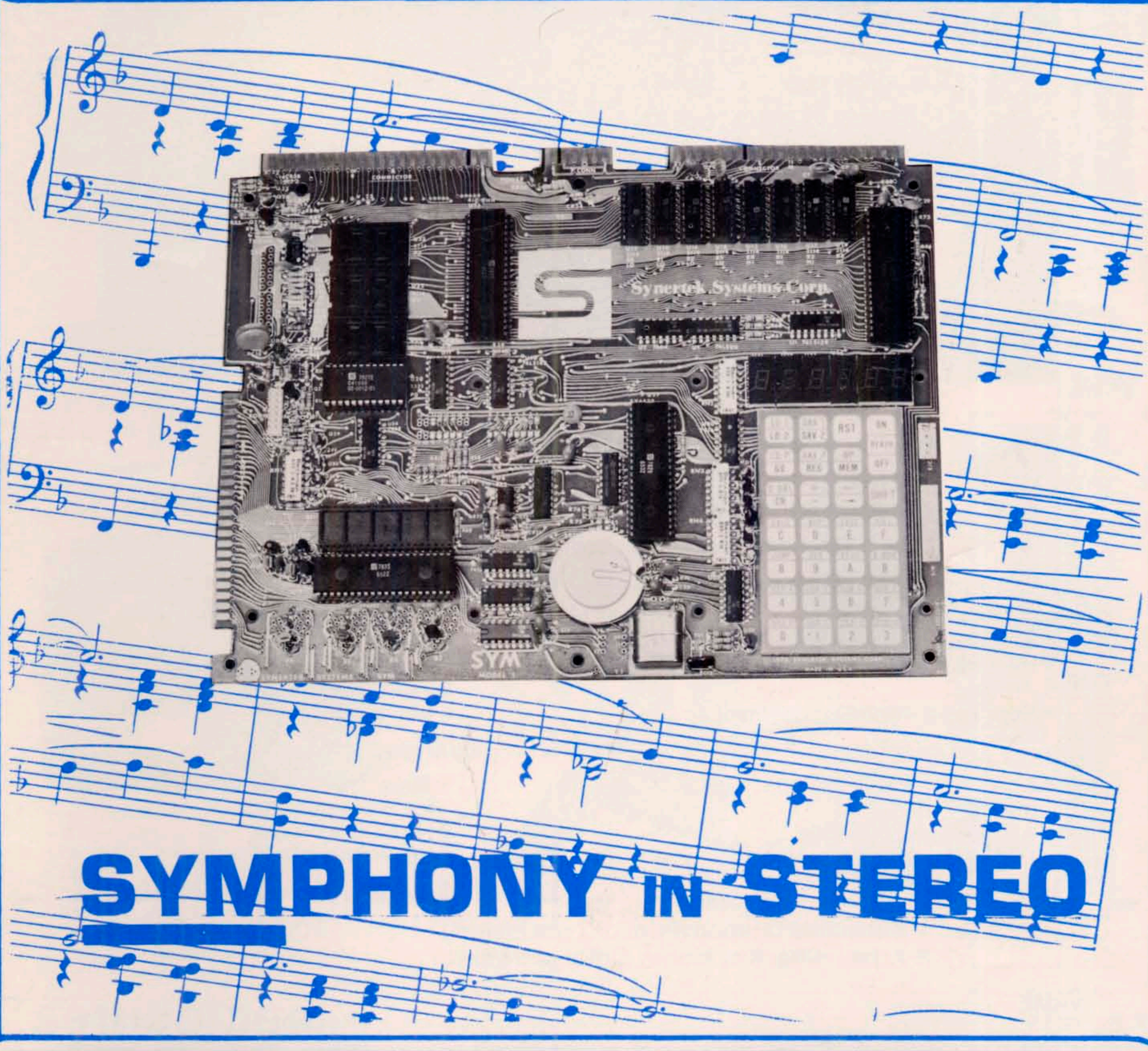
MICRO™

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



SYMPHONY IN STEREO

NO 13 June 1979 ~~$2.00~~ SPECIAL $1.50

PERFECT AIM

ATTRACTIVE FUNCTIONAL PACKAGING FOR YOUR AIM-65 MICROCOMPUTER

- Professional Appearance
- Striking Grey and Black Color Combination
- Protects Vital Components

ENGINEERED SPECIFICALLY FOR THE ROCKWELL AIM-65

- All Switches Accessible
- Integral Reset Button Actuator
- Easy Paper Tape Replacement

EASILY ASSEMBLED

- Absolutely No Alteration of AIM-65 Required
- All Fasteners Provided
- Goes Together in Minutes

MADE OF HIGH IMPACT STRENGTH THERMOFORMED PLASTIC

- Kydex 100*
- Durable
- Molded-In Color
- Non-Conductive

AVAILABLE FROM STOCK

- Allow Three to Four Weeks for Processing and Delivery
- No COD's Please
- Dealer Inquiries Invited

TO ORDER: 1. Fill in this Coupon (Print or Type Please)
2. Attach Check or Money Order and Mail to:

NAME ______________________

STREET ______________________

CITY ______________________

STATE ______________ ZIP ________

Please Ship Prepaid ______ SAE 1-1(s)
@ $43.50 each
California Residents Please Pay
$46.33 (Includes Sales Tax)

enclosures group

753 bush street
san francisco, california 94108

*TM Rohm & Haas

Patent Applied For

MICRO™

June 1979
Issue Number Thirteen

Table of Contents

Staff

Editor/Publisher
Robert M. Tripp

Business Manager
Maggie E. Fisher

Circulation Manager
Carol A. Stark

Distribution
Eileen M. Enos
Janet Santaguida

Micro-Systems Lab
James R. Witt, Jr.
Stephen L. Allen

Comptroller
Donna M. Tripp

Chief Gofer
Fred Davis

MICRO™ is published monthly by:
MICRO Ink, Inc.
34 Chelmsford Street
Chelmsford, Massachusetts
617/256-5515
Mailing address for all correspondence, subscriptions and address changes is:
MICRO
P.O. Box 6502
Chelmsford, MA 01824
Application to mail at second-class postage rates is pending at: Chelmsford, MA 01824.
Publication Number: COTR 395770
Subscription in United States:
$15.00 per year/12 issues.

Advertiser's Index

POWERSOFT, INC.

P. O. BOX 157
PITMAN, NEW JERSEY 08071
(609) 589-5500

products for the APPLE II

ADDRESS FILE GENERATOR — $19.95

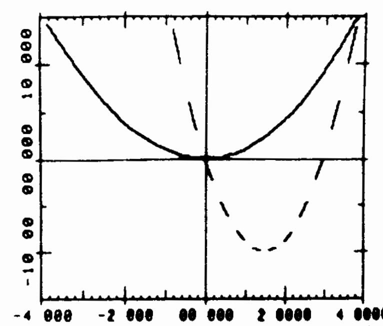
A professional piece of software which allows the user to create four different types of address files: a) Holiday File, b) Birthday File, c) Home Address File, and d) Commercial Address File. The program contains a menu of seven major commands: 1) Create a File, 2) Add to File, 3) Edit File, 4) Display File, 5) Search File, 6) Sort File, and 7) Reorganize File. Most of the major commands have subordinate commands which adds to the flexability of this powerful software system. We doubt you could buy a better program for maintaining and printing address files. REQUIREMENTS: Disk II, Apple Printer Card, 32K of memory with Applesoft Rom Card or 48K of memory without Applesoft Rom Card.

SUPER CHECKBOOK — $19.95

A totally new checkbook program with a unique option . . . Bar Graphs. These bar graphs, outputed to a printer or video screen, provide trend analysis data on code expense, income, expenses, or gain/loss on a month by month basis. The program contains a total of fourteen options: 1) Check/Deposit Entry & Modification, 2) Reconciliation of Checks or Deposits, 3) Sort by Check Number, 4) Sort by Code for Year, 5) Sort by Code for Month, 6) Output Year to Date, 7) Output Month Activity, 8-11) Printer/Video Plot Trend Analysis-Bar Graphs, 12) Account Status, 13) Reconciled Check Status, and 14) Quit. An excellent program for maintaining your checkbook, or that of a small business. REQUIREMENTS: Disk II, 32K of memory with Applesoft Rom Card or 48K of memory without Applesoft Rom Card.

FUNCTION GRAPHS AND TRANSFORMATIONS — $14.95

This program uses the Apple II high resolution graphics capabilities to draw detailed graphs of mathematical functions which the user defines in Basic syntax. The graphs appear in a large rectangle whose edges are X and Y scales (with values labeled by up to 6 digits). Graphs can be superimposed, erased, drawn as dashed (rather than solid) curves, and transformed. The transformations available are reflection about an axis, stretching or compressing (change of scale), and sliding (translation). The user can alternate between the graphic display and a text display which lists the available commands and the more recent interactions between user and program. Expected users are engineers, mathematicians, and researchers in the natural and social sciences; in addition, teachers and students can use the program to approach topics in (for example) algebra, trigonometry, and analytic geometry in a visual, intuitive, and experimental way which complements the traditional, primarily symbolic orientation. REQUIREMENTS: 16K of memory with Applesoft Rom Card or 32K of memory without Applesoft Rom Card.



GAMES

Apple Casino	$ 9.95
Apple Derby	9.95
Apple II Organ	19.95
Cubik	9.95
Radar Interceptor	9.95
Rocket Pilot	9.95
Saucer Invasion	9.95
Space Maze	9.95
Star War	9.95
Swarms	14.95
Wampus Hunt	9.95

SYSTEMS SOFTWARE

Memory Dump	$ 7.45
Program Unload	7.45
File Editor	24.95
Assembler	24.95

(File Editor required for use with Assembler)

APPLICATIONS

Automotive Diagnosis	$14.95
Basic Statistics	9.95
Electrical Engineering I	9.95
Statistics I	14.95
Vector Analysis	9.95

FINANCIAL

Financial Wizard	$ 9.95
Financial Wizard II	9.95
Financial Wizard III	9.95
Financial Wizard IV	9.95

(Programs recorded on cassette tape. Documentation included.)

HARDWARE

Light Pen	$34.95

(Includes Demo-Software)

Call or write for our free SOFTWARE & ACCESSORIES CATALOG

DEALER INQUIRIES INVITED

POWERSOFT, INC.

P. O. BOX 157
PITMAN, NEW JERSEY 08071
(609) 589-5500

- Master Charge and VISA orders accepted.
- Include $1.00 for shipping and handling.
- New Jersey residents add 5% sales Tax.

(Prices subject to change.)

Harmonic Analysis For the Apple

Fourier Analysis is a powerful tool in many fields, but the number of calculation is requires makes it very laborious to perform by hand. A program in Applesoft Floating Point BASIC lets the APPLE II do the work.

Charles B. Putney
1085 Unguowa Road
Fairfield, CT 06430

One of the most important tools of modern technology is the oscilloscope. They can be found everywhere from your local auto repair shop to the hospital operating room and by the thousands in laboratories around the world. The people that use them know that the shape of the squiggly line which appears on the scope screen can tell whether their 'patient' is sick or well and if the former, what to do about it. Sometimes, however, simple visual study of a scope picture does not provide enough information. In this case, the modern day scientist or engineer can resort to quite sophisticated (and expensive) equipment to automatically dissect the picture and to spew out all sorts of numerical data concerning same. This effortless analysis of wave forms is a fairly recent technical development. It was not always quite so easy.

Before the advent of modern electronic computer technology, the analysis of a particular wave form was an all day or all week affair. First, one started with a graph of the function which was scaled to cycle every revolution of 360°. This was divided into equal angular steps and through careful measurements a table of x,y coordinates for each step on the curve was developed. The curve shown in Figure 1 illustrates a representative plot of data which could have been collected from the vibrations of a gasoline engine, torque variations of a synchronous motor or some other physical phenomenon.

The next step was to calculate the coefficients of the terms of a Fourier series. This calculation is a repetitive 'number crunching' exercise and was best done by two or more people armed with mechanical calculators. The team approach was advisable because, with the hundreds or even thousands of calculations required, mistakes were inevitable. At this point, numbers were available for the design of counterbalance weights for your engine, reshaping of poles of the synchronous motor, etc.

The mathematical proof of the fact that any single valued periodic function, such as the one in Figure 1, can be defined by an infinite series and the method for calculation of the coefficients for the terms of the series was developed by Jean Fourier, a French mathematician (1768-1837). The series which he investigated and which is given his name is:

$$
\begin{aligned}
Y = a0 &+ a1(\mathrm{Sin}\ X) + b1(\mathrm{Cos}\ X)\\
&+ a2(\mathrm{Sin}\ 2X) + b2(\mathrm{Cos}\ 2X)\\
&+ a3(\mathrm{Sin}\ 3X) + b3(\mathrm{Cos}\ 3X)\\
&+ a4(\mathrm{Sin}\ 4X) + b4(\mathrm{Cos}\ 4X)\\
&+ a5(\mathrm{Sin}\ 5X) + b5(\mathrm{Cos}\ 5X)\\
&+ \ldots\ldots\ldots \text{etc.}
\end{aligned}
$$

It is easy to see that, if it is desired to carry a calculation for a Fourier series out to say the tenth harmonic, a lot of sines and cosines get into the action, in fact, ten of each for each point of the curve being investigated. The sheer magnitude of the pencil-pushing type calculation tasks that some of the mathematical geniuses of the past, such as Fourier, set before themselves staggers the imagination.

Fortunately, for most engineering problems, the relative significance of harmonics above the fifth are slight and most curves can be defined to sufficient accuracy by a Fourier series with a cutoff at this level. (One of my old text books describes a hydraulic penstock vibration problem involving the beat frequency between the 17th and 18th harmonics of the system!) The accompanying program, written in Applesoft II Floating Point BASIC, calculates a listing of coefficients of each term to the fifth harmonic and continues to show a calculated plot of the input data curve and all five harmonics within a couple of minutes. Admittedly, this does not match the speed of a fast Fourier transform system, but it sure beats the old way.

Harmonic analysis of the data listed for Figure 1 with this program yields the following information:

1. The curve is defined by the equation:

$$
\begin{aligned}
Y = 4.008 &+ 2.39\,\mathrm{Sin}(X) - 0.19\,\mathrm{Cos}(X)\\
&- 0.49\,\mathrm{Sin}(2X) - 0.50\,\mathrm{Cos}(2X)\\
&- 0.13\,\mathrm{Sin}(3X) + 0.12\,\mathrm{Cos}(3X)\\
&+ 0.23\,\mathrm{Sin}(4X) - 0.08\,\mathrm{Cos}(4X)\\
&- 0.07\,\mathrm{Sin}(5X) + 0.07\,\mathrm{Cos}(5X)
\end{aligned}
$$

2. The average of the curve is offset from zero about Y = 4.

If this data and results had been developed with respect to say a vibration problem, it could probably be safely assumed that if the second harmonic vibration component were taken care of, the system would be satisfactory. On the other hand, it might be desired to provide for a flexible mounting to absorb a deflection amounting to the difference between the curve average and the maximum deflection of 7.3. A vertical expansion of the plot of this curve can be obtained by subtracting 4 from the Y component of the input data. This will yield a graph balanced about the X axis as illustrated by the 'average line' in the figure.

Directions for running the program are pretty well built into the listing. After the initial instruction page, the form of the X or angular component must be inputted. (D for degrees or R for Radians followed by a **RETURN**) Then the data for each point of the curve being analyzed must be entered as X,Y (for example, Figure 1, Point 1 would be entered as 30,4.3 **RETURN**) until all points are in the computer, then enter Done, Done **RETURN**. If a mistake is made while entering any of the data points, the program must be restarted. Use a Control C to get out and start over with a **RUN**. Note, that zero degree X and the 360° X are the same from the definition of a periodic wave, so one or the other of these points should be entered but not both of them. It is not necessary to input the data points in order and any 360° span may be used (for example − 180° to + 180°). However, the plotted graph with this program will always come out starting at the zero position.

For those who don't have an APPLE, the program can be used as far as instruction 1225 without missing output of the real important results from the analysis, the term coefficients. While checking out this program, I recalculated examples from several old textbooks and without exception, I found at least one error in the answer listings in each one of them. Needless to say, this created big headaches for the students of that era.

μ

```
]LIST

10   HOME : VTAB 5: HTAB 13: PRINT
      "HARMONIC ANALYSIS": PRINT
20   PRINT " THIS PROGRAM CALCULAT
      ES COEFFICIENTS OF FOURIER S
      ERIES TO THE FIFTH HARMONIC"

30   PRINT "OF PERIODIC FUNCTIONS
      F(Y)=F(X)"
40   PRINT "FUNCTIONS MUST MEET TH
      IS CRITERIA:"
50   PRINT : PRINT "    (A)Y IS NO
      T INFINITE"
60   PRINT "     (B)THERE IS ONLY O
      NE VALUE OF Y FOR EVERY VALU
      E OF X"
70   PRINT "     (C)Y HAS ONLY A FI
      NITE NUMBER OF    MAXIMA OR
      MINIMA"
100  PRINT
110  PRINT "MAKE A TABLE OF X,Y V
      ALUES PICKED FROM  THE GRAPH
       FOR EACH SECTION"
120  PRINT "EVEN IF THE FIRST AND
       LAST HALVES OF THEGRAPH ARE
       SYMETRICAL, X,Y VALUES FOR
      ONEFULL CYCLE MUST BE ENTERE
      D"
122  PRINT "START DATA AT POINT #
      1 NOT POINT #0"
125  PRINT "     HIT ANY KEY AND R
      ETURN TO CONTINUE": INPUT Q$

200  HOME : VTAB 2: PRINT "INPUT
      X,Y VALUES OF POINTS ON GRAP
      H       UNTIL ALL INPUTTED. A
      FTER LAST ENTRY     TYPE 'DON
      E,DONE'"
205  PRINT : PRINT "ARE X DATA PO
      INTS EQUAL STEPS DEGREES(D)
      OR RADIANS(R)?"
206  INPUT D$: IF D$ = "D" THEN R
      = 1
207  PRINT : PRINT " INPUT DATA A
      S X,Y"
210  INPUT X$,Y$
220  IF X$ = "DONE" THEN 1000
300 X =  VAL (X$):Y =  VAL (Y$)
301  REM  CALCULATING SUMS OF COE
      FFICIENTS
305 A0 = Y + A0: IF R THEN X = (X
      / 360) * 6.28318
310  FOR I = 1 TO 5
320 A(I) = Y *  SIN (I * X) + A(I
      )

330 B(I) = Y *  COS (I * X) + B(I
      )
340  NEXT I
350 N = N + 1
360  IF T <  ABS (Y) THEN T =  ABS
      (Y)
390  GOTO 210
1000 A0 = (A0 / N): REM  CALCULAT
      ING COEFFICIENT AVERAGES
1010  FOR I = 1 TO 5
1020 A(I) = (A(I) / N) * 2
1030 B(I) = (B(I) / N) * 2
1040  NEXT I
1100  HOME
1110  PRINT "TERMS OF THE FOURIER
      SERIES ARE:"
1120  PRINT : PRINT "TERM #1-(WIL
      L BE ZERO IF GRAPH IS": PRINT
      "SYMMETRICAL)"
1121  PRINT "    ";A0
1122  PRINT "TERM #2":H = 1: GOSUB
      1200
1123  PRINT "TERM #3": GOSUB 1210

1124  PRINT "TERM #4":H = 2: GOSUB
      1200
1125  PRINT "TERM #5": GOSUB 1210

1126  PRINT "TERM #6":H = 3: GOSUB
      1200
1127  PRINT "TERM #7": GOSUB 1210

1128  PRINT "TERM #8":H = 4: GOSUB
      1200
1129  PRINT "TERM #9": GOSUB 1210
```

Tables I through IV list data points for various standard reference curves. It is interesting to go through them to see the harmonic patterns for each. For example, the triangular wave is an all odd harmonic system. The sawtooth wave, which is the basis for many electronic music generators, is the sum of all harmonics to infinity.

Table I
Three Point Triangular

X°	Y
90 ,	1
180 ,	0
270 ,	−1
360 ,	0

Table II
Triangular

X°	Y
−160 ,	−3
−140 ,	−2
−120 ,	−1
−100 ,	0
− 80 ,	1
− 60 ,	2
− 40 ,	3
− 20 ,	4
0 ,	5
20 ,	4
40 ,	3
60 ,	2
80 ,	1
100 ,	0
120 ,	−1
140 ,	−2
160 ,	−3
180 ,	−4

Table III
Sawtooth

X°	Y
20 ,	8
40 ,	7
60 ,	6
80 ,	5
100 ,	4
120 ,	3
140 ,	2
160 ,	1
180 ,	0
200 ,	−1
220 ,	−2
240 ,	−3
260 ,	−4
280 ,	−5
300 ,	−6
320 ,	−7
340 ,	−8
360 ,	0

Table IV
Square

X°	Y
20 ,	1
40 ,	1
60 ,	1
80 ,	1
100 ,	1
120 ,	1
140 ,	1
160 ,	1
180 ,	0
200 ,	−1
220 ,	−1
240 ,	−1
260 ,	−1
280 ,	−1
300 ,	−1
320 ,	−1
340 ,	−1
360 ,	0

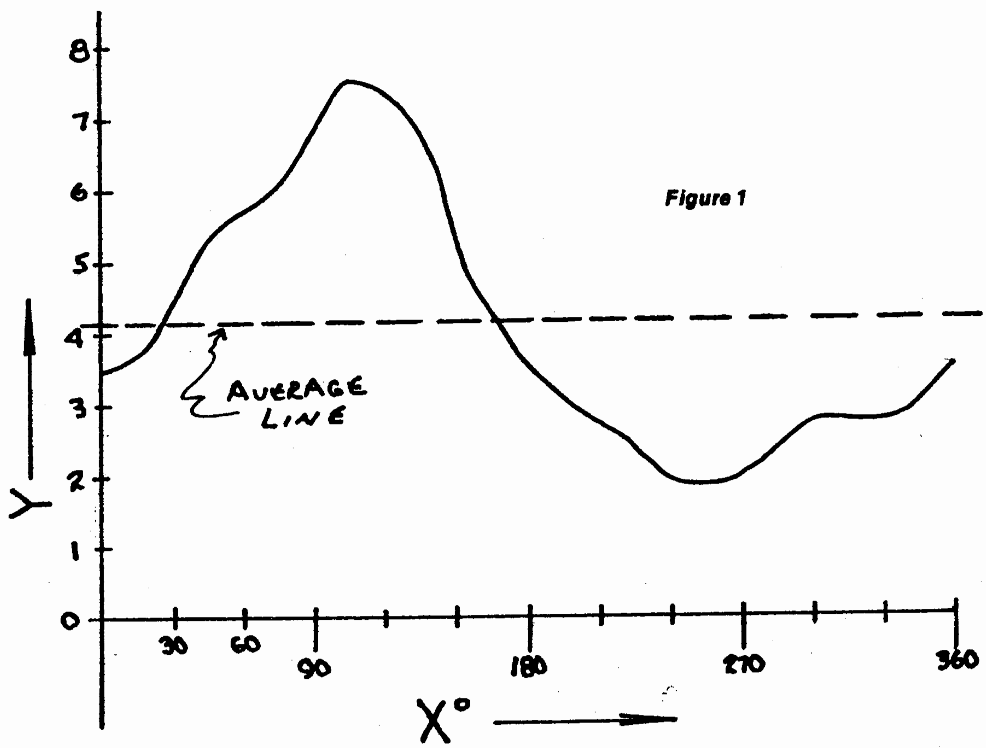


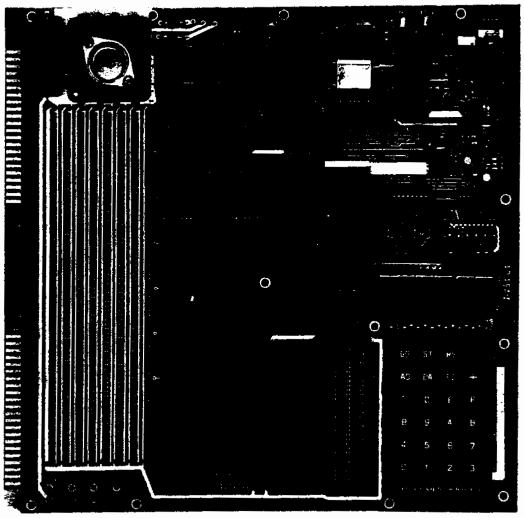
Figure 1

X°	Y	AVE Y
30	4.3	0.3
60	5.7	1.7
90	6.8	2.8
120	7.3	3.3
150	5.2	1.2
180	3.5	−0.5
210	2.7	−1.3
240	1.8	−2.2
270	1.9	−2.1
300	2.7	−1.3
330	2.7	−1.3
360	3.5	−0.5

```
1130  PRINT "TERM #10":H = 5: GOSUB
 1200
1131  PRINT "TERM #11": GOSUB 121
 0
1132  GOTO 1250
1200  PRINT "     ";A(H);" * SIN("
 ;H;"X)": RETURN
1210  PRINT "     ";B(H);" * COS("
 ;H;"X)": RETURN
1225  REM  OK NON GRAPHICS TO HER
 E ALSO CHANGE 1132 GOTO 1132

1250  INPUT "DO YOU WANT A PLOT Y
  OR N?  ";A$
1260  IF A$ = "N" THEN 6010
1290 S = 70:H = 0: REM  SETTING I
 NITIAL SCALE AND CALCULATION
  CONSTANT TO ZERO
1300  PRINT : PRINT : PRINT "PLOT
  OF INPUT DATA CALCULATED TO
  FIFTH    HARMONIC.        Y
 AT 100= ";T: HGR
1301  HCOLOR= 3
1310  HPLOT 0,79 TO 279,79
1320  HPLOT 0,0 TO 0,159
1321  HPLOT 0,79 - S TO 5,79 - S
1322  HPLOT 0,79 + S TO 5,79 + S
1323  HPLOT 269,77 TO 269,81
1330  FOR K = 0 TO 269 STEP 4
1335 X = K / (.75 * 360) * 6.2831
 8531
1340 Y = A0 + A(1) *  SIN (X) + B
 (1) *  COS (X) + A(2) *  SIN
 (2 * X) + B(2) *  COS (2 * X
 ) + A(3) *  SIN (3 * X) + B(
 3) *  COS (3 * X) + A(4) *  SIN
 (4 * X) + B(4) *  COS (4 * X
 ) + A(5) *  SIN (5 * X) + B(
 5) *  COS (5 * X)
1350 Y = (1 / T) * Y * S
1355  IF  ABS (Y) > 79 GOTO 6000
1360  HPLOT K,79 - Y
1390  NEXT K
1500  HCOLOR= 6
1510  PRINT : PRINT : PRINT "FIRS
 T HARMONIC": GOSUB 1600
1520  PRINT : PRINT : PRINT "SECO
 ND HARMONIC": GOSUB 1600
1530  PRINT : PRINT : PRINT "THIR
 D HARMONIC": GOSUB 1600
1540  PRINT : PRINT : PRINT "FOUR
 TH HARMONIC": GOSUB 1600
1550  PRINT : PRINT : PRINT "FIFT
 H HARMONIC": GOSUB 1600
1580  GOTO 2000
1600 H = H + 1: REM  ROUTINE FOR
  PLOTTING HARMONICS
1610  FOR K = 0 TO 269 STEP 4
1620 X = K / (.75 * 360) * 6.2831
 85
1630 Y = A(H) *  SIN (H * X) + B(
 H) *  COS (H * X)
1640 Y = (1 / T) * Y * S
1645  IF  ABS (Y) > 79 THEN 6000
1650  HPLOT K,79 - Y
1660  NEXT K
1670  RETURN
2000  HOME :H = 1: REM  SETTING U
 P COEFFICIENT TABLE
2010  VTAB 5: PRINT "
  COEFFICIENT TABLE"
2020  PRINT : PRINT : PRINT "
  SINES","      COSINES"
2031  PRINT : PRINT "FIRST HARMON
 IC": GOSUB 2100
2032  PRINT : PRINT "SECOND HARMO
 NIC": GOSUB 2100
2033  PRINT : PRINT "THIRD HARMON
 IC": GOSUB 2100
2034  PRINT : PRINT "FOURTH HARMO
 NIC": GOSUB 2100
2035  PRINT : PRINT "FIFTH HARMON
 IC": GOSUB 2100
2040  PRINT : PRINT "CONSTANT = "
 ;A0;"  Y AT 100 = ";T
2045  PRINT : PRINT "HIT SPACE BA
 R FOR REVIEW"
2090 H = 0: GOTO 2200
2100  PRINT A(H),B(H)
2110 H = H + 1: RETURN
2200  REM  REVIEW ROUTINE
2220  IF  PEEK ( - 16384) < 127 THEN
 2220
2230  POKE  - 16368,0
2240  POKE  - 16303,0
2270  IF  PEEK ( - 16384) < 127 THEN
 2270
2280  POKE  - 16368,0
2290  POKE  - 16304,0
3300  GOTO 2220
6000  PRINT "PLOT IS OFF SCALE. T
 RYING AGAIN"
6002 S = S - 10: REM  SHORTEN VER
 TICAL SCALE
6004  FOR K = 1 TO 1000
6006  NEXT K
6008  GOTO 1300
6010  REM  C.B.PUTNEY,
        FAIRFIELD,CONNECICUT
       VER 3/1/79
6020  END
```

SUPERKIM

by MICROPRODUCTS



Here is a powerful microprocessor control system development tool and a complete microcomputer in one low-cost package. The Superkim singleboard computer has more features, more interface and expansion capability with a higher quality design and construction than any other in its class.

Have you got a thousand hours tied up in software for your *KIM-1 BASED control system and now your *KIM-1 is too small? The Superkim is the economical next step for expansion into more RAM, user EPROM and prototype area on one modern, compact, high density, fully assembled and integrated board. The Superkim has more software available than any other singleboard computer since it is totally compatible with KIM-1.

The Superkim has a wide range of appeal to engineers, teachers and industry.

* KIM-1 is a product of MOS Technology.

$445.00

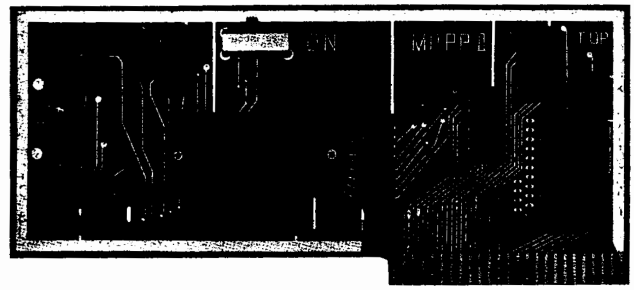
Apple II EPROM

Expand Your ROM Software

Add capability to your system monitor or BASIC for business or other applications. Add commands to the operating system. New operating systems can be put into EPROM memory with our EPROM programmer and plugged directly into your APPLE II board with our EPROM socket adaptor. The **MICROPRODUCTS** EPROM Programmer will program INTEL 2716s, 2758s and other 5-volt replacements for 2716s.

Add to or replace existing APPLE II ROM software with operating systems of your own design. Other software systems similar to PASCAL, FORTH, LISP, APL, FORTRAN, COBOL, ALGOL, other BASIC's, etc. may be incorporated into your APPLE II ROM space.

The EPROM Programmer looks just like memory to the computer and can be configured to program memory locations from 8000 to FFFF for a total of 32K bytes. This means that the EPROMs can be used in computer applications other than the APPLE II, i.e. the **MICROPRODUCTS** Superkim, etc. This turns your APPLE II into a very low cost powerful, software development system.



FEATURES:

- Fully assembled.
- Completely self-contained
- Textool Zero insertion force socket for EPROM
- Onboard 25 volt power supply
- Double sided plated through holes on fiberglass PC board
- Gold plated edge connector
- Fully socketed
- Latest low-power Schottky IC's
- Solder mask

$99.95

MICROPRODUCTS will custom program EPROMs with your program on request

ADVANTAGES:

- Put memory in two empty ROM slots in APPLE II
- Replace memory in existing APPLE II ROM slots
- Add new operating systems to APPLE II
- Programs INTEL 2716 2K byte EPROM's, 2758 1K byte EPROMs and other compatible 5 volt EPROMs
- Put peripheral drivers in permanent memory
- Use APPLE II to program EPROMs for other computers

microproducts
2107 ARTESIA BOULEVARD
REDONDO BEACH, CA 90278
(213) 374-1673

Add a Printer to Your Apple II

With our PC Board that interfaces with the highly popular Southwest Technical Products PR-40 Printer. Both our Printer Interface and PR-40 Printer are available at computer stores.

CENTRONICS TOO!

Printer Interface Apple II to PR-40

Interface is completely assembled, tested and guaranteed. Including: Interconnecting cable, software stored on audio cassette, PC Board which plugs directly into your APPLE II

FEATURES:

- Prints one line at a time when return key is pressed.
- Ideal for writing programs, as you have a complete permanent record of all changes and deletions to your program.
- While in Basic, using the list mode, printer will list the entire program without stopping as the screen scrolls up one line at a time.
- You can refer to an earlier part of your program without the necessity of relisting it on the screen.
- Printer can be called from Basic to print entire contents of video screen.
- When using assembly language mode, one line at a time will be printed in the same format as the video screen.

$49.95

SPECIFICATIONS:
Interface hardware consists of:

- an epoxy fiberglass PC Board
- double-sided
- plated through holes
- silk screen printed legends
- gold plated edge card connector.

microproducts
2107 ARTESIA BOULEVARD
REDONDO BEACH, CA 90278
(213) 374-1673

microproducts
2107 ARTESIA BOULEVARD / REDONDO BEACH / CALIFORNIA 90278 / (213) 374-1673

MICROPRODUCTS/APPLE II

PARALLEL INTERFACE BROCHURE

The MICROPRODUCTS Parallel Ouptput card allows your Apple II* computer to communicate with the outside world.

Applications include:

Printer Interface
Power controller
Tone/Music generator
Plotter Driver
LED/LAMP Driver
Apple II/Superkim Communications interface

Features:

* 8 bits output
* 15 ma output, current sink or source (Can drive L.E.D.s directly)
* TTL or CMOS compatible
* Will go in any slot on the Apple II*
* Data available strobe

General Information:

Data can be transferred to an external device by a STA, STY or STX from assembly language, or a POKE from BASIC. The 8 bits output can drive 2 7-segment L.E.D. displays, relays, SCRs, Printer, or anything which requires up to 8 bits of data.

Application notes and software to drive a Southwest Technical PR-40 printer is available for a nominal $5.00 extra.

Included with the Parallel Interface are instructions on how to interface to a Centronics 779 printer and wiring diagrams for use as a power controller. Additional software and applications notes available.

*APPLE II is a trademark of APPLE COMPUTER, INC.

$44.95

Case of the Missing Tape Counter

The lack of a tape counter on the PET cassette tape unit has led to hours of frustration. The technique presented here provides a fairly automatic method of locating your files on the PET cassette.

William F. Pytlik
6828 Payne
Edwards, CA 93523

The PET has an excellent file management system. Unfortunately, since the PET does not have a tape counter, access to any file or program other than the first requires either an uncanny "touch" to find a file by using FAST FORWARD or an infinite amount of patience weiting for the file management system to find the program at 1-7/8 inches/second. The obvious solution is to use a large number of C-10 or C-5 cassettes. Of course, this solution is costly and requires one to store a large number of tapes.

Fortunately, the PET does have a real time clock and the ability to start and stop the cassette motor via BASIC POKE commands. These two capabilities, combined with the use of constant length files, allow ready access to any program or file on a user created tape.

The use of constant length files implies that every file or program on any cassette has the same space allocated to it regardless of how long the actual program is. This means that some of the tape on the cassette will not actually be used, but the method is much cheaper to use than using C-5/C-10 cassettes for each program/data file.

After experimenting, we found that a maximum FAST FORWARD time of ten seconds is adequate to store the largest program capable to be stored in the PET 8K memory. Of course, this is at the beginning of the tape. As the tape advances, more tape is actually wasted. Still, A C-90 cassette allows approximately 13 large programs or files to be stored and accessed via this method. Access time to the last file on a C-90 cassette is approximately two minutes.

The program shown is pretty self- explanatory and easy to enter.

Usage of this method requires that the program be saved as the first program on every cassette. To use the program, press SHIFT/RUN. After the program is loaded and run, **DO NOT** press STOP/EJECT on the cassette drive. The program will inquire which drive you are using by displaying:

ENTER CASSETTE 1 OR 2

After you enter the number only, the program will present a catalog of all files or programs on that tape. Dummy names will be listed for unused file locations like:

PROGRAM 1
PROGRAM 2

Although a C-90 cassette will hold 13 programs, we have chosen to use only ten.

Next, the program will ask if you wish to read or save a program by displaying:

READ OR CREATE PROGRAM/FILE --
R OR C

The program will then ask which file you wish to read or write by displaying:

WHICH PROGRAM/FILE -- I.E. 1, 2, . . .

If you have entered a number greater than 1, the machine will display:

PRESS F.FWD and HIT RETURN
WHEN READY

(The program will skip the previous step if you ask for program/file number 1 because the tape is already in the correct position). If you enter R to read a file/program, the program will advance the tape to the correct position, stop, and display:

HIT STOP/EJECT AND LOAD AS
USUAL

At this point, you simply load the selected program in the usual manner.

Similarly, if you entered C to create a new file/program, the tape will advance to the selected portion of the tape, stop, and display:

HIT STOP/EJECT
TAPE IS NOW READY TO SAVE NEW
PROGRAM/FILE

Now, you can save any program. If you wish to use descriptive names for your programs or files, just reload this program, change the names in the appropriate data statement, and resave the program as the first file on the cassette.

Since the length of a file allocation is determined by time in seconds (the number 10 in line 300 of the program), the user may change this number to accomodate any length file. Also, since each program occupies a unique well-defined location and the length allocated is for a maximum length file, there is no problem replacing one file/program with another.

We use this method on all our tapes. We also use the program as a subroutine in programs requiring access to other files, i.e., a recipe program. The use of the PET cassette drives becomes simple, quick, and enjoyable, and presents a solution to the case of the missing tape counter mystery.

μ

```
10 REM  THIS PROGRAM ALLOWS THE PET USER TO
20 REM  ACCURATELY POSITION HIS CASSETTE FILES
30 REM  BY USING THE FAST FORWARD FUNCTION OF
40 REM  THE TAPE DRIVE.
50 REM
60 REM
70 PRINT " *** PROGRAM / FILE LOCATOR ***"
80 PRINT
90 INPUT "ENTER CASSETTE 1 OR 2";CA
100 READ X
110 DIM C$(X)
120 FOR I = 1 TO X
130 READC$(I)
140 PRINTC$(I)
150 NEXT I
160 PRINT:INPUT "READ OR CREATE PROGRAM / FILE -- R OR C";R$
170 INPUT "WHICH PROGRAM / FILE --I.E. 1,2,...";WP
180 IF WP = 1 THEN 240
190 REM STATEMENTS 200 AND 210 INITIALIZE THE MOTOR OFF
200 IF CA = 1 THEN POKE 59411,61
210 IF CA = 2 THEN POKE 59456,223
220 PRINT "PRESS F.FWD AND HIT RETURN WHEN READY"
230 REM STATEMENT 240 WAITS FOR RETURN TO BE DEPRESSED
240 GET A1$:IF A1$ = "" THEN 240
250 REM STATEMENTS 260 AND 270 TURN ON SELECTED MOTOR
260 IF CA = 1 THEN POKE 59411,53
270 IF CA = 2 THEN POKE 59456,207
280 T = TI
290 REM STATEMENT 300 WAITS FOR TAPE TO ADVANCE TO SELECTED FILE
300 IF TI<T+(10*60*(WP-1)) THEN 300
310 REM STATEMENTS 320 AND 330 TURN THE MOTOR OFF
320 IF CA = 1 THEN POKE 59411,61
330 IF CA = 2 THEN POKE 59456,223
340 PRINT
350 IF R$ = "R" THEN PRINT "HIT STOP/EJECT AND LOAD AS USUAL"
360 PRINT:IF R$ = "R" THEN 500
370 IF R$ = "C" THEN PRINT "HIT STOP/EJECT"
380 PRINT "TAPE IS NOW READY TO SAVE NEW PROGRAM/FILE"
390 REM CHANGE NUMBER IN STATEMENT 500 TO CHANGE THE MAX
400 REM NUMBER OF PROGRAMS PER CASSETTE
410 REM CHANGE NAMES IN STATEMENTS 510 THRU 600
420 REM TO YOUR PROGRAM NAMES
500 DATA 10
510 DATA "PROGRAM 1"
520 DATA "PROGRAM 2"
530 DATA "PROGRAM 3"
540 DATA "PROGRAM 4"
550 DATA "PROGRAM 5"
560 DATA "PROGRAM 6"
570 DATA "PROGRAM 7"
580 DATA "PROGRAM 8"
590 DATA "PROGRAM 9"
600 DATA "PROGRAM 10"
1000 END
```

The Basic Morse Keyboard

For the HAMs -- here is a way to use your system to make an ASCII keyboard perform as a Morse keyboard. Implemented on an OSI system, the program is in BASIC and should be readily convertible to other systems.

William L. Taylor
246 Flora Road
Leavittsburg, OH 44430

A computer, as with any appliance, should be a useful tool to aid the owner with his daily tasks, or to bring enjoyment.

Being an amateur radio operator and a computer hobbyist, I felt that the computer should aid the operator with his tasks either when operating the station or other activities. From this desire to have the computer as an assistant, I felt that one of the best uses for my computer was to aid me in sending and receiving of the Morse code. With this in mind I went to work developing a program that would allow me to use the ASCII keyboard as a "Morse Keyboard". The program and the interface information in this article will help other amateur radio operators, who own the OSI Challengers with a Model 500 CPU with the PIA port populated, get on the air with the "Morse Keyboard".

First, an explanation of my system is in order. My computer system consists of the system boards sold by Ohio Scientific Instruments. I have the Model 500 CPU with BASIC in ROM. The PIA port is populated with a 6820 PIA, and is addressed at the standard location on the 500 board. The address for the PIA is 63232 decimal or F700 hex on the 500 CPU board. The program was written to service the 6820 at this location. The BASIC program uses the B side of the 6820 as the output and PBO is the specific port. PBO of the PIA is connected to a tone oscillator board to generate the sidetone and a relay driver on the board is used to drive a 12 volt relay that keys the transmitter.

The "BASIC Morse Keyboard" program is written in MicroSoft BASIC and Assembly Language. The Assembly portion of the program is stored in DATA statements, and is entered into user memory with the READ and POKE functions of BASIC. On initialization, the DATA at line 1620 is READ and POKED into memory with the FOR NEXT loop at line 1605. This machine code store subroutine is called at line 15 at the beginning of the program. The machine code routine is stored at hex OCOO. This machine code routine calls up the system monitor to get the ASCII code from the system keyboard. When a key is struck on the keyboard the ASCII equivalent of the letter or number is placed in the accumulator of the 6502 microprocessor. The ASCII character is then stored at hex OFOO where it will be available for the BASIC program to capture it with a PEEK statement. This PEEK statement is located at line 125 of the BASIC program.

The contents of hex OFOO will hold the present keyboard ASCII entry, and after being read with the PEEK statement the character will be stored in the A variable. The contents of variable A is now compared with the contents of a look up table to determine the offset to the Morse element table where the conversion to Morse elements are formed. The ASCII table starts at line 130. The Morse element table starts at line 1500.

The Morse equivalent of the ASCII character is loaded into the string variable A$, and on return from the subroutine the program jumps to a subroutine at line 1000 where the elements of A$ are seperated into the dot-dash elements of Morse code. This seperation is done by loading each seperate element into D$, and if the element is a 1 then a dot is generated in a subroutine at line 1200 through line 1220. If the element read into D$ is the numeral 3 then a jump to the subroutine at line 1300 through 1320 causes a dash to be generated. After each character has been separated and sent to the PIA port the program returns to the input statement line 122. At statement line 122 a jump to the machine code subroutine is executed with the USR function of BASIC. The machine code subroutine causes a jump to the system monitor and the program will loop until a key is depressed on the keyboard.

The subroutine at line 1200 and 1300 generate the Morse elements (dots and dashes). This is done by turning on and off PB0 for a duration of time. For example if a 1 was decoded in the routine at 1000 then PB0 would be turned on (high) for the duration of time contained in the loop at line 1205. This loop (FOR J = 1 TO X:NEXT J) is the dot length. The dot length time element is stored in variable X at line 106. When the loop has timed out then PB0 will be brought low and a return executed. The next Morse element is identified and generated in either the subroutine at 1200 or 1300 depending if it is a dot or a dash. The dot length, dash length, and the length between characters are contained in variables at lines 102 through 109. The H variable at line 102 holds the information that identifies PB0 and turns PB0 on in the subroutines at 1200 and 1300. The R variable contents are used to turn PB0 off. The variable at line 106 is the dot length. The variable at line 108 is the dash length. The variable at line 109 is the length between characters. This variable can be eliminated if desired. The variables X and R can be adjusted for any desired dot dash length. The routine at lines 10, 20 and 30 initialize the PIA. This initialization sets the B port as the output port. The value in variable G at 100 identifies the initial location of the PIA.

The object code subroutine for the program is stored at Hex 0C00. The object code contents are:

```
0C00  20
0C01  ED
0C02  FE
0C03  8D
0C04  00
0C05  0F
0C06  60
```

The tone oscillator board is a straight forward construction project requiring few components. The schematic in Figure 1 shows the schematic for the tone and relay driver board and the components that will be needed to construct the board. Pref board and a wire wrap socket can be used for the construction of the board or you can etch a board if you feel that would be a more desirable method. A printed circuit board layout was not included in this article because it was felt that the user could use any method that was thought best. The connections to the PIA port on the 500 CPU board can be any length of wire. I used ribbon wire and etched a small board that would mate with a Molex male 12 pin connector such as the connectors on the 480 backplane board. This male plug connects to the 500 CPU board at the B side port connector on the 500 CPU board. The power for the Tone board is taken from the computer except the 12 volt DC for the relay. This must be obtained from another source. Be sure that the external power source ground be connected to the tone board ground in order for the relay keying transistor to work correctly.

A note of warning must be given at this point. The memory size must be set to 3071 decimal when bringing up BASIC. This will be for the protection of the machine code routine that is stored above 0C00 Hex.

In conclusion, the program as written does not have any buffer so typing ahead is not possible. This leaves the program open for modifications, such as, installing the buffer. Also I have a version of the program that allows the operator to load ASCII into a memory zone and use this ASCII as preloaded message text. The program and the sidetone keyer works well on the OSI system used at my shack and I have had many pleasurable hours using the "Morse Keyboard" on CW. I wish you the same. Good luck.

μ

Tone Oscillator and Driver Board Parts List

1	1N4001 Diode	
1	Pref board .100 by .100 hole centers	Radio Shack 276-1394
1	555 Timer IC	Radio Shack 276-1723
1	8 Pin wire wrap socket (or 14 pin)	
1	.1 MF Disc capacitor	
1	50 MF Electrolytic capacitor	
3	10K ½ Watt Resistors	
1	2N2222 NPN Transistor	
1	8 Ohm speaker	Radio Shack 40-245
1	12 Volt DC relay	Radio Shack 275-003
1	Male Molex 12 Pin plug KK156	
1	Wire wrap wire 32 Gauge (if used)	
1	Hand wire wrap tool (if used)	

Misc. Wire for connection to computer and external 12 volt power supply. Solder

Note: A 14 pin IC socket can be used for the 555. Only use 8 of the pins. I used a Sigma #62R23-2600 relay for RY1.

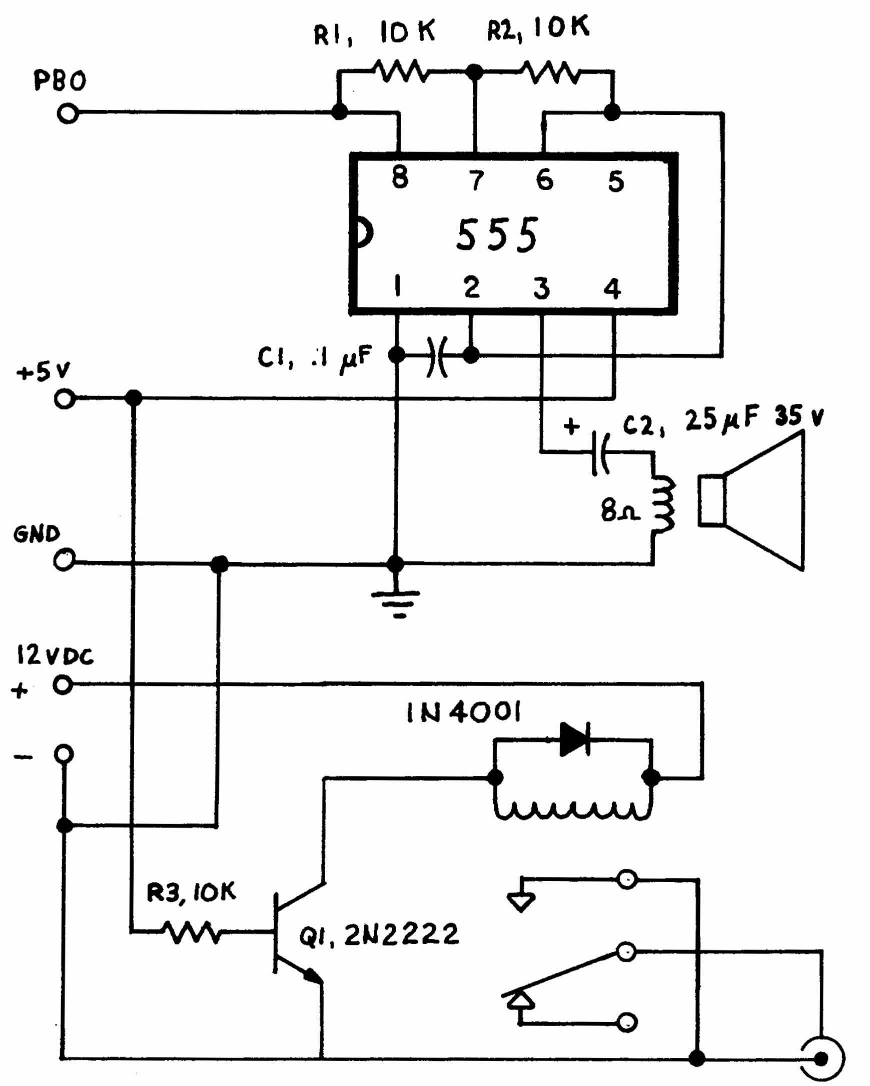


Figure 1

The author wishes to give particular recognition to the article entitled:

"The Morse Master"

which appeared in the January 1979 issue of 73 Magazine, written by William A. Thornburg.

The article provided the concepts upon which this program is based, and gave a program listing for the H8 microcomputer.

```
1 REM MORSE KEYBOARD FEB 1979
2 PRINT" MORSE CODE KEYBOARD"
3 PRINT:PRINT:PRINT:PRINT
5 " ***********READY************"
10 X=63232
15 GOSUB 1600
20 POKE X+1,0:POKE X+3,0: POKE X,0:POKE X+2,255
30 POKE X+1,04:POKE X+3,04
100 G=63232
102 H=1
104 R=0
106 X=25
108 T=100
109 F=150
110 POKE 11,0:POKE 12,12
122 X=USR(X)
125 A=PEEK(3840)
130 IF A= 65 THEN GOSUB 1500
131 IF A= 66 THEN GOSUB 1501
132 IF A= 67 THEN GOSUB 1502
133 IF A= 68 THEN GOSUB 1503
134 IF A= 69 THEN GOSUB 1504
135 IF A= 70 THEN GOSUB 1505
136 IF A= 71 THEN GOSUB 1506
137 IF A= 72 THEN GOSUB 1507
138 IF A= 73 THEN GOSUB 1508
139 IF A= 74 THEN GOSUB 1509
140 IF A= 75 THEN GOSUB 1510
141 IF A= 76 THEN GOSUB 1511
142 IF A= 77 THEN GOSUB 1512
143 IF A= 78 THEN GOSUB 1513
144 IF A= 79 THEN GOSUB 1514
145 IF A= 80 THEN GOSUB 1515
146 IF A= 81 THEN GOSUB 1516
147 IF A= 82 THEN GOSUB 1517
148 IF A= 83 THEN GOSUB 1518
149 IF A= 84 THEN GOSUB 1519
150 IF A= 85 THEN GOSUB 1520
151 IF A= 86 THEN GOSUB 1521
152 IF A= 87 THEN GOSUB 1522
153 IF A= 88 THEN GOSUB 1523
154 IF A= 89 THEN GOSUB 1524
155 IF A= 90 THEN GOSUB 1525
156 IF A= 48 THEN GOSUB 1526
157 IF A= 49 THEN GOSUB 1527
158 IF A= 50 THEN GOSUB 1528
159 IF A= 51 THEN GOSUB 1529
160 IF A= 52 THEN GOSUB 1530
161 IF A= 53 THEN GOSUB 1531
162 IF A= 54 THEN GOSUB 1532
163 IF A= 55 THEN GOSUB 1533
164 IF A= 56 THEN GOSUB 1534
165 IF A= 57 THEN GOSUB 1535
166 IF A= 46 THEN GOSUB 1536
167 IF A= 63 THEN GOSUB 1537
168 IF A= 64 THEN GOSUB 1538
169 IF A= 47 THEN GOSUB 1539
170 IF A= 44 THEN GOSUB 1540
172 GOSUB 1000
175 FOR I= 1 TO F : NEXT I
180 FOR M= 1 TO G9: NEXT M
190 GOTO 122
1000 L= LEN (A$)
1005 FOR I= 1 TO L
1010 R$=MID$ (A$,I,1)
1015 IF R$= "1" THEN GOSUB 1200
1020 IF R$= "3" THEN GOSUB 1300
1025 NEXT I
1030 GOTO 1320
1200 POKE G+2,H
1205 FOR J= 1 TO X: NEXT J
1210 POKE G+2,R
1215 FOR J= 1 TO X: NEXT J
1220 RETURN
1300 POKE G+2, H
1305 FOR J= 1 TO X: NEXT J
1310 POKE G+2, R
1315 FOR J= 1 TO X: NEXT J
1320 RETURN
1500 A$= "13": RETURN
1501 A$= "3111":RETURN
1502 A$= "3131":RETURN
1503 A$= "311" :RETURN
1504 A$= "1"   :RETURN
1505 A$= "1131":RETURN
1506 A$= "331" :RETURN
1507 A$= "1111":RETURN
1508 A$= "11"  :RETURN
1509 A$= "1333":RETURN
1510 A$= "313" :RETURN
1511 A$= "1311":RETURN
1512 A$= "33"  :RETURN
1513 A$= "31"  :RETURN
1514 A$= "333" :RETURN
1515 A$= "1331":RETURN
1516 A$= "3313":RETURN
1517 A$= "131" :RETURN
1518 A$= "111" :RETURN
1519 A$= "3"   :RETURN
1520 A$= "113" :RETURN
1521 A$= "1113":RETURN
1522 A$= "133" :RETURN
1523 A$= "3113":RETURN
1524 A$= "3133":RETURN
1525 A$= "311" :RETURN
1526 A$= "33333" :RETURN
1527 A$= "13333" :RETURN
1528 A$= "11333" :RETURN
1529 A$= "11133" :RETURN
1530 A$= "11113" :RETURN
1531 A$= "11111" :RETURN
1532 A$= "31111" :RETURN
1533 A$= "33111" :RETURN
1534 A$= "33311" :RETURN
1535 A$= "33331" :RETURN
1536 A$= "131313":RETURN
1537 A$= "113311":RETURN
1538 A$= "31113" :RETURN
1539 A$= "31131" :RETURN
1540 A$= "331133":RETURN
1600 FOR R= 3072 TO 3078
1605 READ Q: POKE R,Q: NEXT R
1610 RETURN
1620 DATA 32,237,254,141,0,15,96
```

A SYM-phony in Stereo

All you Symmers who are frustrated because you can not play the music from Star Wars on your systems -- take heart. Here is a program that not only plays music, but plays it in STEREO!

Phillip M. Rinard
2019 Park Ave.
Emporia, KS 66801

Excellent tune player programs for computers abound, but some features of the SYM-1 make it easy to generate stereo music and may be of interest to SYM-1 owners. Such a program also illustrates some of the uses of the on-board UART's (a SY6532 and two SY6522's) and some of the SUPERMON monitor routines. The listing explains the procedures of the program, but a few comments here may be helpful. With no attempt at making use of the memory at the greatest efficiency, each stereo note consists of five bytes: the duration is given by the first byte, then two bytes give the frequency for each of the two stereo tones having that duration. For the program given, the duration is in multiples of about 0.01 second and is timed by counting down in the 6532. The frequency bytes are placed into the latches of the 6522's for use in the timer 1, free-running mode. The 6522's timers generate square-wave outputs with a frequency based on the contents of the latches. The 6532 timer computes when the next 5 bytes should be read and new values placed in the latches.

Two subroutines allow for repeating all or part of the tune. A duration byte of $FF causes a return to the beginning of the tune for a single repeat of the tune up to that point. Upon reading the $FF a second time, the repeat is ignored and the rest of the tune played. If the duration is $00, the tune is over, but immediately begun again as if for the first time. The output port AA used by the 6522 #3 is buffered and thus can drive a speaker after putting about a 200 Ohm resistor between two points of the rightmost buffer's PC holes. As shown on page 4-12 of the SYM Reference Manual, these are points 4 and the one between and below points C and 7. The A port can be buffered by one of the three remaining on-board buffers (or one of your own off the board). Place another 200 Ohm resistor in another set of PC board holes and place the input signal from port A on the exposed wire of the on-board resistor immediately above the transistor. The outputs to both speakers are then available from port AA (see page 4-11 of the SYM Reference Manual for exact pin numbers).

As an illustration of the stereo player, a listing of data for the "Star Wars" music is given. The "notes" of zero frequency provide brief intervals of silence between notes to more realistically immitate a musical instrument. This program and data fit into the 1K of on-board memory provided from the factory. Sockets for 3K more memory are present, as is a socket for a third 6522. If more of this memory is used for extended tunes, then additional programming is necessary similar to that in locations 1B through 21 where "starting" addresses are changed so that as the Y register increments up to 255 the proper note is retrieved. Obviously, an enthusiast could expand on this type of program with the SYM-1 capabilities. μ

SYM-1 STEREO TUNE PLAYER

BY PHILLIP M. RINARD
MAY 1979

MODIFIED BY MIKE ROWE

```
                 SUPERMON REFERENCES

                 ACCESS *     $8B86  ACCESS SUBROUTINE
                 OUTBYT *     $82FA  OUTPUT BYTE SUBROUTINE
                 OUTCHR *     $8A47  OUTPUT CHARACTER TO DISPLAY
                 ACR    *     $A00B  AUXILIARY CONTROL REGISTER
                 ACRX   *     $AC0B  ACR DIFFERENT VIA
                 IER    *     $A00E  INTERRUPT ENABLE REGISTER
                 IERX   *     $AC0E  IER DIFFERENT VIA
                 TOL    *     $A006  TIMER LATCH
                 TOC    *     $A005  COUNTER
                 TOLX   *     $AC06  TIMER LATCH
                 TOCX   *     $AC05  COUNTER
                 STIME  *     $A41E  6532 TIMERS
                 RTIME  *     $A404

0000                     ORG  $0000

0000 00          LOW    =     $00    LOW POINTER
0001 00          HIGH   =     $00    HIGH POINTER
0002 00          REPEAT =     $00    REPEAT COUNTER

                 TUNE TABLE POINTERS

0003 00          TUNES  =     $00    OFFSET FOR FIRST TUNE
0004 70          FIRST  =     $70    FIRST TUNE LOW
0005 02                 =     $02    FIRST TUNE HIGH  0270
0006 70          SECOND =     $2F    SECOND LOW
0007 03                 =     $03    SECOND HIGH  032F
0008 FF          END    =     $FF    END OF TUNE TABLE

                 MAIN PROGRAM

0200                     ORG  $0200

0200 20 86 8B    MAIN    JSR   ACCESS ALLOW WRITING TO SYSTEM RAM
0203 A9 C0               LDAIM $C0    SET ACR6,7 = 1
0205 8D 0B A0            STA   ACR    FOR TIME 1, FREE RUNNING
0208 8D 0B AC            STA   ACRX   WITH OUTPUT ENABLED
020B A9 7F               LDAIM $7F    DISABLE IRQ
020D 8D 0E A0            STA   IER    WITH
```

	DURATION	FREQ ONE		FREQ TWO	
270	05	00	00	00	00
275	60	BC	03	DA	0B
27A	05	00	00	00	00
27F	30	FC	04	F7	09
284	05	00	00	00	00
289	07	ED	05	DA	0B
28E	05	00	00	00	00
293	07	A7	06	4C	0D
298	05	00	00	00	00
29D	07	77	07	EE	0E
2A2	07	00	00	00	00
2A7	60	BC	03	DA	0B
2AC	05	00	00	00	00
2B1	30	FC	04	F7	09
2B6	05	00	00	00	00
2BB	07	ED	05	DA	0B
2C0	05	00	00	00	00
2C5	07	A7	06	4C	0D
2CA	05	00	00	00	00
2CF	07	77	07	EE	0E
2D4	07	00	00	00	00
2D9	60	BC	03	DA	0B
2DE	05	00	00	00	00
2E3	30	FC	04	F7	09
2E8	05	00	00	00	00
2ED	07	98	05	E1	08
2F2	05	00	00	00	00
2F7	07	ED	05	E9	07
2FC	05	00	00	00	00
301	07	98	05	E1	08
306	05	00	00	00	00
30B	60	A7	06	F7	09
310	20	00	00	00	00
315	07	FC	04	00	00
31A	05	00	00	00	00
31F	07	FC	04	00	00
324	05	00	00	00	00
329	07	F7	09	F7	09
32E	FF				

```
0210 8D 0E AC           STA    IERX    IER = 0
0213 A9 00              LDAIM  $00     INIT TUNE TABLE POINTER
0215 85 03              STA    TUNES
0217 A4 03      REST    LDY    TUNES   GET TUNE TABLE POINTER
0219 B1 04              LDAIY  FIRST   GET LOW ADDRESS
021B 85 00              STA    LOW     FROM TABLE
021D C9 FF              CMPIM  $FF     END OF TABLE ?
021F D0 01              BNE    OKAY
0221 00                 BRK            ELSE, RETURN TO MONITOR
0222 C8         OKAY    INY            BUMP POINTER
0223 B1 04              LDAIY  FIRST   GET HIGH
0225 85 01              STA    HIGH
0227 A9 01              LDAIM  $01     INIT REPEAT INDEX
0229 85 02              STA    REPEAT
022B A0 00      RESET   LDYIM  $00     START THE TUNE
022D B1 00      DUR     LDAIY  LOW     READ THE DURATION
022F F0 E6              BEQ    REST    IF ZERO, RESTART THE TUNE
0231 C9 FF              CMPIM  $FF     IF DURATION = FF
0233 D0 06              BNE    OUTB    CHECK TO SEE IF REPEAT HAS
0235 C6 02              DEC    REPEAT  TEST SECOND TIME THROUGH
0237 10 F2              BPL    RESET   REPEAT UNTIL MINUS
0239 30 F2              BMI    DUR     CONTINUE
023B 20 FA 82   OUTB    JSR    OUTBYT  DISPLAY DURATION
023E AA                 TAX            PLACE DURATION IN X
023F C8                 INY            INCREMENT DATA INDEX
0240 B1 00              LDAIY  LOW     READ THE LOWER PORTION OF
0242 20 FA 82           JSR    OUTBYT  FREQUENCY ONE, DISPLAY IT
0245 8D 06 A0           STA    TOL     AND STORE IT IN A LATCH
0248 C8                 INY            INCREMENT THE DATA INDEX
0249 B1 00              LDAIY  LOW     READ THE HIGHER PORTION OF
024B 20 FA 82           JSR    OUTBYT  FREQ. ONE, DISPLAY IT
024E 8D 05 A0           STA    TOC     AND STORE IT IN A LATCH
0251 C8                 INY            INCREMENT THE DATA INDEX
0252 B1 00              LDAIY  LOW     READ THE LOWER PORTION OF
0254 8D 06 AC           STA    TOLX    FREQ 2 AND STORE IT
0257 C8                 INY            INCREMENT THE DATA INDEX
0258 B1 00              LDAIY  LOW     READ THE HIGHER PORTION OF
025A 8D 05 AC           STA    TOCX    FREQ 2 AND STORE IT
025D A9 77      START   LDAIM  $77     START THE 6532 TIMER
025F 8D 1E A4           STA    STIME   ON A 0.01 SECOND COUNT
0262 A9 01      REFR    LDAIM  $01     REFRESH THE
0264 20 47 8A           JSR    OUTCHR  DISPLAY
0267 AD 04 A4           LDA    RTIME   READ THE 6532 TIMER
026A 10 F6              BPL    REFR    AND WAIT FOR TIME OUT
026C CA                 DEX            DECREMENT THE DURATION
026D D0 EE              BNE    START   RESTART TIMER IF NOT ZERO
026F C8                 INY            INCR. THE NOTE INDEX
0270 D0 BB              BNE    DUR
0272 E6 03              INC    TUNES   BUMP TUNES POINTER
0274 E6 03              INC    TUNES   TWICE
0276 4C 17 02           JMP    REST
```

	DURATION	FREQ ONE		FREQ TWO	
32F	15	F7	09	F7	09
334	05	00	00	00	00
339	60	E1	08	C1	11
33E	05	00	00	00	00
343	13	98	05	2E	0B
348	03	00	00	00	00
34D	13	ED	05	DA	0B
352	03	00	00	00	00
357	13	A7	06	4C	0D
35C	03	00	00	00	00
361	15	77	07	EE	0E
366	03	00	00	00	00
36B	50	77	07	EE	0E
370	03	00	00	00	00
375	07	A7	06	4C	0D
37A	03	00	00	00	00
37F	07	ED	05	DA	0B
384	03	00	00	00	00
389	07	A7	06	4C	0D
38E	20	00	00	00	00
393	30	F7	09	F7	09
398	03	00	00	00	00
39D	60	E1	08	C1	11
3A2	03	00	00	00	00
3A7	15	98	05	98	05
3AC	03	00	00	00	00
3B1	15	ED	05	ED	05
3B6	03	00	00	00	00
3BB	10	A7	06	A7	06
3C0	15	00	00	00	00
3C5	03	77	07	DE	01
3CA	05	00	00	00	00
3CF	60	FC	04	3F	01
3D4	20	00	00	00	00
3D9	07	F7	09	FC	04
3DE	05	00	00	00	00
3E3	07	F7	09	FC	04
3E8	05	00	00	00	00
3ED	07	F7	09	FC	04
3F2	00				

Sorting with the APPLE II

Part I

The first of a series of articles which will deal with sorting in general and on the APPLE II in particular. This installment presents some background material, a comparison of three sorting techniques, and a program for implementing the Shell-Metzner sort.

Gary A. Foote
127 Mt. Spring Road
Tolland, CT 06084

Whether you are maintaining complex data bases, compiling mailing lists, or simply keeping track of your checkbook, at some time you will need to sort records. There are a multitude of sorts available — from the agonizingly slow one in the APPLE CHECKBOOK program, through the relatively fast BASIC sort, to my exceedingly fast (by a factor of 200) machine language sort. What makes a sort fast, and which sort is the fastest? These are the questions I will cover in my series on exploring sorting with the APPLE II.

Background

There have been many magazine articles written on sorting. The ones I based my initial investigation on were those in the Nov-Dec 1976 issue of CREATIVE COMPUTING covering the SHELL-METZNER, bubble, delayed replacement, and heapsorts, and the JAN-FEB 1978 issue of the same magazine on the Butterfly-Hart sort. The first article found the Shell-Metzner and heapsorts to be a vast improvement over the bubble and delayed replacement sorts. The second article found the Butterfly-Hart to be even faster. The Shell-Metzner and heapsort are replacement-type sorts; that is, the records are compared to one another and replace each other according to some unique algorhythm. They are relatively small in size and don't rely on much extra storage for their processing. The Butterfly-Hart is a linked list sort. A tree structure is built from the records and broken down into several smaller sorted lists. These lists are then merged to form the final result. This sort is much faster for large numbers of records, but is quite complex and requires extra storage to hold the lists and tree structure. For more details on how these sorts operate, I leave you to refer to the original articles.

I programmed each of these sorts in INTEGER BASIC and compared them by sorting various numbers of random ten character strings. Below were the results.

Table I — Sorting in BASIC

SORT	SORTED WORDS 10	100	500	1000
SHELL-METZNER	1	34	268	647
BUTTERFLY-HART	2	38	266	606
HEAPSORT	1	35	261	600

(All sorting times in seconds)

For further exploration, I decided to use the Shell-Metzner sort because it was easiest to program and most compact. Many things had to be taken into account before implementing this sort in INTEGER BASIC. Because of the limited string support in this BASIC, it is easier to store records to be sorted in memory between the upper end of the data variables and the lower end of the program area, accessing them with PEEK's and POKE's. At first, as I sorted these records, I exchanged the actual records in memory when necessary. This becomes very time consuming because for exchanging two 10 character records, you must move 30 bytes (10 to a work area, 10 from one record to the other, and 10 from the work area back to the other record). A much more elegant technique is to store the address of each record as a member of an array. When an exchange is necessary, you need only exchange the addresses in the array, a total move of 6 bytes (2+2+2) for any size record. When the sort is complete, the addresses of the sorted records can be found sequentially in the array. The first member of the array will point to the lowest sorted record, and the last member to the highest sorted one. The records can be read out in the proper order quite simply, and can easily be sorted in reverse order simply by reading the array backwards. The beauty of this method is that the records have never actually moved and can be read in the original order as simply as the sorted order. This reduction alone increases the speed of the sort by a factor of three for a 100 record sort, and exponentially above that.

My BASIC version is divided into several parts. The first part generates random character strings in memory, depending on the record size and count entered. This is for benchmark tests and can be replaced with your own I/O routine for your application. Line 140 actually puts the random characters in memory, so replacing this line with a REM after your first run allows you to test other sorting methods while using the same records. The second part merely initializes the memory pointer array and prints the unsorted strings. This can also be included in your I/O routine. The third part is the actual SHELL-METZNER sort. The routine can be easily changed if you wish to sort numbers in an array instead of strings in memory. Finally, there is a routine to print the results, and a handy routine from CALL-APPLE for finding the address of a variable in the data area.

SWEET-16 for Size

Never being satisfied, I decided to continue another step and try to program the sort routine in SWEET-16 (as all you APPLE people know, a 16 bit interpreter implemented in ROM). An excellent article in the NOV 1977 issue of BYTE (or the BEST OF BYTE VOL 1) was my source for SWEET-16 information. SWEET-16 was 4 to 9 times faster than the BASIC sort, and very compact due to the powerful instruction set. But due to difficulty in implementing, and because the machine language routine was several orders of magnitude faster, I am not including this material. Don't feel bad. Because I know of no SWEET-16 assembler, writing this program was actually harder than the machine language version.

Machine Language for Speed

The machine language implementation of SHELL-METZNER was not difficult, because I was almost translating directly from each BASIC statement into equivalent functions in machine code. As you can see by the listing in Figure 2, I made extensive use of PAGE ZERO addressing, both to cut down on code and increase speed. I left in BASIC all the I/O routines and setup necessary to prepare the sort, since this is quite easy in BASIC and I already had the program written from the first problem. The actual sort algorhythm is the only part I programmed in machine code. Thus we get the benefit of BASIC for I/O, printing, etc. in 1% of the execution, and the machine code speed for the repetative looping in 99% of the execution.

Using this machine language sort is relatively easy. The BASIC routine in Figure 3 sets up the variables needed by the sort and calls the machine language routine. It can be substituted for the sort routine in the BASIC version in Figure 1 (lines 1000-1900). The sort routine itself (in Figure 2) is loaded at address 300-3C2. This routine is easily relocatable to any other address (say 800 if you are using 300 for another routine). All you need to do is load it where desired and change the last two instructions (2 JMP commands) to reflect your new location. You must, of course, change the CALL in your BASIC program also.

Below is a comparison of my three different implementations of the Shell-Metzner sort.

The maximum number of records you can sort is easily determined by taking the memory size between data high and program low and dividing it by the record size + 2 (the size of the array element needed to hold the pointer to the record). I find with a 32K machine running DOS, I have 18K free. More memory is available if you want to lose DOS of course.

Machine language routines may be more trouble to implement, but with an increase in speed over BASIC by a factor of 200, you cannot ignore them. In Part II I will continue my investigation by exploring sorting APPLESOFT character strings with multiple keys. Until then, happy sorting!

μ

Table II — Comparison of Three Methods

	SORTED WORDS X WORD LENGTH		
METHOD	500 X 10	1000 X 10	3600 X 3
BASIC	268	746	4200 (70 min)
SWEET-16	46	158	—
MACHINE	1	3	21

(All sorting times in seconds)

Figure 1

```
 10 REM ***************************
 20 REM *    SHELL-METZNER SORT    *
 30 REM *       BY GARY FOOTE      *
 40 REM ***************************
 50 CALL -936: PRINT : PRINT "SHELL-METZNER SORT": PRINT
 60 INPUT "ENTER RECORD COUNT AND LENGTH",NUM,LEN
 70 DIM A$(255),A(NUM)
 80 I=J=K=L=M=X=T=Z=LL=II=LM=HM=ADDR=W: REM SAVE SPACE FOR VARIABLES
 90 LM= PEEK (204)+ PEEK (205)*256:HM= PEEK (202)+ PEEK (203)*256
 95 REM
100 REM ****** FILL MEMORY WITH DATA ******
105 REM
110 PRINT : PRINT "CREATING RANDOM STRINGS"
120 IF LM+LEN*NUM<HM THEN 140
130 PRINT "TOO MUCH DATA!!": END
140 FOR X=1 TO LEN*NUM: POKE LM+X, RND (26)+193: NEXT X
150 REM
200 REM ****** INITIALIZE MEMORY POINTER ARRAY ******
205 REM
210 A$="A$": GOSUB 4000
220 FOR X=1 TO NUM:A(X)=(X-1)*LEN+LM+1
230 T=A(X): GOSUB 3000
240 NEXT X
250 REM
```

```
1000 REM ****** SORT ROUTINE ******
1010 REM
1100 PRINT : PRINT "STARTING SORT"
1200 N=NUM:M=N
1300 M=M/2: IF M=0 THEN 1900:K=N-M:J=1
1400 I=J
1500 L=I+M:II=A(I):LL=A(L)
1600 FOR X=0 TO LEN-1:W= PEEK (II+X)- PEEK (LL+X): IF W<0 THEN 1800: IF
     W>0 THEN 1700: NEXT X: GOTO 1800
1700 T=A(I):A(I)=A(L):A(L)=T:I=I-M: IF I>=1 THEN 1500
1800 J=J+1: IF J>K THEN 1300: GOTO 1400
1900 PRINT : PRINT "ENDING SORT"
1910 REM
2000 REM ****** PRINT RESULTS ******
2005 REM
2010 A$="A$": GOSUB 4000
2020 FOR X=1 TO NUM
2030 T=A(X): GOSUB 3000
2040 NEXT X
2050 END
2060 REM
3000 REM ****** STRING PRINT ROUTINE ******
3005 REM
3010 FOR Z=0 TO LEN-1
3020 POKE ADDR+Z, PEEK (T+Z): REM ARRAY A$
3030 NEXT Z: POKE ADDR+Z,30
3040 PRINT X,A$
3050 RETURN
3060 REM
4000 REM ***** FIND VARIABLE'S ADDRESS
4005 REM
4010 ADDR= PEEK (74)+ PEEK (75)*256-1:K= LEN(A$):J= PEEK (204)+ PEEK (205
     )*256-1:L=0: IF A$(K,K)#"$" THEN 4020:K=K-1:L=1
4020 FOR I=1 TO K: IF ASC(A$(I))# PEEK (ADDR+I) THEN 4040: NEXT I
4030 IF PEEK (ADDR+I+L)>1 THEN 4040:ADDR=ADDR+K+4+L: RETURN
4040 FOR I=1 TO 100: IF PEEK (I+ADDR)>1 THEN NEXT I:I=ADDR+I+1:ADDR= PEEK
     (I)+ PEEK (I+1)*256-1
4050 IF ADDR<J THEN 4020: PRINT "VARIABLE ";A$;" NOT FOUND": END
```

Figure 2

```
1000 REM ****** SORT ROUTINE ******
1010 REM
1100 PRINT : PRINT "STARTING SORT"
1200 A$="A": GOSUB 4000
1300 POKE 0,ADDR MOD 256: POKE 1,ADDR/256: REM STORE ARRAY ADDRESS
1400 POKE 2,LEN: REM STORE RECORD LENGTH (MUST BE < 256)
1500 POKE 4,NUM MOD 256: POKE 5,NUM/256: POKE 6,NUM MOD 256: POKE 7,NUM/
     256: REM STORE NUMBER OF RECORDS
1600 CALL 768: REM CALL SORT ROUTINE
1700 PRINT : PRINT "ENDING SORT"
```

Figure 3

```
                    1000 *-----------------------------------*
                    1010 *        SHELL-METZNER SORT         *
                    1020 *         BY GARY A. FOOTE          *
                    1030 *          COPYRIGHT 1979           *
                    1040 *   COMMERCIAL RIGHTS RESERVED      *
                    1050 *-----------------------------------*
                    1060 *
                    1070 *       VARIABLES AND CONSTANTS
                    1080 *
                    1090 * ALL VARIABLES ARE TWO BYTES.
                    1100 * THE LISTED NAME IS THE LOW ORDER BYTE.
                    1110 * THE NAME+1 IS THE HIGH ORDER BYTE.
                    1120 * EX.  I    = LOW  ORDER BYTE
                    1130 *      I+1  = HIGH ORDER BYTE
                    1140 *
                    1150 ADRA .EQ $00         ARRAY A ADDRESS
                    1160 LEN  .EQ $02         RECORD LENGTH
                    1170 N    .EQ $04         NUM OF RECORDS
                    1180 M    .EQ $06         M
                    1190 I    .EQ $08         I (RECORD I)
                    1200 L    .EQ $0A         L (RECORD L)
                    1210 J    .EQ $0C         J
                    1220 K    .EQ $0E         K
                    1230 PTRI .EQ $10         PTR TO ADDR OF A(I)
                    1240 PTRL .EQ $12         PTR TO ADDR OF A(L)
                    1250 ADRI .EQ $14         ADDR OF REC A(I)
                    1260 ADRL .EQ $16         ADDR OF REC A(L)
                    1270 *
                    1280 *      SORT ROUTINE
                    1290 *
                    1300      .OR $300
0300- 46 07         1310 SORT LSR M+1         M = M / 2
0302- 66 06         1320      ROR M
0304- D0 05         1330      BNE SRT1        IF M = 0
0306- A5 07         1340      LDA M+1
0308- D0 01         1350      BNE SRT1        THEN
030A- 60            1360      RTS             DONE!
030B- 38            1370 SRT1 SEC
030C- A5 04         1380      LDA N           K = N - M
030E- E5 06         1390      SBC M
0310- 85 0E         1400      STA K
0312- A5 05         1410      LDA N+1
0314- E5 07         1420      SBC M+1
0316- 85 0F         1430      STA K+1
0318- A9 01         1440      LDA #1          J = 1
031A- 85 0C         1450      STA J
031C- A9 00         1460      LDA #0
031E- 85 0D         1470      STA J+1
0320- A5 0C         1480 SRT2 LDA J           I = J
0322- 85 08         1490      STA I
0324- A5 0D         1500      LDA J+1
0326- 85 09         1510      STA I+1
```

```
0328- 18          1520 SRT3 CLC
0329- A5 08       1530      LDA I         L = I + M
032B- 65 06       1540      ADC M
032D- 85 0A       1550      STA L
032F- A5 09       1560      LDA I+1
0331- 65 07       1570      ADC M+1
0333- 85 0B       1580      STA L+1
0335- A5 00       1590      LDA ADRA      INITIALIZE PTRS
0337- 85 10       1600      STA PTRI      TO ARRAY A
0339- 85 12       1610      STA PTRL      ADDRESS
033B- A5 01       1620      LDA ADRA+1
033D- 85 11       1630      STA PTRI+1
033F- 85 13       1640      STA PTRL+1
0341- A0 02       1650      LDY #2
0343- 18          1660 SRT4 CLC
0344- A5 10       1670      LDA PTRI      PTR TO A(I) =
0346- 65 08       1680      ADC I         ADDR ARRAY A +
0348- 85 10       1690      STA PTRI      2 * I
034A- A5 11       1700      LDA PTRI+1
034C- 65 09       1710      ADC I+1
034E- 85 11       1720      STA PTRI+1
0350- 18          1730      CLC
0351- A5 12       1740      LDA PTRL      PTR TO A(L) =
0353- 65 0A       1750      ADC L         ADDR ARRAY A +
0355- 85 12       1760      STA PTRL      2 * L
0357- A5 13       1770      LDA PTRL+1
0359- 65 0B       1780      ADC L+1
035B- 85 13       1790      STA PTRL+1
035D- 88          1800      DEY           DO 2 TIMES
035E- D0 E3       1810      BNE SRT4      (PTR DISP IS 2 BYTES)
0360- B1 10       1820      LDA (PTRI),Y  II = A(I)
0362- 85 14       1830      STA ADRI
0364- B1 12       1840      LDA (PTRL),Y
0366- 85 16       1850      STA ADRL
0368- C8          1860      INY
0369- B1 10       1870      LDA (PTRI),Y  LL = A(L)
036B- 85 15       1880      STA ADRI+1
036D- B1 12       1890      LDA (PTRL),Y
036F- 85 17       1900      STA ADRL+1
0371- 88          1910      DEY
0372- B1 14       1920 SRT5 LDA (ADRI),Y  COMPARE ONE BYTE IN
0374- D1 16       1930      CMP (ADRL),Y  RECORDS I & L
0376- 90 31       1940      BCC SRT8      I < L
0378- D0 07       1950      BNE SRT6      I > L
037A- C8          1960      INY           I = L
037B- C4 02       1970      CPY LEN       END OF RECORD?
037D- D0 F3       1980      BNE SRT5      NO, NEXT BYTE
037F- F0 28       1990      BEQ SRT8      RECORDS EQUAL
0381- A0 00       2000 SRT6 LDY #0
0383- A5 14       2010      LDA ADRI      A(I) <-> A(L)
0385- 91 12       2020      STA (PTRL),Y
0387- A5 16       2030      LDA ADRL
0389- 91 10       2040      STA (PTRI),Y
038B- C8          2050      INY
038C- A5 15       2060      LDA ADRI+1
```

```
038E- 91 12      2070       STA (PTRL),Y
0390- A5 17      2080       LDA ADRL+1
0392- 91 10      2090       STA (PTRI),Y
0394- 38         2100       SEC
0395- A5 08      2110       LDA I          I = I - M
0397- E5 06      2120       SBC M
0399- 85 08      2130       STA I
039B- A5 09      2140       LDA I+1
039D- E5 07      2150       SBC M+1
039F- 85 09      2160       STA I+1
03A1- 90 06      2170       BCC SRT8
03A3- D0 83      2180 SRT7  BNE SRT3       IF I > 0 THEN STR3
03A5- A5 08      2190       LDA I
03A7- D0 FA      2200       BNE SRT7
03A9- E6 0C      2210 SRT8  INC J          J = J + 1
03AB- D0 02      2220       BNE SRT9
03AD- E6 0D      2230       INC J+1
03AF- A5 0F      2240 SRT9  LDA K+1
03B1- C5 0D      2250       CMP J+1        IF J > K
03B3- 90 0B      2260       BCC JMP2       THEN SORT
03B5- D0 06      2270       BNE JMP1       ELSE SRT2
03B7- A5 0E      2280       LDA K
03B9- C5 0C      2290       CMP J
03BB- 90 03      2300       BCC JMP2
03BD- 4C 20 03   2310 JMP1  JMP SRT2       CHANGE IF RELOCATED
03C0- 4C 00 03   2320 JMP2  JMP SORT       CHANGE IF RELOCATED
                 2330       .EN
```

```
:$300.3C2

0300- 46 07 66 06 D0 05 A5 07
0308- D0 01 60 38 A5 04 E5 06
0310- 85 0E A5 05 E5 07 85 0F
0318- A9 01 85 0C A9 00 85 0D
0320- A5 0C 85 08 A5 0D 85 09
0328- 18 A5 08 65 06 85 0A A5
0330- 09 65 07 85 0B A5 00 85
0338- 10 85 12 A5 01 85 11 85
0340- 13 A0 02 18 A5 10 65 08
0348- 85 10 A5 11 65 09 85 11
0350- 18 A5 12 65 0A 85 12 A5
0358- 13 65 0B 85 13 88 D0 E3
0360- B1 10 85 14 B1 12 85 16
0368- C8 B1 10 85 15 B1 12 85
0370- 17 88 B1 14 D1 16 90 31
0378- D0 07 C8 C4 02 D0 F3 F0
0380- 28 A0 00 A5 14 91 12 A5
0388- 16 91 10 C8 A5 15 91 12
0390- A5 17 91 10 38 A5 08 E5
0398- 06 85 08 A5 09 E5 07 85
03A0- 09 90 06 D0 83 A5 08 D0
03A8- FA E6 0C D0 02 E6 0D A5
03B0- 0F C5 0D 90 0B D0 06 A5
03B8- 0E C5 0C 90 03 4C [20 03]
03C0- 4C [00 03]
:
```

CHANGE IF RELOCATED

SYMBOL TABLE

ADRA	0000	LEN	0002	N	0004
M	0006	I	0008	L	000A
J	000C	K	000E	PTRI	0010
PTRL	0012	ADRI	0014	ADRL	0016
SORT	0300	SRT1	030B	SRT2	0320
SRT3	0328	SRT4	0343	SRT5	0372
SRT6	0381	SRT7	03A3	SRT8	03A9
SRT9	03AF	JMP1	03BD	JMP2	03C0

SOFTOUCH

AT LAST
ENHANCE THE FULL CAPABILITIES OF YOUR DISK II

SOFTOUCH ANOUNCES THE

" DISC MANAGEMENT SYSTEM "

EIGHT PROGRAMS ON DISK TO PROVIDE THE USER WITH A COMPLETE UNDERSTANDING OF THE DISK DRIVE COMMANDS PLUS A UTILITY PACKAGE TO INDEX AND CATAGORIZE ALL PROGRAMS WRITTEN FOR THE APPLE II COMPUTER. THE SYSTEM PROVIDES FULL SEARCH , EDITING AND DATA TRANSFER CAPABILITIES.

A TWENTY-SIX PAGE BOOKLET PROVIDES DETAILED , EDUCATIONAL TECHNIQUES GIVING A THROUGH UNDERSTANDING OF ALL DOS COMMANDS.

INCLUDED ARE SUCH TECHNIQUES AS:

* INITIALIZATION OF TEXT FILES
* WRITING & READING OF TEXT FILES
* CREATING TEXT FILES
* EDITING FILES
* TRANSFERRING DATA TO OTHER DISKETTES
* LISTING OF DATA FILES
* SEARCHING DATA BY VARIOUS OPTIONS

A DOCUMENTED LISTING PROVIDES LINE CHANGES FOR ADAPTING THE SYSTEM TO PREFORM NUMEROUS OTHER APPLICATIONS.

THE BOOKLET IS WRITTEN IN A MANNER WHICH MAY BE USED BY THE NOVICE AS WELL AS THE EXPERIENCED PROGRAMMER. DOZENS OF PROGRAMMING TECHNIQUES ARE SHOWN AND FULLY DOCUMENTED. ALSO INCLUDED SEPARATELY, IS A PROGRAMMERS AID GIVING QUICK REFERENCE TO INTEGER, APPLESOFT AND DOS COMMANDS WITH ILLUSTRATIVE EXAMPLES.

SYSTEM REQUIREMENTS: DISK II & APPLESOFT TAPE OR ROM CARD

PRICE $19.95 (PROCESSED & SHIPPED WITHIN 4 DAYS)

SEND CHECK OR MONEY ORDER TO:

SOFTOUCH
P.O. BOX 511
LEOMINSTER, MASS. 01453

APPLE II® OWNERS:

West Side Electronics introduces the APPLETIME™, a Real Time Clock for the Apple II

The Appletime (Model APT-1) is a single peripheral board which plugs directly into any I/O slot on the Apple II. Timing is done completely in hardware (ie. NOT an interrupt driven clock). Thus, the Appletime continues to operate even when the computer is turned off. Our exclusive Three Way Power System keeps the clock running via its own AC supply, the computer's, or battery backup in case of power failure. Other features include 12/24 Hour selection, AC or crystal timebase, 50/60 Hz, and BCD or ASCII data format. Fully assembled and tested, with instructions and

APT-1 Real Time Clock. **$79.95**

Proto-board for Apple II. Over 1300 holes on 0.1 inch centers for designing your own circuits.

APB-1 Prototyping Board. **$16.95**

VERBATIM 5¼" DISKETTES $34.50

Soft sector, in plastic file case. Box of 10

VISA

WEST SIDE ELECTRONICS
P.O. Box 636
CHATSWORTH, CA. 91311

master charge

We pay all shipping in Continental U.S.A.
Others add 10%; California residents add 6% tax.

PYGMY PROGRAMMING

* APPLE SOFTWARE *

P.O. BOX 3078 • SCOTTSDALE, ARIZONA 85257

APPLE-SCRIBE-2 disk or cassette $49.00
(Word processor with any length line, printing in any shape or form. Requires 16k or larger system)

APPLE-DMS 48k & disk required $49.00
(Universal data management system)

UNLISTABLE-BASIC disk required $39.00
(Make Integer BASIC listings unintelligable)

BCOPY disk & 16, 32 or 48k $19.00
(Single disk drive binary file copier)

DEFLECTIONS disk or cassette $10.00
(Real time color graphics game)

Custom programs . . . write for details!

APPLE II SOFTWARE

Special Introductory Prices

Tabular 4-Way Number Converter
Integer BASIC (8K min). **$3.50**

Apple Pi Life (4K min). **$5.00**

Agenda (16K min). **$5.00**

HIRES Shape Encode (16K min). **$5.00**

Integer BASIC Floating Point
Interface (2K Machine Code). **$6.50**

Each program completely documented.
Prices include postage and handling.
Send SASE for Software List today!
Send check or money order to:

Microspan Software
2213A Lanier Drive
Austin, TX 78758

(Texas residents add 5% sales tax)

Streamlining the C2-4P

Here are three modifications you can make to your OSI C2-4P to raise its speed, increase the cassette throughput, and add reverse video to the display.

James L. Cass
19559 Tulsa Street
Northridge, CA 91326

I am concerned by the paucity of articles on OSI computers in MICRO and hope that this will reverse the trend. I feel that the Challenger 2-4P running speed and ease of modification more than offset its shortcomings. I will describe three modificiations I have made, mainly, raising the CPU clock rate, raising the cassette data rate, and reversing the video presentation.

Raising the CPU Clock Rate

My computer is happily working with a clock frequency a little under 2 MHz (1.9648 actual, 1.96608 nominal) in place of half that, which is the way it was delivered. The CPU clock is taken from the video timing chain, which uses a crystal oscillator near 12 MHz, a divide by three, and then a series of binary dividers to form 15,360 and 60 Hz sync pulses. It was only necessary to move the CPU clock takeoff one stage higher in the timing chain. To do this, move the jumper wire coming from bus pin #18 off IC #E4 pin #13, and onto pin #14. If you intend to make this change, use a small, low power, preferably grounded soldering iron, as recommended for all IC work. Another word of caution: make a long, thorough shakedown run of several operational programs looking for dropped bits from memory. I did this since I have two RAM chips marked "550" (presumably not fast enough to qualify as 450 nsec.), but there was absolutely no hint of dropped bits. Instead, I have very snappy video display operation, slightly fast keyboard repeat, and, best of all, running times cut in half. A machine language LIFE program updates a full screen of 1792 cells 14 times a second!

Doubling the Bit Rate

I successfully doubled the bit rate of my cassette interface from 300 to 600 baud, after speeding up my CPU. I naturally tried 1200 baud; while it seemed to read properly, the load program seemed to choke up on very long (64 to 71 character) lines sometimes and miss the CR and next line. The 555 IC oscillator frequency is doubled from 4800 to 9600 Hz by substitutiong a 0.01 mfd capacitor for the 0.022 and then adjusting the trimpot. A frequency counter is a big help, if not essential. Since the tone frequencies should remain at 1200 and 2400 Hz, an extra divider is needed. The unused half of the 7474 already in the interface works nicely, or you can install a 74163 in the convenient prototyping vacant space, and get several baud rates for printers and the like. Rate selection can be conveniently brought to a switch mounted to the left of the keyboard. Figure 1 shows the circuit using the 7474.

I have found "reversed" video to be much easier to view for extended periods. Also, the black "reversed" characters have less apparent intensity variation, that is, they look evener. The reversed video connection is indicated in the schematics, but there is no provision made in the printed wiring, so that it is necessary to cut a printed conductor. The junction of R 11 and R 23 is moved from Pin 8 to Pins 9-10-11 of the IC at D4. I installed a switch near the keyboard with short, direct small wires, but find that I could have left the wiring at "reversed" with no loss.

Conclusions

Doubling the CPU clock rate and hence the speed of the C2-4P is quite easy to do. The main risk is that some 2114 type RAM chips may be too slow. The data rate in the cassette interface can be doubled to 600 baud, but only with some effort and decrease in reliability; 1200 baud does not work. Reversing the video to display black characters on white is relatively simple and the reversed video format is preferred by everyone. Cassette speed selection and normal-reverse video are conveniently brought to switches installed near the keyboard.

µ

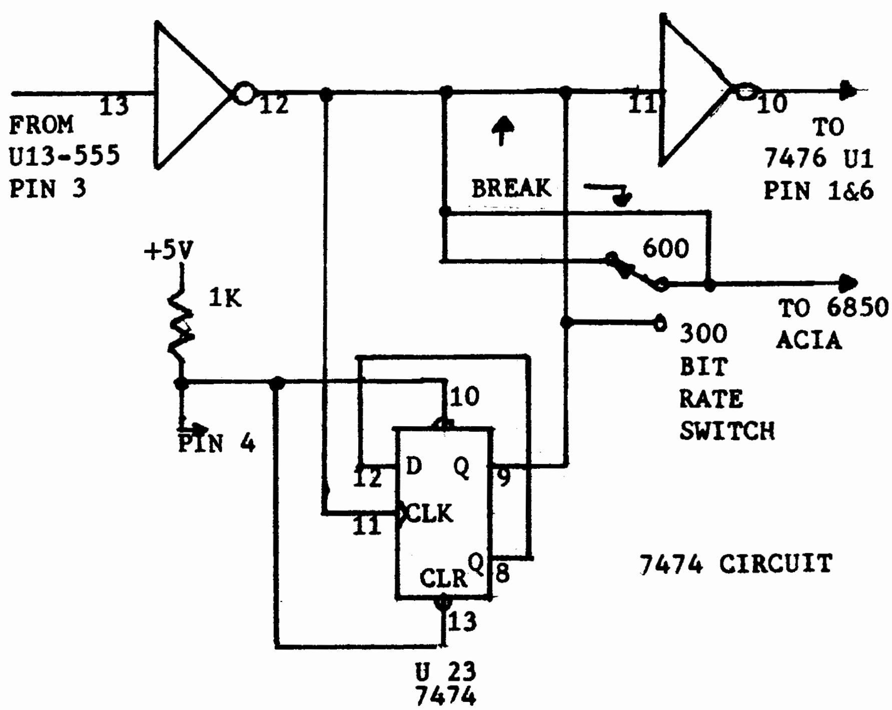


Figure 1

6502 INFORMATION RESOURCES UPDATED

A list of regular publications which have material of interest to 6502 users.

William R. Dial
438 Roslyn Ave.
Akron, OH 44320

Did you ever wonder just what magazines were the richest sources of information on the 6502 microprocessor, 6502-based microcomputers, accessory hardware and software? For several years this writer has been assembling a bibliography 6502 references related to hobby computers and small business systems. The accompanying list of magazines has been compiled from this bibliography. At the top of the list are several publications which specialize in 6502-related subjects. An attempt has been made to give up-to-date addresses and subscription rates for the magazines cited. Subscription rates are for U.S. Other countries normally are higher.

MICRO
$15.00 per year
MICRO
P.O. Box 3
S. Chelmsford, MA 01824

6502 USER NOTES
$13.00 per 6 issues
Eric Rehnke
P.O. Box 33093
Royalton, OH 44133

OHIO SCIENTIFIC — SMALL SYSTEMS JOURNAL
$6.00 per year (6 issues)
Ohio Scientific
1333 S. Chillicothe Rd.
Aurora, OH 44202

PET GAZETTE
Free bi-monthly (Contributions Accepted)
Microcomputer Resource Center
1929 Northport Drive, Room 6
Madison, WI 53704

Robert Purser's REFERENCE LIST OF COMPUTER CASSETTES
Nov. 1978 $2.00/Feb 1979 $4.00
Robert Purser
P.O. Box 466
El Dorado, CA 95623

THE PAPER (PET)
$15.00 per year (10 issues)
The PAPER
P.O. Box 43
Audubon, PA 19407

THE CIDER PRESS (APPLE)
Scot Kamins
Box 4816
San Francisco, CA 94101

STEMS FROM APPLE
Ken Hoggatt
APPLE PORTLAND PROGRAM LIBRARY EXCHANGE
9195 SW El Rose Court
Tigard, OR 97223

APPLE SEED
Bill Hyde
The Computer Shop
6812 San Pedro
San Antonio, TX 78216

KILOBAUD/MICROCOMPUTING
$18.00 per year
Kilobaud Magazine
Peterborough, NH 03458

BYTE
$18.00 per year
Byte Publications, Inc.
70 Main St.
Peterborough, NH 03458

DR. DOBB'S JOURNAL
$15.00 per year (10 issues)
People's Computer Co.
Box E
1263 El Camino Real
Menlo Park, CA 94025

ON-LINE
$3.75 per year (18 issues)
D. H. Beetle
24695 Santa Cruz Hwy
Los Gatos, CA 95030

RECREATIONAL COMPUTING
(formerly PEOPLE'S COMPUTERS)
$10.00 per year (6 issues)
People's Computer Co.
1263 El Camino Real
Box E
Menlo Park, CA 94025

INTERFACE AGE
$18.00 per year
McPheters, Wolfe & Jones
16704 Marquardt Ave.
Cerritos, CA 90701

POPULAR ELECTRONICS
$12.00 per year
Popular Electronics
One Park Ave.
New York, NY 10016

PERSONAL COMPUTING
$14.00 per year
Benwill Publishing Corp.
1050 Commonwealth Ave.
Boston, MA 02215

73 MAGAZINE
$15.00 per year
73, Inc.
Peterborough, NH

CREATIVE COMPUTING
$15.00 per year
Creative Computing
P.O. Box 789-M
Morristown, NJ 07960

SSSC INTERFACE
Southern California Computer Soc.
1702 Ashland
Santa Monica, CA 90405

EDN (Electronic Design News)
$25.00 per year
Cahners Publishing Co.
270 St. Paul St.
Denver, CO 80206

RADIO ELECTRONICS
$8.75 per year
Gernsback Publications, Inc.
200 Park Ave., South
New York, NY 10003

QST
$12.00 per year
American Radio Relay League
225 Main St.
Newington, CT 06111

IEEE Computer
IEEE
345 E. 47th St.
New York, NY 10017

POLYPHONY
$4.00 per year
PAIA Electronics, Inc.
1020 W. Wilshire Blvd.
Oklahoma City, OK 73116

RAINBOW (APPLE)
$15.00 per year
Rick Simpson and Terry Landereau, Editors
P.O. Box 43
Audubon, PA 19407

PET USER NOTES
$5.00 per year (6 or more issues)
PET User Group
P.O. Box 371
Montgomeryville, PA 18936

CONTACT — User Group Newsletter
Gratis to Apple owners
10260 Bandley Drive
Cupertino, CA 95014
(408) 996-1010

SOUTHEASTERN SOFTWARE NEWSLETTER
(APPLE)
10 issues $10.00
George McClelland
Southeastern Software
7270 Culpepper Drive
New Orleans, LA 70126

COMPUTER MUSIC JOURNAL
$14.00 per year (6 issues)
People's Computer Co.
Box E
1010 Doyle St.
Menlo Park, CA 94025

POPULAR COMPUTING
$18.00 per year
Popular Computing
Box 272
Calabasas, CA 91302

MINI-MICRO SYSTEMS
$18.00 per year
Modern Data Service
5 Kane Industrial Drive
Hudson, MA 01749

DIGITAL DESIGN
$20.00 per year
Benwill Publishing Corp.
1050 Commonwealth Ave.
Boston, MA 02215

ELECTRONIC DESIGN
(26 issues per year)
Hayden Publishing Co., Inc.
50 Essex St.
Rochelle Park, NJ 07662

CALL A.P.P.L.E.
$10.00 per year (includes dues)
Apple Puget Sound Program Library Exchange
6708 39th Ave. SW
Seattle, WA 98136

The Color Gun for the Apple II

With some quite inexpensive hardware, you can turn your APPLE II into a color detector -- a device which will automatically determine the colors of any object.

Neil D. Lipson
29 S. New Ardmore Ave.
Broomall, PA 19008

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

Building the Color Gun

To start off with, buy three sensitive cadium sulphide photo cells (physically between 1/4 to 1/2 inch in diameter). If the cells are not equal in sensitivity, they can be equalized easily in software. This is illustrated in the listing. Merely point the gun at a white piece of paper (or at the light source itself if its not too bright) during the calibration procedure. The construction of the gun is very simple. Mount the three cells in a triangle about 2" for each side on a piece of wood or other material. Then place three filters over the cells, with red on paddle (0) cell, blue on paddle (1) cell, and yellow on paddle (2) cell. The purer the filter, the better. Photographic filters are the best, and will give the best results. However, red, blue or yellow clear plastic will work satisfactorily in most situations. Note the use of the REM statements in the program. These are for slowing down the paddle readings just a hair in order to avoid having the readings "overlap". The wiring diagram is shown in Figure 1. Mount the entire setup in some type of barrel or cylinder about 4 inches long, with the inside of the barrel painted white, and glue everything together and seal against light leaks. Plug it into the game paddle after the wiring is complete and you ready to go. For the pin numbers of the paddles, consult your red manual.

The Color Gun Program

Type the program into the Apple in Applesoft 2 and run. The gun will only recognize 6 colors, and when it isn't sure what the color is, it will give you two colors (one primary color and one secondary). This should not happen if the colors are absolutely pure, but most colors are not, so expect this situation more often. Notice the correction algorithm in statement 70 in the program to correct for the blue cell. The cells that I used were somewhat more sensitive to blue than the other colors (which is common of cadium sulfide). This was noticed when the color gun kept saying "orange" (the compliment of blue). The correction algorithm elimates most of this problem. If the gun acts strangely, run it again until it gests a good calibration. It sometimes takes more than one run to get it working properly (usually because it is confused by a bright color nearby).

By fine tuning the software, and using more exact ratios, you can determine many other colors. Given enough ratios to choose from, you can give the color temperature of the object (with high quality cells and filters). The typical photographic filters you can use are the yellow (K2), the red (25 or 25A) and the blue (47). These may be varied if desired to meet the spectral response of the particular cell you buy. You could even use different colors in the filters as long as you adjust the software accordingly. Buy the smallest filter you can (it only has to cover about 1/2 inch diameter), but make sure there is no light leak from the sides of the cells. If you follow these instructions, the gun will work perfectly the first time around. Have fun!

μ

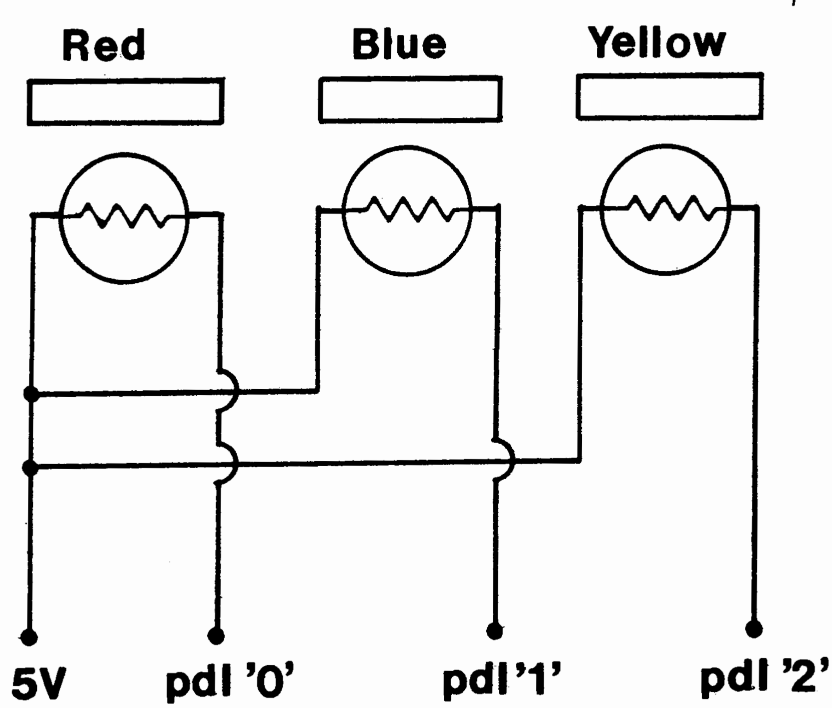


Figure 1

```
1 CALL -936
2 VTAB 10: HTAB 10: PRINT "COLOR
    GUN BY NEIL D. LIPSON"
3 HTAB 15: PRINT "COPYRIGHT 1979
    "
4 HTAB 12: PRINT "ALL RIGHTS RES
    ERVED": FOR I = 1 TO 2000: NEXT
    I
5 REM '0' RED
6 REM '1' BLUE
7 REM '2' YELLOW
10 CALL - 936
15 REM YELLOW,BLUE,RED
20 PRINT : PRINT : PRINT : PRINT

25 GOSUB 1000
30 CALL - 936: PRINT : PRINT
32 A = PDL (0)
35 REM
40 B = PDL (1)
45 REM
50 C = PDL (2)
55 REM
60 A = A * A1
61 B = B * B1
62 C = C * C1
70 B = B / 1.5
100 PRINT "RED CELL = ";A
110 PRINT "BLUE CELL = ";B
115 PRINT "YELLOW CELL = ";C
116 PRINT : PRINT
117 PRINT "THE COLOR IS:": PRINT

118 PRINT "*********************
    "
121 IF C< B AND C< (A) THEN PRINT
    "YELLOW"
123 IF A< B AND A<C THEN PRINT
    "RED"
124 IF A > B AND A> C THEN PRINT
    "GREEN"
125 IF B> A AND B> C THEN PRINT
    "ORANGE"
126 IF C < A AND C > B THEN PRINT
    "PURPLE"
129 IF B < C AND B< (A) THEN PRINT
    "BLUE"
130 PRINT "*********************
    "
131 FOR X = 1 TO 2300: NEXT X
140 GOTO 30
200 END
100 CALL - 936: PRINT
1010 PRINT "POINT GUN AT A WHITE
     SHEET OF PAPER"
1020 FOR I = 1 TO 1500: NEXT I
1030 A1 = PDL (0)
1035 REM
1040 B1 = PDL (1)
1045 REM
1050 C1 = PDL (2)
1055 PRINT "A1=";A1
1056 PRINT "B1=";B1
1057 PRINT "C1=";C1
1060 D1 = A1 * B1 * C1
1070 A1 = D1 / A1
1080 B1 = D1 / B1
1090 C1 = D1 / C1
1100 PRINT "CORRECTION FACTOR FO
    R RED = ";A1
1110 PRINT "CORRECTION FACTOR FO
    R BLUE= ";B1
1120 PRINT "CORRECTION FACTOR FO
    R YELLOW= ";C1
1125 FOR I = 1 TO 2000: NEXT I
1130 RETURN
10000 END
```

ASK the Doctor — Part V

Reading KIM Tapes on the AIM and SYM

The Doctor discusses some problems that arise in trying to load KIM format cassette tapes on the AIM or SYM, and "with a little help from his friends" presents a short routine to get by the SYM "2F" loading bug and a subroutine which mimics the KIM SCANDS routine on the SYM.

Robert M. Tripp, Ph.D.
The Computerist, Inc.
P.O. Box 3
So. Chelmsford, MA 01824

Reading KIM Tapes with the AIM

The AIM 65 has two speeds for reading the KIM format tapes. The normal KIM tape records at about 8 1/3 characters per second. Early in KIM history, Jim Butterfield published "Hypertape", a program that permits KIM formats to be written at higher rates: 2 times, 3 times and even 6 times the normal KIM rate — and still be read by the regular KIM monitor and hardware with no changes! The AIM 65 people recognized the value of the higher speed KIM rates and made their monitor capable of loading either 1 time or 3 times KIM tapes. (The full 6 times would have been very nice, but I guess we can't have everything.) The AIM documentation is very vague about using these KIM formats. To use the KIM loader, you must first set a "user alterable" RAM location A408 as follows:

C7 for normal AIM format tapes
5A for normal KIM format tapes
5B for 3 times KIM format tapes

On power up, C7 is automatically set, so that the AIM format is the default, as one would expect. The A408 location must be set by the user manually to the correct KIM speed value before calling on the tape load or dump routines. This must be reset either manually or by a power up reset to return to the AIM format. If you do not have the correct value in A408 for what you are trying to do, it will not work and, in general, will not give you any indication that it is not working. The KIM Loader/or Dumper is invoked by specifying "K" as the I/O device.

Now that you finally understand how to use the AIM to load a KIM tape, you set A408 to 5A for a regular KIM tape which has your favorite program and run the AIM Load specifying K for the input device, the program identifier that you used when you recorded the tape as the file name, and the correct tape unit. You sit back and wait for the load to complete. But what's this! The AIM is suddenly in some strange state! It may be saying "OUT =" which does not make sense, or even worse may be dumping reems of paper out of the printer! What happened? What happened was that your typical KIM program which uses all available memory on the KIM — locations 0000 to 03FF — has found a small problem with the AIM KIM format load program. The problem is the **STACK.** The programmers who developed the KIM monitor were super-smart in realizing that, given the very limited onboard memory of the KIM, users would often want to load ALL of the RAM, from 0000 to 03FF, right over page zero and page one — stack all. They made the KIM load and dump routines work without using page zero or page one in any way that would not interfere with data in those pages. You might want to examine this code sometime in the KIM monitor listings, as it is quite instructive. The AIM programmers were only smart about the KIM format — not super-smart. They gave the multiple speeds, but did not write the loader is such a way that it could load over the page one stack. So, the loader works fine until it hits the stack that is being used by the loader itself. Then, it **SELF DESTRUCTS!** It over-writes a return address in the stack and then returns to "never-never-land". Where it goes will depend on the byte of data that over-writes the stack. I do not know of any simple solution to this problem. You can, of course, divide your KIM program into two portions: 0000 to 01F0 (or there abouts — I think it bombs at about 01FB but have not done any detailed testing), and 0200 and up. This assumes that you have access to a KIM. If not, my friend, I am afraid you have a real problem.

Reading KIM Tapes with the SYM

The SYM monitor bug which causes loading of KIM tapes to abort when it encounters a "2F" has been documented in previous columns, is "cured" by Skov's program, and is fixed in the new SYM SY1.1 version of the monitor. It will not be discussed here. With the "2F" bug fixed, the SYM still has problems with reading KIM format tapes. The problem is the same as discussed with the AIM above. It can **NOT** load over the end of the page one stack. The SYM has an additional, related problem. It can not read over the last two bytes in page one either. These two bytes are used by the load routine as the indirect pointer to the next location to be loaded. Once your KIM formatted tape hits them — Goodbye! So, we have here the same problem, and the same solution. To load KIM tapes into a SYM, they must be loaded in three segments: 0000 to 00FD, 0100 to (about) 01FC, and 0200 and up. I thought I had a great idea to get around this problem. I dumped my KIM tapes with everything shifted up to start at 0200, with the intention of using a simple SYM Blockmove command to relocate them down to their proper addresses. That is, the KIM tape would be set to load from 0200 to 05FF and then be moved down to 0000 to 03FF. Good idea, right? Well, it may be a good idea, but it doesn't work. Blockmove has the exact same problems as the tape load: it uses the last two locations in page zero as well as subroutines which require access to the page one stack!

AIM/SYM/KIM Tape Summary

While there are obviously some problems in using the KIM format tapes on the SYM and AIM, this format is the only one which is compatible between the three machines, and should be used as a common medium of exchange for programs and data between them. To be "universal" the tapes should be written at the normal KIM speed and should start at location 0200 or above. I am sure that there will be SYM and AIM versions of Hypertape published soon, perhaps in MICRO. Maybe someone will even have the time to come up with a KIM LOAD program for the SYM and the AIM that can be tacked on the front of a tape to be exchanged — in normal speed — and which once loaded will permit the loading of KIM format programs and data into any address (except for those occupied by the loader itself which should be out-of-the-way somewhere), and at the higher speeds. Until then, keep the AIM and SYM loading problems in mind as you make plans to transfer your programs and data from one of the ASK family members to another.

μ

SYM "2F" BUG KILLER

```
        CODE IS COMPLETELY
        RELOCATABLE.              ACCESS  *    $8B86
                                  LOADT   *    $8C78
20 86 8B  START  JSR   ACCESS     CHKT    *    $8E78
20 78 8C         JSR   LOADT      CHAR    *    $00FC
B0 01            BCS   TWOF
60               RTS          SUCCESSFUL LOAD
A9 2F     TWOF   LDAIM $2F    2F ERROR
20 78 8E         JSR   CHKT
A0 00            LDYIM $00
91 FE            STAIY $00FE
E6 FE            INC   $00FE  BUMP POINTERS
D0 02            BNE   OKAY
E6 FF            INC   $00FF  BUMP HIGH
A5 FC     OKAY   LDA   CHAR
20 3F 8D         JSR   $8D3F
B0 EA            BCS   TWOF
60               RTS
```

A Solution to the SYM "2F" Bug

The problem around SYM-1, with KIM-tape compatability, I solved, at first, by writing a small program, that controls the loading. When loading terminates because of a "2F" in the data stream, it can be assumed that it will result in a checksum error too. What the program does, is simply store the "2F" that was the probable cause and then reenter the tape reading as though there had not been an error.

Submitted by

Jan Skov
Majvaenget 7
DK-6000 Kolding
The Netherlands

SYM Static Display

This program is a utility for the SYM-1 that I have found useful in adapting routines for the KIM. It loads the DISBUF from three locations, F5 to F7, corresponding to the three display locations, F9 and FB, used in the KIM monitor. It ends with a JMP SCAND, and thus can be used to replace a JSR SCANDS command as used in a KIM routine. It gives a static display.

```
          SYM STATIC DISPLAY

          SYM REFERENCES

          ACCESS *     $8B86
          SCAND  *     $8906
          SEGSM  *     $8C29
          DISBUF *     $A63F

          COMPLETELY RELOCATABLE

20 86 8B  STATIC JSR   ACCESS ENTER HERE UNLESS ACCESS
A0 03     ALTNTR LDYIM $03    ALREADY SET.  SET UP FOR
B9 F4 00  MOVE   LDAY  $00F4  3 NUMBERS. GET NUMBER INTO A
4A               LSRA         SHIFT
4A               LSRA         LEFT
4A               LSRA         FOUR
4A               LSRA         TIMES
AA               TAX          PUT RESULT IN X
98               TYA          Y HAS NUMBER INDEX
0A               ASLA         MULTIPLY BY 2
A8               TAY          PUT BACK INTO Y
BD 29 8C         LDAX  SEGSM  GET CORRECT SEGMENT CODE
99 3E A6         STAY  DISBUF -01 PUTINTO DISPLAY BUFFER
98               TYA          GET INDEX BACK
4A               LSRA         DIVIDE BY 2
A8               TAY          PUT BACK
B9 F4 00         LDAY  $00F4  NUMBER INOT A
29 0F            ANDIM $0F    MASK
AA               TAX          RESULT INTO X
98               TYA          GET INDEX
0A               ASLA         MULTIPLY BY 2
A8               TAY          PUT BACK
BD 29 8C         LDAX  SEGSM  GET SEGMENT CODE
99 3F A6         STAY  DISBUF PUT INTO DISPLAY BUFFER
88               DEY          DECREMENT INDEX
88               DEY          TWICE
F0 06            BEQ   EXIT   IF HAVE LOADED 6 DIGITS, EXIT
98               TYA          NOT DONE, GET INDEX
4A               LSRA         DIVIDE INDEX BY 2
A8               TAY          PUT BACK
18               CLC          PREPARE TO GO TO MOVE
90 D3            BCC   MOVE   GO!
4C 06 89  EXIT   JMP   SCAND  LIGHT UP DISPLAY AND RETURN

         NOTES:
           F7 CORRESPONDS TO KIM POINTH AT FB
           F6 CORRESPONDS TO KIM POINTL AT FA
           F5 CORRESPONDS TO KIM INH AT F9

           Y KEEPS TRACK OF LOCATION IN DISBUF
             WHEN MULTIPLIED BY 2,
             AND LOCATION IN F5-F7 WHEN NOT.
```

Submitted by

Paul W. Zitzewitz
Department of Natural Sciences
University of Michigan — Dearborn
4901 Evergreen Road
Dearborn, MI 48128

Computer-Determined Parameters for Free-Radical Polymerization

Don't let the title scare you. If you are using your system for complex equation solving, the general techniques presented will be useful -- even if the particular example is not your cup to tea.

Dr. L.S. Reich
3 Wessman Drive
W. Orange, NJ 07052

During the formation of polymers by so-called free-radicals, the following lengthy and "sensitive" equation is generally valid,

$$W(R) = AZ^2RP^{R-1} + ((1\text{-}A)/2)Z^3R(R\text{-}1)P^{R-2} \qquad (1)$$

where, W(R) = weight-fraction of polymer possessing size R; A and P are parameters; and, Z = 1-P (A or P cannot exceed unity). Various methods have been previously attempted to solve eqn. (1) for the parameters, A and P, which are of theoretical and practical importance. Thus, approximate graphical procedures have been employed (Smith et al., **J. Polym. Sci.**, Pt. A-2, **4**, 365 (1966); **J. Phys. Chem.**, 72, 216 (1968); **Ibid.**, **72**, 2933 (1968)) as well as more direct approaches (Reich et al., **J. Appl. Polym. Sci.**, **17**, 3709 (1973)). The purpose of this article is to present a novel trial-and-error computer program whereby parameters A and P (or Z) can be readily obtained from eqn. (1). **Although eqn. (1) applies to polymers, the solution of eqn. (1) involves mathematical procedures which are common to various scientific disciplines and the program presented should therefore be of general interest.**

Prior to running this program, W(R)-R data is entered in line #200. An initial (trial) value of P (PO) is entered in line #65. Since P is generally close to unity, an arbitrary initial value of P should be selected between .995-.999 (regardless of which value is chosen, the computer will search for the correct value in order to obtain final values of A and P).

From eqn. (1), it can be readily seen that a least squares treatment of the data (Y = A1 + A2X) where, Y = W(R)/(1-P)-RP^{R-1} and X = (R-1)/p, will afford best values of intercept (A1) and slope (A2) where, A1 = A and A2 = (1-A)Z/2 (the least squares procedure is given in line #'s 75-120). Then (cf. line #140),

$$Z = 2A2/(1\text{-}A1) \qquad (2)$$

Under the conditions used, more than one solution for Z is possible. However, there can only be one unique physically real solution for A and for Z (or 1-P). The physically unreal solution for Z affords values of A > 1 (which is theoretically impossible). By using line #163, when A > 1, PO is lowered in order to achieve conditions whereby a physically real value of A may be obtained. Other limitations that must be met are: P cannot be greater than unity (see line #150); A1 cannot be less than zero (#170). Another condition (arbitrary) to be met is (line #156), 1-PO > = Z. This ensures that prior to a series of iterative calculations to determine final parameter values, the initial value of P is such that 1-PO > = Z. Then, values of P are gradually increased (line #180) until the following conditions (3) hold (line #'s 160, 168), at which time, A, Z, and the correlation coefficient (from the least squares calculation of A1 and A2) are displayed (line #168),

$$ABS(R) > .9 \qquad (3a)$$

$$\text{and, } 1\text{-}P > = Z \text{ or } ABS(1\text{-}P\text{-}Z)/Z < = 4E\text{-}04 \qquad (3b)$$

Since eqn. (1) is sensitive to changes in data, i.e., small changes in data can result in relatively large changes in A, there is a requirement that correlation coefficient (R) be above .9 (3a). Thus, even though (3b) is satisifed, if (3a) is not, then the screen will display the statement that the data is not accurate enough (in order to afford reasonably significant values) (line #165). In (3b), the second term involving the absolute value may not apply sometimes, but Z-trial values will generally decrease faster than Z-calculated values to that 1-P < = Z halts further iterations and final parameter values are displayed. Further, because of the sensitivity of eqn. (1), when A-values are between -.05 and 0, then A is considered to possess a zero value (line #'s 165, 1000). Insufficiently accurate data may also cause relatively large negative A1-values (#165) and endless loops (iterations) to occur. After 200 such iterations, the screen will display a statement that the data may not be accurate enough (to achieve closed parameter values) (line #155). However, it is also possible that due to an unfortunate choice of a PO value in line #65, more than 200 iterations will be required prior to the display of final values. Hence, the additional statement in line #155 that another value of PO should be entered in line #65 (and another run attempted). If another run is made and a similar situation arises then omitting W(R)-R data at low values of R and/or at very high values of R may afford closed parameter values (the former W(R)-R values generally lie along the steepest part of a distribution curve and are subject to errors in W(R) while the latter values lie along a relatively flat portion of the curve and are subject to errors in R). From the preceding, data in #200 must be derived from precise experimental techniques, which are available, e.g., gel permeation chromatography, due to eqn. (1) sensitivity to relatively small inaccuracies in data (which are prone to occur at the tail ends of a distribution curve).

Prior to the display of final results, values of Z (trial) and Z (calcd.) will be compared in tabular form on the screen (line #'s 68, 155) in order to apprise the viewer of the status of the iterative calculations in progress. After final parameter values have been displayed, values of W(R) (obsd.) and W(R) (calcd.) are compared in tabular form (line #'s 190, 500-520).

Explanatory REM statements are to be found in line #'s 9, 64, 105, 130, 152, 158 and ca. 3-3.5K bytes are required depending upon the amount of data entered (the data is limited to 19 W(R)-R pairs). Applesoft II Basic in ROM was employed and a run, as given in the Program Example section, required ca. 1 min. but this can vary considerably depending upon the accuracy of the data and the initial choice of the PO value (more iterations are necessary when the PO value is further away from the true P value). Finally, it may be noted that the parameter Values A and Z can be used to estimate various pertinent quantities. e.g.. the so-called weight-average degree of polymerization of a polymer which is equal to (3-A)/Z.

```
5 PRINT "THIS PROGRAM ALLOWS THE PRECISE CALCULATION OF
  MOLECULAR WEIGHT DISTRIBUTION PARAMETERS, 'A' & '1-P',
  FOR FREE-RADICAL POLYMERIZATION.  WEIGHT-FRACTION VS.
  DEGREE OF POLYMERIZATION (DP) DATA IS ENTERED IN LINE
  #200. ";
6 PRINT "THE INITIAL VALUE OF 'P' (ARBITRARILY CHOSEN BETWEEN
  .995-.999) IS ENTERED IN LINE  65.  CA. 3-3.5K BYTES ARE
  REQUIRED AND EXPLANATORY 'REM' STATEMENTS ARE IN LINE #'S
  9, 64, 105, 130, 152, 158.": STOP
9 REM #'S 10-60 ALLOW THE FORMATION OF THE ARRAY W(J,K) FOR
  WT.-FRACTION VS. DP DATA IN LINE #200
10 DIM W(20,2)
20 FOR J = 1 TO 50
30 FOR K = 1 TO 2: READ W(J,K)
40 IF W(J,1) = 0 THEN 60
50 NEXT K,J
60 J = J-1
64 REM #65  LISTS THE INITIAL VALUE OF 'P' & #'S 70-100 ALLOW
   FOR A LEAST SQUARES TREATMENT OF THE DATA
65 P0 = .99745
68 PRINT; PRINT"Z-VALUES (TRIAL)"; TAB (21); "Z-VALUES (CALCD.)":
   PRINT "----------------"; TAB (21); "----------------"
70 P = P0
75 FOR I = 1 TO J
80 Y = W(I,1)/(W(I,2)*^(W(I,2) -1)*(1-- P)^2): X = (W(I,2) -1)/P
90 XY = XY + X*Y: XX = XX + X*X: SX = SX + X: SY = SY + Y:
   YY = YY + Y*Y
100 NEXT
105 REM  IN #'S 110, 120 ARE GIVEN THE LEAST SQUARE SLOPE (A2),
    INTERCEPT (A1), & THE CORRELATION COEFF. (R)
110 A2 = (J*XY - SX*SY)/(J*XX - (SX)^2): A1 = (SY/J) - A2*(SX/J)
120 R = (XY - J*(SX/J)*(SY/J))/(SQR(XX - SX*(SX/J))*SQR(YY - SY*
    (SY/J)))
130 REM #'S 140, 150 ALLOW THE CALCULATION OF Z AND THE ADJUSTMENT
    OF P0 IF Z<0
140 Z = 2*A2/(1 - A1)
150 IF P>1 OR Z<0 THEN P0 = P0- .00001: XX = 0: SX = 0:
    SY = 0: YY = 0: GOTO 70
152 REM #155 INDICATES A POSSIBLE ENDLESS LOOP & #156 LOWERS THE
    INITIAL VALUE OF P0 IN LINE #65 IF 1-P0<= Z
155 PRINT 1-P; TAB(21); Z: PC = PC + 1: IF PC>200 THEN PRINT:
    PRINT "THE PROGRAM IS GOING THRU AN ENDLESS LOOP?  THE DATA
    MAY NOT BE ACCURATE ENOUGH!  TRY ANOTHER VALUE OF P0 IN #65
    & SEE IF THERE IS ANY CHANGE!": END
156 IF 1-P0<Z OR Z<0 THEN P0 = P0 - .00001: XX = 0: XY = 0: YY = 0:
    SX = 0: SY = 0: GOTO 70
158 REM #163 LOWERS P0 VALUE IF A1>1 & #'S 160, 165, 168, 180
    ALLOW FOR THE CALCULATION OF FINAL VALUES OF 'A' & '1-P' SHILE
    #170 ACCOUNTS FOR VALUES OF A1<0
160 IF 1-P<=Z OR ABS(1-P-Z)/Z<= 4E-04 THEN A = A1
163 IF A1>1 THEN P0 = P0 - .00001:  XX = 0: XY = 0: YY = 0: SX = 0:
    SY = 0: GO TO 70
```

```
165 IF (ABS(R)<.9 OR A1<-.05) AND (1-P<= Z OR ABS(1-P-Z)/Z<= 4E-04)
    THEN PRINT: PRINT "DATA IS NOT ACCURATE ENOUGH": END
168 IF ABS(R)>.9 AND (1-P<= Z OR ABS(1-P-Z)/Z = 4E-04) THEN PRINT:
    PRINT "VALUES OF 'A' & '1-P' = "; : GOSUB 1000: PRINT A" AND
    "Z" ; AND, CORRELATION COEFFF. = "; : PRINT CC: GOTO 190
170 IF A1<0 THEN 180
180 P = P + .00001 : XX = 0: YY = 0: SX = 0: SY = 0:
    GOTO 75
190 PRINT: PRINT "   W(X),OBSD.";TAB(20); "W(X) ,CALCD.":
200 DATA 7.45E-04,400, 7.62E-04,600,7.3E-04,700,4.41E-04,
    1200,2.9E-04,1500,1.3E-04,2000,2.1E-05,3000
210 DATA 0
500 FOR I = 1 TO J
510 W = (A1 + A2*(W(I,2)-1)/P)*W(I,2)*P^(W(I,2)-1)*(1-P)^2
520 PRINT TAB(4); W(I,1); TAB(20); INT(W*1E06 + .5)/1E06:
    NEXT I: END
1000 IF A<0 AND A>-.05 THEN A = 0: A1 = 0
1010 A = INT(A*100 + .5)/100: Z = INT(Z*1E06 + .5)/1E06:
     CC = INT(R*1E04)/1E04
1020 RETURN
```

Program Example

W(R) vs. R data for the polymer, polystyrene (from gel permeation chromatography techiques), was entered in line #200 and an arbitrary value of PO = .99745 in line #65. Then a run was carried out as follows,

```
COMMAND:   RUN ------>  STATEMENTS 5, 6, and "BREAK IN 6"
COMMAND:   CONT ----->

         " Z-VALUES (TRIAL)       Z-VALUES (CALCD.)

           2.54999986E-03         2.49794192E-03
           2.53999978E-03         2.49536302E-03
           2.52999971E-03         2.49275496E-03
           2.51999963E-03         2.49012109E-03
           2.50999955E-03         2.48747276E-03
           2.49999948E-03         2.48481487E-03
           2.48999941E-03         2.48216115E-03
           2.47999933E-03         2.47951866E-03

      VALUES OF 'A' & '1-P' = .65 AND 2.48E-03; AND,
      CORRELATION COEFF. = .9999

           W(X),OBSD.             W(X),CALCD.

           7.45E-04               7.53E-04
           7.62E-04               7.6E-04
           7.3E-04                7.24E-04
           4.41E-04               4.4E-04
           2.9E-04                2.9E-04
           1.3E-04                1.3E-04
           2.1E-05                2.1E-05  "
```

AIM 6522 Based Frequency Counter

The AIM 65 obviously is going to find its way into the electronics laboratory. Here it is used as a frequency counter.

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

The program listed performs as a six-digit frequency counter. It will count at least as fast as 450 kHz, perhaps faster. A simple interface circuit is shown in Figure 1. Although the signal to be measured could be connected directly to the PB6 pulse counting pin of the 6522, I prefer not to connect strange and unknown signals directly to the computer. In any case, the signal pulses to be counted should really be shaped into the form of a square wave before they appear at PB6.

The counter uses timer T1 in a free-running mode with 50,000 clock cycles between settings of its interrupt flag. The timer T1 is not allowed to interrupt the 6502, rather its interrupt capability is disabled and the flag is "watched" by reading the interrupt flag register, IFR. With $\$14 = 20_{16}$ intervals of 50,000 clock cycles apiece, one gets a total interval of one second. $14 is located in $0000. The T1 timer is loaded with instructions starting at $0230. Note that the number I used is less than 50,000 because my AIM 65 crystal is slow by 244 parts in one million cycles. You may wish to make adjustments with this number also, depending on your system's clock frequency.

The frequency counter works as follows. Timer T2 in its pulse counting mode is initially loaded with $FFFF = 65535. Once it is loaded, timer T1 is started and PB0 is brought to logic O to allow the NAND gate to let pulses through. At the end of the timing interval, described in the preceding paragraph, the gate is closed, the timer T2 is read, the result is subtracted from $FFFF, this number is converted from HEX to BCD, and it is added to the display locations using the ADC instruction in the decimal mode. If, at any time the T2 timer counts through zero, an interrupt request (IRQ) occurs and the display registers are incremented by 65536 = $FFFF + 1, T2 is reloaded with $FFFF, and counting continues. At the end of one second, the total number of counts is displayed by the display subroutine, which, by the way, is identical to the 24-hour clock display routine in the February 1979 issue of MICRO. It is a bit unfortunate that the 6522 designers did not allow the T2 timer to continue producing interrupts without reloading it, because in the time interval between the interrupt request and the reloading of the T2 timer (starting at instruction $0296 in the interrupt routine), a few counts or pulses on PB6 might be missed. This would only be of concern at large counting rates.

The HEX to BCD conversion routine starts at address $025D and ends at address $028E. The 16-bit number representing the number of counts in timer T2 is stored in locations $0010 and $0011. If $PQRS represents this number, then

$$\$PQRS = (P.4096_{10}) + (Q.256_{10}) + (R.16_{10}) + (S.1).$$

If the calculation on the right-hand side of the above equation is done in the decimal mode, the $PQRS will be converted to BCD. In other words, 4096 is **added** to itself P times, 256 is added to itself Q times, 16 is added to itself R times, and 1 is added to itself S times, **all in the decimal mode.** These results are all added together, giving a BCD number. Better routines exist, I am sure, but this one isn't too slow. Note that P,Q,R, and S are each one nibble of the 16-bit number obtained from timer T2. (Has anyone yet suggested calling 16-bit numbers "gobbles," giving nibbles, bytes, and gobbles?) The table starting a $0300 must be loaded into memory for the HEX to BCD conversion to work.

The symbol table given may help you if you wish to modify the program or if you want to change it to run on a microcomputer other than the AIM 65. Also, I would be interested in knowing an exact upper limit for the frequency at which it will operate and in any further improvements to the rate at which it will count. Currently I do not have enough time to do this experimentation myself.

μ

Table I

ADDRESS TABLE FOR THE AIM 65 FREQUENCY COUNTER

$A000 = PBD (ORB)
$A002 = PBDD (DDRB)
$A004 = T1L-L (Read)
$A005 = T1L-H
$A006 = T1L-L (Write)
$A008 = T2L-L
$A009 = T2C-H
$A00B = ACR
$A00D = IFR
$A00E = IER
$A404 = IRQL
$A405 = IRQH
$0000 = Count-to-twenty register
$0001 = Display register, low-order byte
$0002 = Display register, middle-order byte
$0003 = Display register, high-order byte
$0010 = PQ = Low-order byte of count from timer T2
$0011 = RS = High-order byte of count from timer T2
$0340 = Starting address of display subroutine
$0295 = Starting address of IRQ routine

```
0200 A9 LDA #95
0202 8D STA A404
0205 A9 LDA #02
0207 8D STA A405
020A A9 LDA #A0
020C 8D STA A00E
020F A9 LDA #01
0211 8D STA A000
0214 8D STA A002
0217 A9 LDA #60
0219 8D STA A00B
021C A9 LDA #14
021E 85 STA 00
0220 A9 LDA #FF
0222 8D STA A008
0225 8D STA A009
0228 A9 LDA #00
022A 85 STA 01
022C 85 STA 02
022E 85 STA 03
0230 A9 LDA #42
0232 8D STA A006
0235 A9 LDA #C3
0237 8D STA A005
023A CE DEC A000
023D AD LDA A004
0240 C6 DEC 00
0242 2C BIT A00D
0245 50 BVC 0242
0247 A5 LDA 00
0249 D0 BNE 023D
024B EE INC A000
024E 38 SEC
024F A9 LDA #FF
0251 ED SBC A008
0254 85 STA 10
0256 A9 LDA #FF
0258 ED SBC A009
025B 85 STA 11
025D A2 LDX #03
025F A5 LDA 10
0261 29 AND #0F
0263 F0 BEQ 027F
0265 A8 TAY
0266 18 CLC
0267 F8 SED
0268 A5 LDA 01
026A 7D ADC 0300,X
026D 85 STA 01
026F A5 LDA 02
0271 7D ADC 0304,X
0274 85 STA 02
0276 A5 LDA 03
0278 69 ADC #00
```

```
027A 85 STA 03
027C 88 DEY
027D D0 BNE 0266
027F CA DEX
0280 30 BMI 028E
0282 A0 LDY #04
0284 46 LSR 11
0286 66 ROR 10
0288 88 DEY
0289 D0 BNE 0284
028B 4C JMP 025F
028E D8 CLD
028F 20 JSR 0340
0292 4C JMP 021C
0295 48 PHA
0296 A9 LDA #FF
0298 8D STA A009
029B F8 SED
029C 18 CLC
029D A5 LDA 01
029F 69 ADC #36
02A1 85 STA 01
02A3 A5 LDA 02
02A5 69 ADC #55
02A7 85 STA 02
02A9 A5 LDA 03
02AB 69 ADC #06
02AD 85 STA 03
02AF D8 CLD
02B0 68 PLA
02B1 40 RTI
```

```
0340 A5 LDA 01
0342 85 STA 04
0344 A5 LDA 02
0346 85 STA 05
0348 A5 LDA 03
034A 85 STA 06
034C A2 LDX #13
034E 8A TXA
034F 48 PHA
0350 A0 LDY #04
0352 A5 LDA 04
0354 29 AND #0F
0356 18 CLC
0357 69 ADC #30
0359 09 ORA #80
035B 20 JSR EF7B
035E 46 LSR 06
0360 66 ROR 05
0362 66 ROR 04
0364 88 DEY
0365 D0 BNE 035E
0367 68 PLA
0368 AA TAX
0369 CA DEX
036A E0 CPX #0E
036C B0 BCS 034E
036E 60 RTS

<M>=0300 96 56 16 01
< > 0304 40 02 00 00
```

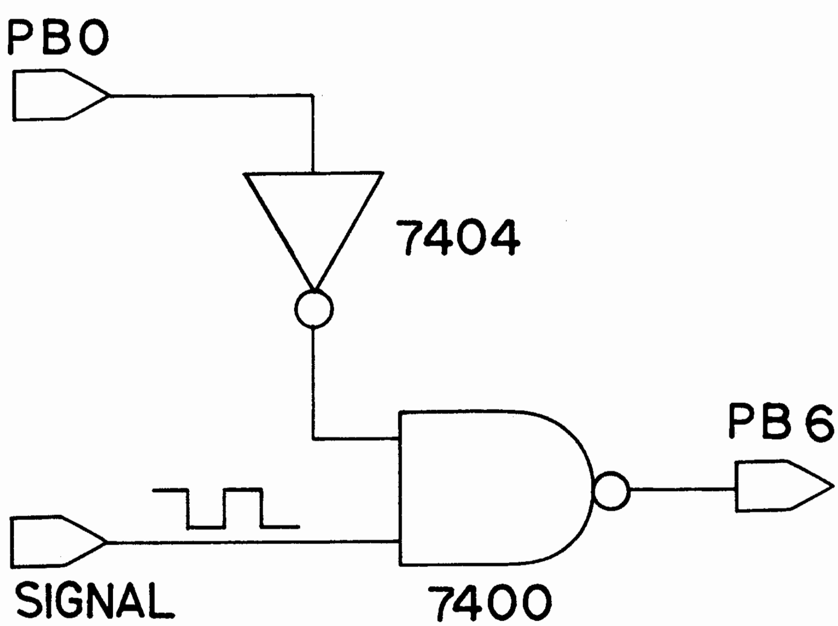


Figure 1

Interface Circuit for the AIM 65
Frequency Counter Using the 6522 VIA

KIM — The Tunesmith

A number of programs have been offered which permit you to play music on your micro. The program presented here also permits you to compose music on your KIM, as well as save it and play it back.

Anthony T. Scarpelli
RR 1, Box 426
N. Windham, ME 04062

Have you ever wanted to compose music, but knew nothing about how to go about doing it? Do you lack a musical instrument and have a tune going through your head and don't know what should go after the first few notes? Well here is a program for a basic KIM-1 that will help you compose a tune, and you don't even have to know how to read or write music.

I have really never learned how to play a musical instrument, and I never have time to practice. Yet every once in a while I want to try out a few notes going on in my head, or I just want to see how a couple of notes sound together, to see if they have any effect on me. So what I did was to develop a program that uses a basic KIM-1 and the speaker circuit shown on page 57 of the KIM-1 User Manual that plays a tune I compose one note at a time. I use the keypad as data entry to place into the program notes of two octaves, including sharp notes, with four possible lengths and a rest or no note. I used the lettered keypads as well as the 9 which looks like a small G for all the notes which are seven in number, basically **A B C D E F** and **G.**

Tunesmith Operation

Once you start the program, you press one of the note letters. It will sound the appropriate note. If you want the sharp for that note, if it has one (B and E do not), press **5.** To get the upper octave of the note you want you press **7,** and if you want the upper octave sharp of the note, press **5** first, then **7.** The keys **1, 2, 4,** and **8** will give you a whole note **(1)**, a half note **(2)**, a quarter note **(4)**, and an eighth note **(8)**. After you choose your note, you choose your length. If you don't want the note, start again, only this time the length is not automatically a half note as it would be when you first start out, you'll have to change it to what you want.

Now that you have your nice note that sounds just right, press **3.** This will save the note and place it in a tune table. To know that the note is indeed saved, the display will flash a **SAVE.** You have to hold the **3** key down until the **SAVE** is seen, though. Now the chosen note will be played and you can pick another note, or a rest which is **0.** The procedure is the same for a possible 72 note tune. If you like your tune and want to write it down, press the + key. The display will show you the first note of the tune, and every time you hit the **3** key, the next will be displayed. If you want to start again, press the **DA** (Do Again) key.

The Tunesmith Program

We can go over the program now. Table I is a listing of the keypad numbers and what they represent. The main program starts at 0200 and initialization goes on to 021A. From 021C to 0228 we test the keypad and 022A to 022E we test for the first time through the program. This step eliminates any noise in the speaker while choosing the first note. 0230 to 0236 gets the program to step through all the notes, and 0238 to 023D delays the program, not only to give you more time to choose a note, but also to put a space between the beginning and ending of the tune. 0242 to 0248 is for the beginning silence. 024A thru 0263 loads the note you have chosen into a temporary location. 0265 to 026E will jump to all the subroutines which we'll explain in a minute. 0271 thru 027B tests for the save key, which you press if you want that particular note. From 027B to 0283 we test for the **DA** key. 0285 to 028F will cause the program to jump to the routine which will allow us to see what notes we have so that they can be written down and saved for the "Top Ten". 0295 to 02A9 sets the save flag, resets the note counter, and because the program goes deep into the stack territory, resets the stack pointer to avoid trouble.

The Get High subroutine is the first one we come to. From 0356 to 035E we test to see if we want a high note. If we don't, we return from the subroutine. If yes, we'll first test to see if it's to be a sharp note that is to go to the next octave. If it is, then from 0366 to 036A we'll load the high sharp note into the temporary location, otherwise from 036F to 0373 we'll load just the next octave note. The Get Sharp subroutine is similar and the Get Length subroutine is simple enough.

The Play Tune subroutine is next. From 0300 to 0306 we set up the first note, then we play it. This is the unsaved note we are trying out. Then we'll test for a save flag from 0313 to 0317, and test for a note or notes in the tunetable up to 031D. If there is one or more notes in the tunetable, from 031F to 0330 we'll play them. If we had a save for the temporary note, we reset the save flag, store a rest so we don't hear the saved note twice, then load the note into the next position of the tuntable, and we'll also put our chosen length into the length table; all this from 0333 to 0345. Since we saved the note, not only do we need some indication that it was saved, we also need to indicate that our finger is on the **3** keypad long enough for the program to catch the keypad entry, so at 0347 we go to the subroutine that displays a big red **"SAVE".** At 034A we play all our notes again, and then go back to the main program to get another note, then back here again so we always hear our tune.

In the Tone subroutine, at 02DD and 02DF we set the ports to outputs; and at 02E2 and E4 we start KIM's internal timer. We load the note frequency, and when it runs down we change the output to its other state, whatever it was. If you hook a speaker circuit on the port as in the KIM manual, a note will be produced as we repeat this procedure every time the timer times out at 02EF; and if we do this for a length of time determined by the note length at 02F9, we have just played a note in our tables or one we're testing out.

Our Save subroutine starts at 03AA where we load a number for a particular time we want to keep the SAVE letters on. Next at 03AE and 03BO we set the direction registers and since we want only 4 digits lit we load the number 4 into the X register. When we store one of six numbers, from 09 to 13 into the location SBD(1742), one of the six digits will be lit, and then if we load a particular hex number representing a letter, number or other shape into another location SAD(1740), then the seven segment display will light. We also need some delay, because if we did not, the display would light and go out in a couple of microseconds, which few of us could see. All this is taken care of from 03B3 to 03CC. And finally we want to end the tune after 72 notes so we will automatically go the the Display Notes routine from 03CE to 03D4. We want to keep count of how many notes we save so at 03D7 we increment the note count.

If we have a nice little tune running through our circuits and we say to ourselves, "Hey, that's a catchy tune that might make the top 40," then we'll need some way of finding out what notes are in the tunetable so that we can write them down. The Display Notes routine does just that. What we want this section to do is to display a lettered note, to show that it is a sharp and/or a high note, and to show what its length is. We want it to stay on the display until we're ready for the next note and we need some indication that the note has changed when we do go to the next note. Finally we want the option of starting again. So here we go.

From 0100 to 010A we test the counters to see if we've reached the end of our tune table, then we take our note and length and put them into a temporary location from 010D to 0115. From 0117 to 011D we check for a rest; if it isn't one then at 011F on we determine what note it is. What I did was to compare the unknown note to the note table and for every wrong comparison increment a count. We also have four groups of 7 notes and to determine what group, I subtract a number until I get a carry flag. This then tells me the group and also the note. The group indicates whether the note is high, sharp, or high/sharp. We load the correct shape for the display on this information. If it was just a rest, at 0180 we load a zero shape. At 018A to 0198 we test for the length and then store the length shape. Up to 01BC we display the shapes as before, only this time, as we go through a test for the next note, and "do again", we keep the

```
                TUNESMITH

                BY ANTHONY T. SCARPELLI
                MAY 1979

                MICRO NUMBER 13
                JUNE 1979
                COPYRIGHT (C) MAY 1979 BY
                THE COMPUTERIST, INC.

                KIM MONITOR REFERENCES

                PAD    *    $1700  DATA REGISTER
                PADD   *    $1701  DATA DIRECTION REGISTER
                TIMER  *    $1704  SET TIMER
                TTIMER *    $1707  TEST TIMER
                STIMER *    $170F  START TIMER
                SAD    *    $1740  SYSTEM DATA REGISTER A
                SADD   *    $1741  SYSTEM DATA DIRECTION A REG
                SBD    *    $1742  SYSTEM DATA REGISTER B
                PBDD   *    $1743  SYSTEM DATA DIRECTION REG B
                KEYIN  *    $1F40  KEYPAD INPUT
                GETKEY *    $1F6A  GET KEYBOARD INPUT

                PAGE ZERO LOCATIONS

0000                   ORG    $0000

                LOW NOTE TABLE

0000 FB         NOTE   =    $FB    G
0001 DF                =    $DF    A
0002 C6                =    $C6    B
0003 BB                =    $BB    C
0004 A6                =    $A6    D
0005 93                =    $93    E
0006 8A                =    $8A    F

                HIGH NOTE TABLE

0007 7B         HINOTE =    $7B    G
0008 6D                =    $6D    A
0009 61                =    $61    B
000A 5B                =    $5B    C
000B 51                =    $51    D
000C 48                =    $48    E
000D 43                =    $43    F

                LOW SHARP NOTE TABLE

000E ED         SHPNOT =    $ED    G SHARP, A FLAT
000F D2                =    $D2    A SHARP, B FLAT
0010 01                =    $01    NO NOTE
0011 B0                =    $B0    C SHARP, D FLAT
0012 9C                =    $9C    D SHARP, E FLAT
0013 01                =    $01    NO NOTE
0014 83                =    $83    F SHARP, G FLAT

                HIGH SHARP NOTE TABLE

0015 74         HISHRP =    $74    G SHARP, A FLAT
0016 67                =    $67    A SHARP, B FLAT
0017 01                =    $01    NO NOTE
0018 56                =    $56    C SHARP, D FLAT
```

```
0019 4C              =    $4C    D SHARP, E FLAT
001A 01              =    $01    NO NOTE
001B 3F              =    $3F    F SHARP, G FLAT

C01C 00              =    $00    UNUSED
001D 00              =    $00
0C1E 00              =    $00
C01F 00              =    $00

0020 02     DELTIM   =    $02    DELAY TIME
0021 00     TIMED    =    $00
0022 00     TIMEC    =    $00
0023 00     SAVFLG   =    $00    SAVE FLAG
0024 00     TLENTH   =    $00    TEMP. LENGTH
0025 00     NOTPTR   =    $00    NOTE POINTER
0026 00     KEYPTR   =    $00    KEY POINTER
0027 00     TNOTE    =    $00    TEMP NOTE
0028 00     HIFLG    =    $00    HIGH FLAG
0029 00     SHPFLG   =    $00    SHARP FLAG
002A 00     NOTNUM   =    $00    NOTE NUMBER
002B 00     PRMNOT   =    $00    PERMANENT NOTE
002C 00     FSTFLG   =    $00    FIRST TIME FLAG
002D 00     PLENTH   =    $00    PERM. LENGTH
002E 00     TNTNUM   =    $00    TEMP. NOTE NUMBER
002F 00     NEXNOT   =    $00    NEXT NOTE
0030 00     DELAYA   =    $00    DELAY A
0031 00     DELAYB   =    $00    DELAY B
0032 00     PNTPTR   =    $00    PERM. NOTE POINTER
0033 00     DELAYC   =    $00    DELAYC
0034 00     TTBPTR   =    $00    TUNETABLE POINTER
0035 00     NTBPTR   =    $00    NOTE TABLE POINTER
0036 00     NOTCNT   =    $00    NOTCNT NOTE COUNT
0037 00     DNTCNT   =    $00    DISPLAY NOTE COUNT
0038 00     TEMNOT   =    $00    TEMP. NOTE
0039 00     TEMLEN   =    $00    TEMP. LENGTH
003A 00     COUNT    =    $00
003B 00     DFOUR    =    $00
003C 00     DTHREE   =    $00
003D 00     DTWO     =    $00
003E 00     DONE     =    $00
003F 00     LNTPTR   =    $00    LENGTH POINTER

            CONSTANTS

0040 01     KEYLNT   =    $01    (1) WHOLE NOTE
0041 02              =    $02    (2) HALF NOTE
0042 04              =    $04    (4) QUARTER NOTE
0043 08              =    $08    (8) EIGHTH NOTE
0044 20     LNTH     =    $20    LENGTH
0045 10              =    $10
0046 08              =    $08
0047 04              =    $04
0048 86     LNSHP    =    $86    (1) LENGTH SHAPE
0049 DB              =    $DB    (2)
004A E6              =    $E6    (4)
004B FF              =    $FF    (8)
004C BD     NTSHP    =    $BD    (G) LETTER SHAPES
004D F7              =    $F7    (A)
004E FC              =    $FC    (B)
004F B9              =    $B9    (C)
0050 DE              =    $DE    (D)
0051 F9              =    $F9    (E)
0052 F1              =    $F1    (F)

0053 00     LETNUM   =    $00    LETTER NUMBER
```

display lit. If we hit the **3** key we jump to a delay which blanks the display. This lets us know a new note has entered the circuits so that we can distinguish two or more same notes in a row. Finally we reset the stack pointer again and display the next note. If we want to start again at any time, we hit the **DA** key and off we go to the beginning again. By the way, the delay subroutine we go to is a good delay to get very long times. It uses the KIM-1's internal timer.

So that's it. I know it is a long program, because of all the explanation, but I want as much understanding as possible, because of the possibilities it holds. The simple tone generation can be replaced with a D/A converter, an erase note mode can be implemented, a larger scale with more lengths and other variables can be developed, and so on. There is no limit. But for a beginning, with a small computer, all you potential Bachs, here it is, go to it.

μ

Table I — Keypad Representations

A = A note
B = B note
C = C note
D = D note
E = E note
F = F note
9 = G note
0 = rest
1 = whole note
2 = 1/2 note
4 = 1/4 note
8 = 1/8 note
5 = sharp
7 = upper octave
3 = save or display next note
DA = Do Again
+ = Display notes

```
0054 0F                  =      $0F
0055 0D                  =      $0D
0056 0B                  =      $0B
0057 09                  =      $09

0058 00         LETTER   =      $00     LETTER SHAPES
0059 F9                  =      $F9
005A BE                  =      $BE
005B F7                  =      $F7
005C ED                  =      $ED

005D            TUNTBL   *      $0060   TUNE TABLE
005D            LNTTBL   *      $00A8   LENGTH TABLE


                DISPLAY NOTE ROUTINE

0100                     ORG    $0100

0100 A9 01      DISNTS   LDAIM  $01     RESET DISPLAY NOTE COUNT
0102 85 37               STA    DNTCNT
0104 A5 37      NXTNOT   LDA    DNTCNT  TEST FOR END
0106 C5 36               CMP    NOTCNT
0108 D0 03               BNE    BEGIN
010A 4C E1 01            JMP    DOAGNB
010D A6 37      BEGIN    LDX    DNTCNT  STORE NOTE
010F B5 60               LDAZX  TUNTBL  AND LENGTH
0111 85 38               STA    TEMNOT
0113 B5 A8               LDAZX  LNTTBL
0115 85 39               STA    TEMLEN
0117 A2 00               LDXIM  $00
0119 A5 38      RPT      LDA    TEMNOT  TEST FOR TEST
011B C9 01               CMPIM  $01
011D F0 61               BEQ    DISZER
011F D5 00               CMPZX  NOTE    TEST FOR NOTE
0121 F0 04               BEQ    SUB
0123 E8                  INX
0124 4C 19 01            JMP    RPT
0127 38         SUB      SEC            TEST FOR FIRST
0128 8A                  TXA            GROUP
0129 E9 07               SBCIM  $07
012B B0 0D               BCS    NXGRPA
012D B5 4C               LDAZX  NTSHP   STORE NOTE SHAPE
012F 85 3D               STA    DTWO
0131 A9 C0               LDAIM  $C0
0133 85 3E               STA    DONE
0135 85 3C               STA    DTHREE
0137 4C 8A 01            JMP    DISLEN

013A 38         NXGRPA   SEC            TEST FOR SECOND
013B 8A                  TXA            GROUP
013C E9 0E               SBCIM  $0E
013E B0 13               BCS    NXGRPB
0140 8A                  TXA
0141 E9 06               SBCIM  $06     STORE NOTE SHAPE
0143 AA                  TAX
0144 B5 4C               LDAZX  NTSHP
0146 85 3D               STA    DTWO
0148 A9 F6               LDAIM  $F6     STORE HI SHAPE
014A 85 3E               STA    DONE
014C A9 C0               LDAIM  $C0
014E 85 3C               STA    DTHREE
0150 4C 8A 01            JMP    DISLEN
```

```
0153 38        NXGRPB SEC            TEST FOR THIRD
0154 8A               TXA            GROUP
0155 E9 15            SBCIM $15
0157 B0 13            BCS   NXGRPC
0159 8A               TXA
015A E9 0D            SBCIM $0D      STORE NOTE SHAPE
015C AA               TAX
015D B5 4C            LDAZX NTSHP
015F 85 3D            STA   DTWO
0161 A9 ED            LDAIM $ED
0163 85 3C            STA   DTHREE
0165 A9 C0            LDAIM $C0
0167 85 3E            STA   DONE
0169 4C 8A 01         JMP   DISLEN

016C 38        NXGRPC SEC            STORE NOTE SHAPE
016D 8A               TXA
016E E9 15            SBCIM $15
0170 AA               TAX
0171 B5 4C            LDAZX NTSHP
0173 85 3D            STA   DTWO
0175 A9 ED            LDAIM $ED      STORE SHARP SHAPE
0177 85 3C            STA   DTHREE
0179 A9 F6            LDAIM $F6
017B 85 3E            STA   DONE
017D 4C 8A 01         JMP   DISLEN

0180 A9 BF     DISZER LDAIM $BF      STORE ZERO SHAPE
0182 85 3D            STA   DTWO
0184 A9 C0            LDAIM $C0
0186 85 3E            STA   DONE
0188 85 3C            STA   DTHREE
018A A2 00     DISLEN LDXIM $00
018C A5 39     RPTB   LDA   TEMLEN TEST FOR LENGTH
018E D5 44            CMPZX LNTH
0190 F0 04            BEQ   GTSHP
0192 E8               INX
0193 4C 8C 01         JMP   RPTB
0196 B5 48     GTSHP  LDAZX LNSHP  STORE LENGTH SHAPE
0198 85 3B            STA   DFOUR
019A A9 80     DIS    LDAIM $80    LOAD DISPLAY
019C 85 33            STA   DELAYC LIGHT TIME
019E A9 7F            LDAIM $7F    SET DIRECTION REGISTER
01A0 8D 41 17         STA   SADD
01A3 A2 04     RPTC   LDXIM $04    SET UP 4 LETTERS
01A5 A0 FF     LITE   LDYIM $FF    AND DISPLAY
01A7 B5 53            LDAZX LETNUM LIGHT LETTERS
01A9 8D 42 17         STA   SBD
01AC B5 3A            LDAZX COUNT
01AE 8D 40 17         STA   SAD
01B1 88        WAIT   DEY          DELAY
01B2 D0 FD            BNE   WAIT
01B4 CA               DEX          GET NEXT LETTER
01B5 10 EE            BPL   LITE
01B7 A4 33            LDYZ  DELAYC DELAY
01B9 88               DEY
01BA 84 33            STYZ  DELAYC
01BC D0 E5            BNE   RPTC
01BE 20 40 1F         JSR   KEYIN  TEST FOR NEXT NOTE
01C1 20 6A 1F         JSR   GETKEY
01C4 C9 03            CMPIM $03
01C6 F0 0C            BEQ   NEXT
01C8 20 40 1F         JSR   KEYIN  TEST FOR START AGAIN
01CB 20 6A 1F         JSR   GETKEY
01CE C9 11            CMPIM $11
```

```
01D0 F0 0F              BEQ    DOAGNB
01D2 D0 C6              BNE    DIS

01D4 20 AC 02  NEXT     JSR    DELAY
01D7 E6 37              INC    DNTCNT INCREMENT DISPLAY NOTE
01D9 A2 FF              LDXIM  $FF    COUNT.  RESET STACK
01DB 9A                 TXS           POINTER
01DC EA                 NOP           PADDING
01DD EA                 NOP
01DE 4C 04 01           JMP    NXTNOT

01E1 A9 00     DOAGNB   LDAIM  $00
01E3 85 36              STA    NOTCNT
01E5 4C 00 02           JMP    NUTUNE


15 16 17
               MAIN PROGRAM

0200                    ORG    $0200

0200 A9 00     NUTUNE   LDAIM  $00    INITIALIZE TUNE
0202 85 23              STA    SAVFLG
0204 85 2A              STA    NOTNUM
0206 85 2C              STA    FSTFLG
0208 A9 01              LDAIM  $01
020A 85 60              STA    TUNTBL
020C 85 A8              STA    LNTTBL
020E 85 27              STA    TNOTE
0210 A9 10              LDAIM  $10
0212 85 24              STA    TLENTH
0214 A9 06     NUNOTE   LDAIM  $06    INITIALIZE NOTE
0216 85 25              STA    NOTPTR
0218 A9 0F              LDAIM  $0F
021A 85 26              STA    KEYPTR
021C 20 40 1F  PLAYB    JSR    KEYIN  TEST KEYPAD FOR NOTE
021F 20 6A 1F           JSR    GETKEY
0222 C5 26              CMP    KEYPTR
0224 F0 2D              BEQ    GTNOTE
0226 C9 00              CMPIM  $00    FOR REST
0228 F0 20              BEQ    GTREST
022A A5 2C              LDA    FSTFLG TEST FOR FIRST TIME
022C C9 00              CMPIM  $00
022E F0 12              BEQ    NOPLAY
0230 C6 26              DEC    KEYPTR SET UP FOR NEXT NOTE
0232 C6 25              DEC    NOTPTR
0234 10 02              BPL    DELYA
0236 30 DC              BMI    NUNOTE

0238 A6 30     DELYA    LDXZ   DELAYA DELAY
023A CA                 DEX
023B 86 30              STXZ   DELAYA
023D D0 DD              BNE    PLAYB
023F 4C 65 02           JMP    SVNOTE

0242 C6 26     NOPLAY   DEC    KEYPTR SET UP FOR NEXT NOTE
0244 C6 25              DEC    NOTPTR
0246 10 D4              BPL    PLAYB
0248 30 CA              BMI    NUNOTE

024A A9 01     GTREST   LDAIM  $01    LOAD REST
024C 85 2C              STA    FSTFLG
024E 85 27              STA    TNOTE
0250 4C 65 02           JMP    SVNOTE
```

```
0253 A9 01      GTNOTE LDAIM $01    LOAD FIRST NOTE FLAG
0255 85 2C             STA   FSTFLG
0257 A6 25             LDXZ  NOTPTR LOAD CHOSEN NOTE
0259 A9 00             LDAIM $00
025B 85 28             STA   HIFLG
025D 85 29             STA   SHPFLG
025F B5 00             LDAZX NOTE
0261 85 27             STA   TNOTE
0263 86 32             STXZ  PNTPTR
0265 20 56 03   SVNOTE JSR   GETHI  GET HIGH NOTE
0268 20 86 03          JSR   GETSRP GET SHARP NOTE
026B 20 DA 03          JSR   GTLNTH GET LENGTH
026E 20 00 03          JSR   PLATUN PLAY NOTE
0271 20 40 1F          JSR   KEYIN  TEST TO SAVE NOTE
0274 20 6A 1F          JSR   GETKEY
0277 C9 03             CMPIM $03
0279 F0 16             BEQ   SAVE
027B 20 40 1F          JSR   KEYIN  TEST OFR START OVER
027E 20 6A 1F          JSR   GETKEY
0281 C9 11             CMPIM $11    DA = DO AGAIN
0283 F0 13             BEQ   DOAGN
0285 20 40 1F          JSR   KEYIN  TEST FOR DIPSLAY NOTER
0288 20 6A 1F          JSR   GETKEY
028B C9 12             CMPIM $12    = +
028D F0 15             BEQ   DNOTES
028F D0 8B             BNE   PLAYB

0291 A9 01      SAVE   LDAIM $01    SAVE NOTE
0293 85 23             STA   SAVFLG
0295 4C 14 02          JMP   NUNOTE

0298 A9 00      DOAGN  LDAIM $00    RESET NOTE COUNTER
029A 85 36             STA   NOTCNT
029C A2 FF             LDXIM $FF    RESET STACK POINTER
029E 9A                TXS
029F EA                NOP          PADDING
02A0 EA                NOP
02A1 4C 00 02          JMP   NUTUNE

02A4 A2 FF      DNOTES LDXIM $FF    RESET STACK POINTER
02A6 9A                TXS
02A7 EA                NOP
02A8 EA                NOP
02A9 4C 00 01          JMP   DISNTS JUMP TO DISPLAY NOTES

                DELAY SUBROUTINE

02AC A5 20      DELAY  LDA   DELTIM GET DELAY VALUE
02AE 85 21             STA   TIMED
02B0 A9 FF      DELA   LDAIM $FF    LOAD TIMER
02B2 8D 04 17          STA   TIMER
02B5 2C 07 17   TEST   BIT   TTIMER TEST TIMER
02B8 10 FB             BPL   TEST   BRANCH IF NOT RUN OUT
02BA C6 22             DEC   TIMEC  REDUCE TIME VALUE
02BC D0 F2             BNE   DELA   START AGAIN
02BE C6 21             DEC   TIMED  REDUCE DELAY VALUE
02C0 D0 EE             BNE   DELA   BRANCH IF NOT DNOE
02C2 60                RTS


02DD                   ORG   $02DD

                TONE SUBROUTINE
```

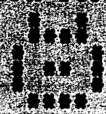
```
02DD A9 01    TONE   LDAIM $01    OPEN PORT
02DF 8D 01 17        STA   PADD
02E2 A9 20    SOUND  LDAIM $20    START TIMER
02E4 8D 0F 17        STA   STIMER
02E7 A6 2B    NOTEX  LDXZ  PRMNOT NOTE FREQUENCY
02E9 CA       NWAIT  DEX
02EA D0 FD           BNE   NWAIT
02EC EE 00 17        INC   PAD    TOGGLE OUTPUT
02EF A9 80           LDAIM $80    TEST COUNTER
02F1 2C 07 17        BIT   TTIMER
02F4 30 03           BMI   TIMOUT
02F6 4C E7 02        JMP   NOTEX
02F9 C6 2D    TIMOUT DEC   PLENTH NOTE LENGTH
02FB D0 E5           BNE   SOUND
02FD 60              RTS


              PLAY TUNE SOBROUTINE

0300                 ORG   $0300

0300 A5 2A    PLATUN LDA   NOTNUM SET UP FIRST NOTE
0302 85 2E           STA   TNTNUM
0304 A9 00           LDAIM $00
0306 85 2F           STA   NEXNOT
0308 A5 27           LDA   TNOTE  PLAY NOTE
030A 85 2B           STA   PRMNOT
030C A5 24           LDA   TLENTH
030E 85 2D           STA   PLENTH
0310 20 DD 02        JSR   TONE
0313 A5 23           LDA   SAVFLG TEST FOR SAVE
0315 C9 01           CMPIM $01
0317 F0 1A           BEQ   SAVEX
0319 A5 2A           LDA   NOTNUM TEST FOR NOTE
031B C9 00           CMPIM $00    (NOT REQUIRED)
031D F0 13           BEQ   RETURN
031F A6 2F    PLAYC  LDXZ  NEXNOT LOAD NEXT NOTE
0321 B5 60           LDAZX TUNTBL
0323 85 2B           STA   PRMNOT
0325 B5 A8           LDAZX LNTTBL LOAD NEXT LENGTH
0327 85 2D           STA   PLENTH
0329 20 DD 02        JSR   TONE   PLAY NOTE
032C E6 2F           INC   NEXNOT SET UP FOR
032E C6 2E           DEC   TNTNUM NEXT NOTE
0330 10 ED           BPL   PLAYC
0332 60       RETURN RTS

0333 A9 00    SAVEX  LDAIM $00    RESET SAVE FLAG
0335 85 23           STA   SAVFLG
0337 A9 01           LDAIM $01    NO PLAY
0339 85 27           STA   TNOTE
033B E6 2A           INC   NOTNUM LOAD NOTE INTO
033D A6 2A           LDXZ  NOTNUM TUNETABLE
033F A5 2B           LDA   PRMNOT
0341 95 60           STAZX TUNTBL
0343 A5 24           LDA   TLENTH LOAD LENGTH
0345 95 A8           STAZX LNTTBL INTO LENGTH TABLE
0347 20 AA 03        JSR   DISPLY
034A 4C 00 03        JMP   PLATUN


0356                 ORG   $0356
```

```
                  GET HIGH SUBROUTINE

0356 20 40 1F     GETHI  JSR    KEYIN  TEST FOR HIGH NOTE
0359 20 6A 1F            JSR    GETKEY
035C C9 07               CMPIM  $07
035E D0 15               BNE    RETRNB
0360 A5 29               LDA    SHPFLG TEST SHARP NOTE
0362 C9 00               CMPIM  $00    (NOT REQUIRED)
0364 F0 09               BEQ    LOADHI
0366 A6 32               LDXZ   PNTPTR LOAD HIGH SHARP NOTE
0368 B5 15               LDAZX  HISHRP
036A 85 27               STA    TNOTE
036C 4C 75 03            JMP    RETRNB (COULD HAVE BEEN RTS)
036F A6 32        LOADHI LDX    PNTPTR LOAD HIGH NOTE
0371 B5 07               LDAZX  HINOTE
0373 85 27               STA    TNOTE
0375 60           RETRNB RTS



0386                     ORG    $0386

                  GET SHARP SUBROUTINE

0386 20 40 1F     GETSRP JSR    KEYIN  TEST FOR SHARP NOTE
0389 20 6A 1F            JSR    GETKEY
038C C9 05               CMPIM  $05
038E D0 0A               BNE    RETRNC
0390 A9 01               LDAIM  $01    LOAD SHARP FLAG
0392 85 29               STA    SHPFLG
0394 A6 32               LDXZ   PNTPTR LOAD SHARP NOTE
0396 B5 0E               LDAZX  SHPNOT
0398 85 27               STA    TNOTE
039A 60           RETRNC RTS



03AA                     ORG    $03AA

                  DISPLAY SAVE SUBROUTINE

03AA A9 80        DISPLY LDAIM  $80    LOAD DISPLAY
03AC 85 33               STA    DELAYC LIGHT TIME
03AE A9 7F               LDAIM  $7F    SET DIRECTION REGISTER
03B0 8D 41 17            STA    SADD
03B3 A2 04        REPEAT LDXIM  $04    SET UP 4 LETTERS
03B5 A0 FF        LIGHT  LDYIM  $FF    AND DELAY
03B7 B5 53               LDAZX  LETNUM LIGHT LETTERS
03B9 8D 42 17            STA    SBD
03BC B5 58               LDAZX  LETTER
03BE 8D 40 17            STA    SAD
03C1 88           WAITY  DEY           DELAY
03C2 D0 FD               BNE    WAITY
03C4 CA                  DEX           GET NEXT LETTER
03C5 10 EE               BPL    LIGHT
03C7 A4 33               LDY    DELAYC DELAY
03C9 88                  DEY
03CA 84 33               STY    DELAYC
03CC D0 E5               BNE    REPEAT
03CE A5 36               LDA    NOTCNT TEST FOR 72 NOTES
03D0 C9 48               CMPIM  $48    48 HEX = 72 DECIMAL
03D2 D0 03               BNE    INCNOT
03D4 4C 00 01            JMP    DISNTS
```

MICROBES

LIFESAVER - MICRO 11

11:10 There should be vertical characters in lines 230 to 290 similar to those in line 220. Also, each line needs a " at the end. Line 310 should read "homeGOTO" not "home GOTO". The correct pattern for the Cheshire Cat is:



AN APPLE II PROGRAM EDIT AID

11:5 Line 32630 should read
J=ASC ... not I=ASC ...

"And here is an addition to make the program run smoother: Add the following lines so that when the screen fills up with text, the Apple will pause and display an "@" in the lower right-hand corner. This will prompt you to hit any key and Apple will clear the screen and continue wherehit left off. This process will continue until until there are no more occur-ences of the search item.

Change: 32690

```
  IF PEEK(37)<18 THEN 32700
```

Add:

```
32692 VTAB 23:TAB 39:PRINT"@"
32695 KEY=PEEK(-16384): IF KEY
        <127 THEN 32695
32697 POKE -16368,0: CALL -936
```

David B. Garson
5163 Willow Wood Road
Rolling Hills Estates, CA 90274

The Ultimate PET Renumber

11:37 A few listing errors:

```
1F15 88     DEY          was C8
1F49 A5 0A  LDAZ         was A9 0A
1F4B F0 2F  BEQ INSF     was F0 17
```

MICRO

```
03D7 E6 36      INCNOT INC    NOTCNT INCREASE NOTE COUNT
03D9 60                RTS

                GET LENGTH SUBROUTINE

03DA A9 03      GTLNTH LDAIM  $03    LOAD LENGTH POINTER
03DC 85 3F             STA    LNTPTR
03DE 20 40 1F   KEYTST JSR    KEYIN  TEST KEYPAD FOR
03E1 20 6A 1F          JSR    GETKEY LENGTH
03E4 A6 3F             LDXZ   LNTPTR
03E6 D5 40             CMPZX  KEYLNT
03E8 F0 05             BEQ    LODLNT
03EA C6 3F             DEC    LNTPTR
03EC 10 F0             BPL    KEYTST
03EE 60                RTS
03EF B5 44      LODLNT LDAZX  LNTH   LOAD LENGTH
03F1 85 24             STA    TLENTH
03F3 60                RTS
```

SYMBOL TABLE 2000 228E

NOTE	0000	HINOTE	0007	SHPNOT	000E	HISHRP	0015
DELTIM	0020	TIMED	0021	TIMEC	0022	SAVFLG	0023
TLENTH	0024	NOTPTR	0025	KEYPTR	0026	TNOTE	0027
HIFLG	0028	SHPFLG	0029	NOTNUM	002A	PRMNOT	002B
FSTFLG	002C	PLENTH	002D	TNTNUM	002E	NEXNOT	002F
DELAYA	0030	DELAYB	0031	PNTPTR	0032	DELAYC	0033
TTBPTR	0034	NTBPTR	0035	NOTCNT	0036	DNTCNT	0037
TEMNOT	0038	TEMLEN	0039	COUNT	003A	DFOUR	003B
DTHREE	003C	DTWO	003D	DONE	003E	LNTPTR	003F
KEYLNT	0040	LNTH	0044	LNSHP	0048	NTSHP	004C
LETNUM	0053	LETTER	0058	TUNTBL	0060	LNTTBL	00A8
DISNTS	0100	NXTNOT	0104	BEGIN	010D	RPT	0119
SUB	0127	NXGRPA	013A	NXGRPB	0153	NXGRPC	016C
DISZER	0180	DISLEN	018A	RPTB	018C	GTSHP	0196
DIS	019A	RPTC	01A3	LITE	01A5	WAIT	01B1
NEXT	01D4	DOAGNB	01E1	NUTUNE	0200	NUNOTE	0214
PLAYB	021C	DELYA	0238	NOPLAY	0242	GTREST	024A
GTNOTE	0253	SVNOTE	0265	SAVE	0291	DOAGN	0298
DNOTES	02A4	DELAY	02AC	DELA	02B0	TEST	02B5
TONE	02DD	SOUND	02E2	NOTEX	02E7	NWAIT	02E9
TIMOUT	02F9	PLATUN	0300	PLAYC	031F	RETURN	0332
SAVEX	0333	GETHI	0356	LOADHI	036F	RETRNB	0375
GETSRP	0386	RETRNC	039A	DISPLY	03AA	REPEAT	03B3
LIGHT	03B5	WAITY	03C1	INCNOT	03D7	GTLNTH	03DA
KEYTST	03DE	LODLNT	03EF	PAD	1700	PADD	1701
TIMER	1704	TTIMER	1707	STIMER	170F	SAD	1740
SADD	1741	SBD	1742	PBDD	1743	KEYIN	1F40
GETKEY	1F6A						

THE MICRO SOFTWARE CATALOG: IX

Mike Rowe
P.O. Box 64
S. Chelmsford, MA 01824

Name: **MASTER CATALOG**
System: **Apple II with disk**
Memory: **32K (min)**
Language: **Applesoft II and machine language**
Hardware: **Apple II, Disk II**
Description: MASTER CATALOG creates a single alphabetized file and listing of each diskette catalog entry by program name, volume number, and program type. MASTER CATALOG will help you locate your programs. A machine language sort will speed the sort by program name or volume number.
Copies: **Just released**
Price: **$8.00**
Includes: Cassette and instructions
Author: **Alan G. Hill**
Available from:
Alan G. Hill
12092 Deerhorn Dr.
Cincinnati, Ohio 45240

Name: **BASIC OPTIMIZER**
System: **Apple II**
Memory: **24K(min) Cassette version, 32K(min) Disk version**
Language: **Interger Basic**
Hardware: **STANDARD (DOS for Disk Version)**
Description: Restructures your basic program by eliminating all remarks, condensing code to long strings, eliminating unneeded line numbers and renaming numeric variables. In affect, the Optimizer creates a production program. You will get a 10% to 40% increase in the speed of execution by running your program through the Optimizer. Now you can write your program with all the remarks you need; give your variables meaningful names. After the program is done, let the Optimizer create a fast efficient production version.
Copies: **Just released**
Price: **$19.95** for Cassette Version. **$25.95** for Disk Version. Texas residents add 5% sales tax
Includes: Cassette Version: Cassette and Instructions. Disk Version: Diskette with sample program and instructions.
Author: **Bruce H. Barber**
Available from:
Bruce H. Barber
11803 Rosewood Drive
Houston, TX 77070

Name: **PET PILOT**
System: **PET**
Memory: **8K**
Language: **BASIC**
Hardware: **Student use: no extra hardware.**
Teacher use: cassette #2
Description: Full Pilot for PET, with full BASIC in C statements & programs to 80K characters. No more memory limitation on program size: you can write real courses in this PILOT.
Copies: **Release date 4-15-79, 5 test sites.**
Price: **$12.00** postpaid in US (prepaid orders only)
Includes: Teacher's Manual Cassette, Reference Card, Licence for 1 machine.
Order Info: Must be prepaid and include PET serial number.
Author: **Dave Gomberg + Martin Kamp**
Available from:
Dave Gomberg + Martin Kamp
7, Gateview Court
San Francisco, CA 94116

Name: **Series One**
System: **PET**
Memory: **8K**
Language: **Not Specified**
Hardware: **Basic 8K PET or 16/32K Full-keyboard PET**
Description: Series One is a collection of 25 programs for the Commodore PET personal computer. For less than one collar each, Series One contains 16 games and 9 general programs. Games include Space Wars, Motorcycle Jump, Saucer Attack, Ping Pong, Bomb Squad, Crack the Safe, Bombs Away, Bite the Wall, Auto Race, Break Away, and six others. Other programs include Mortgage Loan, Perpetual Calendar, Elementary Math, Savings Account, Clock, and more. Most programs take full advantage of the graphics capability of the PET.
Copies: **Not Specified**
Price: **$24.95**
Includes: Not Specified
Author: Not Specified
Available from: Local PET Dealers, or,
ADP Systems
95 West 100 South
Logan, UT 84321

Name: **An 8080 Simulator for the 6502 — KIM-1 Version**
System: **KIM-1**
Memory: **1K**
Language: **Assembly language.**
Hardware: **Unexpanded KIM-1 and (optionally) 8 switches, 1 resistor**
Description: Executes the full 8080 instruction set as though KIM were an 8080-based computer. Supports single-step, trace and run modes and allows monitoring and modification of all internal 8080 registers. User definable input and output ports, breakpoints and access to 6502 subroutines directly from 8080 programs. Up to 224 bytes of 8080 programming space available on an unexpanded KIM-1. Also simulates 8080 interrupts. An excellent training aid for 8080 programming and useful for debugging 8080 code as well as for running non-time dependent 8080 application software. Can be relocated in ROM.
Copies: **90+**
Price: **$18.00** + $1.50 Shipping & Handling. California residents must add 6% sales tax.
Includes: KIM-1 format cassette tape, User Manual, Assembly Source and Object Listings and 8080 Time-Of-Day Clock Demo.
Order Info: Send check or money order.
Author: **Dann McCreary**
Available from:
Dann McCreary
Box 16435-M
San Diego, California 92116

Name: **Light Pen No. 4**
System: **Apple**
Memory: **16K and ROM Board, and Light Pen**
Language: **Apple II Soft**
Description: Program allows user to plot points on the screen in Low-Res, than converts the data to Hi-Res. Plot can be in colors.
Price: **$34.95** + $1.00 postage & handling (PA res. add 6% sales tax)
Includes: Light Pen and 4 other support programs
Copies: **20**
Author: **Neil D. Lipson**
Available from:
Progressive Software
P.O. Box 273
Ply. Mtg., PA 19462

Name: **Morse Code**
System: **Apple**
Memory: **16K**
Language: **Integer Basic**
Description: Program allows user to learn morse code by typing English into computer and having morse code dots and dashes appear on the screen and hearing the beeps (audio) at the same time. Program has transmission speed control.
Copies: **10**
Price: **$9.95** + $1.00 postage & handling (PA residents add 6% sales tax)
Includes: Cassette with instructions
Author: **Ed Hanley**
Available from:
Progressive Software
P.O. Box 273
Ply. Mtg., PA 19462

Name: **GRAFAX**
System: **OSI Challenger IIP**
Memory: **4k (6k optional with buffer)**
Language: **BASIC and machine language**
Hardware Required: **Challenger II(50X cpu,540 video with graphics rom, polled keyboard)**
Description: If you have OSI's rom graphics generator chip then you have been looking for GRAFAX! GRAFAX is designed to give you finger-tip control over the full OSI graphics capabilities. You no longer will need laborious poke list, BASIC string conversions, or machine language kludges just to get something on the screen. GRAFAX uses single key-stroke commands for cursor movement, character selection, 32/64 format, screen save/restore (optional, requires at least 2k ram beyond the basic 4k machine), and cassette save/load. GRAFAX is not an X-Y plotter, but rather a full screen imaging tool carefully devised to free your creative ability for drawing instead of programming.
Copies: **New, just released.**
Price: **$10.00** + 1.00 postage (USA)
Includes: 300 baud cassette (BASIC/machine language source, sample screens), and documentation.
Author: **Mark Bass**
Available from:
Mark Bass
269 Jamison Drive
Frankfort, Illinois 60423

Name: **Polar Coordinate Plot**
System: **Apple**
Memory: **16K and ROM Board**
Language: **Apple II Soft**
Description: A high resolution graphics program which plots polar coordinate equations (4 classic equations and user's own equation). After plot is completed, data (in cartisian and polar coordiante system) will appear on the screen in a summary table form so that the plot can be easily duplicated.
Copies: **50**
Price: **$9.95** + $1.00 postage and handling (PA residents add 6% sales tax)
Includes: Cassette with instructions
Author: **T. David Moteles**
Available from:
Progressive Software
P.O. Box 273
Ply. Mtg., PA 19462

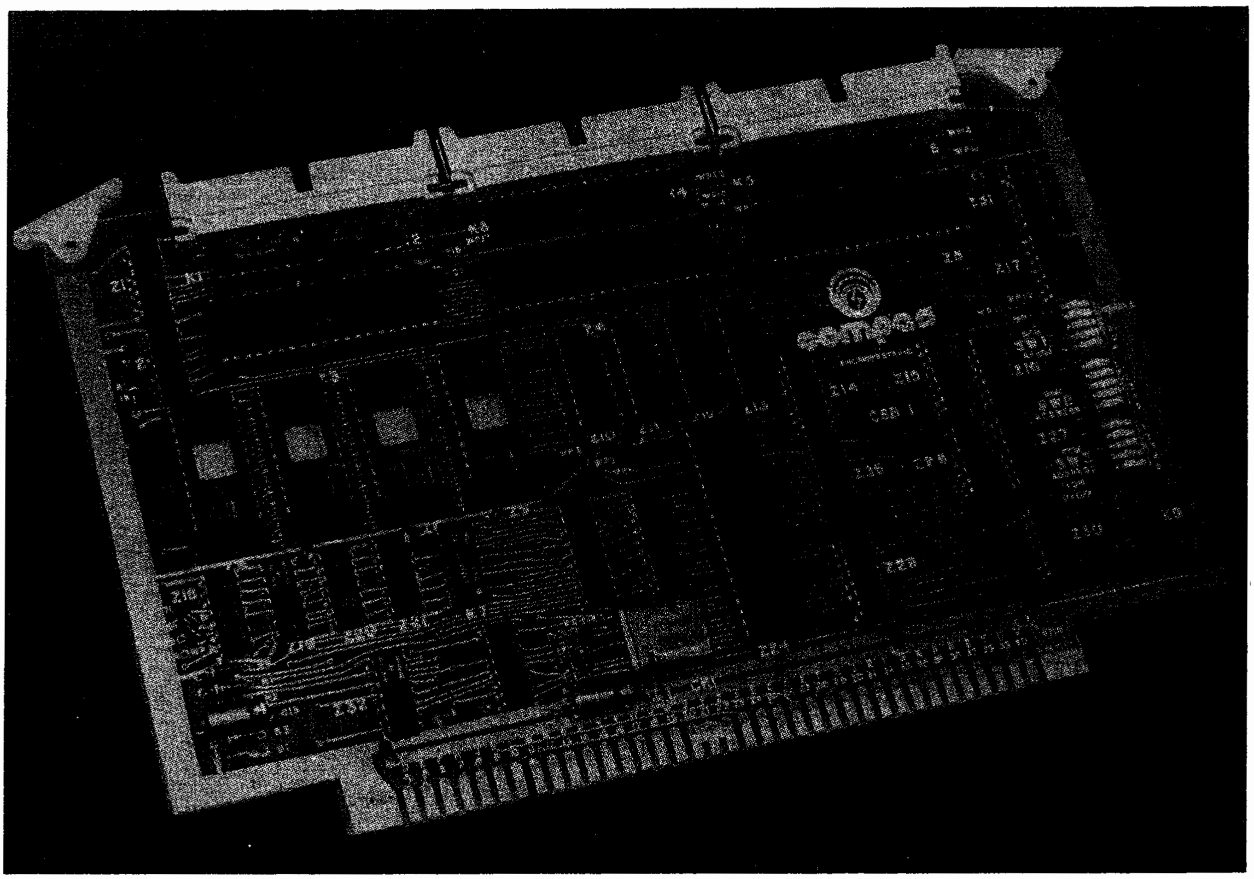
Name: **DISC COPY**
System: **Apple II**
Language: **Applesoft II**
Hardware: **Apple II, Disc II**
Description: For those Apple owners who have only one disc drive but would still like to copy discs that contain Integer Basic and Applesoft programs, this two-part program is a must. It results in an automatic system that can copy a whole discs worth of programs.
Price: **$15**
Copies sold: **Just released**
Author: **Jules H. Gilder**
Available from:
Softsell Associates
2022-79th Street
Brooklyn, NY 11214

compas microsystems

224 S. E. 16th Street
P.O. Box 687
Ames, Iowa 50010
Phone 515-232-8187

single board computers

C/B I



THE 6500 BASED SINGLE BOARD COMPUTER YOU HAVE BEEN LOOKING FOR

An industrial quality 6500 based single board computer produced by the company known for quality 6500 hardware and software. CSB 1 includes a 6502 processor, four ROM/EPROM sockets, 2K bytes RAM(2114), two PIA(6520) and one VIA(6522) chips. All addresses are switch selectable and a single +5 volt power supply is needed. Board size is 6 by 9.75 inches. The gold plated edge connector adheres to the SYSTEM 65 bus standard. Single quantity price is $595 with substantial quantity discounts available. Other CSB board products are available or in design. Call us today for further information on our CSB family or any of our other 6500 support products.

Sym-1: Speak To Me

If you want you computer to carry on some snappy dialog, you are a real dreamer. But, here are some techniques to get you started with limited speech for your micro.

Jack Gieryic
2041 138th Ave. N.W.
Andover, MN 55303

In the February issue of KILOBAUD I came across an interesting article by Robert Bishop. In this article the tape recorder input line was used to sample a voice signal and the tape recorder output line was set high or low depending on the sample. Hopefully the resulting square wave pattern contained enough of the voice's original fundamental to reproduce an understandable signal. A primitive form of speech synthesis indeed but this idea did start the wheels rolling in my head.

Since we are dealing with square waves and not discrete samples at various amplitudes, I thought the sampling frequency would need to be much higher than the theoretical two times the reproduceable frequency. I wrote the accompanying short program which results in slightly over 40,000 samples per second. That's about the limit of the SYM-1 and I figured that should do it for voice. I used my stereo receiver to further help as it has reasonably good amplifiers. I put my cassette deck on record-pause and plugged in a microphone. The tape deck output went to the SYM for sampling on the SYM's normal tape input line. The SYM's tape output line was connected to one of the tape recorder inputs on the stereo receiver.

At this point I began executing the program and listened to the fruits of my labor — sour as they were. The resulting sound reminded me of a small speaker driven to distortion although it was understandable. The recording level control was critical. Remember the SYM sees either a "1" or a "0". The input signal must vary enough to trip the input line. If the volume is too high then too many 1's will be "seen." If the volume is too low then nothing will be heard as the input line will remain a zero. Whistling across (not into) the microphone produced good results. Playing some of my music tapes also was interesting. Only the foreground sounds were "reproduced." The background sounds didn't have enough volume to trip the logic level.

Primitive as this is, it is good enough to use for computer — to-user communications. However, this will require enormous amounts of memory which places a damper on things. The sampling rate of my program would fill 5K of memory in about 1 second. An option would be to reduce the sampling rate. I did this but the results were very poor. Remember this method is using a square wave. At a 10,000 sample/second rate too little information remained and harmonics from the square waves interferred. Change location '225 and '231 to hex 33 and see for yourself. By changing location '234 you can vary the sampling rate. "Ed" will result in 10,000 sample/second. Try "do" and whistle into the microphone. You'll hear the unwanted harmonics so abundant in a square wave.

I had thought of plagerizing the system's tape record/playback routines in order to store the square wave on tape instead of using memory. However, in the high speed format I could only store about 1500 bits per second (185 8 bit characters per second) which is far too low a sampling rate to be useful. Perhaps a floppy disk would have a high enough data transfer rate to be of some use. This is food for thought.

If any of you are still seriously interested then I would suggest an 8 bit analog-to-digital convertor to digitize the data. The data could then be sent through an 8 bit digital-to- analog converter to reproduce the signal. Much lower sampling rates on the order of 5,000 to 8,000 samples/second could be used for voice. However, even at 5,000 8 bit samples per second you would still consume 5K memory in one second.

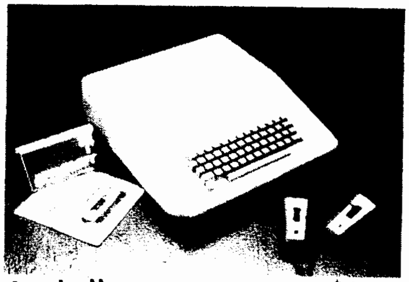
The approach of recording all characteristics of speech for either recognition or future reproduction deserves a reassessment. Perhaps there is some key remaining to be discovered which will enable the computer to use speech with a very limited amount of data. Software may need a greater hardware assist in order to accomplish what seems to be a difficult task. The near future may provide the answer.

SYM SPEAK TO ME

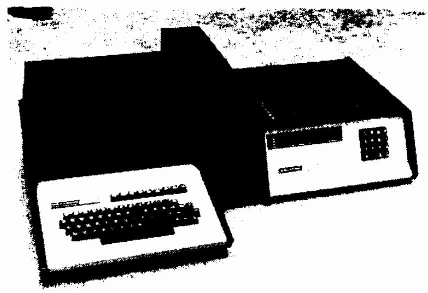
BY JOHN GIERYIC
MAY 1979

PAGE ZERO EQUATE

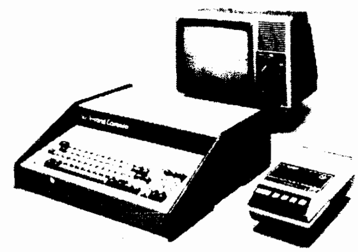
```
    023B            MASK    *      $0001

                  SYM REGISTER EQUATES

023B              VORBX  *      $A000  VIA OUTPUT REGISTER B
023B              VDDRB  *      $A002  VIA DATA DIRECTION REGISTER B
023B              ORB    *      $A402  6532 OUTPUT REGISTER B
023B              DDRB   *      $A403  6532 DATA DIRECTION REGISTER B

0200                     ORG    $0200

0200 A9 07        START  LDAIM  $07    SET UP MASK
0202 85 01               STA    MASK
0204 A9 00               LDAIM  $00    SET DATA DIRECTION FOR INPUT
0206 8D 02 A0            STA    VDDRB  IN VIA
0209 A9 BF               LDAIM  $BF    SET DATA DIRECTION FOR OUTPUT
020B 8D 03 A4            STA    DDRB   6532
020E A9 07               LDAIM  $07    TURN BIT ON - OUTPUT
0210 8D 02 A4            STA    ORB    6532

0213 AD 00 A0     LOOPA  LDA    VORBX  SAMPLE VIA INPUT
0216 29 40               ANDIM  $40
0218 F0 0D               BEQ    LOOPB  IF ZERO, GO CLEAR BIT
021A AD 02 A4            LDA    ORB    IF NOT ZERO, SET BIT
021D 29 F8               ANDIM  $F8
021F 45 01               EOR    MASK
0221 8D 02 A4            STA    ORB    OUTPUT
0224 4C 13 02            JMP    LOOPA

                  CHANGE ABOVE TO JMP DELAY TO CHANGE THE
                  SAMPLE RATE FROM 40,000 TO 10,000 CPS
                  LOCATION 224 = 33
                  ALSO CHANGE LOCATION 231.

0227 AD 02 A4     LOOPB  LDA    ORB    CLEAR BIT
022A 29 F8               ANDIM  $F8
022C EA                  NOP           FOR TIMING
022D 8D 02 A4            STA    ORB    OUTPUT
0230 4C 13 02            JMP    LOOPA

                  CHANGE ABOVE TO JMP DELAY TO CHANGE THE
                  SAMPLE RATE FROM 40,000 TO 10,000 CPS
                  LOCATION 231 = 33
                  ALSO CHANGE LOCATION 224.

0233 A2 ED        DELAY  LDXIM  $ED    96 MICROSECOND DELAY
0235 E8           LOOPC  INX
0236 D0 FD               BNE    LOOPC
0238 4C 13 02            JMP    LOOPA
```



Apple II $1195



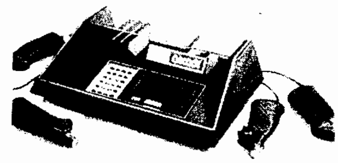
Heathkit H-8 $375



SOL-20 $1600

Which of these computers is for you?

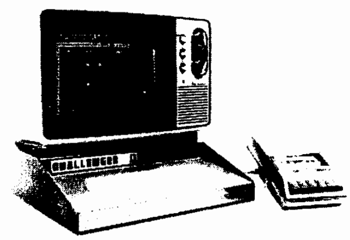
Exidy Sorcerer $895



Bally Arcade $399

CREATIVE COMPUTING magazine is Number 1 in hardware, software and system evaluations. In-depth, thorough evaluations give you the facts **before** you buy. **Creative Computing** was the first to review these now popular systems: Radio Shack TRS-80, Exidy Sorcerer, VideoBrain, Heath H-8, Bally Basic, OSI Challenger, and many others. More important, we also review peripherals and software from independents as well as manufacturers.

Radio Shack TRS-80 $599



Ohio Scientific C2-4P $598

And what are you going to do with it?

CREATIVE COMPUTING has long been Number 1 in applications and software for micros, minis, and time-sharing systems for homes, schools and small businesses. Loads of applications every issue: text editing, graphics, communications, artificial intelligence, simulations, data base and file systems, music synthesis, analog control. Complete programs with sample runs. Programming techniques: sort algorithms, file structures, shuffling, etc. Coverage of electronic and video games and other related consumer electronics products, too.

Just getting started? Then turn to our technology tutorials, learning activities, short programs, and problem solving pages. No-nonsense book reviews, too. Even some fiction and foolishness.

Subscriptions: 1 year $15, 3 years $40. Foreign, add $9/year surface postage, $26/year air. Order and payment to: Creative Computing, attn: Ellen, P.O. Box 789-M, Morristown, NJ 07960. Visa or Master Charge acceptable by mail or phone; call **800-631-8112** 9 am to 5 pm EST (in NJ call 201-540-0445).

CREATIVE COMPUTING also publishes books, games, art prints, and T-shirts for the computer enthusiast. The most popular book of computer games in the world, **Basic Computer Games** is a Creative Computing book — only $8.50 postpaid.

And now, Creative Computing also produces and markets software for personal computers on tape cassette and floppy disk.

If your dealer does not carry the full line of Creative Computing products, write "catalog" on your order and we'd be happy to send you one free.

creative computing

P. O. Box 789-M, Morristown, NJ 07960

Writing For MICRO

If you have information you wish to pass on to the 6502 community, think of **MICRO** first. Since **MICRO** is devoted solely to 6502 material, your article will probably be read by more people than it would be in many of the "general purpose" national magazines. And, **MICRO** pays for articles. You'll never get rich, but you can get enough from an article to buy that extra memory you want. Also, our policy is to give authors additional payments if the material is reprinted. For example, authors whose work appeared in the original six issues of **MICRO** and were reprinted in **"The BEST of MICRO Volume 1"** are receiving double payment — so it can add up.

WHAT TO SUBMIT — MICRO is only interested in things which pertain to the 6502 microprocessor. Since **MICRO** takes the 6502 seriously, it is less interested in games than in useful material such as utility programs, tutorials, resources, interfacing techniques, and so forth. Complete "ready-to-load-and-go" programs are preferred to "think-pieces". Look at what we have published in the past as a guide to what types of articles we prefer.

TEXT — All material should be typed, double-spaced on one side on white paper. Your name and a page number should appear on each page. Try to adhere as much as possible to the format that **MICRO** uses for publication — it saves us a lot of work. Paragraphs are not indented, sub-headings are centered, figures are referenced as Figure 1 (2, 3, . . . n) and tables as Table I (II, III, IV, . . . N). You should suggest a title and may write a brief summary, although this is not necessary.

PROGRAM LISTINGS — All listings should be printed directly from your computer if possible, with a new black ribbon on plain white paper. If your listings are clean enough to print, then we do not have to typeset them, decreasing the chance for typos. Since line numbers used by your editor may be removed, please make all references to your listings by address or PC counter.

DRAWINGS and SCHEMATICS — A good drawing can add a lot of visual interest as well as information to your article. We do not have any special talents in this area, plus we may make mistakes when redrawing your work. So please, if at all possible, submit drawings in ink, ready for reproduction. Since most drawings will be reduced to fit into two columns — approximately four inches, keep this size in mind when planning your work.

PERSONAL INFORMATION — Our readers would like to know something about the authors. If you submit a short note about yourself — your computer experience, talents, interests, plans — we will publish this along with your article. You have done the work, why not enjoy the credit!

RETURN OF MANUSCRIPTS — If you want your manuscript returned, just tell us. We will supply the stamp and envelope — its the least we can do.

PAYMENT — MICRO pays for all articles which we publish. You will be sent a Manuscript Release Form prior to your article being printed. By signing this and returning it to us, you certify that the work you have submitted is yours to sell, and convey all rights to the work to MICRO Ink, Inc. You will receive six copies of your article.

LETTERS, NOTES, ANNOUNCEMENTS, etc. — If you have short pieces of information that do not fit into the article format, **MICRO** has a number of ways of using them. They may be submitted in almost any form, even hand-written, and will be used as the Editor sees fit. There is no payment for these submissions, but full credit is given to the author.

Reading Pet Cassettes Without a Pet

If you have ever wanted to read a program from a cassette written for one 6502 based microcomputer on another type, here is an example which uses a SYM-1 to read a PET cassette. The concepts can be generalized to work with almost any combination of micros.

David P. Kemp
1307 Beltram Court
Odenton, MD 21113

One of the basic problems in obtaining microcomputer software is not that it doesn't exist but that it was written for a machine other than the one it is to be used on. Small programs can be typed in by hand if a hex listing is available, but larger programs are generally distributed on audio cassettes. By virtue of their popularity, the Apple II and PET have the largest pools of published software on cassette, but that doesn't mean that owners of less well established microcomputers like the SYM-1 cannot take advantage of existing programs written for these machines.

All 6502 based microcomputers except the KIM use very simple cassette interface hardware and let the processor do all the work of formatting, encoding and decoding cassette data. This approach has the dual advantages of reducing parts count and increasing flexibility and it means that with suitable software, users of any particular machine can read cassettes written for any other machine (Apple, PET, OSI, AIM, or SYM). This particular program runs on the SYM-1 and reads cassettes written by the PET. It is quite unsophisticated, and doesn't know the difference between various block types such as Beginning of File, End of File, Program, and Data blocks, and it does not strip off countdown bytes or verify checksums. It does check byte parity and will flag any errors; it has been my experience that if there are no parity errors, then the data is OK.

Because the task of converting software from one machine to another is non-trivial, it is assumed that only experienced programmers will have occasion to use **PETCAS,** thus no attempt will be made to explain the program's operation or PET cassette format in detail; however one feature of the program deserves some comment -- the tuning dispay. If an oscilloscope and a D/A converter are available, the display simplifies setting up the program and the recorder controls. With the program running and a PET cassette playing, the scope trace should fall into three distinct levels corresponding to the three possible time periods between active transitions on the tape. If the display is not well clustered or the routine will not work, try exchanging the instructions at locations $6C and $66. (In PET cassettes polarity is significant and this modification effectively reverses the audio signal polarity).

Despite its small size, the program works quite well -- it was originally written to read a third generation analog dubbing of an 8K program, and it accomplished that task in one pass without an error. μ

PET CASSETTE READ ROUTINE FOR SYM-1

BY DAVE KEMP
JANUARY 1979

PETCAS READS A PET FORMAT CASSETTE RECORD AND STORES THE DATA IN A BUFFER STARTING AT $200. TO RUN IT, TYPE:

.G 0

CONTROL IS RETURNED TO SUPERMON AFTER THE FIRST COPY OF THE DATA HAS BEEN READ. LOCATIONS ADH AND ADL POINT TO THE NEXT FREE BUFFER LOCATION (LAST BYTE READ + 1).

```
0075            ADL     *     $00F0  BUFFER POINTER
0075            ADH     *     $00F1
0075            TCNT    *     $00F2  GETBIT TEMPORARY
0075            TPAR    *     $00F3  PARITY GENERATOR TEMPORARY
0075            PECNT   *     $00F4  PARITY ERROR COUNT

0075            PAR     *     $00FE  PARITY ERROR MARKER
                VALUE ARBITRARY

0075            TAPE    *     $A000  CASSETTE INPUT PORT (PB6)
0075            DIGANA  *     $A001  DIGITAL TO ANALOG CONVERTER OUTPUT

0000                   ORG    $0000

0000 A9 02      PETCAS LDAIM $02    SET BUFFER ADDRESS TO $0200
0002 85 F1             STA   ADH
0004 85 F4             STA   PECNT
0006 A9 00             LDAIM $00
0008 85 F0             STA   ADL
000A 20 2F 00   PETCA  JSR   GBYTE  GET A BYTE
000D 30 03             BMI   PETX
000F 4C 00 00          JMP   PETCAS LEADER NOT STABLE YET
0012 C6 F4      PETX   DEC   PECNT  BE SURE LEADER IS VALID
0014 D0 F4             BNE   PETCA
0016 20 2F 00   PETCB  JSR   GBYTE  GET BYTE
0019 30 FB             BMI   PETCB  LOOP UNTIL END OF LEADER
001B B0 04      PETCC  BCS   PETCD  DATA VALID ?
001D A9 FE             LDAIM PAR    NO - PARITY ERROR
001F E6 F4             INC   PECNT  INCREMENT ERROR COUNT
0021 91 F0      PETCD  STAIY ADL    SAVE IT IN BUFFER
0023 E6 F0             INC   ADL    ADVANCE BUFFER POINTER
0025 D0 02             BNE   PETCE
0027 E6 F1             INC   ADH
0029 20 2F 00   PETCE  JSR   GBYTE  GET ANOTHER BYTE
002C 10 ED             BPL   PETCC  CONTINUE IF DATA
002E 60                RTS          EXIT IF SHORTS
```

GET A BYTE OF PET DATA

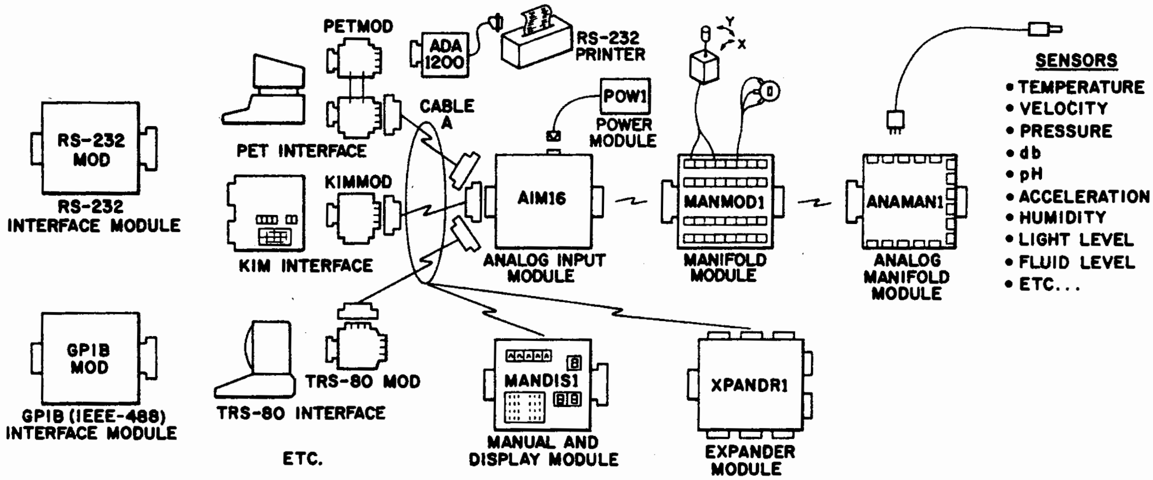
RETURN:

```
                    A = BYTE
                    C = 0 IF PARITY ERROR
                    N = 1 IF SHORTS
                    X CLOBBERED, Y = 0

002F A0 11    GBYTE  LDYIM $11    SHORTS COUNT
0031 20 63 00 GBA    JSR   GETTR  GET TRANSITION TIME
0034 E0 40           CPXIM $40    START BIT ?
0036 B0 08           BCS   GBB    YES - GO GET BYTE
0038 E0 2C           CPXIM $2C    SHORTS ?
003A B0 F3           BCS   GBYTE  NO - START COUNTING AGAIN
003C 88              DEY          YES - DECREMENT COUNT
003D 10 F2           BPL   GBA
003F 60              RTS

0040 A0 09    GBB    LDYIM $09    BIT COUNT
0042 84 F3           STY   TPAR   INITIALIZE PARITY
0044 20 63 00        JSR   GETTR  GET OTHER HALF OF START BIT
0047 20 58 00 GBC    JSR   GETBIT GET A DATA BIT
004A 90 02           BCC   GBD
004C E6 F3           INC   TPAR   ADJUST PARITY
004E 6A       GBD    RORA         PACK IT
004F 88              DEY          DONE ?
0050 D0 F5           BNE   GBC    NO
0052 2A              ROLA         YES - ADJUST DATA
0053 49 FF           EORIM $FF
0055 46 F3           LSR   TPAR   PUT PARITY IN C
0057 60              RTS

              GET A DATA BIT

                RETURN:
                  C = BIT
                  X CLOBBERED, A & Y UNCHANGED

0058 20 63 00 GETBIT JSR   GETTR  GET FIRST TRANSITION
005B 86 F2           STX   TCNT   SAVE IT
005D 20 63 00        JSR   GETTR  GET SECONT TRANSITION
0060 E4 F2           CPX   TCNT   GENERATE BIT IN C
0062 60              RTS

              GET A TRANSITION PERIOD

                RETURN:
                  X = PERIOD
                  A & Y UNCHANGED

0063 A2 00    GETTR  LDXIM $00    INIT COUNTER
0065 2C 00 A0 GETA   BIT   TAPE
0068 E8              INX          INCREMENT COUNTER
0069 70 FA           BVS   GETA   LOOP WHILE HIGH
006B 2C 00 A0 GETB   BIT   TAPE
006E E8              INX
006F 50 FA           BVC   GETB   LOOP WHILE LOW
0071 8E 01 A0        STX   DIGANA OUTPUT TO D/A FOR TUNING
0074 60              RTS
```

CONNECTICUT microCOMPUTER, Inc.

150 POCONO ROAD - BROOKFIELD, CONNECTICUT 06804

(203) 775-9659



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.

AIM162 - Analog Input Module — $249.00
As above plus: greater accuracy - gold plated contacts - pilot light - switch selectable start, enable and ready polarities.

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 — $189.00
Includes one AIM161, one POW1, one ICON and one OCON.

AIM162 Starter Set — $259.00
Includes one AIM162, one POW1, one ICON 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.

APPLE II® PROFESSIONAL SOFTWARE

PIE TEXT EDITOR

PIE (PROGRAMMA IMPROVED EDITOR) is a two-dimensional cursor-based editor designed specifically for use with memory-mapped and cursor-based CRT's. It is totally different from the usual line-based editors, which were originally designed for Teletypes. The keys of the system input keyboard are assigned specific PIE Editor function commands. Some of the features included in the PIE system are: Blinking Cursor; Cursor movement up, down, right, left, plus tabs; Character insert and delete; String search forwards and backwards; Page scrolling; GOTO line number, plus top or bottom of file; Line insert and delete anywhere on screen; Move and copy (single and multiple lines); Append and clear to end of line; Efficient memory usage. The following commands are available in the PIE Text Editor and each is executed by depressing the systems argument key simulataneously with the command key desired:

[LEFT]	Move cursor one position to the left
[RGHT]	Move cursor one position to the right
[UP]	Move cursor up one line
[DOWN]	Move cursor down one line
[BHOM]	Home cursor in lower left left hand corner
[HOME]	Home cursor in upper left hand corner
[-PAG]	Move up (toward top of file) one "page"
[+PAG]	Move down (toward bottom of file) one "page"
[LTAB]	Move cursor left one horizontal tab
[RTAB]	Move cursor right one horizontal tab
[GOTO]	Go to top of file (line 1)
[ARG] n[GOTO]	Go to line 'n'
[+GOTO]	Go to bottom of file (last line + 1)
[-SCH]	Search backwards (up) into file for the next occurence of the string specified in the last search command
[ARG] t[-SCH]	Search backwards for string 't'
[+SCH]	Search forwards (down) into the file for the next occurence of the string specified in the last search command
[ARG] t[+SCH]	Search forward for string 't'
[APP]	Append -move cursor to last character of line +1
[INS]	Insert a blank line beforere the current line
[ARG] n[INS]	Insert 'n' blank lines before the current line
[DEL]	Delete the current line, saving it in the "push" buffer
[ARG] n[DEL]	Delete 'n' lines and save the first 20 in the "push" buffer
[DBLK]	Delete the current line as long as it is blank
[PUSH]	Save current line in "push" buffer
[ARG] n[PUSH]	Save 'n' lines in the "push" buffer
[POP]	Copy the contents of the "push" buffer before the current line
[CINS]	Enable character insert mode
[CINS] [CINS]	Turn off character insert mode
[BS]	Backspace
[GOB]	Gobble - delete the current character and pull remainder of characters to right of cursor left one position
[EXIT]	Scroll all text off the screen and exit the editor
[ARG] [HOME]	Home Line - scroll up to move current line to top of screen
[APP] [APP]	Left justify cursor on current line
[ARG] [GOB]	Clear to end of line

Apple PIE Cassette	**16K**	**$19.95**
TRS-80PIE Cassette	**16K**	**19.95**
Apple PIE Disk	**32K**	**24.95**

6502FORTH · Z-80FORTH 6800 FORTH

FORTH is a unique threaded language that is ideally suited for systems and applications programming on a micro-processor system. The user may have the interactive FORTH Compiler/Interpreter system running standalone in 8K to 12K bytes of RAM. The system also offers a built-in incremental assembler and text editor. Since the FORTH language is vocabulary based, the user may tailor the system to resemble the needs and structure of any specific application. Programming in FORTH consists of defining new words, which draw upon the existing vocabulary, and which in turn may be used to define even more complex applications. Reverse Polish Notation and LIFO stacks are used in the FORTH system to process arithmetic expressions. Programs written in FORTH are compact and very fast.

SYSTEM FEATURES & FACILITIES

Standard Vocabulary with 200 words
Incremental Assembler
Structured Programming Constructs
Text Editor
Block 1/0 Buffers
Cassette Based System
User Defined Stacks
Variable Length Stacks
User Defined Dictionary
Logical Dictionary Limit
Error Detection
Buffered Input

CONFIGURATIONS

AppleFORTH Cassette 16K	**$34.95**
AppleFORTH Disk 32K	**49.95**
PetFORTH Cassette 16K	**34.95**
TRS-80FORTH Cassette 16K	**34.95**
SWTPCFORTH Cassette 16K	**34.95**

ASM/65 EDITOR ASSEMBLER

ASM/65 is a powerful, 2 pass disk-based assembler for the Apple II Computer System. It is a compatible subset of the FORTRAN cross-assemblers which are available for the 6500 family of micro-processors. ASM/65 features many powerful capabilities, which are under direct control of the user. The PIE Text Editor co-resides with the ASM/65 Assembler to form a comprehensive development tool for the assembler language programmer. Following are some of the features available in the ASM/65 Editor Assembler.

PIE Text Editor Command Repetoire
Disk Based System
Decimal, Hexadecimal, Octal, & Binary Constants
ASCII Literal Constants
One to Six character long symbols
Location counter addressing "*"
Addition & Subtraction Operators in Expressions
High-Byte Selection Operator
Low-Byte Selection Operator
Source statements of the form:
[label] [opcode] [operand] [;comment]
56 valid machine instruction mnemonics
All valid addressing modes
Equate Directive
BYTE Directive to initialize memory locations
WORD Directive to initialize 16-bit words
PAGE Directive to control source listing
SKIP Directive to control source listing
OPT Directive to set select options
LINK Directive to chain multiple text files
Comments
Source listing with object code and source statements
Sorted symbol table listing

CONFIGURATION

Apple II	**48K/Disk**	**$69.95**

LISA INTERACTIVE ASSEMBLER

LISA is a totally new concept in assembly language programming. Whereas all other assemblers use a separate or co-resident text editor to enter the assembly language program and then an assembler to assemble the source code, LISA is fully interactive and performs syntax/addressing mode checks as the source code is entered in. This is similar in operation to the Apple II Integer BASIC Interpreter. All error messages that are displayed are in plain, easy to understand English, and not simply an Error Code. Commands in LISA are structured as close as possible to those in BASIC. Commands that are included are: LIST, DELETE, INSERT, PR #n, IN #n, SAVE, LOAD, APPEND, ASM, and a special user-defineable key envisioned for use with "dumb" peripherals. LISA is DISK II based and will assemble programs with a textfile too long to fit into the Apple memory. Likewise, the code generated can also be stored on the Disk, hence freeing up memory for even larger source programs. Despite these Disk features, LISA is very fast; in fact LISA is faster than most other commercially available assemblers for the Apple II. Not only is LISA faster, but also, due to code compression techniques used LISA requires less memory space for the text file. A full source listing containing the object and source code are produced by LISA, in addition to the symbol table

Apple II 32K/Disk $34.95

PROGRAMMA INTERNATIONAL, INC.

3400 Wilshire Blvd.
Los Angeles, CA 90010

(213) 384-0579 · 384-1116 · 384-1117

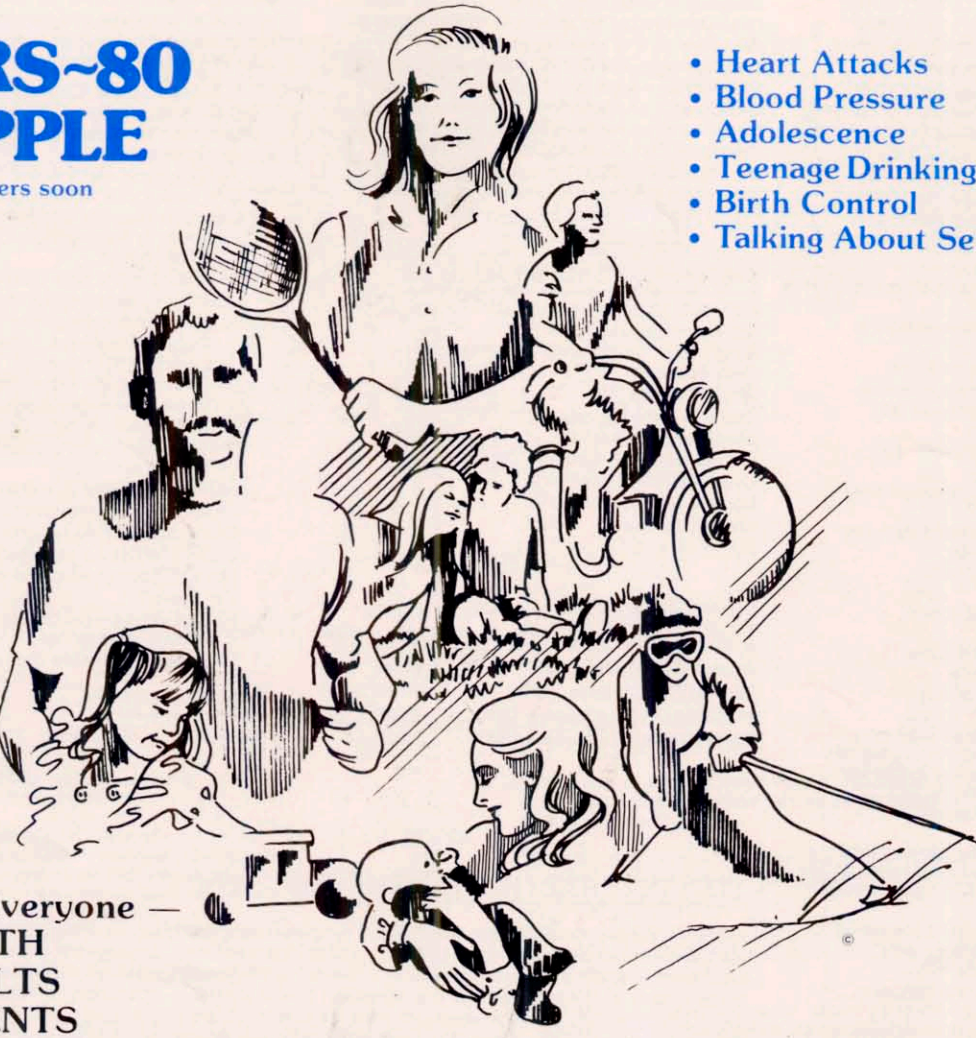
PROGRAMMA Software Products

NEW HEALTH INTEREST PROGRAMS...

VitaFacts™ by Speakeasy

TRS~80
APPLE
and others soon

- Heart Attacks
- Blood Pressure
- Adolescence
- Teenage Drinking & Drugs
- Birth Control
- Talking About Sex



For Everyone — YOUTH ADULTS PARENTS

Speakeasy VitaFacts programs contain up-to-date factual medical information presented in layman's terms about important aspects of your life and health needs. By participating you gain:

— improved knowledge of medical situations, contributing to an improved lifestyle, better family communication and cooperation and better health.
— awareness and appreciation of modern life's stressful situations and how to cope with them.
— a better understanding of family or friends' health problems.
— a heightened sense of the joy of learning.

Help yourself — your family — your friends to better health with the new Speakeasy VitaFacts Series learning programs!

These professionally-prepared programs include an audio cassette, a computer cassette and a booklet. They are available through hundreds of computer stores worldwide at very reasonable prices. Please check with your dealer or contact Speakeasy VitaFacts.

DEALERS: Please call us at the number below for the name of your distributor.

MICROCOMPUTER-ASSISTED INSTRUCTION FOR HEALTHFUL LIVING!

VitaFacts™ by Speakeasy™

Box 909 Kemptville, Ontario, Canada K0G 1J0 (613) 258-3291