. MICRO will not publish a review that simply shows up in the mail, because the act of writing an unsolicited review implies that the author has strong feelings about the product, one way or another, else he would not have troubled to prepare an unsolicited manuscript.

The product must be submitted for review by the manufacturer. This is only fair, because a reviewer should feel free to make negative comments, and a manufacturer should be able to enjoy his monthly MICRO without encountering those negative comments completely unexpectedly.

The manufacturer of a product, or the author of a book or program, must receive a copy of the review, prior to publication. Although manufacturers will not have the right to modify or suppress unfavorable reviews, they will be able to make comments or rebuttals and offer additional insights that the reviewer might have missed.

Software and books submitted for review will become the property of the reviewer. Hardware will be provided, and the reviewer will have the option of purchasing this hardware at dealer cost.

What's in it for me?

More and more frequently, of late, manuscripts have arrived from writers who wished to retain exclusive ownership of their articles. MICRO has received copyrighted manuscripts, and a few authors have declined to fill out the ominously worded MS cover sheet.

Any piece of text can be copyrighted simply by writing "copyrighted by", and your name, and the date, and the magic symbol ©, somewhere near the front of the text. You can copyright your article, your computer program, your life story, or your laundry receipt. It is not necessary to hire a lawyer and send Uncle Sam a draft.

Now, your article remains your article whether it is copyrighted or not, and plagiarism of uncopyrighted material is plagiarism none the less. However, if your article is copyrighted, this means that it cannot be Xeroxed, it cannot be quoted at length, it cannot be transcribed or typeset, and it cannot be printed in any magazine. There is no point in sending a copyrighted article to MICRO, unless you think the editor will enjoy reading it, because enjoy it is about all he can do.

MICRO is a business, like any other corporation, that tries to earn a profit purchasing raw materials, adding value by incorporating these materials into a marketable commodity, and selling the commodity. Articles comprise the raw materials; the magazine is the final product. The added value consists of editing, typesetting, proofing, paste-up, illustration, program listing generation, printing, distribution, and all the other activities that make Chelmsford one frightfully busy place to work.

This means that if you want to retain exclusive ownership of your article, and you also want to share it with others by printing it in MICRO, the hard facts of life require you to request an advertising rate card.

When you submit an article for publication, the editor and publisher collaborate with you to produce a piece that reflects substantial effort on both sides. You receive an initial royalty payment, based upon the size of the piece, in return for the inspiration and effort you have expended on its preparation.

You remain the author, and the article will never appear in print without your byline. You may receive additional payments, called residuals, whenever the article is reprinted or incorporated into another publication such as *Best of MICRO*. Residuals are based on the sales of the auxiliary publication and often exceed the original royalty payment.

Your article will not be published by anyone except MICRO without your permission. In return, of course, you must agree not to publish the article, on your own, without MICRO's permission. As with most any magazine, MICRO would be proud to see your article incorporated into your very own book, presented at a conference, or included in an anthology. It appeared here first, after all, and imitation is a sincere, if unprofitable, form of flattery. Classic works of both fiction and nonfiction have appeared in magazines prior to publication as books.

What if I don't get published?

Rejecting an article is the most difficult task any editor will be called upon to perform, especially when the copy deadline draws near and space remains to be filled. The only common reasons for rejecting articles submitted to MICRO are:

Too short

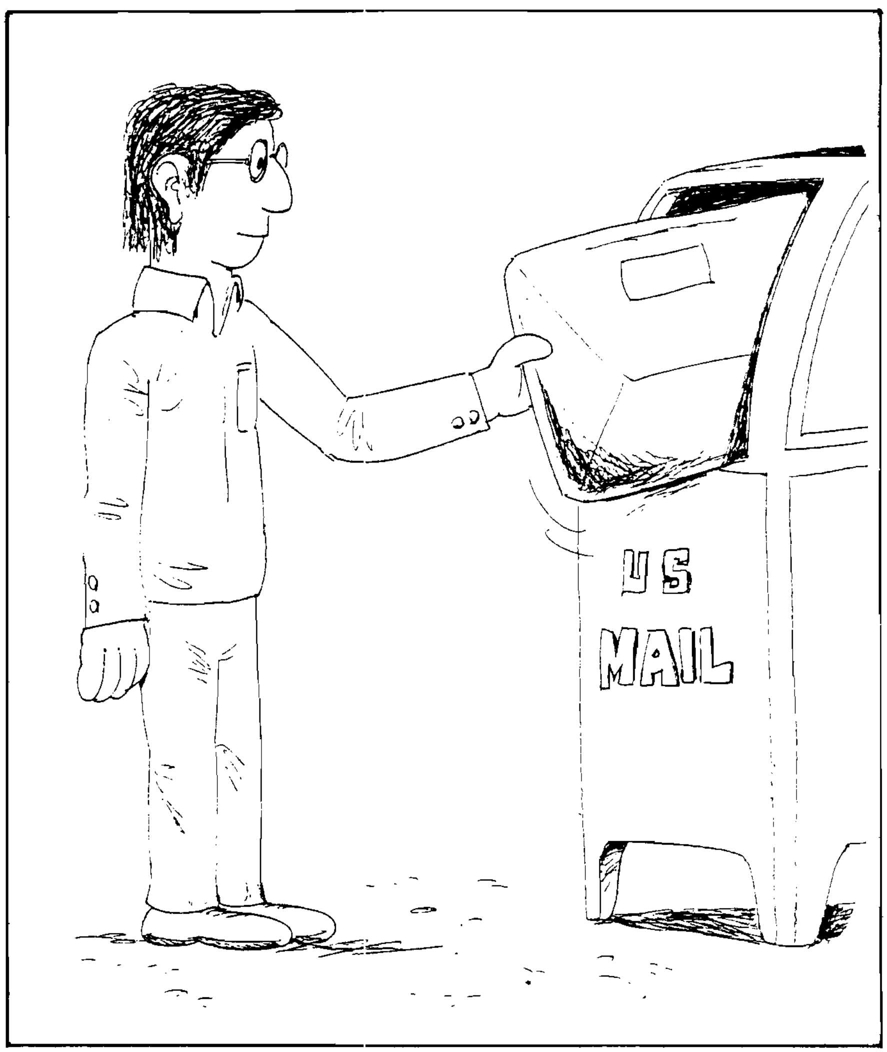
Nothing new

Incomplete

and, very rarely, just entirely too difficult to prepare for publication. These pitfalls are surely easy enough to avoid. If your article is at least one page long and reflects original work, it will be published. If it is incomplete, you will be asked to supply additional material prior to publication. This might involve answers to questions that were raised and left open, comments to accompany program code, or background information most readers would require.

Every so often an otherwise excellent manuscript simply jams the production machinery. One author's draft stubbornly refused to pass through the copying machine. Another included several yards of program listing, in a language the MICRO systems lab was not equipped to reproduce, all printed using blue ink which is invisible to photographic platemaking equipment. By all means avoid blue ink and electrostatic copies. More to the point, take a minute and consider what is required to convert your manuscript into a magazine article. Have you supplied the basic input required? If so, publication is all but guaranteed.

The editor wishes to thank these persons who contributed their thoughts and assistance: Keating Wilcox, Dann McCreary, Dr. Marvin L. De Jong, Philip K. Hooper, Robert M. Tripp. Illustrations by Bruce Conley.



The Basic Switch™

Attention "Old" Pet™ Owners:

Not sure about the ROM Retrofit Kit from Commodore?
Now you can use **both** sets of Commodore ROMs and others as well.

The Basic Switch allows switch selection of **either** ROM set (your original set or your retrofit set) from Commodore. Plus, Model 15-A includes an additional zero insertion force socket allowing easy use of ROMs like the BASIC Programmer's Toolkit ... concurrently.

Model 14-E The economy model of **The Basic Switch.** Stand alone board and harness without case and case hardware. The free standing unit is ready to accept your ROMs.

Model 14-D Same as Model 14-E but includes attractive protective case and mounted **Basic Switch** board.

Note that Model 14 Series does not allow for expansion ROMs like the BASIC Programmer's Toolkit.

Model 15-A **The Basic Switch** plus ... includes expanded cable assembly and zero insertion force socket. Your 15th ROM simply plugs in ... enabled while either ROM set is selected. Socket 15 may be readdressed by the user for additional flexibility.

The Basic Switch is sold in assembled form only. All models are designed for easy attachment to your Pet with a convenient cable assembly. No soldering or drilling is required. **The Basic Switch** mates with a cable assembly at your primary board, and **does not use** the physical connectors of any Pet ports.

Model 15-A allows you to use the BASIC Programmer's Toolkit without the need for the additional $25.00 board or tying up your ports. And since we've designed the 15th socket to be readdressable, watch for more ROM pacs later in the Fall.

The Basic Switch:		With **installed** ROM Retrofit Kit from Commodore:	With BASIC Programmer's Toolkit *
Model 14-E	$64.95	$149.95	–
Model 14-D	$77.95	$162.95	–
Model 15-A	$99.95	$184.95	$149.95

Model 15-A with installed ROM Retrofit and Basic Programmer's Toolkit: $229.95

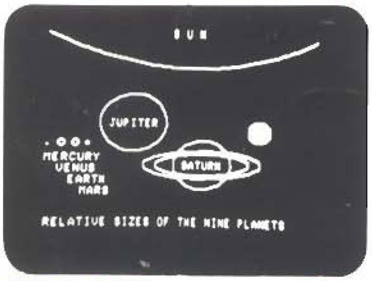
Model 15-A with installed ROM Retrofit and both Toolkits: $274.95

"Old" Pets were shipped with 24 or 28 pin ROMs. You must check which you have, and specify at time of order.

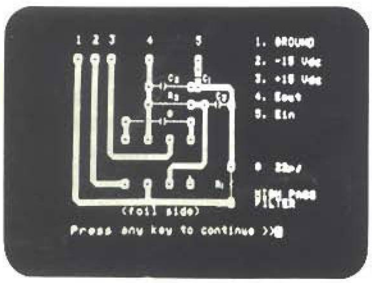
The Basic Switch™ is a product of **Small System Services, Inc.**
900 Spring Garden Street
Greensboro, North Carolina 27403
Telephone 919-272-4867

Pet™ is a trademark of Commodore Business Machines, Inc. of Santa Clara, Calif. The BASIC Programmer's Toolkit is a product of Palo Alto IC's, A Division of Nestar Systems, Inc. North Carolina residents add 4% sales tax. All orders add $2.50 shipping.

Software for the APPLE II



THE PLANETS $15.95



ACTIVE FILTERS $24.95

FOOTBALL PREDICTIONS $19.95

STATE CAPITALS $9.95

FORMAT

PROGRAMMA's FORMAT (Version 1.0) is a command oriented text processor designed to be fully compatible with PIE (PROGRAMMA Improved Editor).

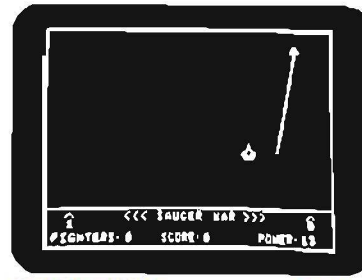
FORMAT's system of imbedded commands (within the text) give it an ease of operation similar to text formatters found on some mini-computers.

FORMAT features right margin justification, centering, page numbering, and auto-paragraph indent.

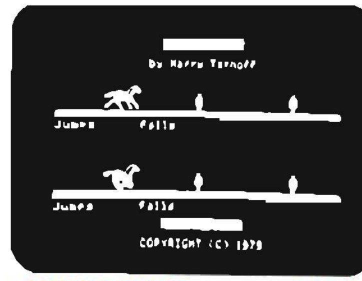
The following commands are available with FORMAT:

.ad		Begin adjusting right margins
.bp	n	Begin page numbered n
.br		Cause a line break
.ce	n	Center next n lines without fill
.fi		Start filling output lines
.fo	t	Foot title becomes t
.he	t	Head title becomes t
.in	n	Indent n spaces from left margin
.li	n	Literal, next n lines are text
.ll	n	Line length including indent is n
.ls	n	Set line spacing to n
.m1	n	Top spacing including head title
.m2	n	Spacing after heading title
.m3	n	Spacing before foot title
.m4	n	Bottom spacing including foot title
.na		Stop adjusting right margins
.nf		Stop filling output lines
.pl	n	Page length is n lines
.pp	n	Begin paragraph= .sp, .fi, .ti n
.sp	n	Space down n lines, except at top
.ti	n	Temporary indent of n
.ul	n	Underline next n input lines

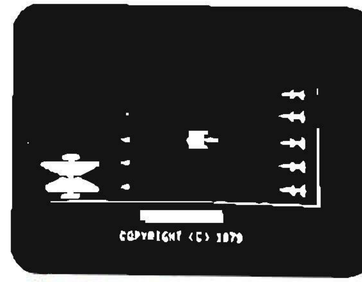
All orders include 3% postage and handling.
Apple II is a registered trademark of Apple Computer, Inc.
California residents add 6% Sales Tax
VISA & MASTERCHARGE accepted.



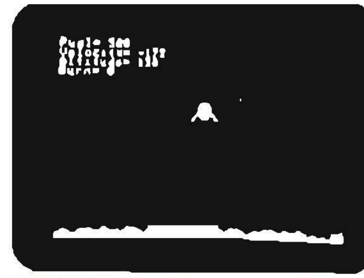
SAUCER WAR $15.95



CANTER DOWNS $15.95

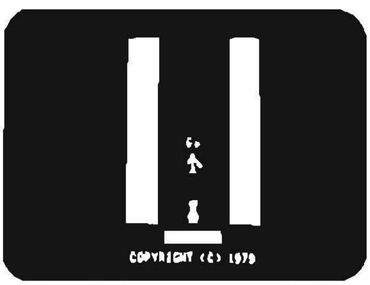


BATTLESTAR I $15.95

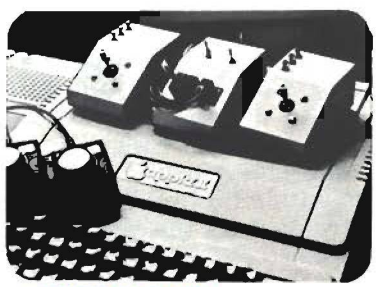


LUNAR LANDER $9.95

PROGRAMMA



SPEEDWAY $15.95



JOY STICK $49.95
EXPAND-A PORT $49.95

PROGRAMMA INTERNATIONAL, Inc.
3400 Wilshire Blvd.
Los Angeles, CA 90010
(213) 384-0579
384-1116
384-1117

Dealer Inquiries Invited

Software Products