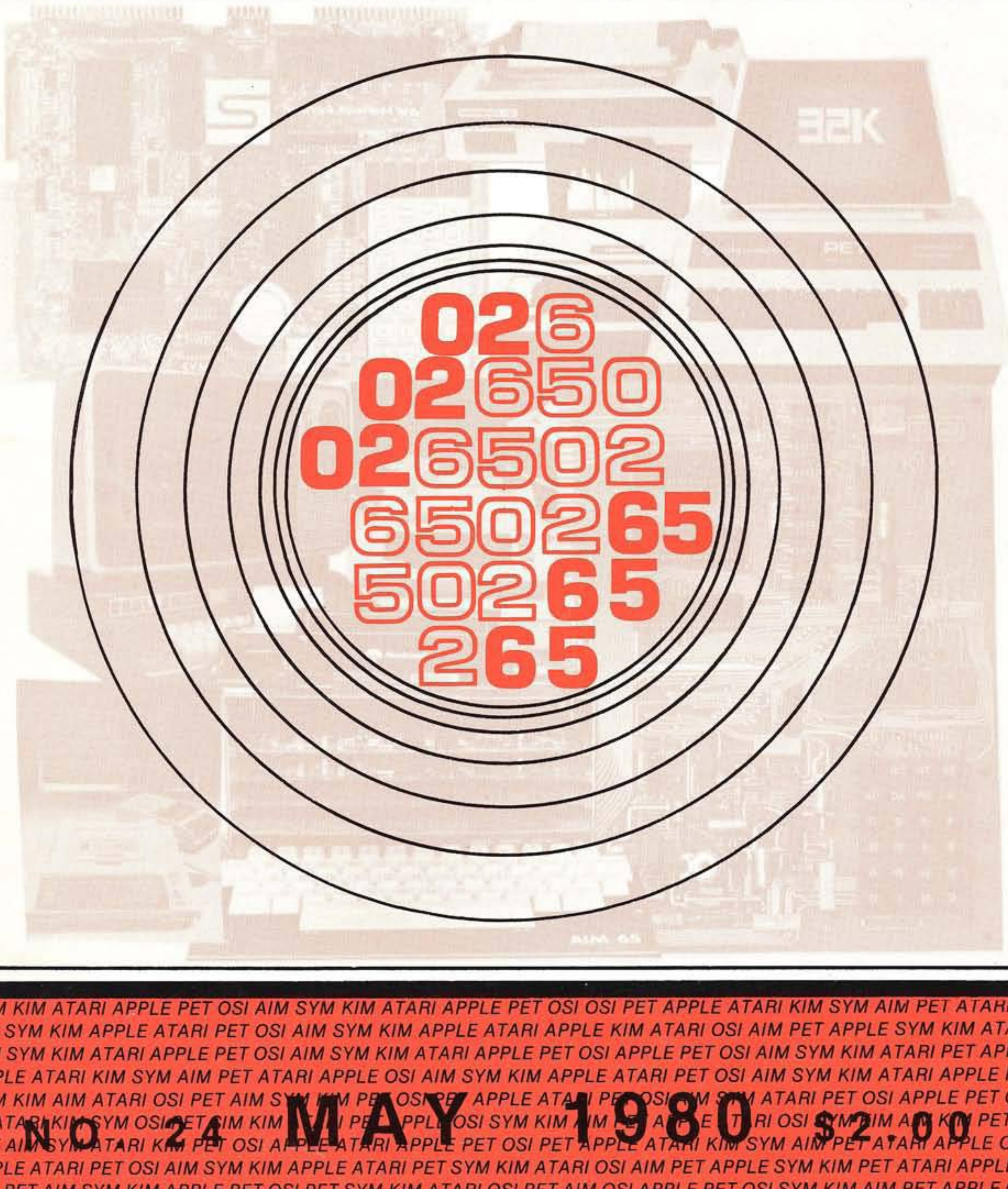
MICRO

THE 6502 JOURNAL



NO. 24 MAY 1980 $2.00

GRAND OPENING...SYSTEMS FORMULATE CORPORATION

Your Microcomputer Information Center

We welcome you to visit our first consumer commercial sales center in the United States. Located in Palo Alto in the Town and Country Shopping Center, Systems Formulate features a full range of microcomputer systems and peripherals for home or commercial use.

During the past few years since our founding in 1978, we have opened three retail stores in Japan stocking premiere American-made microsystems and innovative Japanese systems.

Our new Palo Alto location will feature some exciting surprises for the American interested in microprocessing.

Miplot, the intelligent plotter ...Only $1,200

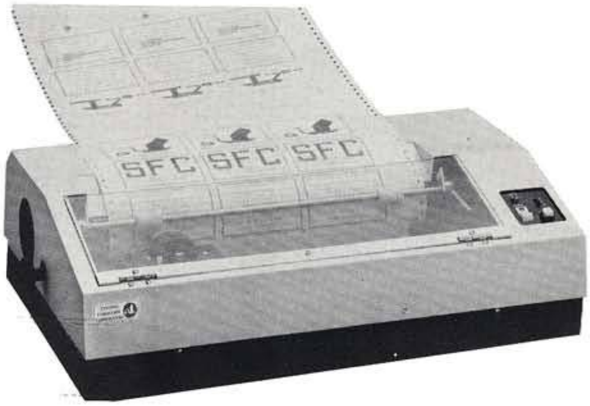
Usable even by those with no plotter experience the Miplot is an ideal graphic output device for the personal and small business computer. It's light, compact, and uses ordinary, hard-tip felt pens. (Interchange and cable available)

Special features include:

- Separate pulse motor drives for x and y axis
- Full range of intelligent functions such as line generator and character generator
- Self-test-mode to verify plotter results
- Printer mode for easy interface testing and data dumping

For research, development and management planning, or use by the personal hobbyist...Miplot is the intelligent plotter worth your attention.

Now on display at Systems Formulate.



Rugged, continuous-feed printers ...Only $1,995

Systems Formulate presents two, new continuous feed printers designed to withstand all the heavy duty use your invoices, statements and statistical analyses can dish out.

Both the SFC-080 and the SFC-136 combine Japanese craftsman ship with many of America's finest components. Like the UMI 8 dot high matrix head, for example.

But that's just the beginning:

- Superior data editing ability
- Your choice of paper size
- Sharp, clear printouts
- Bold double-size characters
- Graphics
- Versitile line spacing
- Paper-saver saving design

Call Systems Formulate and we'll be happy to send you detailed specs or, better yet drop by for a personal demonstration.

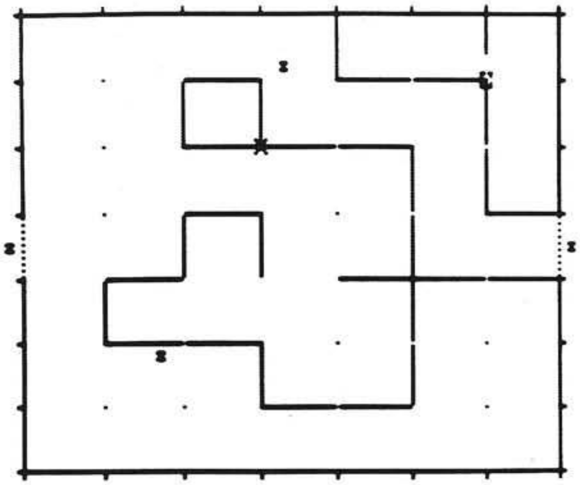
SYSTEMS FORMULATE CORPORATION

**39 Town & Country Village
Palo Alto, California 94301
(415) 326-9100**

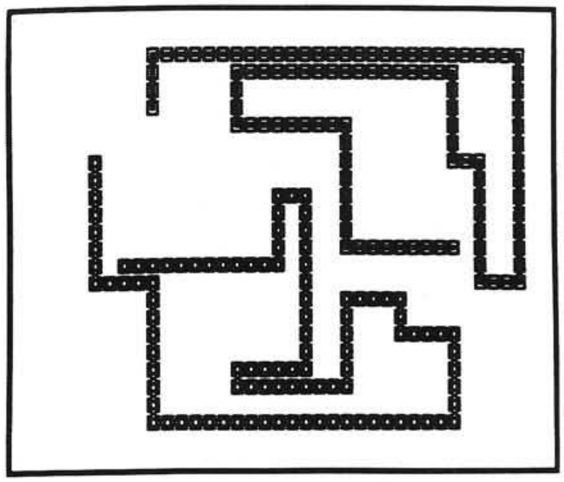
SYSTEMS FORMULATE CORPORATION

Store hours: Monday-Saturday, 10-7
Sunday, 12-6

Software for the Apple II



SCORE:108



SCORE:105

DYNAMAZE—a dazzling new real-time game. You move in a rectangular game grid, drawing or erasing walls to reflect balls into your goal (or to deflect them from your opponent's goal). Every ball in your goal is worth 100 points, but you lose a point for each unit of elapsed time and another point for each time unit you are moving. Control the speed with a game paddle: play as fast as ice hockey or as slowly and carefully as chess. Back up and replay any time you want to; it's a reversible game. By Don Stone. Integer Basic (plus machine language); 32 K; $9.95.

ULTRA BLOCKADE— the standard against which other versions have to be compared. Enjoy Blockade's superb combination of fast action (don't be the one who crashes) and strategy (the key is accessible open space—maximize yours while minimizing your opponent's). Play against another person or the computer. New high resolution graphics lets you see how you filled in an area—or use reversibility to review a game in slow motion (or at top speed, if that's your style). This is a game that you won't soon get bored with! By Don Stone. Integer Basic (plus machine language); 32 K; $9.95.

What is a REVERSIBLE GAME? You can stop the play at any point, back up and then do an "instant replay", analyzing your strategy. Or back up and resume the game at an earlier point, trying out a different strategy. Reversibility makes learning a challenging new game more fun. And helps you become a skilled player sooner.

WORLD OF ODYSSEY—a new adventure game utilizing the full power of Disk II, which enables the player to explore 353 rooms on 6 different levels full of dragons, dwarfs, orcs, goblins, gold and jewels. Applesoft II 48K; $19.95 includes diskette.

PERQUACKEY—an exciting vocabulary game which pits the player against the clock. The object of the game is to form words from a group of 10 letters which the computer chooses at random. The words must be 3 to 10 characters in length with no more than 5 words of any particular length. Each player has only 3 minutes per turn. The larger the words the higher the score. Applesoft II 16K; $9.95.

APPLESHIP—is a naval game in which two players enter their ships in respective oceans. Players take turns trying to blast their opponent's ships out of the water. The first player to destroy their opponent's ships may win the game. A great low-res graphics game. Applesoft II 32K; $14.95.

Available at your local computer store

Call or write for our free SOFTWARE CATALOG

Apple II is a registered trademark of Apple Computer, Inc.

DEALER INQUIRIES INVITED

POWERSOFT, INC.

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

Programs Available on Diskette at $5.00 Additional

- Check or Money Order
- Include $1.00 for shipping and handling
- C.O.D. ($1.00 add'tl. charge)
- Master Charge and VISA orders accepted
- New Jersey residents add 5% sales tax

HDE inc.

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

HUDSON DIGITAL ELECTRONICS INC.

ANNOUNCING THE HDE OMNIDISK 65/8

Now, you can "plug in" the latest in a successful series of flexible disk systems developed by HDE for the KIM, SYM and AIM microcomputers. The OMNIDISK 65/8 is a complete system, using 8 inch soft sectored diskettes with a formatted (IBM Standard) capacity of 256K. Of course, a disk formatting function is included as are system supporting utilities for file renaming, disk packing, copy (dual systems) and others.

TED, a full featured, line oriented editor is standard in KIM and SYM based versions to get you up and running on your project in a hurry. The AIM version uses the on-board editor. With the OMNIDISK 65/8 you can concentrate on your problem, the disk supports you all the way.

OMNIDISK 65/8 is available in an attractive walnut wood cabinet, or unpackaged for OEM applications in dual and single drive configurations. The HDE disk controller is a state-of-the-art 4½" by 6½" card electronically compatible with the 44-pin KIM-4 bus structure. The controller and disk-driver are designed to operate with the popular Shugart 801-R and compatible devices.

The OEM single drive is $1195, the dual, $1895 and the dual in the walnut cabinet, $2200. Price is another reason to step up to the proven quality of an HDE system.

HDE PRODUCTS – BUILT TO BE USED WITH CONFIDENCE
AVAILABLE DIRECT OR FROM THESE FINE DEALERS:

JOHNSON COMPUTER Box 523 Medina, Ohio 44256 216-725-4560	ARESCO P.O. Box 43 Audubon, Pa. 19407 215-631-9052	PLAINSMAN MICROSYSTEMS Box 1712 Auburn, Ala. 36830 800-633-8724	LONE STAR ELECTRONICS Box 488 Manchaca, Texas 78652 612-282-3570	PERRY PERIPHERALS P.O. Box 924 Miller Place, N.Y. 11764 516-744-6462

MICRO™

MAY 1980
Issue Number 24

Table of Contents

Staff

Editor/Publisher
Robert M. Tripp

Associate Editor
Mary Ann Curtis

Circulation Manager
Carol A. Stark

Art/Advertising Coordinator
Terry Spillane

Comptroller
Donna M. Tripp

Production Assistant
L. Catherine Bland

MICRO™ is published monthly by MICRO INK, Inc., Chelmsford, MA 01824. Tel. 617/256-5515.
Second Class postage paid at Chelmsford, Ma 01824.
Publication Number: COTR 395770.
Circulation: Paid subscriptions: U.S.: 3800, Foreign: 350; Dealers: U.S.: 4500, Foreign: 1900.
Subscription rates: U.S.: $15 per year. Foreign, surface mail: $18 per year.
For air mail rate, change of address, back issue or subscription information write to: MICRO, P.O. Box 6502, Chelmsford, MA 01824.

Advertiser's Index

More About 16 Bits

Last month's article by Randall Hyde, "The SY6516 Pseudo-16 Bit Processor" [MICRO 23:36] is an interesting combination of a year-old rumor and author fantasy.

More than a year ago, Synertek considered developing a part to be called the SY6516. Several different SY6516's were proposed which significantly differed from each other. It was decided not to develop any of the proposed 6516's. Therefore, it is not "almost ready to ship".

Of course, Synertek does have several other development programs running in both the peripheral and CPU areas. The SY6545 improved CRT controller and the SY6591 floppy disk controller, each with the 6502 bus, will be available later this year.

This letter may not be the action referred to in last month's editorial. A simple phone call to Synertek from Micro to discuss the SY6516 would have forestalled publication of this article which has created so much confusion and annoyance among 6502 enthusiasts.

Michael Smolin
Strategic Marketing Manager,
Synertek, Inc.

Editor's Note: The intent of the above mentioned article, and my editorial 'The Value of 16 Bits"[MICRO 23:9], was to spark reader interest in improved versions of our 6502 microprocessor. My intent was **not** *to cause anyone "confusion and annoyance" and to the degree that this has occurred, I apologize. I did, by the way, attempt to get some information from my local distributor, but without any success. Now that I have a contact at Synertek who is aware of this type of project, I will certainly check out any Synertek related material in the future.*

It is heartening to hear that the article and editorial did generate interest in an improved 6502. Several readers have written with their suggestions. If you have any ideas, please send them in. We will present the best ideas in an article in a few months.

Robert M. Tripp

ROADRUNNER - A Math Drill for Second - Graders

Remember 'rationalizing' the purchase of your microcomputer on the grounds that 'it would be good for the kids'? Well, he are some suggestions on using your Apple for Computer Assisted Instruction.

Peter A. Cook
1443 N. 24th Street
Mesa, AZ 85203

Computer Assisted Instruction (CAI) will spread rapidly in the field of education as the use of small computers becomes more widespread. Children in their earliest school years can enjoy using a keyboard to learn the traditional skills, if programs are prepared with imagination and care. The greatest challenge in designing these programs is the fact that young children too easily become bored and lose interest.

Making a Game of It

My second-grader was having trouble beating the clock during timed math drills at school, so I devised a program which presents him wih 50 practice problems on an Apple II computer. Random additon and subtraction problems such as the following are displayed on the screen.

```
 11      9      3     12
- 2    - 5    + 4    + 0
---    ---    ---    ---
```

The largest number used is 12, for either operand or result. This can be changed as the needs of the child dictate.

To increase the interest level and heighten the student's motivation, several elements have been added which turn the drill into a game, called "Roadrunner". These added features provide the child with a visible goal, immediate reinforcement, and a certain amount of pressure.

Goal orientation is provided by six animal names which illuminate in sequence as each group of ten problems is completed. The lowest level is the snail, for the beginning problems with a long time interval, progressing to the roadrunner for the last problem with the shortest interval.

The player is informed at the end of each problem whether or not his answer is correct. If so, the next problem is presented. If not, the problem sequence is stopped, and the player is given as many chances as needed to determine the right answer. The game will not end until the correct answer has been given, so the student isn't left hanging as to what the proper answer should be.

Pressure is applied in the form of a time countdown for each problem. The time interval starts at 20 seconds for the first ten problems, then decreases by two seconds after each set of ten problems is completed.

The game stops running if an answer is not given before the time elapses, or when an answer is incorrect. Once the correct answer is supplied, or when 50 problems are completed, the animal name corresponding to the highest level attained begins flashing. This is accompanied by several beeps from the speaker, to officially announce the end of the game.

The Program

Coding the Applesoft program was fairly straightforward with the exception of the time countdown. A random number is first selected between 0 and 24. If it is greater than 12, then 12 is subtracted from it, and the problem will be a subtraction problem. A second random number is then selected between 0 and 12. In the case of an addition problem, the sum of the two numbers is checked to be sure it is no higher than 12. For a subtraction problem, the second number is checked to be sure it is less than the first number. The problem is then presented on the screen, and a reply is requested.

The usual method for inputting a reply to a question in Applesoft, using the INPUT or GET statement, will not work in this case because they cause the program to stop and wait indefinitely until a reply is keyed in. Memory location -16384 contains the ASCII value of the last key depressed, plus 128, provided that the keyboard strobe (-16368) has been reset to zero.(*Applesoft BASIC Programming Reference Manual,*Apple Computer Inc., 1978) By PEEKing this location repeatedly during a time delay loop, as in program lines 60 through 66, the computer will know whether or not an answer has been keyed in prior to the time interval elapsing. Since a reply could have one or two digits, the return key is used to signify the end of data input and to stop the timer from counting down.

Once the reply is received, it is tested for correctness, and the appropriate message is printed. When ten problems have been completed, the animal name and time delay are changed, and the above process is repeated.

Several changes can be made to the program to make the game more or less challenging, depending on the age and ability of the child. Numbers larger or smaller than 12 can be programmed in by changing H in line 2. The time interval, 20 seconds, is defined by T in line 4, and the decrease after each ten problems, 2 seconds, is subtracted from U in line 84.

One caution needs to be mentioned which could cause some frustration if not explained before using the program. The process of reading the keyboard using PEEK (-16384) depends on the particular time during the cycle that a key is depressed, and for how long. It seems to work about 95 per cent of the time. Watch the screen to be sure each digit is printed before pressing the next key during the time countdown, or an incorrect answer will be accepted. Sometimes the desired key must be pressed one or two additonal times. The time interval is purposely long enough to allow for this. Also, if the wrong key is pressed, you cannot back up and correct it.

That completes the description. Type in the program, have your second-grader RUN it, and see if he can answer the problems fast enough to become a ROADRUNNER.

```
]LIST0
0  REM  "ROADRUNNER", PETE COOK,
     OCT 79
2 H = 12: REM HIGHEST NUMBER
4 T = 20: REM LONGEST TIME, SECON
     DS
6  DIM W$(6): FOR W = 1 TO 6: READ
     W$(W): NEXT
8  DATA SNAIL, TURTLE, CHIPMUNK, RAB
     BIT, COYOTE, ROADRUNNER
10  REM  PRINT HEADINGS:
12  HOME : HTAB 11: PRINT "R O A
     D R U N N E R": PRINT : HTAB
     9: PRINT "50 ADD/SUBTRACT PR
     OBLEMS"
14  POKE 34,5: REM TOP MARGIN
16  POKE 33,22: REM WIDTH, ALTERN
     ATES FROM LEFT HALF TO RIGHT
      HALF
18 P = 1:X = 0:Y = 1:Z = 0:U = T:
      REM RESETS VARIABLES FOR NE
     W GAME
20  REM  PRINT ANIMALS:
21  VTAB 10: FOR W = 1 TO 6: IF W
      = Y THEN  INVERSE : IF Z =
     1 THEN  FLASH
22  PRINT W$(W): NORMAL : PRINT :
      NEXT
23  IF Z = 1 THEN  FOR C = 1 TO 5
     : FOR D = 1 TO 10: NEXT D: PRINT
      CHR$ (7);: NEXT C: GOTO 90:
      REM  5 BEEPS
24  REM  BLANK LAST PROBLEM, PRIN
     T NEW NUMBER AND TIME REMAIN
     ING:
25  VTAB 6: CALL  - 868: REM  BLA
     NK LINE
26  POKE 32,17: REM  LEFT MARGIN
27  FOR C = 1 TO 20: FOR D = 1 TO
     60: NEXT D: CALL  - 912: NEXT
      C: REM  SCROLL UP ONE LINE
28  POKE 32,0: VTAB 6: PRINT "NUM
     BER: ";P: POKE 32,17: FOR D =
      1 TO 1000: NEXT : REM  DELAY

29  VTAB 6: HTAB 3: PRINT "SECOND
     S: ";U
30  REM  SELECT NUMBERS:
31 S$ = "+ ":L = H:A = 0
32 M =  INT ( RND (1) * 100): IF
     M > 2 * H THEN 32: REM TOP N
     UMBER
34 N =  INT ( RND (1) * 100): IF
     N > H THEN 34: REM BOTTOM NU
     MBER
36  IF M > H THEN 42: REM SUBTRAC
     T
38 S = M + N: IF S > H THEN 34
40  GOTO 46
42 L = M - H: IF N > L THEN 34: REM
     TOP NUMBER MUST BE LARGER
44 S = L - N:S$ = "- ":M = L
45  REM  PRINT PROBLEM:
46  FOR D = 1 TO 1000: NEXT : REM
     DELAY
50  VTAB 9: HTAB 8: IF M < 10 THEN
      HTAB 9: REM RIGHT JUSTIFY
52  PRINT M: HTAB 6: PRINT S$;
54  IF N < 10 THEN  HTAB 9
55  PRINT N: HTAB 6: PRINT "----"
56  REM  INPUT ANSWER, TEST IT:
57 V = U * 18: REM  MULTIPLIER FO
     R ACTUAL SECONDS
58  PRINT : PRINT : HTAB 3: PRINT
      "ANSWER? ";: REM  NO CURSOR
60 I =  PEEK ( - 16384): POKE  -
     16368,0: REM  READ KEYBOARD,
      RESET KEYBOARD STROBE
62  IF I = 141 THEN  VTAB 15: GOTO
      74: REM  RETURN KEY INDICATE
     S ALL DIGITS RECEIVED
64  IF I > 127 THEN A = A * 10 +
      VAL ( CHR$ (I - 128)): VTAB
      14: HTAB 11: PRINT A: REM
     WEIGHT KEYSTROKES FOR UNITS,
      TENS
66 V = V - 1: IF V > 0 THEN  VTAB
      6: HTAB 13: PRINT " ";: HTAB
      12: PRINT  INT (V / 18): GOTO
      60: REM  BLANK SECOND DIGIT
      OF SECONDS REMAINING
70 Z = 1: VTAB 16: HTAB 5: PRINT
      CHR$ (7);"TOO LATE!"
72  PRINT : PRINT : HTAB 3: INPUT
      "ANSWER? ";A
74  IF A = S THEN 78: REM CORRECT
      ANSWER
76 Z = 1: PRINT : HTAB 5: PRINT "
     WRONG, TRY AGAIN!": GOTO 72
78 P = P + 1: IF P = 51 THEN Z =
     1:Y = 6
80  PRINT : HTAB 5: PRINT "RIGHT!
     "
82  POKE 32,0: IF Z = 1 THEN 21: REM
      Z STOPS GAME
84 X = X + 1: IF X > 9 THEN X = 0
     :Y = Y + 1:U = U - 2: REM CH
     ANGE ANIMAL AND TIME INTERVA
     L AFTER 10 PROBLEMS
86  GOTO 21
90  VTAB 24: INPUT "ANOTHER GAME
     (Y/N)? ";I$: IF I$ = "Y" THEN
      POKE 33,40: HOME : GOTO 16
92  POKE 34,0: POKE 33,40: HOME :
      END
```

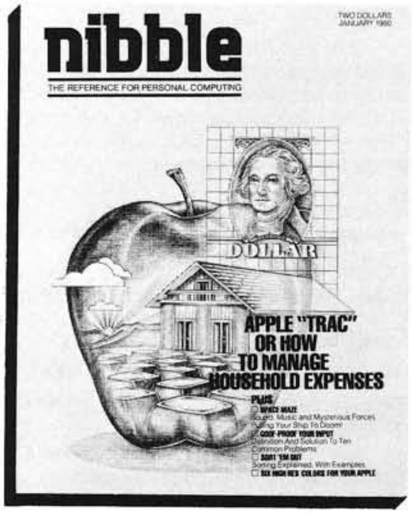
```
                R O A D R U N N E R

             50 ADD/SUBTRACT PROBLEMS
NUMBER: 32                       SECONDS: 9

SNAIL                               12
                                  - 4
TURTLE                            ----

CHIPMONK                         ANSWER?  7

RABBIT                           WRONG, TRY AGAIN.

COYOTE                           ANSWER?  8

ROADRUNNER                       RIGHT!
```

Figure 1: **End of game, Problem 32 was answered incorrectly. "RABBIT" flashes to show highest level achieved.**

Roadrunner Variables List

A	Answer
C	Counter
	Delay
H	Highest Number
I	Input
I$	Input
L	Top number for subtraction
M	Top number
N	Bottom number
P	Problem number
S	Sum or difference
S$	Sign
T	Maximum time interval
U	Decreased time interval
V	Actual seconds remaining
W$(6)	Winning animal
W	Subscript
X	Counts ten problems
Y	Level attained
Z	Ends game

INTRODUCING . . . NIBBLE THE REFERENCE FOR APPLE COMPUTING



NIBBLE IS:
A SOFTWARE GUIDE for high quality Applications Programs for your Home and Business.

NIBBLE IS:
A REFERENCE GUIDE to new Programming Methods.

NIBBLE IS:
A BUYERS GUIDE for making purchase decisions on new products.

NIBBLE IS:
A CONSTRUCTION PROJECT COOKBOOK for adding function and value to the system you already own.

NIBBLE IS:
A COMMUNICATIONS CLEARING HOUSE for users, vendors, and associations.

Each issue of NIBBLE features at least one significant new application program of commercial quality. The programs in NIBBLE are surrounded with articles which show how to USE the programming methods in your OWN programs.

Examples of upcoming articles:
☐ Modeling and Forecasting Your Business ☐ Build a Two-Tape Controller for $12 ☐ Arcade Shooting Gallery — Save Your Quarters! ☐ Data Base Management System I, II, III

And many many more! NIBBLE will literally "Nibble Away" at the mysteries of your system to help you USE IT MORE. In 1980, the principal featured system is the Apple II.

Try a NIBBLE

nibble
BOX 325 Lincoln, Mass. 01773

I'll try NIBBLE!
Enclosed is my $15 for 8 issues.
☐ **check** ☐ **money order**

Name ______________________

Address ______________________

City ______________________

State ______________ Zip ________

Plotting with Special Character Graphics

A primer on, and program for, generating plot mode type graphics with special characters. Applicable to the PET, Challenger, and other micros.

Dale DePriest
611 Galen
San Jose, CA 95106

Microcomputers that support graphics are basically one of two types. One type supports their graphics with a special set of graphics characters that are printed or poked on the screen thereby drawing the picture you were trying to portray. Examples of this type of computer are the Challenger, Sorcerer, and Pet. The second type of graphics support divides the screen into small squares or rectangles which are turned on or off by specifying their address in a matrix system. These points are said to be "plotted" on the screen in the same fashion that you would plot on a piece of graph paper. Examples of this type of machine include the Compucolor, Intecolor and TRS-80.

If you are an owner of the first type of computer and have ever been envious of the people who own the second type or would like to use graphics programs written for the second type then this is the article for you! A program will be presented that allows you to duplicate the plot mode graphics and allows you to create your own expanded graphics mode.

First let us take a minute and ensure that everyone understands plot mode graphics. I will use the TRS-80 for an example. They have divided the screen into 128 points across (horizontally) and 48 points down (vertically). To identify any single point you must specify its location with two numbers; the first will identify how many points over from the left edge of the screen your point is and the second number will identify how may points down from the top edge. To turn a point on you would use the instruction SET (X,Y) where X is the distance across (0-127) and Y is the distance down (0-47). To draw a line across the screen you would have to write a program that would specify a value for Y and included a loop that would increment or decrement X until the line was drawn.

For example, consider the following program:

```
10 Y =6
20 FOR X =0 TO 63
30 SET (X,Y)
40 NEXT X
```

This program would plot a line near the top of the screen beginning at the left edge and extending across to the center of the screen.

The TRS-80 has three instructions that support their graphic; one to turn a point on, one to turn a point off, and one to examine a point to see whether it is on or off.

Some people call this graphics mode "high resolution" or simply HIRES graphics since this is the smallest increment of data that the programmer has control of. HIRES graphics capability of the TRS-80 is 128 × 48. A low resolution graphics capability exists also and is inherent in all computers whether they have graphics capability or not. This LOWRES graphics mode is simply to use a character form the keyboard (such as X) and draw pictures with it. Most people have used this technique when playing around with a typewriter. In this mode the TRS-80 would have a resolution of 64 × 16. (The number of characters per line by the number of lines.) To use this mode of graphics you would simply use print statements.

Intecolor and Compucolor use a method very much like that used by Radio Shack. They use a sequence of PLOT instructions to accomplish their graphics capability. In addition, they have some subplot modes which allow you to draw lines and bars without the necessity of writing loops like the one I showed you for the TRS-80. This capability gives the programmer a very powerful tool for fancy graphing such as vector mode, point to point lines, bar graphs, etc.

Most home computers display the screen from a memory somewhere in the 64K that a programmer has access to. The BASIC interpreter program simply places the information in the proper address of this memory and the hardware of the computer constantly displays this memory on the TV tube that you look at. For this reason, if you know where this refresh RAM (memory) is located, then you can simply use the BASIC PEEK instruction to find out what is there and the POKE instruction to put anything you choose on the screen, whether it be keyboard characters or graphics characters. Normally each character on the screen occupies one byte of data in the refresh RAM. This means that there are 256 possible characters that can be displayed. The way that each manufacturer uses these 256 characters is one of the major differences between home computers.

The TRS-80 is no exception. They reserve 64 of these characters for the keyboard, 64 of these characters are used for the double width characters and the other 128 are the graphics characters. Mathematically it can be shown that the 128 characters are enough to contain all possible combinations of 6 bits ($2^6 = 128$); therefore what Radio Shack did was divide a character into 6 pieces arranged in a two by three group.

Compucolor and Intecolor devote two bytes of data to each character on the screen since they must also include color information. This also gives then the ability to devote 256 characters to graphics, 8 bits in a two by four group, and thus they have a higher resolution capability than does Radio Shack.

Now that you understand plot mode character graphics let's see how we can duplicate this graphics mode with the special character graphics. The demonstration programs and the special graphics subroutine we'll be looking at will run unmodified on a PET computer but I'll try to include enough notes so that it should be easy to modify these routines for any computer that uses Microsoft BASIC as long as the graphics symbols are available.

The symbols that we are going to use are shown in Figure 1. Since we are going to divide our characters into four bits we will need 2^4 or 16 characters. PET has thoughtfully provided all of these characters on the keyboard, although some will have to be used with the RVS key. Challenger doesn't provide these characters from the keyboard but you should be able to find them listed in the graphics program book. The first one is simply a space.

Since we plan to poke these characters on the screen, we'll need to know the decimal equilalents of all of these characters. The following subroutine will build an array of all the characters shown in figure 1 by using the decimal equivalents of these characters in the data statements.

```
32000  DIM X2 (15) :FOR Y1=0 TO
32010  READ X2(Y1):NEXT:RETURN
32020  DATA 32,126,124,226,123,97,
       225,236,108,127,225,251,
       98,252,254,160
```

If you don't have a PET, change the DATA statement per your machine documentation. Be sure to enter them in the order shown in figure 1.

My subroutines all use the variables X, Y, plus these variables with numbers. This is done to minimize the impact on variables you may be using in your program. Variable definitions are given in the table below.

X The horizontal coordinate of the point

Y The vertical coordinate ot the point

X1 The decimal address of the character that the point is in.

X2 The original data at the address X1

X3 A flag to tell the subroutine what kind of plotting is desired (see text)

Y1 The pointer into the array containing our plot character

Y2 A flag indicating which one of the four points in the character that X,Y points to

X2(Y1) The array of possible plot characters

Now let's look at the program in detail. Line 50 gets rid of any ambiguity about the value of X; first by making sure that it is an integer and then by making sure that the point is on the screen. The number 79 is one less than twice the number of characters you have across your screen. A good value to use on the Challenger 1P is 47; Challenger 2 would use 127.

Line 52 does the same thing for Y. The number 49 represents one less than twice the number of lines.

Line 54 generates the address of the character we are interested in and peeks the current value. It then searches to try and match this value with the array that we set up earlier. The number 40 is the number of characters on the Pet line. For a Challenger 1P this must be 32. 32768 is the decimal address of the starting location of the Pet memory map for screen refresh. Your system documentation should tell you where yours is located. For the Challenger 1P this starting location depends on the TV overscan but 53349 should be a good place to start.

Line 56. If the search is unsuccessful and X3 is a zero, we'll assume Y1 = 0, thereby overwriting any data that is already on the screen. Otherwise, we will abort the plot and preserve the data on the screen.

Lines 58 and 60 find the proper quadrant of the character.

Let's skip lines 63, 64 and 66 for now.

Line 68 does the actual plotting by oring the old data pointer and the quadrant pointer. If you've gotten this far and you suddenly find that your machine won't, or that there are two numbers together, then please drop me a line and I'll give you a program that does the same thing with logical IF tests. Be sure to tell me what kind of machine you have.

The program we have just discussed will simulate the TRS-80 SET instruction except that we can also have control over what happens should our plot program encounter a normal print character. To demonstrate this, consider the following example:

```
5 GOSUB 32000
10 Y = 6
20 FOR X = 0 TO 39
30 GOSUB 50
40 NEXT X : END
```

If you have entered the two subroutines prior to this, then this program will draw a line half way across the screen near the top, just like the Radio Shack program did. Now, remove these lines (5 through 40) and enter the following program.

```
10 PRINT CHR$(147):REM CLRS
SCREEN
20 GOSUB 32000
30 GOTO 100
100 FOR X = 0 TO 79
110 Y = 24 + 15 * SIN(X/5)
120 GOSUB 50
130 NEXT X
140 END
```

When you run this you should have a nice sine wave appear on the screen.

Radio Shack has a RESET instruction also that allows them to turn off a bit. Some of the time this is used to simulate a ball or bullet for animation purposes. Since this program is in Basic which is inherently slow for this sort of thing, I have provided for a special feature to allow simulating this kind of action. Please add this one line to your earlier program.

```
115 POKE X1, X2
```

This turns off the bit that was just turned on by poking back the original value. Once you have tried this program, please be sure to remove line 115. This is not the only use of the RESET instruction, however. We should be able to simulate this instruction also. Now we can discuss the rest of the main subroutine.

Line 62 holds the key to the power of this subroutine. By setting X3 to a particular value we can use this subroutine to do many plotting functions. We have already discussed the values of X3 = 0 and X3 = 1.

Line 64 is required since the ON instruction cannot work with X3 = 0.

Line 70 is the place we will jump to if X3 = 2. Add the following lines to your program and try it again.

```
140 IF X3 = 2 THEN END
150 X3 = 2
160 GOTO 100
```

As you can see, X3 = 2 simulates the RESET instruction very well. You should now save your program tape.

This same routine provides more advanced functions as well which are similar to those supplied by Compucolor. For example, if X3 = 3, then a test is made on the bit at the X,Y coordinate. If it is off, we'll turn it on but if it's already on, we'll turn it off. This decision is made with line 66.

Line 72 is the line we will get to if X3 = 4. This will cause us to enter the X-bar graph mode. Consider this program:

```
5 GOSUB 32000
10 PRINT CHR$(147)
```

```
15 Y = 6
20 X3 = 4
25 X = 39
30 GOSUB 50
35 END
```

It draws the same horizontal line we saw earlier but we didn't have to write a loop. It was also a little faster.

Line 74 is where you will end up when X3 = 5. This enters the Y-bar graph mode and vertical lines will be drawn.

Of course several more variations could be derived depending on your application. For example, if X3 = 6 pointed to line 76, then
76 GOSUB 68:X = X - 1:Y = Y + 1:GOTO 50
would draw a diagonal line. Or alternately,an adaptation of line 66 could provide a status that would indicate whether one bit was on or off.

Final Considerations

This routine is intended to reside near the front of your using program since subroutines at the beginning of programs execute faster than those near the end. The initialization subroutine only executes once, so placing it at the end simply gets it out of the way.

Notes on the Challenger 1P

In addition to the changes described in the text, you will have to make the following changes to run this program on the Challenger 1P. Line 54 will have to be broken into two lines since it is more than 72 characters long. The same is true of line 32020. Although Challenger has a very extensive set of graphics characters, they really blew it when using the sixteen characters described in this article. Four are missing. This prevents not only a clean implementation of this program but alos prevents another use for these characters such as that of creating large lettering. The best compromise may be the following statements:

```
32020 DATA 32, 168, 166, 155, 167, 156, 170, 175

32030 DATA 165, 169, 157, 177, 154, 178, 176, 161
```

Challenger should consider changing their ROM; perhaps changing 171 through 174.

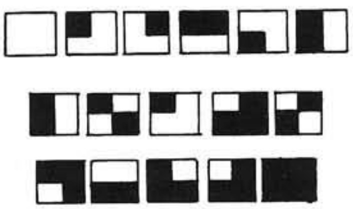


Figure 1

Main Subroutine

```
50   X = INT(X): IF X   79 OR X   0 THEN RETURN
52   Y = INT(Y): IF Y   49 OR Y   0 THEN RETURN
54   X1 = INT(X/2) + 40*INT(Y/2) + 32768: X2 = PEEK(X1): FOR Y1 = 0 TO
     15: IF X2 = X2(Y1) THEN 58
56   NEXT: Y1 = 0: IF X3 THEN RETURN
58   Y2 = 1: IF X/2 - INT(X/2) THEN Y2 = 2*Y2
60   IF Y/2 - INT(Y/2) THEN Y2 = Y2*4
62   ON X3 GOTO 68,70,66,72,74
64   GOTO 68
66   IF X2(Y1 OR Y2) = X2(Y1) THEN 70
68   POKE X1, X2(Y1 OR Y2) : RETURN
70   POKE X1,X2(Y1 AND 15 - Y2) : RETURN
72   GOSUB 68 : X = X - 1 : GOTO 50
74   GOSUB 68 : Y = Y + 1 : GOTO 52
```

SYM - 1 BASIC "GET" Command

Everything you need to know to implement the 'GET' function in SYM - 1 BASIC. The use of the GET function is discussed and several examples are provided.

George Wells
1620 Victoria Place
La Verne, CA 91750

The SYM-1 BASIC Interpreter provides for an unused "GET" token which always produces a Function Call error (FC) whenever it is encountered in a program. GET is an alternate form of INPUT except that it only inputs one character for each call and that one character can be any keyboard character including control characters and lower case letters. The first section of this article describes a simple procedure to implement this very useful command. The second section explains in detail how it works and the the third section offers some examples of BASIC subroutines utilizing the GET command.

Section One
Implementing the GET Command

Step 1: Deposit and Verify the code in the OBJECT LISTING. If it consistently will not Verify, read Section 2 before proceding.

Step 2: Enter the following monitor command:

```
.SD A600,A664
```

Step 3: Jump to BASIC:

```
.J O
```

Step 4: Enter and RUN a BASIC program such as:

```
100 PRINT "HIT ANY KEY:"
110 GET A$
120 PRINT ASC(A$)
130 GOTO 100
```

The GET command is always used to input a character string which will normally have a length of one. (A double-quote (") or a NULL results in a length of zero which causes an FC error to occur. See Section 2.) Of course, the string variable can be either simple or an element of a matrix, but only one variable is allowed for each GET and it cannot be used in a Direct Command. When GET is encountered in a running program there is no prompt "?" and prompt strings are not allowed. This is intentional to allow for several characters in a row to be typed in, in response to several GET's or for a loop which examines the characters for errors as they are typed. It is therefore normal to precede GET with a PRINT statement to serve as a prompt.

Section Two
Detailed Explanation of GET Implementation

The assembly language program to implement GET is stored in two sections of RAM that are unused by both the Monitor and BASIC. The first of these is the first 32 bytes of System RAM which are normally allocated as the Scope Buffer but are not changed in any way as long as none of the hex keypad buttons are pushed (except, of course, RST and DEBUG ON and OFF). These 32 bytes are located at \$A600-\$A61F. The second section of RAM is the 16 bytes located on page zero at \$E8-\$F7. The code can be entered into your SYM-1 and verified using the object code listing or if you have Synertek's RAE-1, you can enter the source code as it appears in the assembly listing. After it is assembled the block of code belonging on page zero must be moved there from page \$0F with the monitor command:

```
B E8,FE8-FF7
```

The code can not be assembled directly on page zero since RAE-1 also uses that block of memory. If you happen to have EPROM in your system you can also relocate the code there (delete line 300 JMP GET.COMD.3). In order to activate GET, the System Output Vector (\$A664,5) must be changed from its present value, assumed to be the Terminal Output routine (TOUT = \$8AA0), to the GET command processor (GET.COMD = \$A600). This vector can be changed at the monitor level with the simple command:

```
.SD A600,A664
```

or it can be done in BASIC with:

```
POKE 42596,0: POKE 42597,166
```

which can be either a Direct Command or part of a program. If you decide to relocate the code to some other address than \$A600 then be sure to use the correct address when changing the System Output Vector. Please be aware of the fact that the System RAM is write protected after a warm start to BASIC (G O) until after a LOAD or SAVE command is attempted (if you have the new Monitor ROM) or until a call to ACCESS is made some other way, for example, with QQ = USR(&"8B86",0) or unless the jumper at 45-MM is removed. Incidentally, since BASIC passes the program size and file ID information to the Tape routines through the System RAM, the first LOAD or SAVE after a warm start won't work.

To understand how the GET command is processed look at the assembly language listing. Each time BASIC attempts to print any character, this routine will be entered. If the character to be printed is a carriage return, which is the case when any error is encountered, then further testing is performed to see if it is a Function Call error and then if it was caused by a GET token. If any of the proper conditions are not met then a jump is made to the Terminal Output routine or to whatever special output routine you might have.

Assuming that all the conditions for the GET command are met, then twelve bytes are taken off the stack to account for the series of JSR's involved in printing the error message. Next, the BASIC Input Buffer is set up as it would be if a single character were entered in response to an INPUT command. However, the routines that normally bring characters into the Input Buffer are bypassed because they ignore all control characters (except BELL) and change lower case letters to upper case. Instead, the Input Buffer is loaded directly by the GET command processor so that all characters will be allowed. In addition, a double-quote is automatically inserted before the typed character so that commas, colons and spaces will also be properly interpreted. After the typed character a zero is inserted which is the End-of-Line token. There remains an ambiguity over two characters which can be typed in, namely, NULL and double-quote ("), both of which will be interpreted as a string of zero length. The NULL looks like the End-of-Line token and the double-quote looks like the End-of-String character. If you are not concerned with this ambiguity in your application, skip the remainder of this section.

There are two ways to avoid the ambiguity between double-quote and NULL. First you can change the assembly language instruction on line 350 from AND #$7F to ORA #$80 and then subtract 128 from each character after the GET statement. Example: Change BASIC program line 630 to:

```
630 CHAR$ = CHR$(ASC(CHAR$)-128)
```

The second way to handle this is by inserting three instructions between lines 350 and 360 of the assembly program as follows:

```
CMP #$22
BNE +2
ORA #$80
```

But this will require relocating the code to accomodate the additional bytes of program. (Due to a minor error in RAE-1, the branch must be entered as BNE = +3.) In this case, only a double-quote has its most significant bit set. It is not necessary to subtract 128 as long as you treat the ASCII code for double-quote as 162 instead of 34. Also, line 630 of the BASIC program should be deleted.

Section Three
Examples of Using GET

The remainder of this article will describe several BASIC subroutines which can be used to simulate the INPUT function for integer, numeric and string variables. Also described is a means to disable the BREAK key to make it possible to write programs that are incapable of being clobbered by the operator. This is an especially important feature when running programs for the novice. If you've had the frustrating experience of trying to leave your computer in the hands of the kids to play games only to have them forget to press RETURN after every input and not press RETURN without some input, then you know what a boon this can be. It can save you from having to reload a program because the kids have unknowingly deleted lines of program by typing in numbers while in Command Level.

OBJECT LISTING

```
.V E8-F7
00E8 20 58 8A 29 7F 85 1F A2,F0
00F0 1D A0 00 84 20 4C EA C9,50
 0650
.V A600-A61F
A600 C9 0D D0 08 E0 08 D0 04,6A
A608 C0 36 F0 03 4C A0 8A BA,83
A610 8A 69 0B AA 9A A9 2C 85,1F
A618 1D A9 22 85 1E 4C E8 00,DE
 0DDE
.
```

```
LIST

 10 PROMPT$ = "INPUT A STRING: "
 11 GOSUB 600
 12 PRINT PHRASE$
 13 GOTO 10
 20 PROMPT$ = "INPUT A NUMBER: "
 21 GOSUB 500
 22 PRINT NUMBER
 23 GOTO 20
 30 PROMPT$ = "INPUT AN INTEGER: "
 31 GOSUB 400
 32 PRINT NUMBER%
 33 GOTO 30
 35 :
 95 REM  *** SUBROUTINE TO ACTIVATE "GET" ROUTINE ***
 100 QQ = USR(&"8B86",0): REM ALLOW ACCESS TO SYSTEM RAM
 110 POKE 42596,0: POKE 42597,166: REM CHANGE OUTPUT VECTOR TO "GET"
 120 RETURN
 185 REM  *** SUBROUTINE TO DISABLE "BREAK" KEY ***
 195 REM SIMULATE MONITOR COMMAND: .SD 862D,A667
 200 QQ = USR(&"8B86",0): REM ALLOW ACCESS TO SYSTEM RAM
 210 POKE 42570,103: POKE 42571,166: REM STORE INSVEC+1 IN P3
 220 POKE 42572,45: POKE 42573,134: REM STORE $862D (CLC-RTS) IN P2
 230 QQ = USR(&"861D",0): REM EXECUTE STORE DOUBLE BYTE COMMAND
 240 RETURN
 285 REM  *** SUBROUTINE TO ENABLE "BREAK" KEY ***
 295 REM SIMULATE MONITOR COMMAND: .SD 8B3C,A667
 300 QQ = USR(&"8B86",0): REM ALLOW ACCESS TO SYSTEM RAM
 310 POKE 42570,103: POKE 42571,166: REM STORE INSVEC+1 IN P3
 320 POKE 42572,60: POKE 42573,139: REM STORE $8B3C (TSTAT) IN P2
 330 QQ = USR(&"861D",0): REM EXECUTE STORE DOUBLE BYTE COMMAND
 340 RETURN
 395 REM  *** SUBROUTINE TO INPUT AN INTEGER ***
 400 GOSUB 500: REM INPUT A NUMBER
 410 IF ABS(NUMBER) > 32767 THEN 400: REM REPEAT IF OUT OF RANGE
 420 NUMBER% = INT(ABS(NUMBER))*SGN(NUMBER): REM DROP FRACTIONAL PART
 430 RETURN
 495 REM  *** SUBROUTINE TO INPUT A NUMBER ***
 500 GOSUB 600: REM INPUT A STRING
 510 NUMBER = VAL(PHRASE$): REM CONVERT STRING TO NUMBER
 520 RETURN
 595 REM  *** SUBROUTINE TO INPUT A STRING ***
 600 PRINT: PRINT PROMPT$;: REM PRINT PROMPT ON NEW LINE
 610 PHRASE$ = "": REM DELETE PHRASE
 620 GET CHAR$
 630 IF LEN(CHAR$) = 0 THEN CHAR$ = CHR$(34): REM CHANGE NULL STRING TO "
 640 IF ASC(CHAR$) <> 8 THEN 680: REM BRANCH IF NOT BACK-SPACE
 650 IF LEN(PHRASE$) = 0 THEN PRINT RIGHT$(PROMPT$,1);: GOTO 620
 660 PHRASE$ = LEFT$(PHRASE$,LEN(PHRASE$)-1): REM DELETE LAST CHARACTER
 670 PRINT " "; CHR$(8);: GOTO 620
 680 IF ASC(CHAR$) = 10 THEN 600: REM START OVER IF LINE-FEED
 690 IF ASC(CHAR$) = 13 THEN PRINT: RETURN: REM DONE IF CARRIAGE RETURN
 700 PHRASE$ = PHRASE$ + CHAR$
 710 GOTO 620
 795 REM  *** SUBROUTINE TO DE-ACTIVATE "GET" ROUTINE ***
 800 QQ = USR(&"8B86",0): REM ALLOW ACCESS TO SYSTEM RAM
 810 POKE 42596,160: POKE 42597,138: REM CHANGE OUTPUT VECTOR TO "TOUT"
 820 RETURN
OK
```

The BASIC program lisitng contains two parts. The first part (lines 10-35) contains sample drivers for the three types of INPUT's and the second part (lines 95-820) contains the actual subroutines. The first subroutine (GOSUB 100) changes the output vector to point to the assembly language program which of course must be loaded prior to entering BASIC. The last subroutine (GOSUB 800) can be used to switch the output vector back to its normal state. The second and third subroutines can be used to disable and enable the BREAK key. These routines use part of the Monitor Store Double Byte Command to change the Input Status Vector because it is impossible to do the same thing in pure BASIC since the status would be checked between the two POKE's and would result in the program going to an undesired place. The BREAK is disabled by simply pointing it to a routine that always returns a status clear.

The subroutine beginning at line 600 simulates the INPUT command for a character string. The first thing it does is print a prompt string which should be defined prior to calling the subroutine. The name of the prompt string is PROMPT$ (or PR$). Next, the string which will contain the typed characters is cleared. Its name is PHRASE$ (or PH$). Then a loop is entered which GETs the typed characters one at a time and examines them before it puts them into the PHRASE$ string to see if they are any of the following special characters:

1. NULL (same as double-quote) is changed to ''.

2. Back Space deletes previous character.

3. Line Feed deletes entire line.

4. Carriage Return ends the input.

No test is made to limit the number of characters to 255. Therefore, typing in 256 characters is a way to "BREAK" a program that has the BREAK key disabled since it will cause a Long String Error (LS).

The subroutine beginning at line 500 simulates the INPUT command for a number. It does this by calling the string input subroutine and using the BASIC VAL function to put the string into the variable called NUMBER (or NU). If the string does not convert correctly into a number, no error is generated, instead that portion of the string up to the error is used (or zero if it is completely wrong). However, if the magnitude of the number is too large for BASIC an Overflow Error(OV) results. This is another way to "BREAK" a program even with the BREAK key disabled.

The subroutine beginning at line 400 simulates the INPUT command for an integer. It does this by calling the number input subroutine and using the BASIC INT function to convert it to an integer called NUMBER• (or NU•). If the number is too large to be an integer, the prompt is repeated to avoid an error. Also, the fractional part of a negative number is dropped instead of rounding up to the next larger integer (absolute value).

Obviously, similar sorts of routines can be written to accomodate any particular requirements you might have. One word of caution: at the lower baud rates BASIC can't keep up with a fast typist. Using the BREAK disable subroutine will keep the program from aborting but might result in incorrect characters being read. However, if they are read incorrectly they will also be echoed incorrectly, so backspace over any errors and retype. At 4800 baud, BASIC can easily keep up with all but the fastest typist. At 110 baud it isn't hard to get incorrect reads, but even then it's not likely to be a problem with a novice operator. However, if you are running at 110 baud it is probably because you are running on a teletype in which case you will have to handle the character deletes with something other than a back-space.

```
>ASSEMBLY LISTING

                 0010 GET.TOKEN  .DE $9B         BASIC "GET" TOKEN
                 0020 FC.ERROR   .DE $08         BASIC "FC ERROR" TOKEN
                 0030 INPUT.COMD .DE $C9B9       BASIC INPUT COMMAND INTERPRETER
                 0040 INP.BUFFER .DE $1E         BASIC INPUT BUFFER
                 0050 TOUT       .DE $8AA0       MONITOR TERMINAL OUTPUT ROUTINE
                 0060 INTCHR     .DE $8A58       MONITOR INPUT TERMINAL CHARACTER
                 0070
                 0080            .DS
                 0090            .BA $A600
                 0100
                 0110 ;    *** PROGRAM TO IMPLEMENT SYM-1 BASIC "GET" COMMAND ***
                 0120
A600- C9 0D      0130 GET.COMD   CMP #$0D               TEST FOR CARRIAGE RETURN
A602- D0 08      0140            BNE GET.COMD.1         AND BRANCH IF NOT.
A604- E0 08      0150            CPX #FC.ERROR          TEST FOR FC ERROR AND
A606- D0 04      0160            BNE GET.COMD.1         BRANCH IF NOT.
A608- C0 36      0170            CPY #L,GET.TOKEN+GET.TOKEN   TEST FOR GET AND
A60A- F0 03      0180            BEQ GET.COMD.2               BRANCH IF SO.
A60C- 4C A0 8A   0190 GET.COMD.1 JMP TOUT               IF NOT, CONTINUE OUTPUT.
                 0200
A60F- BA         0210 GET.COMD.2 TSX                    TAKE 12 BYTES OFF STACK.
A610- 8A         0220            TXA                    ALREADY IN BINARY MODE AND
A611- 69 0B      0230            ADC #12-1              CARRY SET, SO ADD 11.
A613- AA         0240            TAX
A614- 9A         0250            TXS
A615- A9 2C      0260            LDA #',                STORE COMMA IN FRONT OF
A617- 85 1D      0270            STA *INP.BUFFER-1      BUFFER (NEEDED BY BASIC).
A619- A9 22      0280            LDA #'"                STORE QUOTE IN BUFFER TO
A61B- 85 1E      0290            STA *INP.BUFFER        ALLOW AUTO STRING INPUT.
A61D- 4C E8 00   0300            JMP GET.COMD.3         CONTINUE ON PAGE ZERO.
                 0310
                 0320            .BA $E8                STORE $E8 CODE AT $FE8,
                 0330            .MC $FE8               MOVE WITH:  B E8,FE8-FF7
00E8- 20 58 8A   0340 GET.COMD.3 JSR INTCHR             INPUT A CHARACTER.
00EB- 29 7F      0350            AND #$7F               CLEAR PARITY BIT.
00ED- 85 1F      0360            STA *INP.BUFFER+1      PUT IT IN BUFFER.
00EF- A2 1D      0370            LDX #INP.BUFFER-1      X NEEDED BY BASIC.
00F1- A0 00      0380            LDY #0                 Y=0 NEEDED BY BASIC.
00F3- 84 20      0390            STY *INP.BUFFER+2      END-OF-LINE TOKEN.
00F5- 4C EA C9   0400            JMP INPUT.COMD+49      CONT INTO BASIC.
                 0410            .EN

LABEL FILE:  [ / = EXTERNAL ]

/GET.TOKEN=009B        /FC.ERROR=0008        /INPUT.COMD=C9B9
/INP.BUFFER=001E       /TOUT=8AA0            /INTCHR=8A58
GET.COMD=A600          GET.COMD.1=A60C       GET.COMD.2=A60F
GET.COMD.3=00E8
//0000,00F8,0FF8
>
```

A Simple Temperature Measurement Program and Interface

Using a micro for temperature measurement démonstrates some of the problems and some of the solutions involved in interfacing to the real world.

Marvin L. DeJong
Dept. of Math & Physics
The School of the Ozarks
Point Lookout, MO 65726

Temperature measurements at least as precise as +1°C can be made with the circuit shown in Figure 1 and the program listed in Table 1. The 555 timer integrated circuit operates in conjunction with a FENWAL GB41P2 thermistor as a temperature-to-frequency converter. The pulses from the circuit in Figure 1 are counted with the T2 counter/timer on the 6522 Versatile Interface Adapter. A machine language subroutine measures the number of pulses in one second, while a BASIC program converts the frequency to temperature.

The relationship of the temperature of the thermistor to the frquency of the puslses at PB6 is non-linear. A temperature -Vs- frequency curve for our system is shown in Figure 2. You must make such a clibration curve for the system to work. A calibration curve is obtained by immersing the thermistor and a previously calibrated thermometer in some fluid and making measurements of the frequency as the temperature of the fluid is changed. We used water, heat, and ice cubes to produce our calibration curve. The frequency measurement program in Table 2 is used to measure the pulse frequency as a function of temperature. If you want to use this system as an air themometer, then the fluid should be air. You will have to wait for nature to provide the necessary temperature changes. Temperatures below and above those shown on our calibration curve (Figure 2) may be included, depending on your intended application. Provided components with low temperature coefficients are used in the 555 timer circuit, the precision of the temperature measurements made by the program will depend largely on the quality of the calibration data you obtain for your circuit. The thermistor may be located in some remote location and connected to the 555 timer circuit by a twisted wire pair.

The program listed in Table 1 requires the user to input 20 frequency -temperature points from the calibration curve. The program can be easily modified to input more or less data. With the calibration data in memory, it calls the machine language subroutine to measure the frequency of the pulses from the interface circuit in Figure 1. Using the measured frequency and the calibration data, it performs a quadratic interpolation calculation to conver the frequency measurement to a temperature. It also converts the Celsius temperature to a Fahrenheit temperature and outputs both. In the BASIC program, statements number 50, 60 and 70 serve to get the frequency using the machine language subroutine. We are using AIM 65 BASIC, and the techniques necessary to call the machine language subroutine may vary from machine to machine. In any case,

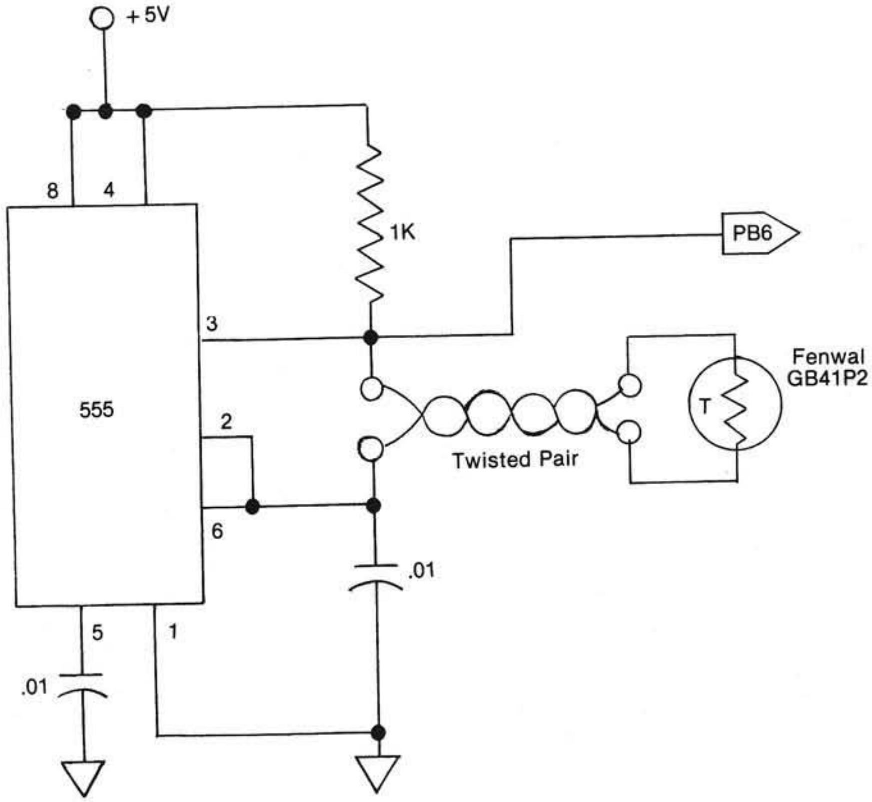


Figure 1: **Using the 555 Timer as a Temperature-to-Frequency Converter**

statement number 50 pokes the starting address of th machine language subroutine into a location where AIM 65 BASIC can find it. Statement number 60 actually produces the subroutine jump. The variable Y means nothing in statement 60. In statement 70 the BASIC program obtains the frequency from the two bytes in memory where the machine language subroutine stored it, namely 49 = $31 and 50 = $32 in zero page.

Because of the way the quadratic interpolation formula is applied to the incoming frequency data, it is a good idea to make the first calibration point entered into the BASIC program be F = 0, T = −100 or some other low temperature below the range where you wish to operate. The other calibration points, from your calibration curve, are entered **in order** from low frequency-low temperature to high frequency-high temperature. For example, our first few data points entered were:

0, −40
1000, −10
2550, 0.5
3000, 5.0

A close inspection of our calibration curve in Figure 2 shows that the first two sets of points are a dummy point (0, −40) and an extrapolated point (1000, −10). Note that the data are entered in pairs, frequency first, temperature second.

For reference purposes, let's review very briefly the quadratic interpolation formula that is used. Given a function T(F) defined at three points, (F^i, T^i), (F^j,T^j), and (F^k,T^k), we must find the value of the function at an arbitrary point F, assuming that the curve through the three points is a second degree equation (quadratic) in F. The equation is:

$$T = T_j + U \quad [-R^2T_i + (R^2 - L^2)T_j + L^2T_k]$$

$$+ U^2 \; \frac{(R+L)}{RL} \quad [RT_i - (R + L)T_j + LT_k],$$

where,

$R = F_j - F_i$, $L = F_k - F_j$, and
$U = (F - F_j)/(R + L)$.

Refer to Figure 3 for a graphical interpretation of quadratic interpolation. In the program, the value of j (J in BASIC) that is chosen is such that F exceeds F^j but is less than F^k. Then i = j − 1 and k = j + 1. Thus the points F^j and F^k always bracket F.

Now a few comments on the machine language subroutines used in the programs in Tables 1 and 2. These routines are identical. They allow the T2 counter/timer on the 6522 to count pulses for a number of 50,000 clock cycle intervals. The number of such intervals is determined by the byte of data in location $OFO7 in the program. $14 = 20 such intervals give a total counting period of one second. Clearly this number may be changed to count pulses for either 0.1 s, 10.0 s, or some other time interval if

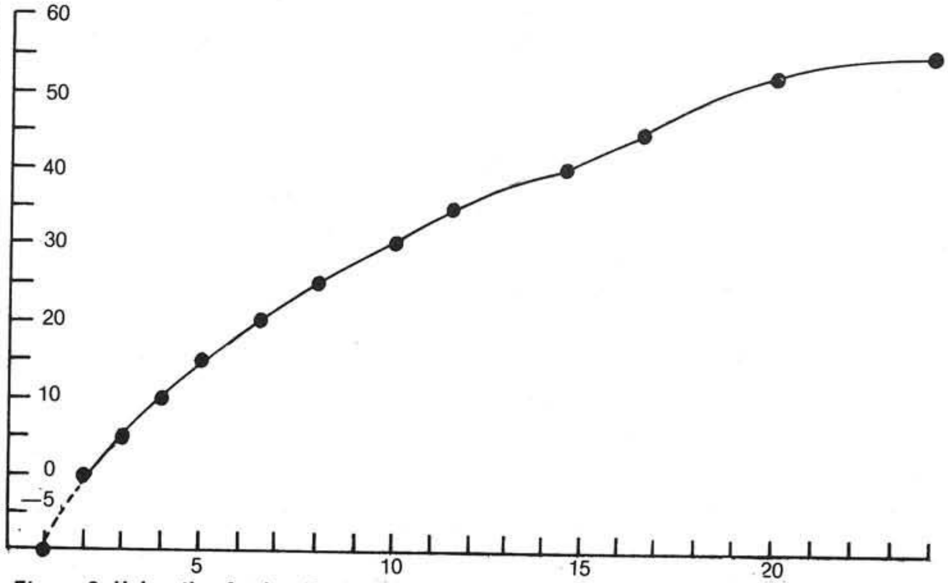


***Figure 2:* Using the Analog Devices 537J as a T/F Converter. Resistances are in Ohms and Capitances are in Microfarads.**

Table 1 . A simple frequency-to-temperature conversion program.

```
 10 DIM F(20), T(20)
 20 FOR J = 1 TO 20
 30 INPUT F(J),T(J)
 40 NEXT J
 50 POKE ∅4,∅∅: POKE ∅5, 15
 60 Y=USR(∅)
 70 Y=PEEK(49): Z=PEEK(50)
 80 FRQ=256*Z+Y
 90 FOR J = 1 TO 20
100 IF FRQ<F(J) THEN 120
110 NEXT J
120 I=J-1: K=J+1
130 L=F(J) - F(I)
140 R=F(K) - F(J)
150 U=(FRQ - F(J))/(R + L)
160 TC=T(J) + U/(R*L)*(-R*R*T(I)+(R*R-L*L)*T(J)+L*L*T(K))
170 TC=TC+U*U*(R+L)/(L*R)*(R*T(I)-(R+L)*T(J)+L*T(K))
180 TF = 9/5*TC + 32
190 TC= INT(TC + .5): TF= INT(TF + .5)
200 PRINT " ";TF;"F ";TC;"C"
210 GO TO 60
220 END
```

```
$OFOO D8        START  CLD          $OF1C 2C OD AO  TEST  BIT IFR
 OF01 A9 60            LDA $60       OF1F 50 FB            BVC TEST
 OF03 8D OB AO         STA ACR       OF21 AD 04 AO         LDA T1CL
 OF06 A9 4D            LDA $4D       OF24 C6 30            DEC CNTR
 OF08 8D 06 AO         STA T1LL      OF26 DO F4            BNE TEST
 OFOB A9 14            LDA $14       OF28 38               SEC
 OFOD 85 30            STA CNTR      OF29 A9 FF            LDA $FF
 OFOF A9 C3            LDA $C3       OF2B ED 08 AO         SBC T2CL
 OF11 8D 05 AO         STA T1LH      OF2E 85 31            STA PLSLO
 OF14 A9 FF            LDA $FF       OF30 A9 FF            LDA $FF
 OF16 8D 08 AO         STA T2LL      OF32 ED 09 AO         SBC T2CH
 OF19 8D 09 AO         STA T2CH      OF35 85 32            STA PLSHI
                                     OF37 4C D1 CO         JMP BASIC*
```

*Used in AIM 65 BASIC. Other BASICs may use a different return-from-subroutine technique.

Table 2. A program to measure frequency using BASIC and machine language.

```
10 POKE Ø4,ØØ: POKE Ø5, 15
20 Y = USR(Ø)
30 FRQ = 256*PEEK(5Ø) + PEEK(49)
40 PRINT FRQ
50 GO TO 20

$0F00 D8        START  CLD          Clear decimal mode.
 0F01 A9 60            LDA $60      Set up ACR so T1 runs free and
 0F03 8D 0B A0         STA ACR      T2 counts pulses.
 0F06 A9 14            LDA $14      The program will count pulses for
 0F08 85 30            STA COUNT    $14 = 20 intervals of 50,000 clock
 0F0A A9 4D            LDA $4D      cycles.  T1 is loaded with $C34D,
 0F0C 8D 06 A0         STA T1LL     since $C34D + 2 = 50,000.  IFR6 will
 0F0F A9 C3            LDA $C3      be set every 50,000 clock cycles.
 0F11 8D 05 A0         STA T1LH     Clear IFR6 and start T1 running.
 0F14 A9 FF            LDA $FF      Set up T2 to start counting down
 0F16 8D 08 A0         STA T2LL     from $FFFF.
 0F19 8D 09 A0         STA T2CH     Start counting pulses on PB6.
 0F1C 2C 0D A0  LOAF   BIT IFR      Has T1 timed out yet?
 0F1F 50 FB            BVC LOAF     No, then wait in this loop.
 0F21 AD 04 A0         LDA T1CL     Read T1CL simply to clear IFR6.
 0F24 C6 30            DEC COUNT    Decrement interval counter.
 0F26 D0 F4            BNE LOAF     Count pulses for another interval if
 0F28 38               SEC          interval counter has not reached zero.
 0F29 A9 FF            LDA $FF      If it has reached zero, obtain the
 0F2B ED 08 A0         SBC T2CL     number of pulses from T2 by subtracting
 0F2E 85 31            STA PULSLO   the number in T2 from $FFFF.
 0F30 A9 FF            LDA $FF      Result into locations $0031 and $0032.
 0F32 ED 09 A0         SBC T2CH
 0F35 85 32            STA PULSHI
 0F37 4C D1 C0         JMP BASIC    AIM 65 return to BASIC command.
```

necessary. The programs in Tables 1 and 2 will count to a maximum of 65535 pulses in one one-second interval at a maximum rate of 500,000 Hz, the limit of the 6522. Note that the total counting interval ma be more or less than one second by say ten microseconds. This error amounts to less than one count if the incoming pulse rate is less than 65,535 Hz, and is of no consequence for this application. The listing in Table 2 is useful as a frequency counter with no regard to our frequency-to-temperature conversion program listed in Table 1. That is, the program in Table 2 is a stand-alone frequency counting program which may be used to count the frequency of pulses arriving at PB6, provided these are TTL level pulses similar to those provided by the 555 temperature-to-frequency circuit. A clever programmer will note that if IFR5, the T2 interrupt flag, is read, the T2 counter becomes a 17 bit counter, extending the range listed above by a factor of two. We did not program this feature into the programs in Tables 1 and 2.

Now that you can measure temperature, let's see what interesting applications you can come up with, and please let us hear from you. Of course, the first thing you will want to do is put the thermistor under your tongue and measure your body temperature. Analog Devices sells a T/F converter (AS537) that provides a linear relationship between T and F. We now describe how to interface it to your computer.

The connection diagram for the AD537 is shown in Figure 4. Again, the T2 timer/counter on the 6522 is used to measure the frequency of the pulses coming from

A Program to Measure Temperature with the AD537 Interface

```
10 POKE Ø4,ØØ: POKE Ø5,15
20 Y = USR(Ø)
30 FRQ = 256*PEEK(5Ø) + PEEK(49)
40 TC = (FRQ - 2731)/1Ø
50 TF = TC*9/5 + 32
60 TF = INT(TF + .5)
70 PRINT " "; TF; "F"; TC; "C"
80 GO TO 2Ø
```

the AD537. With the values shown, the AD537 will produce a linear relationship between frequency and absolute temperature (Kelvin degrees) of 10Hz/°K. At room temperature (about 300°K) the frequency will be 3000 Hz. The 15 k potentiometer in Figure 4 is adjusted to give the correct temperature. The adjustments are easier if the 15 k potentiometer is replaced by a 9.1 k resistor in series with a 2 k potentiometer to trim the total resistance to about 10 k ohms.

To convert from absolute temperature (°K) to Celsius temperature, we make use of the formula [°C = °K − 273.1]. Then we can convert to Fahrenheit with the formula [°F = (°C)(9/5) + 32]. The entire process is handled with the BASIC program listed in Table 3. This program also calls the machine subroutine listed in Tables 1 and 2.

The AD537 is a versatile device. It can also be used as a very fine voltage-to-frequency converter with only a few external components. Analog Devices appears to share my philosophy that the fewer external components around, the less likely it is for me to have problems. In any case, with the same integrated circuit you can make youself a voltmeter. The same machine language subroutine will provide the necessary software, and a simple BASIC calling program will place the decimal point and output the answer. You should be sure to obtain the specification sheets on the AD537 if you get one. They contain a lot of useful and vital information. For example, the Ad537 can be operated in a remote location with a two-wire connection. Several of them can be multiplexed because the pulse output pin is an open-collector connection. The AD537 is much more expensive than a 555 timer, and jAnalog Devices may require a minimum order. Perhaps the members of your computer club can get together and place an order. Write: *Analog Devices*, 1 Industrial Park, P.O. Box 280, Norwood, Ma. 02062.

If you do not have a BASIC interpreter on your computer, then the machine language output subroutines given in Tables 4, 5, and 6 may be used with the programs in Tables 2 and 3 to output the frequency and temperature information. (Note that in order to measure temperature with the 555 timer circuit, a BASIC interpreter is an absolute essential.) SYM-1 and KIM-1 users can use the binary-to-BCD conversion routine in Table 4, together with their own subroutine that displays six numbers on the 7-segment LEDs, to display the frequency of the pulses measured by the machine language program in Table 2. The JMP BASIC instruction at $0F37 would be replaced by the following instructions:

```
20 60 0F    JSR DCML
20 ?? ??    JSR DISPLAY
4C 00 0F    JMP START
```

where DISPLAY is the user's routine to display six digits. AIM 65 owners will want to use all the subroutines in Tables 4, 5, and 6 to output the BCD digits representing the frequency to the AIM 65 twenty character display. To use the subroutines with the AD537 interface and the program in Table 3, you must first subtract $0AAB (2731) from the measured pulse rate to convert it to Celsius, and then output the BCD digits, remembering for yourself where the decimal place is. Good luck.

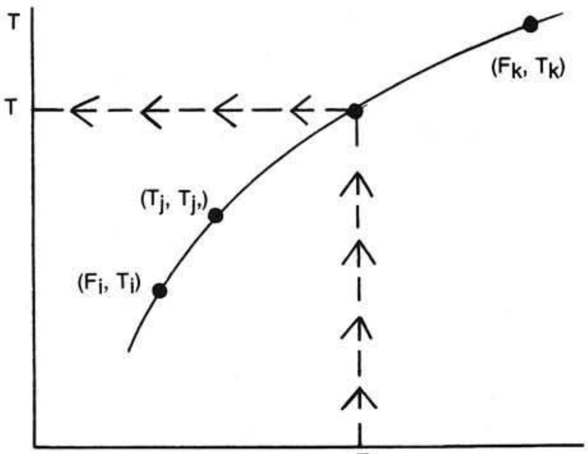


Figure 3: **Graphical Interpretation of Quadratic Interpolation**

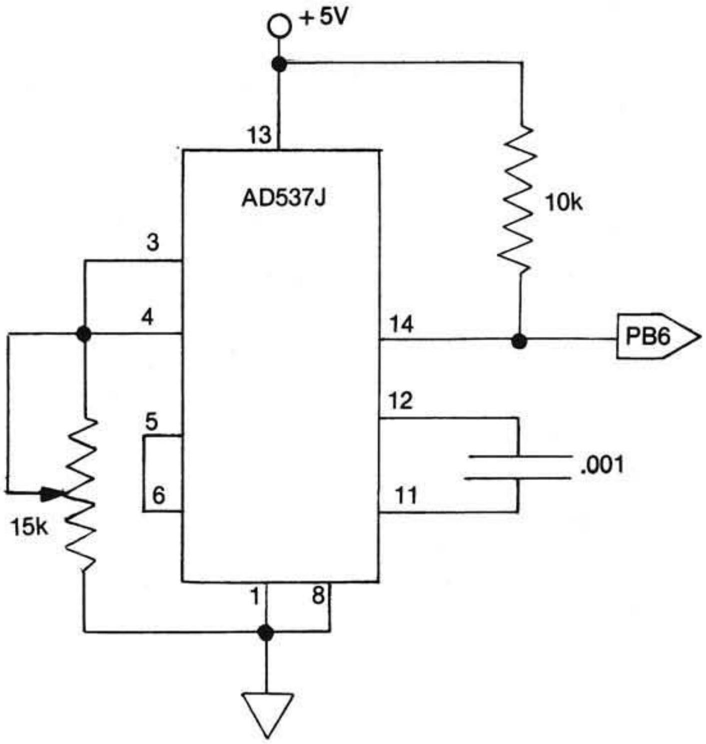


Figure 4: **Frequency-to-Temperature Conversion Curve for the 555 Circuit.**

Table 4. A subroutine to convert a 16-bit binary number to six BCD digits.

$0031 = PLSLO; contains low-order byte of 16-bit number to be converted.

$0032 = PLSHI; contains high-order byte of 16-bit number to be converted.

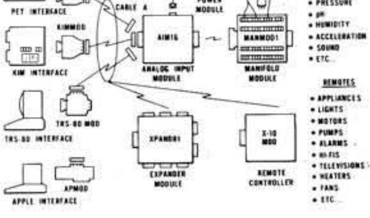
$0F60	A9 00	DCML	LDA $00	Clear memory locations for the
0F62	85 01		STA BCDLO	BCD number
0F64	85 02		STA BCDMI	
0F66	85 03		STA BCDHI	
0F68	F8		SED	Set decimal mode for subsequent
0F69	A0 10		LDY $10	additions. Y contains number of
0F6B	06 31	THERE	ASL PLSMO	bits to be converted. One bit of
0F6D	26 32		ROL PLSMI	the number is shifted into the
0F6F	A2 FD		LDX $FD	carry bit at a time. X serves as
0F71	B5 04	MORE	LDA BYT,X	a counter for a triple-precision
0F73	75 04		ADC BYT,X	addition.
0F75	95 04		STA BYT,X	
0F77	E8		INX	
0F78	D0 F7		BNE MORE	
0F7A	88		DEY	Get the next bit.
0F7B	D0 EE		BNE THERE	When Y = 0, the conversion is
0F7D	D8		CLD	complete.
0F7E	60		RTS	Return to calling program.

Table 5. A subroutine to convert six BCD digits to ASCII and call an output subroutine.

```
$0F80 A2 06      ASCII    LDX $06       X contains the number of BCD digits.
 0F82 A9 00               LDA $00       Our out-character (OUTCH) subroutine
 0F84 85 04               STA LOC       requires LOC to start at $00.
 0F86 A5 03      NEXT     LDA BCDHI     Get the most-significant nibble
 0F88 29 F0               AND $F0       of the BCD number.  The BCD digits
 0F8A 4A                  LSR A         will be output from the most-
 0F8B 4A                  LSR A         significant to the least significant.
 0F8C 4A                  LSR A         Move high-order nibble to the low-
 0F8D 4A                  LSR A         order nibble.
 0F8E 18                  CLC
 0F8F 69 30               ADC $30       Adding $30 to a BCD digit converts
 0F91 20 A5 0F            JSR OUTCH     it to ASCII.  Output the character.
 0F94 A0 04               LDY $04       Get another nibble.
 0F96 06 01      AGN      ASL BCDLO
 0F98 26 02               ROL BCDMI
 0F9A 26 03               ROL BCDHI
 0F9C 88                  DEY
 0F9D D0 F7               BNE AGN
 0F9F CA                  DEX           Get another digit?
 0FA0 D0 E4               BNE NEXT      Yes.
 0FA2 60                  RTS           No.
```

Table 6. A subroutine to display six digits on the AIM 65 display.

```
$0FA5 09 80      OUTCH    ORA $80       ASCII character is in the accumulator.
$0FA7 85 05               STA TEMP      Set bit seven and store temporarily.
$0FA9 8A                  TXA           Save X.
$0FAA 48                  PHA
$0FAB A6 04               LDX LOC       LOC contains location of the digit
$0FAD A5 05               LDA TEMP      on the 20 character display
$0FAF 20 7B EF            JSR OUTDD1    Use AIM 65 monitor routine.
$0FB2 E6 04               INC LOC
$0FB4 A5 04               LDA LOC
$0FB6 C9 06               CMP $06       Have all six characters been output?
$0FB8 90 04               BCC AHEAD
$0FBA A9 00               LDA $00       Yes.  Clear LOC.
$0FBC 85 04               STA LOC
$0FBE 68         AHEAD    PLA           Get X back.
$0FBF AA                  TAX
$0FC0 60                  RTS           Return to the calling routine.
```

Shorthand Commands for Superboard II and Challenger C1P BASICs

This article shows how to intercept the BASIC's input routine and how to implement a shorthand notation.

Henk J. Wevers
Cloeckendaal 38
6715 JH EDE
The Netherlands

As a superboard or Challenger IP owner, you surely have noticed the large amount of adds for extra software for the Apple, PET and TRS 80 machines and you hoped for just some of these goodies to show up for your own computer.

Well, no such luck, so far. So, now we have to do the job ourselves. One of the advertised options for the TRS 80, single stroke instructions, looked nice and I started to program something like that for the OSI machine. The result is presented here, and the shorthand routine is almost alway present in my machine during program development.

Before describing how the job was done, let's first have a look at what this routine does exactly. After loading the program, type

POKE 536,34:POKE 537,2

and instead of typing the instruction letter by letter, you can enter it by hitting the ESCAPE KEY and another key after that. The last key determines which instructin is entered. For instance, if you want to enter RIGHT$, hit the ESCAPE key first. On the display the cursor will change to an arrow to warn you that the next entry will be an instruction instead of a single character. Now hit C and you have just entered RIGHT$ as the display shows you.

All instructions are accessible in this way, and by altering the table in memory locations 0280 through 02C3 you may even choose your own shorthand codes.

There are a few things about the Microsoft Basic for the OSI machines that should be known before you can fully understand the program.

First, if Basic asks for an input, the input routine is accessed by a vector located in memory locations 0218 and 0219 (hex) or 536 and 537 decimal. You can intercept the input by changing these locations and so routing the input through your own routines, and that is the way I did the job.

Secondly, to use the token system in their Basic, Microsoft put a table containing all possible instruction in their program starting at location A084 hex. The instructions are separated in the table by the last character of ever instruction having bit 7 set. If you strip off bit 7 of the token, you have the relative position of the instruction in the table. If we look at the instruction END with token 80, then this one has the first position in the table (actually position 0, since we count from zero). RIGHT$ with Token C3 (hex) has the hex position of 43 in the table.

Third to consider is that the input buffer is located at hex location 13 and up in page zero. X serves as the buffer pointer during input.

And lastly, location 0200 is used as the relative cursor location. The routine WRCHAR at BFC2 puts the character in location 0201 on the screen, at location D300 + the contents of 0200. You can use this in your own programs; I found it very handy. Now to the program. Most of it will be clear to you now.

First a character from the keyboard or cassetport is input, and if it is not 'ESCAPE' we return it to the A register. Basic can't tell the difference between this routine and the original one. If the character is 'ESCAPE', the cursor is changed from underline to right arrow and another character is input. The shorthand table (0280 through 02C3) is scanned for a match, and if a match is found, the X register will contain the token for that command with bit 7 stripped off. If no match is found another (shorthand) character is input. The start of the instruction is searched for in the Basic table and then the instruction is output to the screen and stored in the input buffer, character by character. If bit 7 of a character is set, (signaling the end of the instruction) this process is stopped, bit 7 stripped off and the last character processed. Another character or shorthand command can then be input.

By now you have noticed the strange BIT $07A9 instruction around locations 0263 and 0274. It is a short way of entering a routine with different contents of the accumulator. For instance, if you enter the OUTCH routine via locations 0262 - 0263 - 0266, you have the character in A output, but entering the routine via 0264-0266 you have the 'BELL' character in A and so output. 025E and 026F switch between the two, depending on the input buffer being full.

Now let's look at the shorthand table starting at location 0280. The last two characters of the addresses also give the token for the instruction. I have programmed this table for you in a way that I have found convenient for the location of the shorthand commands on the keyboard. If you want to program this layout yourself, just enter the ASCII value in the table for the shorthand key you want. For example, if you want the Q-key to be shorthand for 'THEN', only put 51, the ASCII code for Q, in location 02A0, the location for 'THEN'.

The last thing to explain is the choice of where to put this routine in memory. I used locations 0222 and up, because these locations are unused by BASIC and the monitor, and they are not affect by either a cold or a warm start. If you have hit the BREAK key you have to change the input vector again by proper POKING as described earlier.

I hope this little routine will make programming a little easier for you as it did for me. Imagine being able to RUN, LIST, SAVE, and LOAD with one simple keystroke! Most likely, you have exceeded the maximum line length by using a ? instead of PRINT, so you had to type the line all over again after a list, because the program wouldn't load. This routine shows PRINT on the screen after 'ESCAPE' and ? so you will always see what you are doing. Good luck!

Second part of Memloc is taken for command in that location.

MEMLOC	COMMAND	SHORTHAND	CODE(HEX)
0280	END	H	48
0281	FOR	Q	51
0282	NEXT	G	47
0283	DATA	O	4F
0284	INPUT	I	49
0285	DIM	J	4A
0286	READ	U	55
0287	LET	!	21
0288	GOTO	R	52
0289	RUN	'CR'	0D
028A	IF	D	44
028C	GOSUB	T	54
028D	RETURN	Y	59
028E	REM	"	22
028F	STOP	^G	07
0290	ON	:	3A
0291	NULL	^E	05
0292	WAIT	^A	01
0293	LOAD	L	4C
0294	SAVE	K	4B
0295	DEF	^D	04
0296	POKE	A	41
0297	PRINT	?	3F
0298	CONT	^B	02
0299	LIST	'RUBOUT'	7F
029A	CLEAR	^C	03
029B	NEW	'LF'	0A
029C	TAB(	.	2E
029D	TO	W	57
029E	FN	^F	06
029F	SPC(	;	3B
02A0	THEN	F	46
02A1	NOT	)	29
02A2	STEP	E	45
02A3	+	+	2B
02A4	-	-	2D
02A5	*	*	2A
02A6	/	/	2F
02A7	^	^	5E
02A8	AND	5	35
02A9	OR	%	25
02AA	>	>	3E
02AB	=	=	3D
02AC	<	<	3C
02AD	SGN	(	28
02AE	INT	6	36
02AF	ABS	&	26
02B0	USR	'	27
02B1	FRE	7	37
02B2	POS	8	38
•2B3	SQR	9	39
02B4	RND	Ø	30
02B5LOG	LOG	$	24
02B6	EXP	4	34
02B7	COS	2	32
02B8	SIN	1	31
•2B9	TAN	3	33
02BA	ATN	#	23
02BB	PEEK	S	53
02BC	LEN	M	40
02BD	STR$	B	42
02BE	VAL	,	2C
•2BF	ASC	N	4E
02C0	CHR$	V	56
02C1	LEFT$	Z	5A
02C2	RIGHT$	C	43
02C3	MID$	X	58

SHORTHAND COMMAND FOR OSI CHALLENGER IP and SUPERBOARD

```
0222    20 BA FF  SHORT1    JSR   IN          INPUT CHAR FROM KEYB. OR TAPE
0225    C9 1B               CMPIM $1B         IS IT 'ESCAPE' ?
0227    F0 01               BEQ   SHORT2      YES? BRANCH
0229    60                  RTS               NO, RETURN TO BASIC WITH ECHAR
022A    98        SHORT2    TYA               SAVE Y
022B    48                  PHA               AND
022C    8A                  TXA               X REGISTERS
022D    48                  PHA
022E    A9 12               LDAIM $12         LOAD 'ARROW' TO
0230    8D 01 02            STA   CURSOR      CHANGE CURSOR
0233    20 C2 BF            JSR   WRCHAR      DO IT
0236    A2 43     SHORT3    LDXIM $43         LOAD MAX TABLE INDEX
0238    20 BA FF            JSR IN            INPUT SHORTHANDCOMMAND
023B    DD 80 02  SHORT4    CMPX  TABLE       COMPARE WITH TABLE
023E    F0 06               BEQ   SHORT5      FOUND IT? BRANCH
0240    CA                  DEX               DECREMENT INDEX FOR NEXT TRY
0241    10 F8               BPL   SHORT4      IF NOT DONE? LOOP BACK
0243    4C 36 02            JMP   SHORT3      NO MATCH? IGNORE AND LOOP BACK
0246    E8        SHORT5    INX               COME HERE WITH TABLE INDEX IN X
0247    A0 FF               LDYIM $FF         PREPARE FOR LOOKUP IN COMMAND T
0249    CA        SHORT6    DEX               COMMAND FOUND?
024A    F0 08               BEQ   SHORT8      YES? BRANCH
024C    C8        SHORT7    INY               NO SKIP CURRENT COMMAND IN TABL
024D    B9 84 A0            LDAY  $A084,Y     DONE YET?
0250    10 FA               BPL   SHORT7      NO, LOOP BACK
0252    30 F5               BMI   SHORT6      YES? GO AND TRY NEXT ITEM
0254    68        SHORT8    PLA               GET INPUTBUFFER INDEX BACK
0255    AA                  TAX               AND  STORE IT IN X REG
0256    C8        SHORT9    INY               GET READY TO STORE COMMAND IN B
0257    B9 84 A0            LDA $A084,Y       GET COMMAND CHAR
025A    30 0F               BMI   SHORT10     IF LAST CHAR OF COMM, BRANCH
025C    E0 47               CPXIM $47         INPUTBUFFER FULL?
025E    B0 04               BCS   +04         YES? BRANCH TO SHORT9A + 1
0260    95 13               STAX  $13         STORE CHAR IN INPUTBUFFER
0262    E8                  INX               INCR. BUFFERPOINTER
0263    2C A9 07  SHORT9A   BIT   $07A9       SKIP OR LOAD 'BELL' IN A
0266    20 E5 A8            JSR   OUTCH       OUTPUT CHAR
0269    D0 EB               BNE   SHORT9      BRANCH ALWAYS
026B    29 7F     SHORT10   ANDIM $7F         LAST CHAR. STRIP OF HIGH BIT
026D    E0 47               CPXIM $47         INPUTBUFFER FULL?
026F    B0 04               BCS   + 04        YES, BRANCH
0271    95 13               STAX  $13         STORE CHAR IN INPUTBUFFER
0273    E8                  INX               INCR BUFFERPOINTER
0274    2C A9 07            BIT   $07A9       SKIP OR LOAD 'BELL' IN A
0277    20 E5 A8            JSR   OUTCH       OUTPUT CHAR
027A    68                  PLA               RESTORE
027B    A8                  TAY               Y REGISTER AND
027C    4C 22 02            JMP    SHORT1     LOOP BACK

0280 - 02C3        TABLE
```

```
POKE 536,34:POKE 537,2  PUTS SHORTHAND ON AND
POKE 536,186:POKE 537,255 OFF
```

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 ________

SAE 1-1 PLEASE SHIP PREPAID ________ SAE 1-1(s)
@ $43.50 each
California Residents Please Pay $46.33 (Includes Sales Tax)

SAE 1-2 PLEASE SHIP PREPAID ________ SAE 1-2(s)
@ $46.50 each
California Residents Please Pay $49.52 (Includes Sales Tax)

enclosures group

771 bush street
san francisco, california 94108

***TM Rohm & Hass** **Patent Applied For**

A Formatted Dump Routine for the AIM 65

This HEX dump utility permits the user to control the formatting of the dump to conform to his printer's capabilities.

W.E. Wilson
Washungton State U.
Pullman, WA 99164

The Dump routine in the AIM 65 Monitor produces a continuous character string and thus is not very readable. The dump format is essentially not fit for human consumption. The serious AIM 65 user who needs a memory dump is thus limited to using the Monitor "M" command, which only dumps four locations at a time. A more useful and efficient dump routine with a variable output format was needed by the author and thus the following program was written.

The Formatted Dump routine will dump memory over the range specified in response to the "FROM = " and "TO = " parameters. The number of bytes in each line of the dump is specified in response to "/". All input and output is in hexidecimal. Each line of the dump gives the starting address of the first byte in the line, a space, 1st byte, space, 2nd byte, etc. The standard AIM-65 printer will handle $05 bytes per line and an 80 column TTY type unit will handle up to $16 (22) bytes per line.

The dump routine makes extensive use of the routines in the AIM-65 Monitor as well as RAM locations reserved for the Monitor. No locations outside of the Monitor area, except for the dump routine itself, are used by the dump routine. Thus the dump routine may be located at any convenient place in RAM and will not affect any other software. The following dumps demonstrate the use of the routine.

AIM-65 MONITOR ROUTINES USED IN DUMP PROGRAM

E7A3 = Print "FROM = " and get address in $A41C/D.

E83E = Print " " (blank).

E910 = Move address from $A41C/D to $A41AB.

E7A7 = Print "TO = " and get address in A41C/D.

E837 = Print "/".

E785 = Get two hex digits and store in A419.

EA13 = Print "CRLF".

EA46 = Print one hex byte = Two ASCII characters.

EB58 = LDAY - Simulates LDA (N), Y without page 0.

E182 = AIM-65 Monitor Re-entry.

A Formatted Dump Routine for the AIM-65
List 1

```
0112 4C JMP 0F90

0F90 20 JSR E7A3
0F93 B0 BCS 0F90
0F95 20 JSR E83E
0F98 20 JSR F910
0F9B 20 JSR E7A7
0F9E B0 BCS 0F9B
0FA0 20 JSR E83E
0FA3 20 JSR E837
0FA6 20 JSR E785
0FA9 20 JSR EA13
0FAC AD LDA A41C
0FAF 38 SEC
0FB0 ED SBC A41A
0FB3 48 PHA
0FB4 AD LDA A41D
0FB7 ED SBC A41B
0FBA 30 BMI 0FF6
0FBC D0 BNE 0FC1
0FBE 68 PLA
0FBF F0 BEQ 0FF6
0FC1 AD LDA A41B
0FC4 20 JSR EA46
0FC7 AD LDA A41A
0FCA 20 JSR EA46
0FCD AE LDX A419
0FD0 A0 LDY #00
0FD2 20 JSR E83E
0FD5 A9 LDA #1A
0FD7 20 JSR EB58
0FDA 20 JSR EA46
0FDD C8 INY
0FDE CA DEX
0FDF D0 BNE 0FD2
0FE1 20 JSR EA13
0FE4 AD LDA A419
0FE7 18 CLC
0FE8 6D ADC A41A
0FEB 8D STA A41A
0FEE 90 BCC 0FF3
0FF0 EE INC A41B
0FF3 4C JMP 0FAC
0FF6 4C JMP E182
```

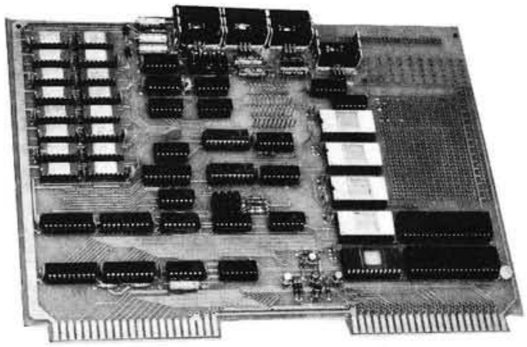
```
RUN
FORMATTED DUMP ROUTINE FOR THE AIM-65
ENTER VIA F3 FUNCTION KEY =↑
SPECIFY : FROM=,TO=,/=(CHRS/LINE)
    CHRS/LINE=TWO HEX DIGITS
<↑>FROM=B000 TO=B020 /05
B000 4C A3 CE 4C 7F
B005 B2 FE BE D1 C0
B00A 5D B6 5B B5 FF
B00F BA 66 B7 BB B9
B014 D9 BD EF B9 13
B019 B8 13 B7 EB B6
B01E 96 B7 30 B6 F6

<↑>FROM=B000 TO=B020 /08
B000 4C A3 CE 4C 7F B2 FE BE
B008 D1 C0 5D B6 5B B5 FF BA
B010 66 B7 BB B9 D9 BD EF B9
B018 13 B8 13 B7 EB B6 96 B7

<↑>FROM=B000 TO=B020 /10
B000 4C A3 CE 4C 7F B2 FE BE D1 C0 5D B6 5B B5 FF BA
B010 66 B7 BB B9 D9 BD EF B9 13 B8 13 B7 EB B6 96 B7
```

DRAM PLUS™

DRAM PLUS [16K RAM]: TCB-101-16
DRAM PLUS [32K RAM]: TCB-101-32

DRAM PLUS offers the most powerful memory expansion capabilities available for the ASK family of microcomputers. Its many important features include:

- **16K or 32K Dynamic RAM with all refresh handled on the board and completely transparent to the host microcomputer.**
- **Memory does not have to be addressed as a single 16K or 32K segment. 4K segments of memory may be placed on 4K boundaries.**
- **Up to 16K ROM/EPROM with provision for four ROMs or EPROMs: 2716/2516 2K EPROMs, 2732/2532 4K EPROMs, or 2332 ROMs. These may be mixed on the board.**
- **Two Versatile Interface Adapters, each with two 8-bit I/O ports, additional handshaking lines, two timers, and a serial-to-parallel shift register.**
- **Prototyping Area has space for adding circuits: memory write protection, floppy disk controller, communications devices, A/D or D/A, etc.**
- **EPROM Programmer handles all four types of EPROM: 2716/2516 2K and 2732/2532 4K.**
- **Simple Power Requirements of +5 volts at 1 amp and +12 volts at 150 milliamps. On board regulators permit unregulated power to be used.**



Dynamic RAM Memory: The RAM is composed of compact 4116 type dynamic memory. Each 4116 chip contains 16K bits, organized as 16K addresses with one bit of information per address. An 8 bit byte of memory is obtained by addressing 8 memory chips in parallel. Eight 4116 memory chips provide 16K bytes of memory. DRAM PLUS has provision for two sets of ram chips for a total RAM capacity of 32K bytes. All of the memory is socketted and each board is tested for the full 32K capacity. The only difference between the 32K and 16K versions is that 16K bytes of memory have been removed from the 16K version board. This memory may be added at any time.

EPROM and/or ROM Memory: There is provision for up to four EPROMs and/or ROMs to be added. These may be a mixture of the following types:
2516/2716 and 2532/2732 EPROMs or 2332 ROM.
The XX16 EPROMs contain 2K bytes and the XX32 contain 4K bytes. The 2332 is a 4K ROM. Using the 4K parts, up to 16K of read-only memory may be added.

Versatile Interface Adaptor [VIA]: Contains two 8-bit programmable I/O ports with additional handshaking lines; two timers; and a serial-to-parallel/parallel-to-serial shift register. The VIA may be used to interface keyboards, printers, and many other devices to the DRAM PLUS and/or the host system. A 24 page data sheet on the 6522 VIA is included in the documentation package.

EPROM Programmer: The VIA's are used in conjunction with some additional circuitry on the DRAM board to program all four types of EPROM. A separate programming socket is provided, a regulator circuit provides the programming voltage from a +27 volt input, and voltage to the EPROM is controlled by the program to prevent accidental damage to the EPROM.

Transparent Refresh: All of the circuitry for refreshing the dynamic RAM is contained on the board and operates in a manner that makes it completely transparent to the host microcomputer. All of the refresh memory accessing is done during Phase One, leaving the memory completely available to the host microcomputer during Phase Two. No "clock stretching" or "wait states" are required.

RAM Memory Addressing: Although the RAM is packaged as one or two 16K segments, provision has been made on DRAM PLUS for the memory to be addressed at four separately defined 4K boundaries per 16K segment. There are some restrictions on the set of boundaries that may be used within any 16K segment. Address bits A12 and A13 must not be the same for any of the 4K segments within a 16K segment. This results in a type of "Chinese Menu" selection. One 4K segment may be selected from each column of the following table, which lists the starting address of the 4K boundaries in hexidecimal:

0000	1000	2000	3000
4000	5000	6000	7000
8000	9000	A000	B000
C000	D000	E000	F000

An examination of the table will show that any four contiguous blocks will automatically come from different columns. If blocks were selected at 1000, 2000, and 3000, then the fourth block would have to be 0000 (which is highly unlikely on an AIM/SYM/KIM), 4000, 8000, or C000, for that 16K segment of memory.

Prototype Area: A prototyping area provides space and support for the addition of special circuits. The actual prototyping grid is approximately 2" by 2-3/4" and consists of a matrix of 13 by 28 holes spaced for standard sockets and IC's. The area is designed so that wirewrap or solder sockets may be used. The address and data lines are readily accessible to this area and convenient +5V and ground runs are provided. Connections to this area may be made through a separate connector facility which can support a standard connector with up to 50 pins.

MICRO Bus Compatible: The connections between the DRAM PLUS and the AIM/SYM/KIM follow the same conventions used by the original KIM-4 mother board. DRAM PLUS may be interfaced via a simple cable or the MOTHER PLUS.

General Information: The board is high quality, double sided with two sets of gold plated fingers with the same positioning as the connectors on the AIM/SYM/KIM. The board is the same size and shape as the SYM and KIM: 7-7/8" wide (excluding the edge connectors) by 10-3/4" long. All IC's are socketted to make field repair and servicing simple. Full documentation consists of instructions, schematics, program listings, data sheets and application notes. A Memory Test and an EPROM Programming Program are provided on a cassette tape which loads and works on the AIM/SYM/KIM. The DRAM PLUS Manual is available separately for $10.00, and this cost may be applied towards the purchase price.

Trade-in: You may be able to trade-in your Rev B **MEMORY PLUS** board toward a **DRAM PLUS**. Write/Call for details.

DRAM PLUS with 16K RAM [TCB-101-16] $295.00 US/Canada, or $325.00 All Other Countries, plus shipping and handling.
DRAM PLUS with 32K RAM [TCB-101-32] $395.00 US/Canada, or $435.00 All Other Countries, plus shipping and handling.
Shipping/Handling: US $2.50 Canada $6.00 All Other Countries $18.00

This is one page from our eight page 1980 Catalog. Send name and address for your copy of the compete 1980 Catalog.

The COMPUTERIST, Inc., 34 Chelmsford Street, Chelmsford, MA 01824 617/256-3649

MICRO SUBSCRIBERS SPECIAL

THANK YOU, COMMODORE...

Just about a year ago, Skyles Electric Works introduced a 15 inch wide keyboard with all PET functions on nice, plump, full-sized keytops with torsion spring action. S.E.W. knew that most owners of the PET 2001-8 couldn't remain happy with the undersized keytops and unsatisfactory spring action.

Now you have announced that, effective in January of 1980, all PETs and CBMs would feature full sized keyboards.

So, thank you, Commodore, for confirming that S.E.W. was right all along.

Meanwhile, S.E.W. hasn't been resting...and now offers with the **Big KeyBoard** an 18 inch ribbon cable at no extra charge...and a cassette tape containing BASIC and machine language programs that convert the **Big KeyBoard** to ASCII or typewriter operation, including lower case characters, and upper case through the shift key. Even a shift lock, quotations in the proper place, numbers across the top row.

All keytops double shot for lifetime durability. Switching action uses gold cross-point contacts; torsion springs are gold plated. Housing is black heavy gauge aluminum. Entire unit can be rinsed to clean under water and left to dry. 120 day warranty. **Only $125.00***

MICRO SUBSCRIBERS SPECIAL TILL MAY 15, 1980 $99.50*

Wouldn't You Love Your PET 2001-8 Even More With the Skyles Big KeyBoard?

Enclosed is my MICRO address label showing subscription number, please send ___Big Keyboard(s). Enclosed is a check $______*. Charge my VISA, Mastercharge (circle one) #__________________ Expires________

**California residents: please add 6% or 6.5% sales tax as required.*

Skyles Electric Works

231E South Whisman Road
Mountain View, CA 94041
(408) 735-7891

New and Better PET User Port Printer Routines

A series of programs are presented which drive any TTL, parallel, or ASCII printer from the PET's user port.

Michael Tulloch
103 White Cr.
Niceville, FL 32578

This article describes three programs which drive a printer from PET's user port. Any TTL, parallel, ASCII, printer can be driven. Two of the programs are in machine language and one is in BASIC.

Although there are several IEEE to serial and IEEE to parallel adaptors available for the Pet, the user's port is often needed to drive an ASCII device. In my case I saved $100 dollars by using the user port to drive my Trendcom printer. No hardware (except a cable and two connectors) is required. The software is equally simple: A printer driver with hand shake and a screen reader.

There are several reasons to drive your printer or other ASCII device from the user port. First, it is quick and easy. Second, some of the IEEE to ASCII adaptors respond to any and all device addresses. Third, if you already have an adaptor, the user port allows a tempory installation without interfering with existing devices. Another reason is that it allows you to have better and more direct control over the output. Both data and hand shaking can be done explicitly with software. Finally, and for me most importantly, it saves money. Just $2.19 for a ribbon cable and two junkbox connectors, had me printing

In general the following two programs comprise a screen printer. Two parameters can be adjusted by the calling program (or as direct commands): Start point, and ÷ of rows (if implemented in RAM), Thus a specific area, or window, of the screen can be printed. The two programs are named: 1. Printer Driver and 2. Screen Reader. For timing reasons Printer Driver is implemented only in machine language. Screen Reader, however, can be implemented either in machine language (Version A) or BASIC (Version B).

```
10000 POKE850,13:SYS849:
      FORR=0TO23:FORC=0TO39
10010 A=PEEK(32768+C+R*40)
10015 IF A=18 AND C=0 THEN STOP
10020 IF A<=31 THEN A=A+64::GOTO10060
10030 IF A<=63 THEN 10060
10040 IF A>127 THEN A=A-127:GOTO10060
10045 A=A+32
10060 POKE850,A:SYS826
10070 NEXT C:POKE850,13:SYS826
10080 NEXT R
10090 RETURN
READY.
```

Figure 1

Let's start with the easy one first— the BASIC Screen Reader. Figure 1 is a listing of this routine. Line 1000 clears the small printer buffer by making a carriage return and calling Printer Driver (located at 826 in this example). Line 10005 forms the screen reading loops with R the Row counter and C the Column counter. Here only eleven lines are printed. The Screen Value is placed into A by line 10010. Lines 10020 through 10045 convert the screen value to its equivelent ASCII code. Notice that graphic characters are printed as lowercase letters if they are on letter keys. Reversed letters are printed as not reversed letters, and not all graphic characters are printed. Figure 2 gives a sample of print out for the PET character set. The equivalant screen values are though 255. Version A is the machine language equivalent of Screen Reader. It's principle advantage is that it runs hundreds of times faster. In fact, on my Trendcome 100, which prints bidirectionally, you can't even see it hesitate between lines. At the Trendcom's 40 char/sec rate, a full screen of 1000 characters is printed in 30 seconds. Not Bad!

Another advantage is that you can hide it in the second cassette buffer and load it in only once. The BASIC version has to be attached to your program somehow.

A flow chart is shown in fiqure 3. It is annotated for the machine langage version. Figure 4 shows the dissassembled code. Figure 5 gives the HEX code as out put by the PET monitor program.

Block I initializes all registers. The screen read address is initialized to 32767. This is one less than the upper left screen start address value. Memory location 995 ($03E3) is the number of Rows to be printed. It's used as the Row counter. A column counter is stored in 992 (03E0). It is initalized with 40 and 40 is held in the X register for later use.

Block II increments the screen read address. Block III gets the screen value occupying the screen read address. This value is stored in location 996. Block IV is the adjustment routine. This is different from the scheme used in the BASIC program. Instead of using subtraction, addition is used. Although the logic is inverted from the BASIC program, the value adjustments are the same. Critical temporary storage rigisters in addition to the program itself are listed in table I. The adjusted value is passed to the Printer Driver (Block V).

When control returns to Screen Reader the colums counter is decrimented. If the column counter is not equal to zero, then it is reset to the value stored in the X register (normally 40). Rows are then decrimented (Block IX)

Block X checks for the row counter equal to zero. If it is not, then a new screen value is read. If it is then that the program returns control to the calling routine.

The Printer Driver routine dissassembled listing is shown in figure 6. The PET monitor HEX dump is given in figure 7. Printer Driver takes a value (here stored in 85210), places it on the user port output lines, provides a data ready output pulse and waits for an ACKnowledge pulse. NOTE: If no ACK pulse is returned the program will continue to hold the PET off line. You must assure an ACK pulse will always be returned!

The above description of Printer Driver also sets up the Via registers. Each time it is entered it resets these registers for its own use. Only the E84C register is restored to its original value. Further, the routine inhibits interrupts. If an interrupt were to occur during the brief time a data ready pulse was being given, multiple outputs could be caused (and it does happen). There is also the chance that the ACK pulse would be missed, leaving PET in Limbo. Unfortunately this bit of protection has an adverse side affect. PET's internal clock will not keep correct time. Depending upon the amount of printing and your printer's characteristics this error can be substantial.

There are several improvements which could be made to these routines. Reversed character handling could be added to the "A" version of Screen Reader. Blanks could be output for nonprintable graphics characters. Codes could be developed for cursor command characters. You will probably want to make changes for your particular printer. There is room within Printer Driver to add a delay loop or NOPs to stretch the output pulse. Finally, Printer Driver can be used alone by passing ASCII values directly. Simply use PET's ASC ( ) command and Poke location 852.

Table I

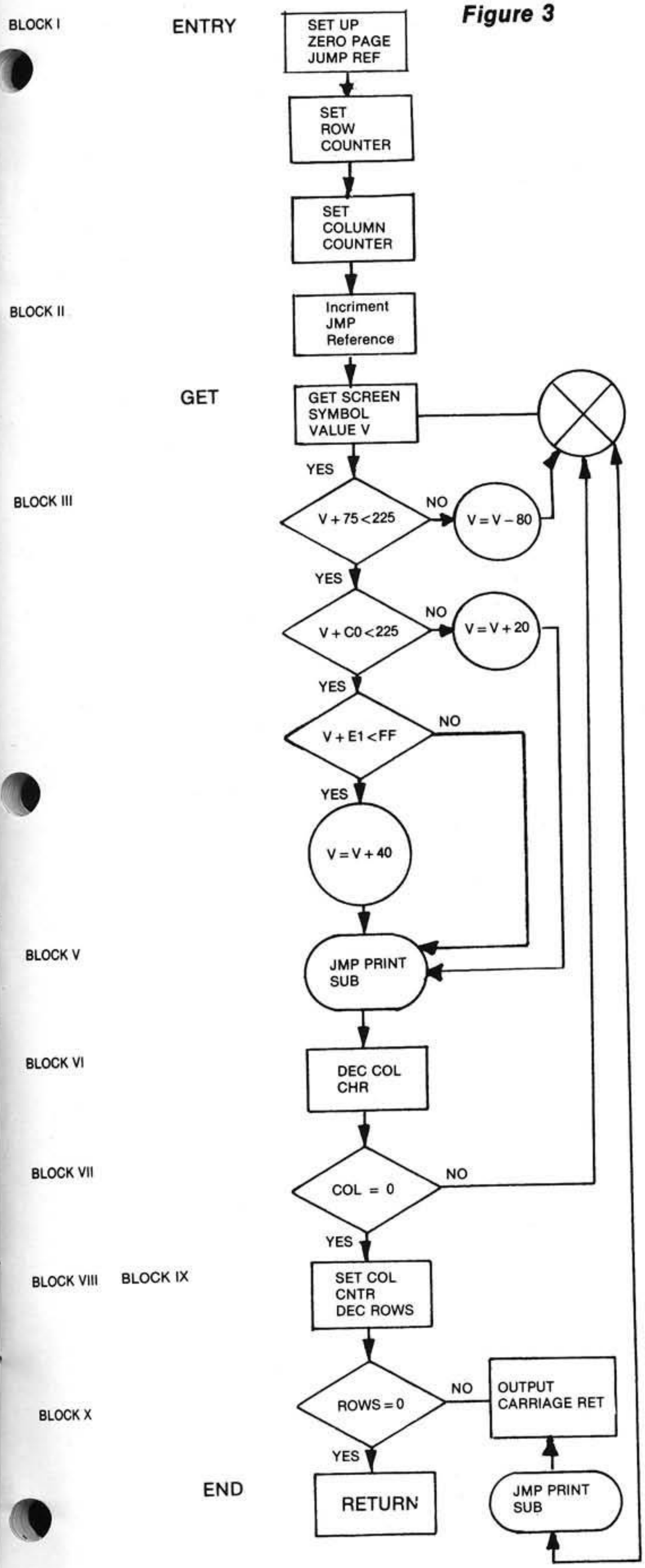
Decimal	Hex	Function
1.2	$1.2	Screen read address
992	$03E0	Column Counter
993	03E1	Row counter
995	03E3	Row input value
996	03E4	Screen value
852	0354	Value of output charater

```
64@ 65A 66B 67C 68D 69E
70F 71G 72H 73I 74J 75K
76L 77M 78N 79O 80P 81Q
82R 83S 84T 85U 86V 87W
88X 89Y 90Z 91[ 92\ 93]
94^ 95_ 32  33! 34" 35#
36$ 37% 38& 39' 40( 41)
42* 43+ 44, 45- 46. 47/
480 491 502 513 524 535
546 557 568 579 58: 59;
60< 61= 62> 63? 96` 97a
98b 99c 100d 101e 102f 103g
104h 105i 106j 107k 108l 109m
110n 111o 112p 113q 114r 115s
116t 117u 118v 119w 120x 121y
122z 123{ 124| 125} 126~ 127
128 129 130 131 132 133
134 135 136 137 138 139
140 141
142 143 144 145
146 147 148 149 150 151
152 153 154 155 156 157
158 159 65A 66B 67C 68D
69E 70F 71G 72H 73I 74J
75K 76L 77M 78N 79O 80P
81Q 82R 83S 84T 85U 86V
87W 88X 89Y 90Z 91[ 92\
93] 94^ 95_ 32  33! 34"
```

```
35# 36$ 37% 38& 39' 40(
41) 42* 43+ 44, 45- 46.
47/ 480 491 502 513 524
535 546 557 568 579 58:
59; 60< 61= 62> 63? 96`
97a 98b 99c 100d 101e 102f
103g 104h 105i 106j 107k 108l
109m 110n 111o 112p 113q 114r
115s 116t 117u 118v 119w 120x
121y 122z 123{ 124| 125} 126~
127 128 129 130 131 132
133 134 135 136 137 138
139 140 141
142 143 144
145 146 147 148 149 150
151 152 153 154 155 156
157 158 159 65A 32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32  32  32  32  32
32  32
```

Figure 2

Figure 3



SCREEN READER

```
START  28846          END    28985

TIME 160338           DATE 081779
```

Figure 4: **Machine Language Listing of Version A Screen Reader. Note that the listing is for a high memory location. Addresses found at the following hex lines must be changed to relocate the program: $70f9, $70fc, $7108, $7113, $7129, $7134, $7137.**

```
28846 70AE  A0 00    LDYIM  00    0
28848 70B0  A2 FF    LDXIM  FF    255
28850 70B2  8E 01 00STX     0001  1
28853 70B5  A2 7F    LDXIM  7F    127
28855 70B7  8E 02 00STX     0002  2
28858 70BA  AE E3 03LDX     03E3  995
28861 70BD  8E E1 03STX     03E1  993
28864 70C0  A2 28    LDXIM  28    40
28866 70C2  EA       NOP
28867 70C3  8E E0 03STX     03E0  992

28870 70C6  18       CLC
28871 70C7  AD 01 00LDA     0001  1
28874 70CA  69 01    ADCIM  01    1
28876 70CC  8D 01 00STA     0001  1
28879 70CF  AD 02 00LDA     0002  2
28882 70D2  69 00    ADCIM  00    0
28884 70D4  8D 02 00STA     0002  2
28887 70D7  18       CLC
28888 70D8  B1 01    LDAIY  01    1
28890 70DA  8D E4 03STA     03E4  996

28893 70DD  69 7F    ADCIM  7F    127
28895 70DF  B0 1F    BCS    1F    31
28897 70E1  AD E4 03LDA     03E4  996
289 1 70E4  69 C0    ADCIM  C0    192
28902 70E6  B0 25    BCS    25    37
28904 70E8  AD E4 03LDA     03E4  996
28907 70EB  69 E1    ADCIM  E1    225
28909 70ED  B0 37    BCS    37    55
28911 70EF  AD E4 03LDA     03E4  996
28914 70F2  69 40    ADCIM  40    64
28916 70F4  EA       NOP
28917 70F5  EA       NOP
28918 70F6  8D 54 03STA     0354  852
28921 70F9  20 3A 71JSR     713A  28988
28924 70FC  4C 18 71JMP     7118  28952
28927 70FF  EA       NOP
28928 7100  EA       NOP
28929 7101  AD E4 03LDA     03E4  936
28932 7104  38       SEC
28933 7105  E9 80    SBC    80    128

28935 7107  18       CLC
28936 7108  4C D9 70JMP     70D9  28890
```

```
28939 710B  EA        NOP
28940 710C  EA        NOP
28941 710D  AD E4 03LDA   03E4   996
28944 7110  18        CLC
28945 7111  69 20     ADCIM  20    32
28947 7113  4C F6 70JMP   70F6   28918
28950 7116  EA        NOP
28951 7117  EA        NOP

28952 7118  CE E0 03DEC   03E0   992
28955 711B  D0 AA     BNE    AA    170
28957 711D  8E E0 03STX   03E0   992
28960 7120  CE E1 03DEC   03E1   993
28963 7123  D0 09     BNE    09    9
28965 7125  60        RTS
28966 7126  AD E4 03LDA   03E4   996
28969 7129  4C F6 70JMP   70F6   28918
28972 712C  EA        NOP
28973 712D  EA        NOP

28974 712E  EA        NOP
28975 712F  A9 0D     LDAIM  0D    13
28977 7131  8D 54 03STA   0354   852
28980 7134  20 3A 71JSR   713A   28986
28983 7137  4C C6 70JMP   70C6   28870
```

Figure 5

```
READY.

  B*
US $ PC   SR AC XR YR SP
.;  USED 30 38 7E 31 FE
.   M 70AE,7137
         0  1  2  3  4  5  6  7
.:  70AE A0 00 A2 FF 8E 01 00 A2
.:  70B6 7F 8E 02 00 AE E3 03 8E
.:  70BE E1 03 A2 28 EA 8E E0 03
.:  70C6 18 AD 01 00 69 01 8D 01
.:  70CE 00 AD 02 00 69 00 8D 02
.:  70D6 00 18 B1 01 8D E4 03 69
.:  70DE 7F B0 1F AD E4 03 69 C0
.:  70E6 B0 25 AD E4 03 69 E1 B0
.:  70EE 37 AD E4 03 69 40 F4 EA
.:  70F6 8D 54 03 20 3A 71 4C
.:  70FE 71 EA EA AD E4 03 38
.:  7106 B0 18 4C DA 70 EA EA AD
.:  710E E4 03 18 69 20 4C F6 70
.:  7116 EA EA CE E0 03 D0 A9 8E
.:  711E E0 03 CE E1 03 D0 09 60
.:  7126 AD E4 03 4C F6 70 EA EA
.:  712E EA A9 0D 8D 54 03 20 3A
.:  7136 71 4C C6 70 A9 FF 8D 43
.   X
>S
```

```
28995 7143  A9 FE     LDAIM  FE    254

            PRINTER DRIVER

START  28986           END    29051

TIME 160614            DATE 09/17/79
```

Figure 6

```
28986 713A  A9 FF     LDAIM  FF    255
28988 713C  8D 43 E8STA   E843   5945
28991 713F  AD 4C E8LDA   E84C   59468
28994 7142  48        PHA
28995 7143  A9 FE     LDAIM  FE    254
28997 7145  8D 4C E8STA   E84C   5946
29000 7148  AD 4B E8LDA   E84B   59467
29003 714B  29 E3     ANDIM  E3    227
29005 714D  8D 4B E8STA   E84B   59467
29008 7150  EA        NOP

29009 7151  EA        NOP
29010 7152  78        SEI
29011 7153  AD 54 03LDA   0354   852
29014 7156  8D 41 E8STA   E841   59457
29017 7159  AD 4C E8LDA   E84C   5946
29020 715C  29 1F     ANDIM  1F    31
29022 715E  09 C0     ORAIM  C0    192
29024 7160  8D 4C E8STA   E84C   59468
29027 7163  EA        NOP
29028 7164  EA        NOP

29029 7165  EA        NOP
29030 7166  EA        NOP
29031 7167  AD 4C E8LDA   E84C   59468
29034 716A  09 E0     ORAIM  E0    224
29036 716C  8D 4C E8STA   E84C   59468
29039 716F  EA        NOP
29040 7170  18        CLC
29041 7171  EA        NOP
29042 7172  AD 4D E8LDA   E84D   59469
29045 7175  29 02     ANDIM  02    2

29047 7177  F0 F9     BEQ    F9    249
29049 7179  68        PLA
29050 717A  58        CLI
29051 717B  60        RTS
29052 717C  EA        NOP
29053 717D  EA        NOP
29054 717E  EA        NOP
29055 717F  00        BRK
29056 7180  00        BRK
```

```
READY.
RUN

C*    PC   SR AC XR YR SP
.;   C5ED 30 38 7E 31 FE
.    M 713A,7180
          0  1  2  3  4  5  6  7
FSUSD MONITOR
LOADING
READY.
RUN

C*    PC   SR AC XR YR SP
.;   C5ED 30 38 7E 31 FE
.    M 713A,7180
          0  1  2  3  4  5  6  7
.:   713A A9 FF 8D 43 E8 AD 4C E8
.:   7142 48 A9 FE 8D 4C E8 AD 4B
.:   714A E8 29 E3 8D 4B E8 EA EA
.:   7152 78 AD 54 03 85 41 E8 AD
.:   715A 4C E8 29 1F 09 C0 8D 4C
.:   7162 E8 EA EA EA EA AD 4C E8
.:   716A 09 E0 8D 4C E8 EA 18 EA
.:   7172 AD 4D E8 29 02 F0 F9 68
.:   717A 58 60 EA EA EA 00 00 24
.    X
READY.
>S
```

Figure 7

DATA CAPTURE 3.0

Is DATA CAPTURE just another smart terminal program for your Apple II(TM) or Apple II Plus(TM)?

NO. It is a GENIUS TERMINAL PROGRAM and is designed to be used with the Micromodem II(TM).

Tired of watching data and programs scroll off the screen forever? Then DATA CAPTURE is the program for you.

- ANYTHING that appears on the screen of your Apple II can be captured. Any program or data.
- You can then save what you have captured to disk, dump it to your printer or even do simple editing with DATA CAPTURE.
- You can use DATA CAPTURE to compose text offline for later transmission to another computer. Think of the timeshare charges this will save you.
- Use DATA CAPTURE with the Dan Paymar Lower Case Adapter and you can enter lower case from the keyboard for transmission to another computer or capture both upper and lower case.
- A program is also included to convert your programs to text files for transmission using DATA CAPTURE.
- You receive two versions of the program. One is for upper case only and one for both upper and lower case use with the above adapter.

DATA CAPTURE will save you money if you are using a timesharing system because you can compose messages offline for later transmission. You can also quickly capture data for later reading, printing or editing. Requires DISK II(TM), APPLESOFT II(TM).

Price $29.95

If your local dealer does not have DATA CAPTURE then order directly. We ship DATA CAPTURE within 3 working days of receipt of order and welcome your personal check. We also accept Visa and Master Charge. Add $49.95 if you would also like to order the Dan Paymar Lower Case Adapter at the same time.

Ask for a catalog of our software.

* Apple II, Apple II Plus, Disk II and APPLESOFT II are trademarks of Apple Computer Company.

* Micromodem II is a trademark of D.C. Hayes Associates, Inc.

SOUTHEASTERN SOFTWARE
7270 Culpepper Drive
New Orleans, LA 70126

504/246-8438   504/246-7937

Symbol-Table Sorter/Printer for the AIM Assembler

by Mel Evans

[MICRO 20:43]

"After more extended use of the program,I have found the following pair of bugs.

The first can be cured by replacing the code shown in Figure 1A (occurring at the end of subroutine SORT) by that in Figure 1B. The old code works often, but not always. The new code always works.

The second bug won't show until you start getting fancy with your source code. I was mistaken in thinking that memory locations 003C, 003D contain the address of the last symbol found during assembly. Instead, they contain the address of the last *active* symbol. With straightforward code, these will be one and the same. But suppose you have written your last subroutine (let's call it SUBZ) and then decide to initialize a couple of zero-page addresses (starting at ZP1) as in Figure 2A. After assembly, the last symbol will be SUBZ, but the last *active* symbol with be ZP1. And with this stored in 003C, 003D, you will get a very short listing!

The problem could be solved by re-writing the program to avoid using 003C, 003D. But, ther's a simpler solution, as shown in Figure 2B. Add a new symbol, LAST, as the last byte of the program. (This is a good practice anyway. After assembly, the address of LAST tells you precisely how much memory the program needs.) The, after initialization and any other housekeeping, add the line "* = LAST". This makes "last active" equal "LAST", and the listing comes out complete."

Submitted by: Mel Evans
ERIM, P.O. Box 8618
Ann Arbor, MI 48107

```
JSR CRCK
JSR INCADR
BMI PRNT1
BEQ PRNT1
JSR GAP
RTS
```

Figure 1A: Old SORT Code

```
JSR CRCK
TXA
BNE FIN
JSR INCADR
BNE PRNT1
DEX
BNE PRNT1
FIN JSR GAP
RTS
```

Figure 1B: New SORT Code

```
SUBZ
  .
  .
RTS
;
*=ZP1
.DBY $0A0B
;
.END
```

Figure 2A: Wrong "Last Active"

```
SUBZ
  .
  .
LAST RTS
;
*=ZP1
.DBY $0A0B
;
*=LAST
;
.END
```

Figure 2B: Right "Last Active"

Microbes

and

Updates

Expanding the SYM - 1 ... Adding an ASCII Keyboard

by Robert A. Peck

[MICRO 21:5]

"As we discussed, here is a corrected version of my progrm listing. Somehow the hex locations column of this listing was not used for the article. [Sorry about that - MICRO] Typos corrected on final version including label "DISP" change to WAIT2 at location 206 (minor), incorrect object code fixed at line 222 to **20 47 8A** .... Last was pointer to KSTAT at line 240 which should be **39**."

Submitted by: Robert A. Peck
P.O. Box 2231
Sunnyvale, CA 94087

```
0200  20 88 81  GKEY   JSR  SAVER    SAVE REGISTERS
0203  AD 01 A8         LDA  A801     GET PARALLEL ASCII
0206  F0 24            BEQ  WAIT2    UNLESS NONE,THEN BRANCH
0208  85 F1            STA  00F1     STORE IT A WHILE
020A  A9 10            LDA  #$10     DEBOUNCE CONSTANT
020C  85 EF            STA  00EF     DEBOUNCE
020E  C6 F0     WAIT1  DEC  00F0     SMALL LOOP
0210  D0 FC            BNE  WAIT1
0212  C6 EF            DEC  00EF     LARGE LOOP
0214  D0 F8            BNE  WAIT1
0216  20 03 89  SCANA  JSR  IJSCNV   SCAN DISPLAY(USE SCANVEC)
0219  2C 01 A8         BIT  A801     IS KEY STILL DOWN?
021C  30 F8            BMI  SCANA    WAIT FOR KEY RELEASE
021E  A5 F1            LDA  00F1     KEY UP, PROCESS KEY
0220  29 7F            AND  #$7F     STRIP KEY STROBE BIT
0222  20 47 8A         JSR  OUTCHR   SEND INTO DISBUF
0225  A5 F1            LDA  00F1     GET IT AGAIN
0227  29 7F            AND  #$7F     STRIP IT AGAIN
022A  4C B8 81         JMP  RESXAF   RETURN WITH ASCII IN A
022C  A9 10     WAIT2  LDA  #$10     IF NO KEY,
022E  85 EF            STA  00EF     SCAN DISPLAY
0230  20 03 89  SCANB  JSR  IJSCNV   THRU SCANVEC
0233  C6 EF            DEC  00EF     A NUMBER OF TIMES
0235  D0 F9            BNE  SCANB    THEN GO BACK
0237  F0 CA            BEQ  GKEY     AND LOOK AGAIN
0239  AD 01 A8  KSTAT  LDA  A801     READ ASCII INPORT
023C  0A               ASLA          SHIFT MSB INTO CARRY
023D  60               RTS           RET, CFLAG=1 IF KEY DN.

0240  20 86 8B  INIT   JSR  ACCESS   UNPROTECT SYSRAM
0243  A9 00            LDA  #00      MODIFY
0245  8D 61 A6         STA  A661     KEYBOARD
0248  A9 02            LDA  #02      INPUT
024A  8D 62 A6         STA  A662     VECTOR
024D  A9 39            LDA  #$39
024F  8D 67 A6         STA  A667     KEYPRESS
0252  A9 02            LDA  #02      STATUS
0254  8D 68 A6         STA  A668     VECTOR
0257  4C 03 80         JMP  WARM     WARM ENTRY,MONITOR
```

Graphics and the Challenger C1P, Part 5

This final installment in the series discusses plotting techniques and moving characters.

William L. Taylor
246 Flora Road
Leavittsburg, OH 44430

The ability to have characters and have them move in our game programs is a must. How do we accomplish this task? It is a simple task to implement. We do it with a technique called plotting.

C1P Plotting Technique

In order to have any character move on our C1P's Monitor screen, we must first know where we wish our character to start, the angular directions in which it is to move, and where the character's movement will end. If you will examine the example of the plotting diagram in Figure 1, you can see the angular directions in which the character can be made to move on the monitor screen. These angular directions are relative to any point on the screen, i.e., relative to a certain position on the Video RAM. If, for example, the starting location were 54000 decimal, the zero point would be 54000 decimal and all movement would be relative to that point.

As stated in an earlier part of this series, we can cause a character to be placed on the screen of our C1P with a BASIC POKE statement. We move the character that has been placed on the screen with a BASIC FOR/NEXT loop. In the explanation of plotting and how to develope animated characters we will use the functions of BASIC to develope our programs and to describe animation methods.

To begin our explanation, let's use decimal location 54000 as an example again as a starting point. A BASIC program would use this decimal location as a variable content. For example, 10 A = 54000. Now that we have a starting point, we can move the graphics character in any direction shown in the diagram in Figure 1. For example, if you wish the character to move in a vertical direction, with a BASIC subroutine we can get the character displayed and moved. In order to explain how this proceedure works, please refer to the BASIC program subroutines in Listing 1, along with Figure 1.

If we wish to have an animated character (one that moves) we must first know the start, end, and path of the character, as stated before. The character must be made to appear at a point along the path of angular movement. The character is then displayed for some duration of time. Next it is erased from its present position and then displayed at a new position on the monitor screen. This process must be continued for the desired distance along the plotted path that we have chosen. These criteria can be executed with BASIC or Machine Language porgrams. Since we are primarily programming in BASIC, we will develop some BASIC routines to show how the character can be produced and moved on the C1P's monitor screen.

The BASIC routine in section one of Listing 1 will be used to generate an animated character that will primarily move from near center screen downward to near the bottom of the screen. This subroutine begins at BASIC progran line 5. Here the REM statement tells the user that this is a routine to generate the movement of a character downward. Line 10 is the real beginning of the subroutine. At line 10 the A variable is loaded with the decimal beginning of the memory location where the character will first be displayed. Notice that this line forms part of a FOR/NEXT loop. Also notice that this loop will be incremented by a total of 32 counts for every pass through the program. This is done with the STEP function of BASIC. The FOR/NEXT loop at line 10 actually sets the limits of movement of the character. These limits are in the

For an angular movement in any direction,use the value in the chart to cause movement in that direction. It must be understood that the decimal beginning and ending must be caluclated because for each pass through the loop with a step function, the variable will be incremented by the amount in the step value.

Study the remainder of the modules in Listing 1, from our discussion you should be able to see just how these subroutines work. Load the programs into your C1P and watch the action on the screen. This will show you the results. The diagram in Figure 2 gives the complete memory map for a C1P. This is for a 25 by 25 character format. Use this diagram for all your plotting to find any location on the C1P monitor screen.

Now that we have seen some examples of how a moving character is made to move on the C1P's screen, let's use some of these techniques to develop a program that has some moving character elements that form a game. Listing 2 shows a game program that uses moving elements. These are: a starship and lasar cannon shots directed at the starship. All the techniques that we have discussed, that give the sensation of motion, are used in Listing 2. Please refer to this listing as we discuss the inner workings of the program's operation.

The program is presented, as I have said, as a game. The starship moves across mid-screen and the cannons are placed at each bottom corner, and at mid-bottom of the screen. The keyboard keys 5,6, and 7 are used to fire the cannons. A hit score total is printed out at the top of the monitor screen for the player.

This program is straight-forward and each module is identified by REM statements. This discussion will deal mainly with graphics and the keyboard routines, so please continue to refer to Listing 2. The remainder of the program should be self-explanatory.

The program from line 300 through 347 forms the main line BASIC module. It is used to draw and move the starship across the screen. The polling routine for the keyboard is located from line 335 to line 344. If a 5,6, or 7 key is pressed, a GOSUB to a cannon shot routine will result in a shot at the starship. Key 5 causes a shot from bottom right upward diagonally to top left of the screen. A 7 key results in a shot from bottom left to top right. A 6 key results in a true vertical shot.

The position of the starship is always contained in the K variable. This location is always checked in each shot routine at lines 415, 462, and 525. If a hit occurs, the program jumps to line 600 where an explosion of the starship will be displayed at the screen location contained in variable K. Next a hit score will be placed on the screen. The hit count will be checked for 10 hits. If so the player will be informed that he has completed the exact number of hits and has won the game. If the player has less than 10 hits, the program returns through RETURNs to the exact main-line program at line 300.

This program uses more of the elements contained in the Character Generator ROM. These elements are the elements that are used to draw the starship. Their decimal equivalents are 9 and 12, and are written into video memory with the POKE statement at line 310. After a delay at line 320, the starship is erased and placed at the next location in the FRO/NEXT loop from line 300. The cannon shots are primarily POKEd to screen memory, displayed for some duration of time and then erased. This process continues until the FOR/NEXT loop has been incremented to its maximum value.

Conclusion

If you have followed all five parts of this series, I believe that you should now have sufficient knowledge of your C1P's graphics capabilities. I hope that you now also have a better understanding of the polled keyboard, and how to use these capabilities with BASIC programming to produce real working programs that will be enjoyable to use. Hopefully you have learned with me through these efforts and I will see some of your programs published in the pages of MICRO in the near future. With that, I will conclude this series of articles and I hope that these programs and ideas will be as much fun for you as you read and experiment, as they have been for me in the writing. Good luck with your programming and with your writing.

```
SECTION 1)
5 REM MOVE CHARACTER DOWN
10 FOR A = 53776 TO 54160 STEP 32
20 POKE A, 161
30 FOR B = 1 TO 50 : NEXT B
40 POKE A , 32
50 NEXT A

SECTION 2)
60 REM MOVE CHARACTER UP
70 FOR A = 54160 TO 53763 STEP- 32
80 POKE A, 161
90 FOR B = 1 TO 50 : NEXT B
100 POKE A , 32
110 NEXT A

SECTION 3 )
120 REM MOVE CHARACTER RIGHT
130 FOR A = 53776 TO 53787
140 POKE A , 161
150 FOR B = 1 TO 50 : NEXT B
160 POKE B , 32
170 NEXT A

SECTION 4 )
180 REM MOVE CHARACTER LEFT
190 FOR A = 53776 TO 53763 STEP -1
200 POKE A , 161
210 FOR B = 1 TO 50 : NEXT B
220 POKE A , 32
230 NEXT A
```

List 1

Photographs for this series were provided by William L. Taylor, Jr.

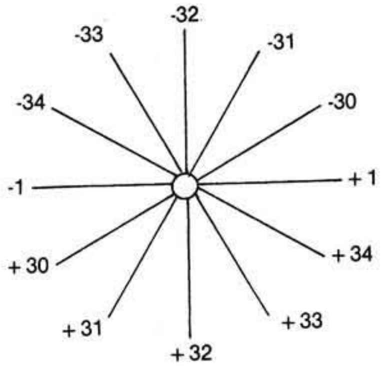
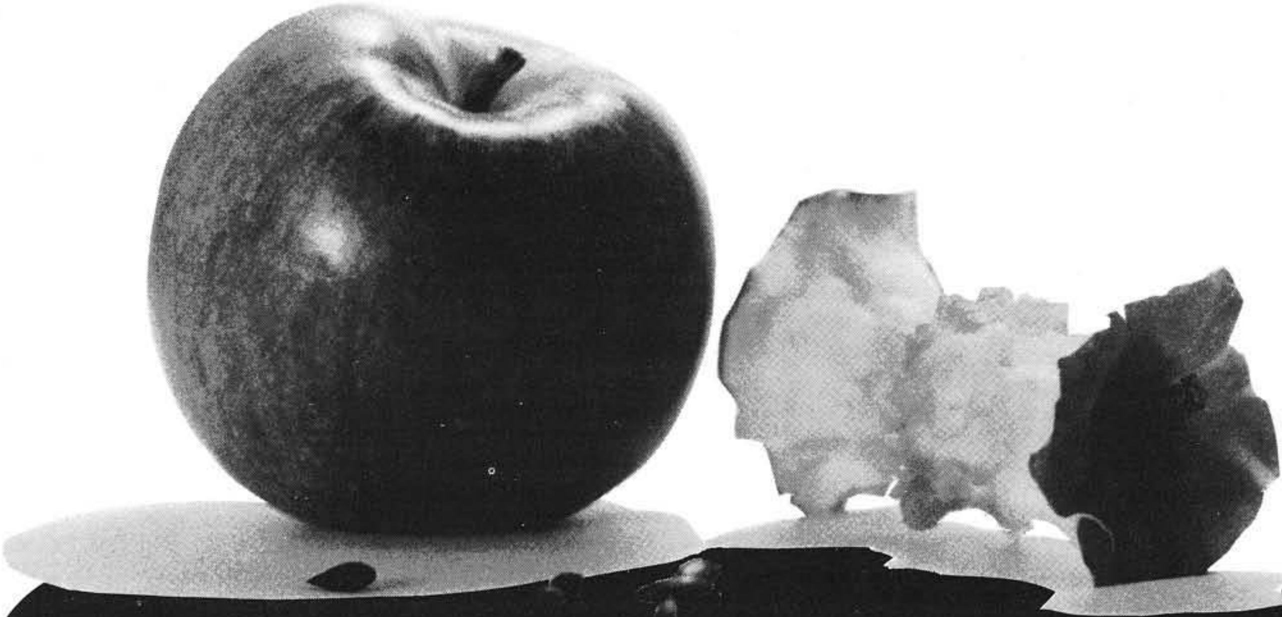


Figure 1: **Plotting Directions for the C1P**

```
LIST1-500

 1 REM DEMONSTRATION PROGRAM FOR ANIMATED ELEMENTS ON
 2 REM BY WILLIAM L. TAYLOR 12/3/1979          OSI C1P
 3 PRINT"STAR SHIP ATTACK"
 4 PRINT:PRINT" DESTROY THE STARSHIPS WITH KEYS 5,6,7"
 5 PRINT" YOU GET 10 SHOTS":PRINT
 8 FOR S=1 TO 10000:NEXT S
 10 L=2:GOTO350
 299 REM DRAW STARSHIP AND KEYBOARD POLLING ROUTINE
 300 FOR K=53763 TO 53787
 310 POKE K,9:POKE K+1,12
 320 FOR J=1 TO 50:NEXT J
 330 POKE K,32:POKE K+1,32
 335 POKE 530,1:POKE 57088,127
 337 C=C+1
 342 IF PEEK(57088)=253 THEN GOSUB 400
 343 IF PEEK(57088)=251 THEN GOSUB 500
 344 IF PEEK(57088)=247 THEN GOSUB450
 345 NEXT K
 347 GOTO 300
 349 REM CLEAR SCREEN
 350 FOR T=-3 TO 32:PRINT:NEXT T
 360 GOTO 300
 399 REM DRAW RIGHT VERTICAL SHOT
 400 FOR T1=54147 TO 53403 STEP -31
 410 POKE T1,249
 415 IF T1=K THEN GOSUB 600
 420 FOR T2=1 TO 5:NEXT T2
 425 POKE T1,32
 430 NEXT T1
 438 RETURN
 449 REM DRAW LEFT VERTICAL SHOT
 450 FOR T3=54171 TO 53379 STEP -33
 460 POKE T3,255
 462 IF T3=K THEN GOSUB 600
 463 FOR T4=1 TO 5:NEXT T4
 464 POKE T3,32
 465 NEXT T3
 470 RETURN
 475 POKE T4,32
 499 REM DRAW VERTICAL SHOT
 500 FOR T5=54158 TO 53390 STEP-32

OK

LIST4_500-800

 500 FOR T5=54158 TO 53390 STEP-32
 520 POKE T5,248
 525 IF T5=K THEN GOSUB 600
 530 FOR T7=1 TO 5:NEXT T7
 540 POKE T5,32
 550 NEXT T5
 580 C=0:RETURN
 599 REM CHECK SHOT HIT DRAW EXPLOSION AND DISPLAY HITS
 600 E=0:U=53731
 610 POKE U+E,32
 620 FOR A=1 TO 10:POKE K,A:NEXT A
 630 E=E+1:IF E<20 THEN 610
 640 IF E=20 THEN POKE 53455,72:POKE 53456,73:POKE 53457,84
 650 POKE 53458,32:POKE 53459,L+47
 660 L=L+1
 662 IF L<10 THEN RETURN
 665 IF L=10 THEN PRINT" ALL TARGETS DESTROYED"
 670 PRINT" YOU HAVE SAVED THE UNIVERSE"
 680 PRINT" WANT TO PLAY AGAIN  YES OR NO"
 690 INPUT A1$
 700 IF A1$="YES" THEN 1
 710 IF A1$="NO"THEN END
OK
```



Lower Case and Punctuation in APPLESOFT

Do you need to get lower case and punctuation into your BASIC strings? Then, try these programs.

James D. Childress
5108 Springlake Way
Baltimore, MD 21212

Introduction

While computer people may adapt to all caps, the general public still uses, and apparently likes, lower case. Printing with lower case is more familiar, more readable and more acceptable. Thus, we who work with computers should provide lower case in any printout that we expect or hope laymen to read. After all, computers should adapt to people; people should not have to adapt to computers.

Also, who is there among us who hasn't wondered at how the APPLE handles punctuations in strings? In INPUT's, we have found to our dismay that a "JONES, JOHN" results in an error message saying "?EXTRA IGNORED" and later finding the string variable as only "JONES" with nothing to tell us which Jones that may be. What wouldn't we give to get quotation marks and commas in the places we want?

So much for what should be or what we want. The APPLE doesn't have lower case and seems rather whimsical about punctuation. Well, face it; there were a number of compromises made in the design of the APPLE and Applesoft. Of course, some of these deficiencies can be conquered by money. We can buy one of the lower-case boards and live more or less happily ever after. Unfortunately, we do not all or always have the option of buying a solution to a problem; most of us have more problems than money. And there are not always solutions for sale.

An alternative approach is an Applesoft program to produce the desired lower case and punctuation. I have looked for such a program and I found two possibilities (there likely are others but I am not acquainted with them):

1. Val J. Golding in "Lower Case Routine for Integral Data Printer," *Call-Apple*, v.2, p. 11 (April/May 1979) gave a program to poke lower case characters into strings in the string array memory space.

2. Another program was published in *Contact*, v.1, p.5 (May 1978); this program pokes lower case into the beginning of program memory space.

Both of these are quite limited. Note: both should work for punctuation problems within the same limitations.

Neither of these enables one to enter lower case or problem punctuations conveniently into string variables, nor to print statement strings in an Applesoft program as desired. The program given in the listing in Figure 1 does the job for string variables and the one given in Figure 2, for strings in print statements.

Use and Operation

The heart of these programs is the same as in the cited programs: use of the GET command to sneak things around the interpreter. The GET command handles input character by character so that each can be manipulated.

(The identical GET routine is used for both programs—lines 63010 to 63120 in the first and lines 63140 to 63150 in the second. Only one typing needs be done, a hint not to be ignored.)

The first program is intended for use as a subroutine. For example, a statement such as

30 INPUT "ACCOUNT NAME";NAME$(1)
can be replaced directly by

30 PRINT "ACCOUNT NAME";: GOSUB 63000:NAME$(1) = BB$

In a run, the program would appear to behave normally except that there would be no ?EXTRA IGNORED's and NAME(1) would look quite strange on the CRT monitor (",/7%2 #!3%" for "lower case") and as lower case only on the printer.

In both programs, capitals are entered in a manner similar to the operation of MUSE's word processor program Dr. Memory. A ctrl-A makes the next letter only captial; an ctrl-C makes all the following letters capital until either a ctrl-S or the end of the string. Unlike Dr. Memory, the control characters are not displayed. Instead, the capitalized letters are shown in inverse video. I like this way of doing things. If you would prefer the opposite video, just interchange the words NORMAL and INVERSE in lines 63020-63040 and 63080 and add an INVERSE to line 63000 in Figure 1. You could do even more to tailor to your personal tastes; change the control characters, change the default operation from lower case to capitals, etc. These custom fittings are left as an exercise.

Another feature common to both programs is the motion of the cursor. The backspace works but that is all. And it will move the cursor back no further than the initail position. However, therein lurks a minor nuisance; if you try to backspace beyond that limit, the immediately preceeding character will be wiped out or replaced by a white block. This is of no consequence; ignore it.

Since the string variables subroutine runs as a part of your program, you have to keep labels straight. This subroutine uses only AA$, AZ$, BB$, BB, BZ$, and ZZ and has no FOR loops. Also note that only the usual limitation applies for the length of strings.

In the use of the second program, you append it to the program in which you want to put lower case. A RUN 63000 initiates things; you simply give the line number in which lower case is wanted. The first string in that line is printed, terminated by ## to indicate the length limit. The cursor below this line indicates the place for the change. You can insert anything but we assume that a mixed capital and lower case rendition of the line above is what you will want. In any case, the length cannot be exceeded. If you go over the limit, the excess will be ignored. If you put in less, the remainder will be filled with spaces. If you don't want to change that particular string, simply hit RETURN.

Figure 1

```
63000 BB$ = "":BZ$ = "":BB = 0:ZZ
       = 0
63010  GET AA$:AZ$ = AA$: IF  ASC
      (AA$) = 13 THEN  NORMAL : GOTO
      63130
63020  IF  ASC (AA$) = 1 THEN ZZ =
      1: INVERSE :BB = 0: GOTO 630
      10
63030  IF  ASC (AA$) = 3 THEN BB =
      1: INVERSE : GOTO 63010
63040  IF  ASC (AA$) = 19 THEN BB
       = 0: NORMAL : GOTO 63010
63050  IF ZZ = 1 OR BB = 1 THEN Z
      Z = 0: GOTO 63080
63060  IF  ASC (AA$) < 65 OR  ASC
      (AA$) > 90 THEN 63080
63070 AA$ =  CHR$ ( ASC (AA$) + 3
      2)
63080 BZ$ = BZ$ + AZ$: PRINT AZ$;
      : IF BB = 0 THEN  NORMAL
63090 BB$ = BB$ + AA$: IF  ASC (B
      B$) = 8 AND  ASC (AA$) = 8 THEN
       PRINT " ";
63100  IF  LEN (BB$) <  = 2 AND  ASC
      (AA$) = 8 THEN BB$ = "":BZ$ =
      "": GOTO 63010
63110  IF  ASC (AA$) = 8 THEN BB$
       =  LEFT$ (BB$, LEN (BB$) -
      2)
63120  GOTO 63010
63130  PRINT : RETURN
63140  END
```

Figure 2

```
62999  END
63000  HOME : VTAB (3): PRINT "LO
      WER CASE INSERTION PROGRAM":
       PRINT : PRINT
63010 LMAX = 62999: PRINT "NUMBER
       OF FIRST LINE TO BE RE-": INPUT
      "WRITTEN  ";LT: PRINT
63020  PRINT :M = 256 *  PEEK (10
      4) +  PEEK (103) + 2
63030 LN = 256 *  PEEK (M + 1) +
       PEEK (M): IF LN >  = LMAX OR
      LN > LT THEN 63320
63040  IF LN <  > LT THEN M = 256
       *  PEEK (M - 1) +  PEEK (M -
      2) + 2: GOTO 63030
63050 K = 0:LL = 0:UL = 0
63060  FOR J = M + 2 TO M + 255:T
      ST =  PEEK (J): IF TST = 0 THEN
      M = J + 3: GOTO 63030
63070  IF TST = 58 THEN K = 0
63080  IF TST = 186 OR TST = 132 THEN
      K = 1
63090  IF K = 1 AND LL > 0 AND TS
      T = 34 THEN UL = J - 1: GOTO
      63120
63100  IF K = 1 AND LL = 0 AND TS
      T = 34 THEN LL = J + 1
63110  NEXT
63120 BB$ = "":BZ$ = "":BB = 0:ZZ
       = 0
63130  FOR I = LL TO UL: PRINT  CHR$
      ( PEEK (I));: NEXT : PRINT "
      ##"
63140  GET AA$:AZ$ = AA$: IF  ASC
      (AA$) = 13 THEN  NORMAL : GOTO
      63260
63150  IF  ASC (AA$) = 1 THEN ZZ =
      1: INVERSE :BB = 0: GOTO 631
      40
63160  IF  ASC (AA$) = 3 THEN BB =
      1: INVERSE : GOTO 63140
63170  IF  ASC (AA$) = 19 THEN BB
       = 0: NORMAL : GOTO 63140
63180  IF ZZ = 1 OR BB = 1 THEN Z
      Z = 0: GOTO 63210
63190  IF  ASC (AA$) < 65 OR  ASC
      (AA$) > 90 THEN 63210
63200 AA$ =  CHR$ ( ASC (AA$) + 3
      2)
63210 BZ$ = BZ$ + AZ$: PRINT AZ$;
      : IF BB = 0 THEN  NORMAL
63220 BB$ = BB$ + AA$: IF  ASC (B
      B$) = 8 AND  ASC (AA$) = 8 THEN
       PRINT " ";
63230  IF  LEN (BB$) <  = 2 AND  ASC
      (AA$) = 8 THEN BB$ = "":BZ$ =
      "": GOTO 63140
63240  IF  ASC (AA$) = 8 THEN BB$
       =  LEFT$ (BB$, LEN (BB$) -
      2)
63250  GOTO 63140
63260  IF BB$ = "" THEN 63310
63270  PRINT : FOR I = LL TO UL
63280 DD$ =  MID$ (BB$,I - LL + 1
      ,1):MM =  ASC (DD$)
63290  POKE I,MM
63300  NEXT
63310 UL = 0:LL = 0: PRINT : GOTO
      63110
63320  PRINT : PRINT "NUMBER OF N
      EXT LINE TO BE REWRITTEN": INPUT
      "(ENTER 0 TO END PROGRAM)  "
      ;LT
63330  IF LT = 0 THEN  END
63340  GOTO 63020
```

After a RETURN, the next string in the same line will appear, ready to be changed. When all the strings of that one line have been dealt with, you are asked for the number of the next line.

As mentioned above, lower case if displayed by the APPLE ad keyboard symbols other than letters. These print properly as lower case on a printer that prints lowercase. If you want to display, say, a table so that you can check data prior to printing, you need to program the display table and the printout table seperately. For convenience in doing this, both programs provide an all-caps string BZ$ ad well as the corresponding string BB$ with lowercase.

Program Design

The GET routine, essentially the whole of Figure 1, has already been mentioned. The GET command is follwed by a series of IF's to implement the control character,backspace and RETURN functions. These are straight-forward and self-explanatory.

The second program, Figure 2, consists of three parts. The first, lines 63020-63300, pokes the new string into the program into the program memory space.

Concluding Remarks

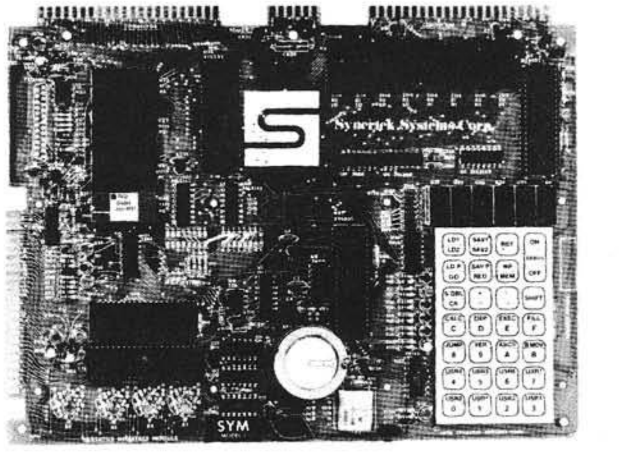
Although written for Applesoft, these programs can be adapted to other BASIC's.The first presents no problems. However, the program memory space search routine in the second will require modification for other computers. This modification should not be too difficult to implement for other Microsoft BASIC' s.

SYM-1, 6502-BASED MICROCOMPUTER

- FULLY-ASSEMBLED AND COMPLETELY INTEGRATED SYSTEM that's ready-to-use
- ALL LSI IC'S ARE IN SOCKETS
- 28 DOUBLE-FUNCTION KEYPAD INCLUDING UP TO 24 "SPECIAL" FUNCTIONS
- EASY-TO-VIEW 6-DIGIT HEX LED DISPLAY
- KIM-1* HARDWARE COMPATIBILITY

The powerful 6502 8-Bit MICROPROCESSOR whose advanced architectural features have made it one of the largest selling "micros" on the market today.

- THREE ON-BOARD PROGRAMMABLE INTERVAL TIMERS available to the user, expandable to five on-board.
- 4K BYTE ROM RESIDENT MONITOR and Operating Programs.
- Single 5 Volt power supply is all that is required.
- 1K BYTES OF 2114 STATIC RAM onboard with sockets provided for immediate expansion to 4K bytes onboard, with total memory expansion to 65, 536 bytes.
- USER PROM/ROM: The system is equipped with 3 PROM/ROM expansion sockets for 2316/2332 ROMs or 2716 EPROMs
- ENHANCED SOFTWARE with simplified user interface
- STANDARD INTERFACES INCLUDE:
  - —Audio Cassette Recorder Interface with Remote Control (Two modes: 135 Baud KIM-1* compatible, Hi-Speed 1500 Baud)
  - —Full duplex 20mA Teletype Interface
  - —System Expansion Bus Interface
  - —TV Controller Board Interface
  - —CRT Compatible Interface (RS-232)
- APPLICATION PORT: 15 Bi-directional TTL Lines for user applications with expansion capability for added lines
- EXPANSION PORT FOR ADD-ON MODULES (51 I/O Lines included in the basic system)
- SEPARATE POWER SUPPLY connector for easy disconnect of the d-c power
- AUDIBLE RESPONSE KEYPAD



Synertek has enhanced KIM-1* software as well as the hardware. The software has simplified the user interface. The basic SYM-1 system is programmed in machine language. Monitor status is easily accessible, and the monitor gives the keypad user the same full functional capability of the TTY user. The SYM-1 has everything the KIM-1* has to offer, plus so much more that we cannot begin to tell you here. So, if you want to know more, the SYM-1 User Manual is available, separately.

SYM-1 Complete w/manuals	**$229.00**
SYM-1 User Manual Only	**7.00**
SYM-1 Expansion Kit	**60.00**

Expansion includes 3K of 2114 RAM chips and 1-6522 I/O chip.

SYM-1 Manuals: The well organized documentation package is complete and easy-to-understand.

SYM-1 CAN GROW AS YOU GROW. Its the system to BUILD-ON. Expansion features that are soon to be offered:

***BAS-1 8K Basic ROM (Microsoft)**	**$89.00**
***KTM-2 TV Interface Board**	**319.00**

**We do honor Synertek discount coupons*

QUALITY EXPANSION BOARDS DESIGNED SPECIFICALLY FOR KIM-1, SYM-1 & AIM 65

These boards are set up for use with a regulated power supply such as the one below, but, provisions have been made so that you can add onboard regulators for use with an unregulated power supply. But, because of unreliability, we do not recommend the use of onboard regulators. All I.C.'s are socketed for ease of maintenance. *All boards carry full 90-day warranty.*

All products that we manufacture are designed to meet or exceed industrial standards. All components are first qualtiy and meet full manufacturer's specifications. All this and an extended burn-in is done to reduce the normal percentage of field failures by up to 75%. To you, this means the chance of inconvenience and lost time due to a failure is very rare; but, if it should happen, we guarantee a turn-around time of less than forty-eight hours for repair.

Our money back guarantee: If, for any reason you wish to return any board that you have purchased directly from us within ten (10) days after receipt, complete, in original condition, and in original shipping carton; we will give you a complete credit or refund less a $10.00 restocking charge per board.

VAK-1 8-SLOT MOTHERBOARD

This motherboard uses the KIM-4* bus structure. It provides eight (8) expansion board sockets with rigid card cage. Separate jacks for audio cassette, TTY and power supply are provided. Fully buffered bus.

VAK-1 Motherboard — **$139.00**

VAK-2/4 16K STATIC RAM BOARD

This board using 2114 RAMs is configured in two (2) separately addressable 8K blocks with individual write-protect switches.

VAK-2 16K RAM Board with only 8K of RAM (½ populated)	**$239.00**
VAK-3 Complete set of chips to expand above board to 16K	**$125.00**
VAK-4 Fully populated 16K RAM	**$325.00**

VAK-5 2708 EPROM PROGRAMMER

This board requires a +5 VDC and ±12 VDC, but has a DC to DC multiplyer so there is no need for an additional power supply. All software is resident in on-board ROM, and has a zero-insertion socket.

VAK-5 2708 EPROM Programmer — **$249.00**

VAK-6 EPROM BOARD

This board will hold 8K of 2708 or 2758, or 16K of 2716 or 2516 EPROMs. *EPROMs not included.*

VAK-6 EPROM Board — **$119.00**

VAK-7 COMPLETE FLOPPY-DISK SYSTEM (See March Issue of Micro)

VAK-8 PROTYPING BOARD

This board allows you to create your own interfaces to plug into the motherboard. Etched circuitry is provided for regulators, address and data bus drivers; with a large area for either wire-wrapped or soldered IC circuitry.

VAK-8 Protyping Board — **$39.00**

POWER SUPPLIES

ALL POWER SUPPLIES are totally enclosed with grounded enclosures for safety, AC power cord, and carry a full 2-year warranty.

FULL SYSTEM POWER SUPPLY

This power supply will handle a microcomputer and up to 65K of our VAK-4 RAM. ADDITIONAL FEATURES ARE: Over voltage Protection on 5 volts, fused, AC on/off switch. Equivalent to units selling for $225.00 or more.

Provides +5 VDC @ 10 Amps & ±12 VDC @ 1 Amp

VAK-EPS Power Supply — **$119.00**

KIM-1* Custom P.S. provides 5 VDC @ 1.2 Amps and +12 VDC @ .1 Amps

KCP-1 Power Supply — **$39.00**

SYM-1 Custom P.S. provides 5 VDC @ 1.4 Amps

VCP-1 Power Supply — **$39.00**

*KIM is a product of MOS Technology

RNB ENTERPRISES INCORPORATED

2967 W. Fairmount Avenue
Phoenix AZ. 85017
(602)265-7564

master charge · VISA

MICRO™

PO Box 6502
Chelmsford, Mass 01824

617-256-5515

"The BEST of MICRO Volume 1" contains all of the important material from the first six issues of **MICRO** in book form.

"The BEST of MICRO Volume 2" contains all of the important material from the second six issues [#7 to 12] of **MICRO** in book form.

"ALL of MICRO Volume 2" is all six issues of Volume 2, issues 7 to 12, at a special reduced price for a limited time while supplies last.

Back Issues:

Issues 7 to 12: ....

Issues 13 on: ....

All payments must be in US dollars.
Make checks payable to: MICRO
Foreign payments in International Money Order or cash.

Subscription: One Year = 12 issues. Circle correct category and write amount in space provided.

Surface:

United States	$15.00	
All Other Countries	$18.00	
Air Mail:		
Central America	$27.00	
Europe/So. America	$33.00	
All Other Countries	$39.00	$ ....
"BEST of MICRO Volume 1"		
Surface	$7.00	
Air Mail	$10.00	$ ....
"BEST of MICRO Volume 2"		
Surface	$9.00	
Air Mail	$13.00	$ ....
"ALL of MICRO Volume 2"		
Surface	$9.00	
Air Mail	$13.00	$ ....

No.	Surface @ $1.75 each Air Mail @$2.75 each	=	$ ....
No.	Surface @ $2.25 each Air Mail @ $3.25 each	=	$ ....
TOTAL			$ ....

If you are a subscriber, attach label or write subscription number here: ....

Name: ....

Address: ....

....

City: .... State: .... Zip: ....

Country (if not U.S.): ....

Help **MICRO** bring you the info you want by completing this short questionnaire.

Microcomputers Owned/Planning to Buy: AIM SYM KIM PET APPLE OSI Other: ....

Peripherals Owned/Planning to Buy: Memory Disk Video Printer Terminal Other: ....

Microcomputer Usage: Educational Business Personal Control Games Other: ....

Languages Used: Assembler BASIC FORTH PASCAL Other: ....

Your comments and suggestions on **MICRO**: ....

....

Club/Group User Registration Form

Name:.................................... President: ....

Location:....................................

No. of Members:....................................

Meeting algorithm (date, time, place):....................................

....

....

Publications:....................................

....

Aim/Purpose of the group:....................................

....

For Current Information, Contact:....................................

....

....

Software/Hardware Catalogue Entry

Do you have a software or hardware package you want publicized? Our Software and Hardware Catalogues offer a good opportunity to receive some free advertisement. These regular features of MICRO are provided both as a service to our readers and as a service to the 6502 industry which is working hard to develop new and better software and hardware products for the 6502 based system. There is no charge for listings in these catalogues. All that is required is that material for the listing be submitted in the listing format. All information should be included. We reserve the right to edit and/or reject any submission. We might not edit the description the same way you would, so please, be brief and specific.

Name:..

System:..

Memory:..

Language:..

Hardware:..

Description:..

....

....

....

....

....

....

Copies:..

Price:..

Author:..

Available from:..

....

....

Classified Ad

Classified ads provide an economical way to announce new products or sales promotions, generate product interest, enhance visibility and promote good will. MICRO clusters large format classified ads at high impact locations throughout each issue. Because classifieds represent a service to readers, MICRO must restrict each advertiser to a single, six-line insertion per issue. The nominal $10 charge reflects our preparation costs and must be prepaid.

Description:..

....

....

....

....

....

....

Name:..

Address:..

City,State:........................Zip: ....

Other

If you are interested in Writing for MICRO, becoming a MICRO dealer or advertising in MICRO, please indicate below and the information will be mailed to you.

Dealer Information Package: ☐

Advertiser's Media Package: ☐

MICRO Writer's Guide: ☐

....

First
Class
Stamp

MICRO™

P.O. Box 6502
Chelmsford, MA 01824

SYM - 1 Sends Morse Code

Now you can use your SYM as a Morse Code teaching tool, automatic I.D'er or 'canned' message sender.

Ralph R. Orton
16015 San Fernando Mission Blvd.
Granada Hills, CA 91344

Although many Morse Code oriented programs have been written ranging from simple message loops to quite flexible code reading routines, I have not yet seen any written specifically for the SYM-1. The following will fill this gap with a sending program that could be used as a teaching tool, automatic I.D.'er or as a short cut for sending often sent messages. About 25 words can be stored with the 1K memory that comes with the SYM-1. An additional 50 words can be stored for each additional 1K memory added; thus, the 4K board R/W memory capability could store a total of about 175 words. This may not seem like a lot yet; teaching code at 5 wpm (words per minute), one would have over one half hour of steady material. Even at 13 wpm, you would have over 10 minutes of practice; no easy task for a learner! Figure 1 is a simple circuit for interfacing to the SYM-1 to provide an audio code indicator. Headphone jacks for several people could possibly be paralleled instead of the loud speaker. Other interfaces are left to the needs of the reader.

Pressing an 'O' on the keyboard enters a 'DIT' into memory. A '1' enters a 'dah' and a '2' enters a 'space' (enter 1 between letters and 3 between words). Spaces between parts of a letter need not be entered as they are provided by the program. Entering a '3' ends a message segment. This is only required if a series of messages are being entered. (See list of key memory locations.) As dits, dahs and spaces are entered from the keyboard on the SYM-1°'s, 1's and 2's appear on the display, indicating the data entered. Entry errors can be corrected by entering an 'E' for each entry to be erased. For example, if two erroneous entries had just been made, pressing 'E' twice would cause 'E' to be displayed twice. This indicates that the two prior entries had been erased (see figure 2). Upon completion of data entry, press the 'GO' key and your message will be sent.

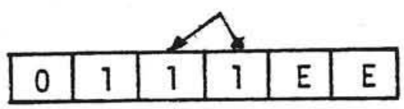
A popular method of teaching code is to send letters at a fast rate but leaving larger than normal spaces between letters until the learner has reached the desired plateau of proficiency.

The rate modification table can be used to determine data to be entered for desired combination of letter speed versus words per minute. Dit delay factor is entered at 0091 and the space factor is entered at 0076.

If continuous loop has been programmed at 004A through 004C, then code will be sent until such time reset is accomplished. If multiple message has been programmed, then a "GO" "CR" at the end of each segment will cause the nest segment to be sent.

It should be noted that a GO command at 002D will cause a new start to occur regardless of the mode selected. Thus, it is not necessary to reprogram old data unless it has been lost due to a newer entry.

These Characters Have Been Erased



04A thru 004C: These locations control the mode of operation.

4C 35 00 Gives continuous loop message. Be sure to put enough spaces at the beginning or end to identify the start of each loop through.

4C A2 00 Gives single or multiple message as desired. For multiple messages, key-in 'GO' 'CR' to start next message.

0053: Data at this location determines times Dit delay will be executed per 'Dit'.

0067: Data at this location determines times Dit delay will be executed per 'DAH'.

0076: Data at this location determines times Dit delay will be executed per 'Space'.

007D: Data at this location determines times Dit delay will be executed per silence between parts of a letter of spaces.

0091: Data at this location determines times delay programmed by "DIVFAC" (Division Factor) will be executed. (e.g. if 'Divfac' = 1024 then 1 loop = 1.024 ms disregarding instruction time error. Part of "Ditdly" routine.

0093: Data at this location determines divison factor to be used by internal timer. 1C = ÷ 1; 1D = ÷ 8; 1E = ÷ 64; 1F = ÷ 1024. Part of "Ditdly routine."

Modified word rate	13	17	21	25	29
	Standard word rate				
5	0E	14	1A	20	27
8	06	0A	0E	11	15
11	02	05	08	0B	0D
14		03	05	07	50
17		01	03	05	06
20				03	04
23				02	03
26					02
	Dit delay factor				
	5A	45	38	2F	28

RATE MODIFICATION TABLE

The timing in the table is based on the following relationships for standard code:

1. A dit is a reference unit of time.
2. A dah = 3 dits
3. Average letters = 6.2 dits
4. Spaces in a letter = 1 dit
5. Spaces between letters = 3 dits
6. Words = 5 letters & appropriate spaces

"Space" multiplication factor =

$$\frac{60 + d - d\,(43\ Wm\ - 3)}{d(7W - 3)}$$

Where d = dit time of standard words per minute rate
Wm = words per minute of the desired modified rate.
"dit time" =

$$\frac{60}{50Ws - 7}$$

Where Ws = words per minute of the desired standard rate.

The above formulas neglect the operation times of the SYM-1 but for practical purposes are quite accurate. The results must be converted to Hex for use in the program, introducing a rounding error which is also normally inconsequential. Greater accuracy is obtainable ofcourse, but the author leaves it to those with the desire to make the needed changes.

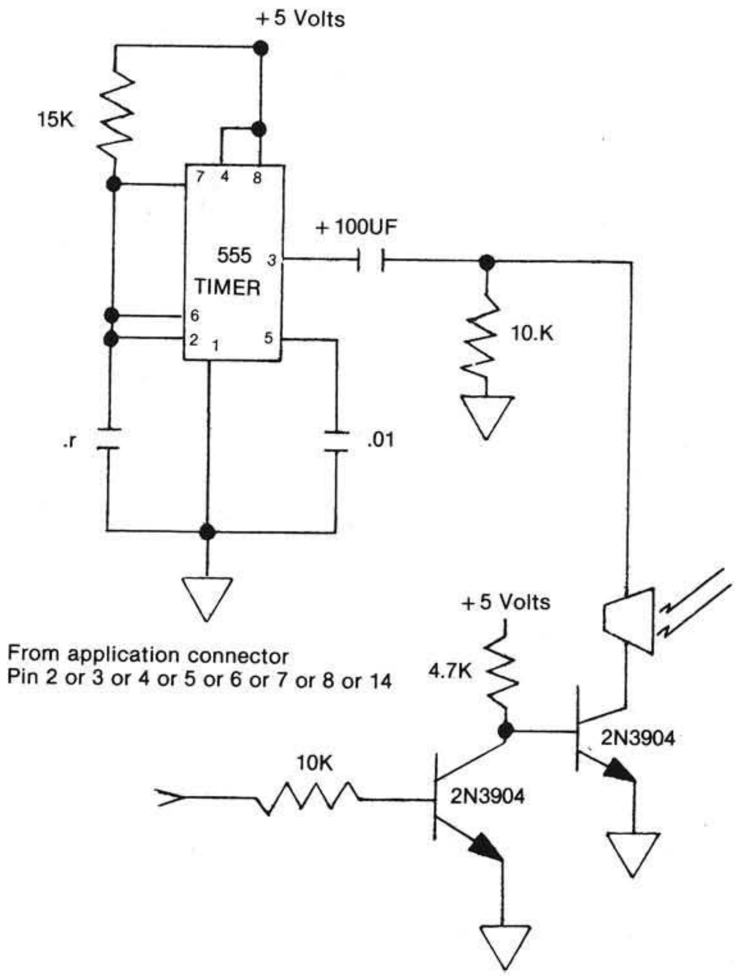


Figure 1

```
                         EQUATE LIST

      00A6               WORDS  *     $000A
      00A6               INCHR  *     $8A1B
      00A6               ACCESS *     $8B86
      0000                      ORG   $0000

      0000 A0 00         LOAD   LDYIM $00
      0002 20 86 8B             JSR   ACCESS
      0005 20 1B 8A      SHOKEY JSR   INCHR
      0008 99 00 02             STAY  $0200  WORDS ARE STORED STARTING AT $0200
      000B C9 47                CMPIM $47    WAS 'GO' KEY PRESSED ?
      000D F0 1E                BEQ   START  IF YES - START SENDING CODE
      000F C9 45                CMPIM $45    WAS 'E' PRESSED ?
      0011 F0 0D                BEQ   ERASE  IF YES - DO ERASE ROUTINE
      0013 C8                   INY
      0014 C0 00                CPYIM $00    256 WORDS COMPLETED ?
      0016 F0 03                BEQ   BASELD IF SO, INCREMENT HI BYTE OF BASE
      0018 4C 05 00             JMP   SHOKEY

      001B E6 0A         BASELD INCZ  WORDS
      001D 4C 05 00             JMP   SHOKEY

      0020 88            ERASE  DEY
      0021 C0 FF                CPYIM $FF    PAGE CROSSING ?
      0023 F0 03                BEQ   SUBASE IF YES - DECREMENT HIGH BYTE OF BASE
```

```
0025 4C 05 00          JMP    SHOKEY
0028 C6 0A      SUBASE DEC    WORDS
002A 4C 05 00          JMP    SHOKEY
002D A9 FF      START  LDAIM $FF     START OF SENDING CODE
002F 8D 01 A0          STA   $A001
0032 8D 03 A0          STA   $A003
0035 A9 02      KEEPON LDAIM $02     'MODE' JUMPS HERE FOR LOOP
0037 85 3D             STAZ  $3D
0039 A0 00             LDYIM $00
003B B9 00 02   CODE   LDAY  $0200   CODE WAS STORED STARTING AT $0200
003E C9 30             CMPIM $30     IS IT A DIT ?
0040 F0 0B             BEQ   DIT     IF SO - GO TO DIT ROUTINE
0042 C9 31             CMPIM $31     IS IT A DAH ?
0044 F0 1B             BEQ   DAH     IF SO - GO TO DAH ROUTINE
0046 C9 32             CMPIM $32     IS IT A SPACE CHARACTER ?
0048 F0 2B             BEQ   SPACE   IF SO - GO TO SPACE ROUTINE
004A 4C A2 00   MODE   JMP   SEGMNT  NONE OF ABOVE , DECIDE MODE
004D A9 00      DIT    LDAIM $00
004F 8D 01 A0          STA   $A001   SET OUTPUT LOW
0052 A9 01             LDAIM $01     LOAD 1 FOR DIT DELAY
0054 85 FF             STA   $00FF   STORE FOR USE BY 'DITDLY'
0056 20 90 00          JSR   DITDLY
0059 A9 FF             LDAIM $FF
005B 8D 01 A0          STA   $A001   SET OUTPUT HIGH AGAIN
005E 4C 7C 00          JMP   SILENT
0061 A9 00      DAH    LDAIM $00
0063 8D 01 A0          STA   $A001   SET OUTPUT LOW
0066 A9 03             LDAIM $03     LOAD FOR 3 DIT DELAYS
0068 85 FF             STA   $00FF   STORE FOR USE BY 'DITDLY'
006A 20 90 00          JSR   DITDLY
006D A9 FF             LDAIM $FF
006F 8D 01 A0          STA   $A001   SET OUTPUT HIGH AGAIN
0072 4C 7C 00          JMP   SILENT  QUIET BETWEEN CHARACTERS
0075 A9 01      SPACE  LDAIM $01     LOAD $0076 FOR DESIRED SPACE
0077 85 FF             STA   $00FF      LENGTH AND STORE FOR USE BY 'DITDLY'
0079 20 90 00          JSR   DITDLY
007C A9 01      SILENT LDAIM $01     LOAD FOR 1 DIT DELAY
007E 85 FF             STA   $00FF   STORE FOR USE BY 'DITDLY'
0080 20 90 00          JSR   DITDLY
0083 C8         INCMEM INY           MOVE POINTER TO NEXT CHARACTER
0084 C0 00             CPYIM $00     PAGE CROSSING ?
0086 F0 03             BEQ   BASEGO  IF YES - INCREMENT HIGH BYTE BASE
0088 4C 3B 00          JMP   CODE
008B E6 3D      BASEGO INC   CODE    +02
008D 4C 3B 00          JMP   CODE    GET NEXT CHARACTER
0090 A9 47      DITDLY LDAIM $47     LOAD $0091 WITH DESIRED DIT TIME
0092 8D 1F A4   DIVFAC STA   $A41F
0095 2C 05 A4   TIMER  BIT   $A405
0098 10 FB             BPL   TIMER   KEEP CHECKING FOR DELAY COMPLETED
009A E8                INX           DONE - INCREMENT DIT COUNTER
009B E4 FF             CPX   $00FF   SHOULD WE DITDLY AGAIN ?
009D D0 F1             BNE   DITDLY
009F A2 00             LDXIM $00     RESET 'X' REGISTER
00A1 60                RTS           BACK TO WHERE YOU CAME FROM
00A2 00         SEGMNT BRK           STOP UNTIL "GO" "CARRIAGE RETURN
00A3 4C 83 00          JMP   INCMEM  NOW SEND NEXT MESSAGE
ID=
```

An EDIT Mask Routine in Applesoft BASIC

This article describes some techniques for producing formatted output using Edit Masks. The programs permit you to produce professional looking output.

Lee Reynolds
801 NE 18th Ct. 109
Ft. Lauderdale, FL 33305

My work as a professional programmer in business applications has often called for the use of what are called "edit masks", in such languages as COBOL, DIBOL, and the Commercial Subroutine Package of Data General FORTRAN. I have found the edit mask capability in these languages quite useful, and so I decided to write a routine in Applesoft Basic that I could use at home on my Apple II.

I should begin by first giving a brief explanation of what an edit mask is, for those readers who have never encountered the term before. An edit mask might be defined as a string of characters which specify operations on a number so as to produce an output string that contains the number's digits re-formatted for printing in certain specific ways. Some of the most common operations that can be carried out on any given number by means of edit masks are the following: (1) "suppressing" of zeroes, by replacing them with blanks in the output string, (2) inserting of a decimal point in a fixed position of the output string, (3) inserting of comma in the string to express thousands, million, etc., (4) placing a dollar sign before the leftmost digit of the number string, and (5) appending a minus sign to the end of the string if the input number is negative.

The edit mask is used as a sort of "picture" of what the output string should be like after carrying out operations such as the above on the number to be edited. In order to achieve this, there are defintie rules for the edit routine's interpretation of the characters that make up the mask. Perhaps the best way of explaining this is to give some examples of my routine's use.

The routine itself, on the following listing, is contained between line numbers 100 to 580. The statements preceding 100 are a "driver" routine you can use to input your edit mask and number to be editied in order to experiment with various types of editing.

The editing routine is called by means of a GOSUB 100. There are two arguments that must be passed to it: NUM is the number to be edited, and MASK$ is the edit mask string. NUM can contain any number of digits up to 9. I have made no provision for editing numbers that must be expressed in "scientific notation" with an Exponent field.

The result of the masking will be passed back to the calling program in the string OUT$, whose length is the same as MASK$.

There are six special characters which can appear in MASK$ that are treated in a distinctive way: these are the digit 9, the digit 0, the period, the comma, the minus sign, and the dollar sign. The mask can contain other characters also, but more about this later.

The digit 9 is the "numeric replacement" character. This means, wherever a 9 is present in the mask, it will be replaced in the result field (OUT$) by the corresponding digit of NUM, if any, in that position.

Thus, suppose we define MASK$ = "99999", and assume the number to be edited is NUM = 352. Then the result, after calling the edit routine, will be OUT$ = "352". (Note the two blanks preceding the ASCII digit 3. This is because the length of the mask exceeds the length of the number to edit by two.)

Next, the digit 0 is the "zero-suppress" character. This means wherever a 0 appears in the mask, it will be replaced in the result field by the corresponding digit of NUM only if that digit is not a zero; if the digit is a zero, then the corresponding position in the result field will be a blank.

To give an example, suppose MASK$ = "990990" and the number to be edited is NUM = 120563. Then the result will be OUT$ = "12 563". The zero in NUM was suppressed.

The most common usage of the zero-suppress character in a mask is to surpress leading zeroes of a number. Thus a mask of "00099" would suppress the first three digits of any five-digit number if they were zeroes, but would print them if they were not. Due to the way my routine operates, it turns out that leading zeroes are always suppressed, anyway. If you would rather change this feature of the routine, I will describe later how you could go about doing so.

The period in a mask is usually used as the decimal point position. It is what is called an "insertion character" in the mask because it is always inserted in the result field exactly in its corresponding position in the mask.

Let's consider some examples of masks containing a period, and what the result will be. Suppose our mask is "999.99", and our number to be edited is 312.44; then, as you would expect, the result will be OUT$ = "312.44". Next suppose we use the same mask but NUM = 33.6. The result is OUT$ = " 33.60". There is a blank in position one and a zero in the last position. (If the last character of the mask had been a 0 instead of a 9, then the last character in the result would have been a blank.) Now, let's suppose that NUM = 124.556. In this case there is one more digit to the right of the decimal point in the number to edit that there is in the decimal part of the mask. When this, or something similar happens, my routine will truncate the extra digit(s), and replace it (them) by an asterisk to signal field overflow. The result then is OUT$ = "124.5*".

My routine follows a similar rule in case the number of digits to the left of the decimal point in NUM exceeds the number allowed in MASK$. For example, if NUM = 1256.7, then the result will be OUT$ = "*56.70".

By the way, since it is conceivable that you might, either by mistake or be design, include two or more periods in your mask, the routine will treat only the rightmost period in the mask as the decimal point position. All other periods will be treated as insertion characters, and so will appear in the corresponding positions of the result field as they expected.

Next, let's consider the comma in an edit mask. An example of a mask containing two commas is the following: MASK$ = "99,999,999". If your number to edit contains either 7 or 8 digits, then the result field will contain both commas in the appropriate places, as you would expect. However, with 6 or fewer digits in NUM, either the first or both commas will be suppressed and replaced by blanks. Examples: if NUM = 1234567, the OUT$ = " 1,234,567"; and if NUM = 1234, then OUT$ = "   1,234" (note the five blank characters preceding the digit 1); and lastly, if NUM = 123, then there will appear seven blanks preceding the digit 1: OUT$ = "   123".

Thus we see that the comma is a special sort of insertion character which is suppressed if there are no preceding digits of the number to be edited.

Now consider the dollar sign used as an edit mask character. I have defined this character's usage in a special way. IF the dollar sign is the very first character in the mask, then it is treated as what is called a "floating dollar signt". That means that the dollar sign in the result field will "float" to the right, far enough so as to immediately precede the leftmost digit of NUM. Some exaples: if MASK$ = "$99,999.99" and NUM = 11.45, then the result of editing is OUT$ = " $11.45" (note that there are four blanks preceding the dollar sign in the result field). And if NUM = 2321, then we have this result: OUT$ = " $2,321.00" (one blank preceding the dollar sign).

Please note that I have defined this usage of the dollar sign as a "floating" dollar sign only when it is the first character in the mask. If it occurs elsewhere in the mask, then it becomes an insertion character.

The last special usage character in a mask is the trailing minus sign. If the mask contains a minus sign as the very last character, then the rightmost position of the result field will be a minus sign when the number to edit is negative, or will be blank if the number is positive. Examples: if MASK$ = "99,999.99-" and NUM = -1453.62, then the resultant OUT$ = " 1,453,62-". While if NUM = 2246.7, then we have OUT$ = " 2,246.70".

If a minus sign appears in a mask in any other position, it is treated as an insertion character. Thus, for example, you could format a date, MMDDYY = month, day, and year with the following mask: MASK$ = "09-99-99". If NUM = 101479, then OUT$ = "10-14-79".

You might be wondering what will happen if you edit a negative number using a mask which does not contain a trailing minus sign. It depends upon whether you have allotted enough digit positions in the mask to accommodate a leading minus sign. If you have then the minus sign will take the place of the first position containing a nine, zero, or comma that immediately precedes the leftmost digit of NUM. If you have not allotted enough digit positions in the mask, then my routine will print the asterisk signaling field overflow.

Now, any character other than the six special cases discussed above may also appear in a mask. In that case the character becomes an insertion character. Suppose you define

MASK$ = "$BAL. DUE AS OF SEP/'78: '99,999.99"

If NUM = 1324.57, then the result of masking will be:

OUT$ = "BAL. DUE AS OF SEP/'78: $1,324.57"

From the above example, you can see that you are only restricted in using edit masks by your imagination, perhaps after making modifications to my routine. For example, you will note that the year in the above mask is '78 not '79. It could not be '79 because the 9 is a numeric replacement character and in this case would have been blanked out. However, if you change the numeric replacement character to some other more convenient character (perhaps an ampersand?) then this difficulty could be avoided.

As already mentioned, another modification you might wish to make is to allow outputting of leading zeroes in a numeric field if the corresponding edit characters are 9's. To do this, you need to make three changes to the routine.

```
455 IF I-1 > =II AND MID$
            (MASK$,I-1,1) =
            "9" then 480
500 IF N$ = " " THEN N$ = "0"
525 IF N$ = " " THEN 460
```

When you incorporate this routine into your own programs, you may wish to change the names of some of the local variables used by it in order not to conflict with your own use of the same names. So here is a list of all variables used by my routine.

Variables

MASK$	the string containing the edit mask.
NUM	the input number to edit
NUM$	NUM converted to a string
LM	length of MASK$
LN	length of NUM$
PM	position of rightmost decimal point in MASK$ (or zero if none)
PN	position of decimal point in NUM$ (zero if none)
RM	number of digit positions right of decimal point in MASK$
RN	number of digits right of decimal point in NUM$
QM	number of digit positions left of decimal point in MASK$
QN	number of digits left of decimal point in NUM$
FD%	flag telling whether mask has floating dollar sign (1 if yes, 0 if no)
MF%	flag telling whether mask has trailing minus sign (1 if yes, 0 if no)
NF%	flag telling whether NUM is negative (1) or positive (0)
M$	current character of MASK$ being processed
N$	current character of NUM$ being processed
I	loop variable and temporary variable
J	pointer to current digit in NUM$
II	first position in MASK$ to process
I2	last position in MASK$ to process

One final note: in using the driver routine to experiment with various edit masks, you should remember that if your mask will contain commas or colons, then you must enclose the entire mask by quotation marks, or else Applesoft will drop part of your mask when it executes the INPUT statement.

Notes on Converting to other Basics

I am not familiar with any other Basics for microcomputers. I do, however, have some acquaintance with the Basic languages for two mini-computers—the DEC PDP-II and the Data General Nova 3. With this as background, I have compiled the following list of possible modifications you might have to make to my routine to get it to work on other 6502 machines other than the Apple.

1.) Applesoft allows variables to have names with more than two characters, although only the first two are used to distinguish between between different names. If your Basic does not allow this, you will have to change some of the names that my routine uses.

2.) Some Basics don't allow multiple statements per line, or if they do, the statement separator might not be the colon; two common alternatives are the back slash or the exclamation point.

3.) If your Basic does not have the "ON...GO TO" statement, then line number 85 will have to be replaced with something else, perhaps a couple of "IF...THEN GOTO..." statements.

4.) Not all Basics allow "NEXT" statements which do not specify the loop variable to end "FOR" loops. There are several lines in my program that may necessitate this type of change: 160, 190, 240, 280, 340, and 550. In all of these cases the implied FOR loop variable is "I".

5.) You may have to DIMension your strings in your Basic program, as is true in Apple's Integer Basic, but not Applesoft.

6.) String concatenation in Applesoft is accomplished with string expressions joined by means of the plus (+) sign; your Basic may use the ampersand (&).

7.) In comparing strings, Applesoft uses the combination of less than and greater than signs (<>); perhaps, as in Integer Basic on the Apple, you are only allowed to test inequality with the number sign (#).

8.) Please note that I have several statements in my program of the following general form: IF X THEN... This is "shorthand" for the equivalent IF X <> 0 THEN... I also have a number of statements like the following: IF...THEN 100 (where 100 can be any statement number). This is a "shorthand" for IF...THEN GOTO 100. I don't know whether all Basics allow the abbreviated forms that I use.

9.) I have made use of the following string functions: STR$, LEFT$, RIGHT$, MID$, and LEN. Your Basic might call these by different names, or have different syntax rules about their arguments. Here are the Applesoft syntactic definitions for these functions, which you should keep in mind if you have to convert to different usages on your computer:

STR$(X)
: converts the number X to a string

LEFT$(A$,N)
: returns the leftmost N characters of string A$

RIGHT$(A$,N)
: returns the rightmost N characters of string A$

MID$(A$,M,N)
: returns the N consecutive characters of string A$, starting at position M

LEN(A$)
: returns the number of characters in string A$

These are all the differences between Applesoft and other Basics that I am aware of, although there may be more. At any rate, it should not be difficult to convert my program to any other machine's Basic.

```
]LIST

10  REM  ROUTINE TO EDIT A NUMBER
     , NUM, WITH AN EDIT MASK, MA
     SK$
20  HOME : PRINT "EDIT MASK ROUTI
     NE": PRINT : PRINT "   THE E
     DIT MASK CAN CONTAIN ANY INS
     ER-": PRINT "TION CHARACTERS
     , PLUS FOLLOWING SPECIAL"
30  PRINT "CHARACTERS:": PRINT "
      IF $ IS FIRST CHAR., IT IS
     TREATED AS": PRINT "A FLOATI
     NG DOLLAR SIGN"
40  PRINT "  IF - IS LAST CHAR.,
     IT WILL BE OUTPUT": PRINT "I
     F NUMBER TO EDIT IS NEGATIVE
     , OR RE-": PRINT "PLACED BY
     BLANK IF POSITIVE"
50  PRINT "  9 CORRESPONDS TO A D
     IGIT TO PLACE IN": PRINT "TH
     AT POSITION OF THE MASK": PRINT
     "  0 CORRESPONDS TO A NONZER
     O DIGIT TO"
60  PRINT "PLACE IN THAT POSITION
     . IF YOU WANT A": PRINT "COM
     MA OR COLON IN THE MASK, ENC
     LOSE THE"
65  PRINT "ENTIRE MASK IN QUOTES
     TO INPUT IT.": PRINT
70  INPUT "EDIT MASK? ";MASK$
75  INPUT "NUMBER TO EDIT? ";NUM:
      GOSUB 100: PRINT "EDITED NU
     MBER:";OUT$
80  PRINT : INPUT "1=NEW NUMBER,
     2=NEW MASK AND NUMBER?";N
85  ON N GOTO 75,70
90  GOTO 80
100 NUM$ =  STR$ (NUM):LN =  LEN
     (NUM$):LM =  LEN (MASK$):QM =
     0:QN = 0:RM = 0:RN = 0:PN =
     0:PM = 0:NF% = 0:MF% = 0:FD%
      = 0:DF% = 0
110 OUT$ = "": IF NUM < 0 THEN NF
     % = 1: REM  SET FLAG TELLING
      WHETHER INPUT NUMBER IS NEG
     ATIVE
120  IF  RIGHT$ (MASK$,1) = "-" THEN
     MF% = 1: REM  SET FLAG TELLI
```

```
      NG WHETHER INPUT MASK HAS TR
      AILING MINUS SIGN
130  IF  LEFT$ (MASK$,1) = "$" THEN
      FD% = 1: REM  SET FLAG TELLI
      NG WHETHER INPUT MASK HAS FL
      OATING DOLLAR SIGN
140  FOR I = 1 TO LM: REM  FIND P
      OSITION OF DECIMAL POINT IN
      MASK
150  IF  MID$ (MASK$,I,1) = "." THEN
      PM = I
160  NEXT : IF FD% = 0 THEN DF% =
      1: REM  IF NO FLOATING DOLLA
      R SIGN IN MASK, SET FLAG SAY
      ING "$" ALREADY OUTPUT TO ED
      ITED FIELD
170  FOR I = 1 TO LN: REM  FIND P
      OSITION OF DECIMAL POINT IN
      NUMBER TO EDIT
180  IF  MID$ (NUM$,I,1) = "." THEN
      PN = I
190  NEXT
200  IF PN THEN RN = LN - PN: REM
       IF DECIMAL POINT IN NUMBER,
       COMPUTE # DIGITS RIGHT OF D
      ECIMAL PT.
210  IF PM = 0 THEN 250: REM  IF
      DEC. PT. IN MASK, FIND # DIG
      IT POSITIONS RIGHT OF IT
220  FOR I = LM TO PM STEP  - 1
230  IF  MID$ (MASK$,I,1) = "0" OR
       MID$ (MASK$,I,1) = "9" THEN
      RM = RM + 1
240  NEXT
250  IF PN = 0 AND PM = 0 THEN 30
      0
260  IF RM = RN THEN 300
270  IF RM < RN THEN 290
280  FOR I = RN TO RM - 1:NUM$ =
      NUM$ + "0": NEXT : GOTO 300:
       REM  ZERO-FILL RIGHTMOST DE
      CIMAL POSITIONS OF NUM$
290 I = LN - RN + RM - 1:NUM$ =  LEFT$
      (NUM$,I) + "*": REM  TRUNCAT
      E NUM$ TO MATCH MASK, PUT "*
      " IN RIGHTMOST DIGIT
300 QN =  LEN (NUM$) - RM: IF PN THEN
      QN = QN - 1: REM  GET # DIGI
      TS LEFT OF DEC. PT. IN NUMBE
      R, IGNORING DEC. PT., IF ANY

310  IF NF% AND MF% THEN QN = QN -
      1: REM  IGNORE MINUS SIGN IN
       NUMBER IF TRAILING MINUS IN
       MASK
320  FOR I = 1 TO LM: IF I = PM THEN
      350: REM  FIND # DIGITS IN M
      ASK LEFT OF DEC. PT.
330  IF  MID$ (MASK$,I,1) = "0" OR
       MID$ (MASK$,I,1) = "9" THEN
      QM = QM + 1
340  NEXT
350  IF QM >  = QN THEN 370: REM
       TRUNCATE NUMBER ON LEFT, MA
      KING LEFTMOST DIGIT "*"
360 I =  LEN (NUM$) - QN + QM - 1
      : IF NF% AND MF% THEN I = I -
      1: REM  DROP MINUS SIGN ALSO
       IF IGNORED BEFORE
365 NUM$ = "*" +  RIGHT$ (NUM$,I)
      :QN = QM
370 I1 = 1: IF FD% THEN I1 = 2: REM
       WILL IGNORE ANY FLOATING DO
      LLAR SIGN IN MASK
380 I2 = LM: IF MF% THEN I2 = LM -
      1: REM  WILL IGNORE ANY TRAI
      LING MINUS IN MASK
385  IF NF% AND MF% AND  LEFT$ (N
      UM$,1) = "-" THEN QN = QN +
      1: REM  IF NUMBER'S MINUS SI
      GN WAS IGNORED BEFORE, PUT I
      T BACK IN
390  IF PN THEN NUM$ =  LEFT$ (NU
      M$,QN) +  RIGHT$ (NUM$,RM): REM
       DROP DEC. PT. FROM NUMBER S
      TRING
400  IF NF% AND MF% AND  LEFT$ (N
      UM$,1) = "-" THEN NUM$ =  RIGHT$
      (NUM$, LEN (NUM$) - 1): REM
       DROP MINUS SIGN IF TRAILING
       MINUS IN MASK
410 J =  LEN (NUM$): FOR I = I2 TO
      I1 STEP  - 1:M$ =  MID$ (MAS
      K$,I,1):N$ = " ": IF J > 0 THEN
      N$ =  MID$ (NUM$,J,1)
420  IF M$ <  > "," THEN 490
430  IF N$ <  > "-" THEN 450
440 OUT$ = N$ + OUT$:J = J - 1: GOTO
      550
450  IF N$ <  > " " THEN 480
460  IF DF% THEN 440: REM  IF FLO
      ATING DOLLAR SIGN ALREADY OU
      TPUT, GO INSERT BLANK
470 DF% = 1:OUT$ = "$" + OUT$: GOTO
      550
480 OUT$ = M$ + OUT$: GOTO 550
490  IF M$ <  > "9" THEN 520
500  IF N$ = " " THEN 460: REM  I
      F ALL DIGITS OF NUMBER OUTPU
      T, GO OUTPUT FLOATING DOLLAR
```

```
     SIGN OR BLANK
510  GOTO 440: REM  GO OUTPUT THE
     DIGIT
520  IF M$ < > "0" THEN 480: REM
     GO OUTPUT CURRENT CHARACTER
     IN MASK
530  IF N$ < > "0" THEN 500: REM
     GO OUTPUT BLANK OR DIGIT
540 N$ = " ": GOTO 440: REM  OUTP
    UT BLANK
550  NEXT : IF DF% = 0 THEN OUT$ =
     "$" + OUT$: REM  IF FLOATING
     DOLLAR NOT OUTPUT, APPEND I
     T ON LEFT
555  IF DF% AND FD% THEN OUT$ = "
     " + OUT$: REM  IF DOLLAR SI
     GN ALREADY OUTPUT, PUT BLANK
      IN PLACE OF MASK'S DOLLAR S
     IGN
560  IF MF% = 0 THEN  RETURN : REM
      ALL DONE IF NO TRAILING MIN
     US IN MASK
570 N$ = " ": IF NF% THEN N$ = "-
    ": REM  BLANK IF POSITIVE, M
    INUS SIGN IF NEGATIVE
580 OUT$ = OUT$ + N$: RETURN
```

PET Keysort Update

Two changes are presented to improve the PET Keysort.

Rev. James Strasma
120 West King Street
Decatur, IL 62521

After further use and testing, I decided to make two changes to my program KEYSORT as printed in MICRO:23. First, I've added to the intelligence of the himem setter in lines 550-630 of the source listing. Previously, my copy wasted about 100 bytes of memory by setting himem lower than it needs to be. Now it is set just at the start of the sort. The new source listing would read:

```
.550      lda *him+1;cut himem?
.560      seo
.570      sbo #h,sart
.580      boo sav    dont lower himem
.590      bne cut    ;do lower it
.600      lda *him   depend on lo byte
.605      sbo #L,sart
.610      boo sav    ;if already lower
.615cut   lda #L,sart ;out lo, then hi
.620      sta *him
.625      lda #h,sart
.630      sta *him+1
```

This is an addition of 3 lines and 5 bytes.

The other change is in the way KEYSORT handles nulls. Logically, they should have a value below any other character. The original KEYSORT treats them this way. However, that leads to a problem with partially filled Basic arrays. All the undefined array elements start out as nulls, and end up after a sort at the 'bottom' of the array, where the significant elements were before. I elected to redefine nulls as larger than 'Z', so they stay at the 'top' of the array, where they were before the sort. The necessary changes are made in lines 5180-5220 of the source listing. The label 'null' is deleted from line 5180. A new label, 'same' is added to line 5220. Then 4 new lines are added after line 5420. These are like lines 5180-5200, except that the destinations are opposite. The new lines read:

```
.5422nullcpx#1
                    ;put nulls @top of$s
.5424     beq two((
                    ;to keep them out
.5426     bpL one((
                    ;of the way of prgm
.5428     bmi same
                    ;jump
```

This change adds 4 lines and 8 bytes to the program. Unfortunately it also alters many other parts of the object code in order to stay just below himem, so you'll need to check the enclosed new object listing carefully against your copy of the former version.

After these changes are made, the system call address for the sort is lowered, to sys (31828). The option setting addresses are unchanged. If you'd rather not make these changes yourself, updated copies for any loation in memory, or for old ROMs are available directly from the author for $5.00. Please specify the address and ROM set you prefer to use.

```
. 7C50 00 00 00 00 A5 35 38 E9
. 7C58 7C 90 10 D0 06 A5 34 E9
. 7C60 54 90 08 A9 54 85 34 A9
. 7C68 7C 85 35 20 98 7D A5 2C
. 7C70 85 15 A5 2D 85 16 A5 02
. 7C78 C9 24 F0 06 A9 80 85 00
. 7C80 85 01 A0 00 B1 15 C5 00
. 7C88 F0 06 A9 80 C5 00 D0 08
. 7C90 C8 B1 15 C5 01 30 01 C8
. 7C98 98 AA A5 03 C9 25 F0 04
. 7CA0 A9 09 85 04 A5 05 C9 23
. 7CA8 F0 04 A9 00 85 06 A0 02
. 7CB0 B1 15 18 65 15 85 18 C8
. 7CB8 B1 15 65 16 85 19 A5 2F
. 7CC0 C5 19 F0 02 B0 0E A5 2E
. 7CC8 C5 18 F0 02 B0 06 E0 02
. 7CD0 B0 11 90 17 E0 02 B0 0B
. 7CD8 A5 18 85 15 A5 19 85 16
. 7CE0 18 90 9F A0 04 B1 15 C9
. 7CE8 01 F0 08 20 98 7D A2 80
```

```
. 7CF0 4C 57 C3 A0 06 B1 15 85
. 7CF8 12 88 B1 15 85 13 18 6A
. 7D00 85 0E A5 12 6A 18 69 01
. 7D08 85 0D A5 0E 69 00 85 0E
. 7D10 A5 15 18 69 04 85 23 A5
. 7D18 16 69 00 85 24 A5 12 85
. 7D20 0B A5 13 85 0C A5 0E F0
. 7D28 03 4C AA 7D A5 0D C9 01
. 7D30 D0 F7 A5 0B 85 1B A5 0C
. 7D38 85 1C 20 B3 7F A0 00 B1
. 7D40 15 85 20 C8 B1 15 85 21
. 7D48 C8 B1 15 85 22 A5 23 18
. 7D50 69 03 85 18 A5 24 69 00
. 7D58 85 19 B1 18 91 15 88 B1
. 7D60 18 91 15 88 B1 18 91 15
. 7D68 38 A5 0B E9 01 85 0B A5
. 7D70 0C E9 00 85 0C C9 00 D0
. 7D78 57 A5 0B D0 53 A5 07 85
. 7D80 1B A5 08 85 1C 20 B3 7F
. 7D88 A5 20 A0 00 91 15 C8 A5
```

```
.: 7D90 21 91 15 C8 A5 22 91 15
.: 7D98 A2 24 BD DB 7F 48 B5 00
.: 7DA0 9D DB 7F 68 95 00 CA 10
.: 7DA8 F1 60 38 A5 0D E9 01 85
.: 7DB0 0D A5 0E E9 00 85 0E 85
.: 7DB8 1C A5 0D 85 1B 20 B3 7F
.: 7DC0 A0 00 B1 15 85 20 C8 B1
.: 7DC8 15 85 21 C8 B1 15 85 22
.: 7DD0 A5 0D 85 09 A5 0E 85 0A
.: 7DD8 A5 09 85 07 A5 0A 85 08
.: 7DE0 18 26 09 26 0A A5 0A C5
.: 7DE8 0C F0 05 90 0D 4C 88 7E
.: 7DF0 A5 09 C5 0B 90 04 F0 5E
.: 7DF8 B0 F3 A5 09 85 1B A5 0A
.: 7E00 85 1C 20 B3 7F A0 00 B1
.: 7E08 15 85 17 C8 B1 15 85 18
.: 7E10 C8 B1 15 85 19 18 A5 15
.: 7E18 69 03 85 15 A5 16 69 00
.: 7E20 85 16 A0 02 B1 15 85 1C
.: 7E28 88 B1 15 85 1B 88 B1 15
.: 7E30 85 1A A5 19 85 1F A5 18
.: 7E38 85 1E A5 17 85 1D 20 D7
.: 7E40 7E A5 14 C5 17 90 0F F0
.: 7E48 0D 18 A5 09 69 01 85 09
.: 7E50 A5 0A 69 00 85 0A A5 09
.: 7E58 85 1B A5 0A 85 1C 20 B3
.: 7E60 7F A0 02 B1 15 85 1C 88
.: 7E68 B1 15 85 1B 88 B1 15 85
.: 7E70 1A A5 22 85 1F A5 21 85
.: 7E78 1E A5 20 85 1D 20 D7 7E
.: 7E80 A5 14 C5 17 F0 02 B0 1E
.: 7E88 A5 07 85 1B A5 08 85 1C
.: 7E90 20 B3 7F A0 00 A5 20 91
.: 7E98 15 C8 A5 21 91 15 C8 A5
.: 7EA0 22 91 15 4C 25 7D A5 07
.: 7EA8 85 1B A5 08 85 1C 20 B3
.: 7EB0 7F A5 15 85 18 A5 16 85
.: 7F90 17 60 B1 1E 85 14 A9 00
.: 7F98 85 17 60 A9 00 85 14 B1
.: 7FA0 1B 85 17 60 85 14 B1 1E
.: 7FA8 85 17 60 E0 01 F0 EC 10
.: 7FB0 E1 30 D8 A5 1B 85 1E A5
.: 7FB8 1C 85 1F 18 26 1B 26 1C
.: 7FC0 A5 1B 18 65 1E 85 1B A5
.: 7FC8 1C 65 1F 85 1C A5 1B 18
.: 7FD0 65 23 85 15 A5 1C 65 24
.: 7FD8 85 16 60 00 00 00 00 00
.: 7EB8 19 A5 09 85 1B A5 0A 85
.: 7EC0 1C 20 B3 7F A0 00 B1 15
.: 7EC8 91 18 C8 B1 15 91 18 C8
.: 7ED0 B1 15 91 18 4C D8 7D A0
.: 7ED8 00 84 0F A6 06 D0 0B A5
.: 7EE0 1D 85 10 A5 1A 85 11 18
.: 7EE8 90 70 A5 04 D1 1E F0 08
.: 7EF0 C8 C4 1D 90 F7 4C EB 7C
.: 7EF8 84 0F CA F0 02 B0 F1 C8
.: 7F00 D1 1E F0 05 C4 1D 90 F7
.: 7F08 C8 88 98 38 E5 0F 85 10
.: 7F10 E6 0F A5 0F 18 65 1E 85
.: 7F18 1E A5 1F 69 00 85 1F A0
.: 7F20 00 84 0F A6 06 A5 04 D1
.: 7F28 1B F0 08 C8 C4 1A 90 F7
.: 7F30 4C EB 7C 84 0F CA F0 02
.: 7F38 B0 F1 C8 D1 1B F0 05 C4
.: 7F40 1A 90 F7 C8 88 98 38 E5
.: 7F48 0F 85 11 E6 0F A5 0F 18
.: 7F50 65 1B 85 1B A5 1C 69 00
.: 7F58 85 1C A5 10 C5 11 F0 08
.: 7F60 B0 0C 85 0F A2 01 D0 0C
.: 7F68 85 0F A2 00 F0 06 A5 11
.: 7F70 85 0F A2 02 C9 00 F0 33
.: 7F78 A0 00 B1 1B D1 1E D0 24
.: 7F80 C8 C4 0F 90 F5 E0 01 F0
.: 7F88 09 10 10 A9 00 85 14 85
```

Expand KIM - 1 Versatility in Systems Applications

Techniques and programs are presented which permit the simple addition of six sense switches or an ASCII keyboard to the KIM.

Ralph Tenny
P.O. Box 545
Richardson, TX 75080

The KIM-1 microcomputer, produced by MOS Technology, Commodore and Rockwell International, is a single-board computer which gained early popularity with hobbyists. It also was adopted by industry for small controller applications. Some of these computers have been expanded into fairly large systems, in colleges as well as industry. One reason for the easy acceptance of the KIM-1 was the on-board keypad and six digit display. These features, along with a slow but extremely reliable audio cassette interface for program storage, made KIM-1 one of the first microcomputers which did not require an operator interface more expensive than itself.

The on-board keyboard and seven-segment display, which permits system operation without an teletype of terminal, is implemented in a way which permits addition of both an ASCII external keyboard and sense switches. Fig. 1 shows the key-pad implementation where U24 enables one of three banks of seven keys, and U2 (an MCS6530 programmable interface device detects a key closure in any one of the seven switch columns.

The keyboard encoding scheme works as follows: U2 is programmed for output on lines PB1-PB4 to dribe U24, a four line-to-ten-line decoder which has active-low outputs. Note that the least significant bit of U2's B port (PBO) is not used in the keyboard drive, so values written to Port B are incremented by two to select the next higher keybank. For example, writing 0016 to Port B selects Row keys, 0216 enables Row 1 and 0416 selects Row 2.

On Port A of U2 (lines PA0-PA6), which are programmed as inputs, a closure of (for example) key 8 will cause a logic zero to be input on PA5 whenever key Row 1 is active (low). KIM's operating system software then decodes Row 1/PA5 as key 8 and returns the value 0816 in the accumulator.

A fourth keybank (Row 3) is also implemented by this matrix, but the standard KIM-1 has only the TTY/KYBD switch installed on this row. FIG. 1 shows six additional switches implemented on Row 3; with proper programming, these can be used as sense switches or imput lines for address vectors in an expanded interrupt scheme. Listing 1 gives an example of the programming required to detect activity on Row 3 inputs.

The programming strategy required for any such inputs is to enable PAO-PA6 lines for input and sequentially activate the driving lines (outputs of U24 in this case) to their on (low) state. The program then reads all input lines, masks and inverts the data and returns to the calling program which tests the accumulator for any "one" bits. It is then the programmer's responsibility to repeat the scan periodically and test to see if the same data is present (a noise spike would be gone on a second scan) or has changed after some period of time. This testing allows for switch bounce—multiple closures of the contacts—a characteristic of all switches. Very good switches will bounce for a minimum of one or two milliseconds, while worn or cheap switches may bounce for up to 25 milliseconds. On the other hand, any operator who is trying to make a very short switch closure will find it difficult to release a switch earlier than 50 milliseconds after closure. Consequently, reading keys with software is a fine art!

LISTING I

```
A9 00      LDA #$00    SET PADD (KEY INPUT LINES)
8D 41 17   STA PADD    FOR INPUT
A9 3F      LDA #$3F    SET PBDD (ROW DEFINITION)
8D 43 17   STA PBDD    FOR OUTPUT
A9 06      LDA #$06    ENABLE KEYBOARD
8D 42 17   STA PBD     ON ROW 3
AD 40 17   LDA PAD     READ SENSE SWITCHES
29 7E      AND #$7E    MASK OFF TTY/KYBD SWITCH
49 7E      EOR #$7E    INVERT SWITCH DATA
A2 00      LDA #$00    DISABLE
8E 42 17   STX SBD     KEYBOARD
60         RTS         RETURN TO CALLING PROGRAM
```

Any keyboard with ASCII outputs is likely to have both a debounced output and a strobe which becomes active when there is a key pressed and the data has been debounced. Typically, the key data is active high (positive logic), but the strobe can be either active high or active low. The ASCII keyboard input described here does not use the strobe; instead, the key matrix is scanned in the same manner as is the normal KIM keypad. Fig. 2 shows the necessary connections—a pull-down transistor for each output bit of the keyboard. Any logic "one" data from the keyboard will input a low on the same lines as the KIM keypad. Note that some keyboards output only six bits, so the strobe can be implemented on Column G.

Listing 2 shows a "bare bones" scan program which will return to the calling program as did Listing 1. The basic scheme here is to initialize the accumulator to FF_{16} and get the input data by a logic AND with the input port. The data is then inverted (Exclusive OR) and tested for any logic one bits. Note that the calling program could also permanently set the port for input and somewhat abbreviate the program segment shown. If the strobe is implemented on Column G as mentioned above, the 6502 BIT instruction followed by a test of the overflow status bit (BVC or BVS) will identify strobe activity. Note that the onboard keypad must not be active when the ASCII keyboard is being used, and that the normal KIM keypad scan routines will not properly interpret the ASCII input.

LISTING II

```
A9 80             LDA #$80    ENABLE KEYBOARD
8D 41 17          STA PADD    INPUT LINES
A9 FF             LDA #$FF    INITIALIZE ACCUMULATOR
2D 40 17          AND SAD     INPUT POSSIBLE KEYBOARD BITS
49 7F             EOR #$7F    INVERT ANY BITS PRESENT
F0 02             BEQ OUT     TEST FOR DATA PRESENT
A9 80             LDA #$80    SET FLAG FOR NO INPUT
60           OUT  RTS         RETURN TO CALLING PROGRAM
```

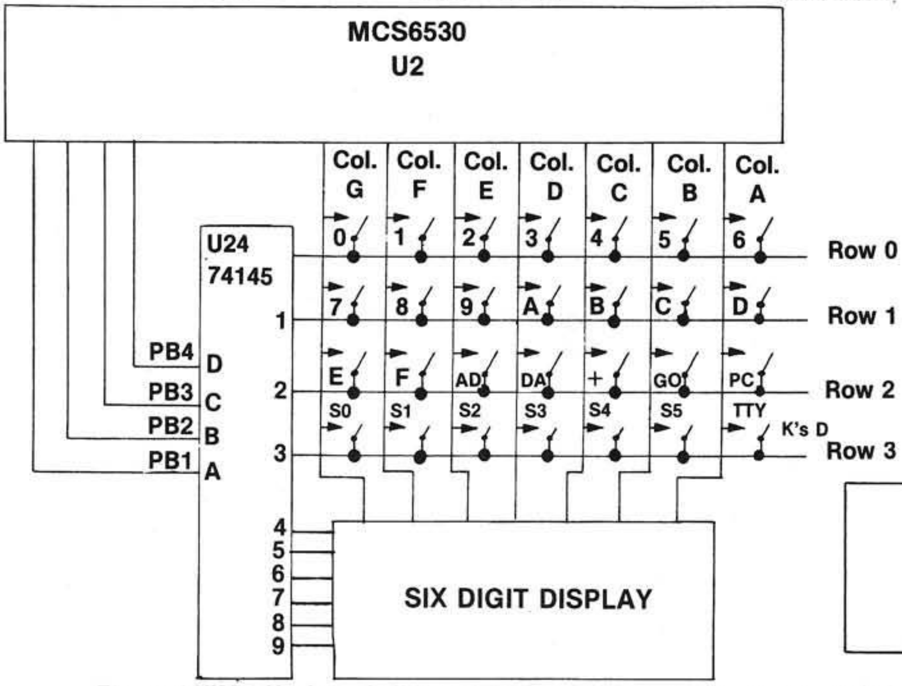


Figure 1: **KIM-1 Keyboard allows six sense switches to be added.**

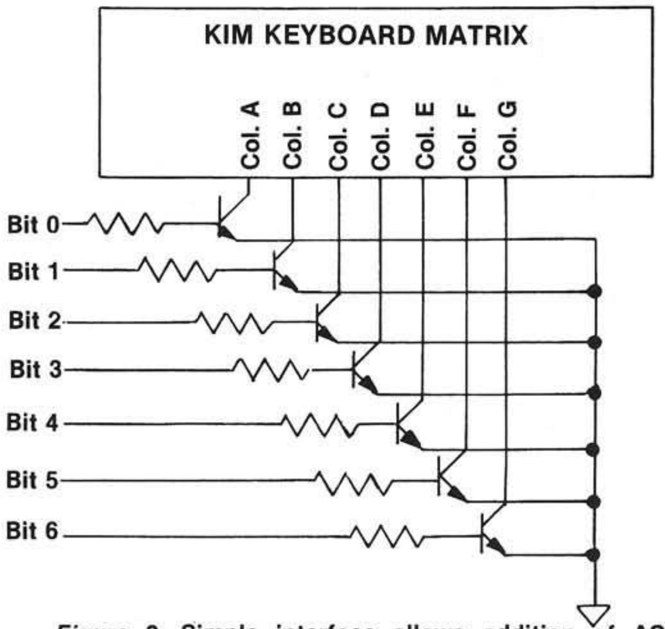


Figure 2: **Simple interface allows addition of ASCII keyboard to basic KIM-1.**

MICRO Club Circuit

MICRO continues its soon to be monthly feature on 6502-related clubs. We are continuing to publish the names, locations, and activities of groups that could be of interest to our readers.

If you are involved in such a club have your representative register your group with us. In return for this registration we will send a free one year subscription of MICRO to your club's library. Include information regarding the club's name, location, algorithm, publications, purpose, officers, number of members, contact person, etc. Your club will then automatically appear in any club updates. If you are already registered please be sure to keep us current on your club's activities.

We would like this feature to be as helpful to our readers as possible. We welcome any information that will be of interest to other clubs: ie. what your club does, how it got started, what is published, your meeting format, purpose, etc.

We are publishing a complete list as of March. Please keep the updates coming! Start increasing your membership and give your group new exposure by telling others about yourselves.

If any of the following information is in error or outdated, please notify us.Address any questions or information to:

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

OSI User's Group

Meets at Aristocraft on the first Thursday of the month (7-9:00 p.m.):
314 5th Avenue
New York, New York.
David Gillette, President. "Mutual aid and sharing of information."

The Big Apple User's Group

Meets on the last Tuesday of each month,at:
55-A Locust Avenue
New RochelleNew York
Tony Cerreta, President. "Exchange of ideas, growth in the field, production of hardware and software."

Apple Pi Computer User's Group

Meets first Thursday of each month at:
Colorado School of Mines
Cecil H. Green Bldg
Room 280
Boulder, CO.
Scott Knaster, President. "Spread information, use of documentation library and a software library for research and trading."

Apple User's Group

Meets on the third Thursday of each month (7:30 p.m.) at:
Computerland of Walnut Creek
1815 Ygnacio Valley Road
Walnut Creek, CA.
Hank Couden, President. "Foster knowledge and use of the Apple Computer."

Original Apple Corps

Meets second Sunday of the month (12:00 Noon) at:
Cal State University at Long Beach
Lecture Hall 151
Contact:
Kip J. Reiner,
19041-2 Hamlin Street
Reseda, CA 91335
"Expand the knowledge of Apple Computers. Software and Hardware."

Greater Lafayette Apple User's Group

Meets on the second Wednesday of each month (7:00 p.m.) at:
Digital Technology
10 North Third
Layfayette, IN. 47901
Jon W. Backstrom, President. "Library of public domain software. Exchange program. Want to educate members on successful programming skills. Workshops."

Salem (Oregon) Area Computer Club

Meets on the first Monday of each month. On odd numbered months, meetings are held at:
McKinley Community School
461 McGilchrist Street SE
Salem, OR.
and on even numbered months at:
The Computer Pathways Unlimited Retail Store
831 Lancaster Drive NE
South End-Lancaster Mall
Salem, OR.
Contact:
DougWalker
4554 Jan Ree Drive NE
Salem, OR 97303
"Each meeting features a presentation by a club member or an invited guest, followed by a 'bull session'."

The Apple Cart

A special interest group of American Mensa. For more information contact:
C. Brandon Gresham, Jr.
23 Van Buren Street
Dayton, OH 45402
"Hardware and software information. Software exchange. Promote the creation of well written and well documented software."

Apple Sac

Meets first Tuesday and third Wednesday of the month (7:00 p.m.) at:
Woodbridge School
5761 Brett Drive
North Highlands, CA
For further information contact:
Bill Norris, President
8074 Ruthwood Way
Orangevale, CA 95662
"To provide members with information, discounts at group rates, a place to exchange programs, ideas, techniques, hardware modifications and to feature guest speakers."

Ohio Scientific Users NW

Meets second Friday of each month at:
Data Systems Plaza
975 SE Sandy Blvd
Portland, OR 97214
Meetings are at 7:30 p.m.

The Santa Barbara Apple User's Group
Meets at:
2031 De La Vina
Santa Barbara, CA 93105

Las Cruces Computer Club
Meets the first Thursday of the month (7:30 p.m.) at:
SouthWest Computer Center
Suite 7
121 Wyatt Drive
Las Cruces, N.M. 88001
John Martellaro, President. Contact him at:
2929 Los Amigos, Apt.B
Las Cruces, New Mexico 88001

Apple-Siders of Cincinnati
Meets the second Tuesday of the month (7:30 p.m.) at:
The University of Cincinnati
Med. Science Bldg.
Contact:
John Anderson
5707 Chesapeake Way
Fairfield, OH 45014

Denmark 6502 Club
A country wide 6502 microprocessor club is being formed. Please contact for further information:
E.Skovgaard
Nordlundsvej 10
DK-2650 Hvidoure
Denmark
"Systems are reviewed and demonstrated. Developing a software library."

Carolina Apple Core
Meets third Tuesday (7:30 p.m.) of the month for general meeting. Other meetings are held on specific topics. Contact Joe Budge, President at:
P.O.Box 31424
Raleigh, NC 27622
"General support of the Apple User."

North London Hobby Computer Club
For more information contact:
Stephanie Bromley
The Polytechnic of North London
Holloway, London N7 8D8

Computer Club in Belgium
DeVlaamse Minicomputerclub
Lambrechtshoekenlaan 171b6
2060 Merksem, Belgium

Apple Group - New Jersey
Meets the fourth Friday of every month (7:00 p.m.) at:
Union County Tech. Institute
1776 Raritan Road
Scotch Plains, N.J.

PACS PET User Group
Meets the third Saturday (11:00 a.m.) every month at:
Science Building
LaSalle College
20th and Onley Avenue
Philadelphia, PA 19191

Washington Apple Pi
Meets the fourth Saturday (9:30 a.m.) every month at:
George Washington University
Rm. 206, Tompkins Hall
23rd and H streets NW
Washington, DC
You may write to this club at:
Washington Apple Pi
P.O.Box 34511
Washington, DC 20034
"Publishes a monthly newsletter."

South Carolina Apple
Meets second Tuesday of the month (7:30 p.m.) at:
The Byte Shop
1920 Blossom Street
Columbia, SC
You may address your inquiries to:
P.O.Box 70278
Charleston Heights, SC 29405

WAKE—
(Washington Area Kim Enthusiasts)
Meets the third Wednesday (7:30 p.m.) of every month at:
McGraw-Hill Continuing Education Center in Washington DC.
Contact Ted Beach at
5112 Williamsburg Boulevard
Arlington, VA 22207
for further information.

Miami Apple User's Group
Contact David Hall, Secretary at:
2300 NW 135th Street
Miami, FL 33167

Sun Coast Apple Tree
Meets the first and third Thursday of the month (7:00 p,m,) at:
The Computer Store
21 Clearwater Mall
Clearwater, FL 33516

Central Ohio Apple Computer Hobbyists
(COACH) Meets the third Saturday of each month (1:00 - 5:00). Contact:
Tom Mimlitch
1547 Cunard Road
Columbus, Ohio 43227

Apple Dayton
Meets the second Wednesday of odd numbered months and the second Thursday of even numbered months (7:30 p.m.) at:
Computer Solutions
Contact: Robert W. Rennard at
2281 Cobble Stone Court
Dayton, OH 45431

Madison Pet User's Club
Meets monthly at:
Washington Square Building
1400 East Washington Avenue
Madison, WI 53913
Contact: Ben A. Stewart
501 Willow
West Baraboo, WI 53913

Micro and Personal Computer Club of St. Louis
Meets monthly at:
Futureworld, Inc.
12304 Manchester Road
St. Louis, MO 63131
Contact: Mr. Kunihiro Tanaka

Tulsa Computer Society
Meets the last Tuesday of each month (7:30 p.m.) at:
Tulso Vo-Tech School
Seminar Center
3420 S. Memorial Drive
Tulsa, OK
This society also has an Apple User Group. For more information please write to:
The Tulsa Computer Society
P.O.Box 1133
Tulsa, OK 74101

The Apple Corps
Meets the second Saturday of each month (2-5:00 p.m.) at:
Greenhill School
14255 Midway Road
Dallas, TX

Appleseed
Meets monthly at:
The Computer Shop
6812 San Pedro
San Antonio, TX 78216

Apples Brit.Columbia Computer Society
Meets the first Wednesday of each month. Contact:
Gary B. Litte
101-2044 West Third Avenue
Vancouver, British Columbia
Canada V6J 1L5

Honolulu Apple User's Society
Meets the first Monday of each month at the Computerland Store in Honolulu. Contact:
Bill Mark
98-1451-A Kaahumanu Street
Aiea, Hawaii 96701

Has anyone heard from the following clubs? Are they still active? Any current information would be appreciated!

The MicroComputer Investor's Assoc.
The New England Apple Tree
Apple User Group of Europe

Applelist
Meets the second Wednesday of the month (7:30 p.m.) at:
Computerland
Skiff Street
Hamden, Conn.
Contact:
Marc B. Goldfarb
55 Pardee Place
New Haven, Conn. 06515
"Promote greater literacy on Apple II, Publish Newslad (ASAP) and aid new users."

THREE GOOD REASONS YOU SHOULD READ

COMPUTE. The Journal for Progressive Computing.™

ISSUE 1, FALL, 1979

ISSUE 2, JANUARY/FEBRUARY, 1980

ISSUE 3, MARCH/APRIL, 1980

ONE GOOD REASON TO READ

compute II.

The Single-Board COMPUTE.™

ISSUE 1, APRIL/MAY, 1980

The 6502 Resources

COMPUTE. and compute II.

6 Issue (annual) subscription: U.S. $9.00, CANADA $12.00 U.S. Important Note: Beginning with Issue 4, **COMPUTE** covers Pet, Apple and Atari. The Single Board Computer Gazette moves to our new magazine, **compute II. The Single-Board COMPUTE,** covering Sym, KIM, AIM, OSI, and 1802 based machines. 6 Issue (annual) subscription: U.S. $9.00, CANADA $12.00 U.S. **Subscribe to both through June 30, 1980** for $15.00, U.S.; $18.00 U.S. in CANADA. **COMPUTE. P.O. Box 5119, Greensboro, N.C. 27403, (919) 272-4867** Publications of Small System Services, Inc., 900-902 Spring Garden Street, Greensboro, N.C. 27403.

The MICRO Software Catalog: XX

Mike Rowe
Box 6502
Chelmsford, MA 01824

Name: **The Designer System**
System: **Apple II or Apple II +**
Memory: **48K**
Language: **ROM APPLESOFT**
Hardware: **Apple II w/DISK II**

Description: The Designer is a HIRES graphics macro-operating system that provides the user with line and curve creation with game paddles (or Joysticks) and single keystroke ease. Lines, circles, arcs, ellipses, rectangles, areas, etc. may be quickly drawn, modified, and saved to Disk as completed or unfinished drawings. Both HIRES pages are used to provide 2 position animations. Typical uses are computer art, graphic game setups, visual presentations, and showing off. Sometimes called "the poor man's graphics tablet" this program does your complex hplotting for you. ERROR-FREE-GUARANTEE

Price: **$24.95**
Includes: Disk with DEMOS and Manual, guarantee
Author: **Jeff Johnson, Apple Jack**
Available: Your dealer or Apple Jack
12 Monterey Drive
Cherry Valley, MA 01611

Name: **ACTS**
System: **Apple II**
Memory: **32K RAM with ROM Spplesoft**
Language: **Applesoft and Machine Language**
Hardware: **Apple II, Disk II, D C Hayes Micromodem.**

Description: The Apple Communication Transfer System (ACTS) and an Apple (equipped with a disk drive, ROM Applesoft, and a C C Hayes micromodem) will transfer over the telephone Apple programs in all three languages. Exchange programs with others without leaving your home. No program modifications, self adapting and easy to use. Automaticlly stores the transfered program on the receiving Apple's disk, ready for use. The entire ACTS system, on a disketter with complete documentation, retails for only $14.50. All proceeds derived from the sale of ACTS will go toward the procurment of micro hardware for the Northeast Ohio Apple Bulletin Board System.

Copies: **30 plus**
Price: **$14.50 on disk**
Includes: System diskette and full documentation
Author: **Northeast Ohio Bulletine Board System**
Available: The NEO/ABBS
P.O. Box 4731
Clevelnd, Ohio 44126

Name: **Road Race**
System: **Apple II**
Memory: **16K min**
Language: **Integer Basic and Machine Language**
Hardware: **Game paddles or joysticks**

Description: Real-time simulation of Grand Prix Road Racing. Two players race around a 2.25 mile course, or one player races against a computer driven car. HIRES display shows through-windshield view of race course.

Price: **Cassette $15.00, disk $20**
Author: **Stan Erwin**
Available: **Stan Erwin**
5410 W. 20th Street
Indianapolis, IN
46224

Name: **Space Shuttle Landing Simulator**
System: **Apple II**
Memory: **48K**
Language: **Machine language and Applesoft**

Description: Slightly improved version of program advertised in November 1979 of MICRO. Give system config.

Copies: **250 plus**
Price: **Applesoft RAM $15.00 on cassette, Applesoft ROM $17.00 on casette, Diskette version $21.00. State which.**
Author: **John Martellaro**
Available: Harvey's Space Ship Repair
P.O.Box 3478 Univ. Park
Las Cruces, NM 88003

Name: **Restaurant Evaluation**
System: **Apple II**
Memory: **16K**
Language: **Applesoft II**
Hardware: **Disk II, Printer (both optional)**

Description: Evaluates potential restaurant/night club sites and thereby reduces the margin of risk involved in purchasing a new or existing business. All the necessary percentages and formulas are programmmed to evaluate whether a potential site will be profitable or not. The program is also structured for use by present restaurateurs to evaluate whether or not their present business is operating at cost and profit effeciency. Calculates montly gross, computes monthly loan rates (or mortgage), and reports weekly, monthly and annual net profit/loss in dollar amounts and percentages.

Copies: **25 +**
Price: **$19.95 Diskette plus $1.95 P&H First Class Mail, Check or Money Order.**
Includes: Diskette and full documentation
Author: **M. Goldstein**
Available: Mind Machine, Inc.
31 Woodhollow Lane
Huntington, N.Y. 11743

Name:	**Trace/Debug-Monext**
System:	**SYM-1**
Memory:	**2K (for cassette version)**
Language:	**Assembler**
Hardware:	**Standard SYM (w/CRT)**

Description: This program adds 15 commands to SYM's monitor including: Trace, Disassemble, Relocate, Find, ASCII dump, Stack dump, etc. The "T" command sets up its own operating environment supporting commands such as, Go, Skip, Continue, Single Step, Memory/Register examine/modify, ect. As SYM executes each instruction of the user program, an NMI is generated. IF the address of the instruction is "valid" — neither in SYM's monitor not the extension — and if it is not is a "skip" range, a disassembly/register listing is printed. This program as a whole is clean and operates transparently under SYM's OS. SASE for complete specs and examples.

Copies:	**Just released**
Price:	**Object listing** **$14.95** **Cassette** **$15.95 @$3800 or specify** **EPROM (2716)** **$49.95 @$F000 or specify** **Commented Source** **$9.95**
Author:	**Jeff Holtzman**
Available:	Jeff Holtzman 6820 Delmar-203 St. Louis, MO 63130

Name:	**LEM LANDER**
System:	**Apple II**
Memory:	**32K**
Language:	**Applesoft**
Hardware:	**Disk II**

Description: Lem Lander is a real-time version of the popular lunar lander game. This disk-based game includes nine landscapes to try your hand at landing on. Your high-resolution LEM is controlled through space via the paddle knowb (thrust) and the buttons (rotation).

Copies:	**One for you**
Price:	**$14.95**
Author:	**Barry Cox**
Available:	Barry Cox 444 Myers Avenue Harrisonburg, VA 22801

Name:	**UTIL-DS**
System:	**Apple II**
Language:	**Machine language and Applesoft**
Hardware:	**Apple II**

Description: UTIL-DS is a collection of several machine language utility routines and one Applesoft utility routine. The Applesoft utility is a sophisticated formatting routine for numeric output. The routine converts numeric values into a character string for printing. The user of the routine specifies the maximum length of the resulting string and the number of decimal places to appear in the result. Positive and negative numbers can be converted by the routine. Comma are inserted in the integer portion of the number. The machine language utilities consist of several routines to improve the error handling capabilities of Applesoft programs (e.g. resume execution at the statement following the one in error), a machine language to Applesoft interface utility, a routine to selectively clear arrays and a routine for loading machine language programs into RAM along with an Applesoft program.

Copies:	**Just released**
Price:	**$35.00 (Texas residents add 5% sales tax)**
Includes:	Routines on diskette, a sample program to demonstrate numeric formatting and documentation.
Author:	**Robert F. Zant**
Available:	Decision Systems P.O.Box 13006 Denton, TX 76203

Name:	**Dynatext Editor**
System:	**PET/CBM, ROM**
Memory:	**16K or more**
Language:	**Basic, plus machine-language repeat key**
Hardware:	**Commodore 2022 or 2023 Printer (optional)**

Description: Authorized PET version of "Context Editor", as printed in Kilobaud Magazine 5/79. Enhanced and changed in many ways for the PET. Uses cassette or disk. Has all the desirable features of most good word processors, such as global search and replace, right justification, cursor editing, etc. Plus dynamic formatiing, the ability to print in any desired shape. Holds 7 pages of text at once in a 32K PET.

Copies:	**5, Just Authorized**
Price:	**$5.00 for cassette, program and instructions.**
Author:	**James Strasma, based on work by Law & Mitchell**
Available:	Rev. James Strasma 120 West King Street Decatur, IL 62521

Name:	**Higher Graphics II**
System:	**Apple**
Memory:	**32K and disk drive**
Hadware:	**Apple I**

Description: A collection of programs and shape tables that lets any programmer create detailed and beautiful high resolution displays and animation effects. Make your programs come alive by utilizing the full graphical capabilities of the Apple II. The package contains:

Shape Maker - create shapes with this easy to use shape table generator. Start new shape tables or add to existing ones. Correct shapes as they are being produced. Delete unwanted shapes from the table. Display any/all shapes with any scale or rotation at any time.

Table Combiner - pull shapes from existing general purpose tables and add the ones you want into a new special purpose table. May combine shapes from any number of tables. All shapes can be viewed or deleted.

Screen Creator - place your shapes on the hig-res screen. Add areas of color and text to make detailed displays or game boards for high resolution games. A screen can be created in minutes with this easy to use program. Utilizes any number of shape tables and allows screen to be saved at any time.

Shapes - four shape tables with over 100 shapes are provided. Included are alphanumerics, chess figures, card symbols (club, spade, etc), tanks, planes, spaceships, ships, cars, trees, mountains, buildings, etc. Add the shapes you like to your own table.

High Res Text - how to use high resolution graphics in your program. Animation effects and display techniques.

Price:	**$24.00 Retail**
Available:	Synergistic Software 5221 120th Avenue SE Bellevue,WA 98006

Name:	**HYPNOSIS**
System:	**Apple-1 disk drive**
Memory:	**32K**
Language:	**Integer Basic**

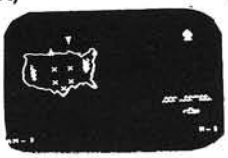
Description: Hypnosis is a program that uses Apple's video and sound capabilities to aid in suggestive relaxation, behavior modification and trance induction. Visual and auditory patterns are fully variable for shape, color and frequency matching of the subject's alpha brain wave rhythm. Designed for health professionals and students of the medical, psychological and social sciences.

Copies:	**250 plus**
Price:	**$20.00**
Includes:	Diskette, program and manual
Author:	**E.J. Neiburger**
Available:	Andent Inc. 1000 North Avenue Waukegan, IL 60085

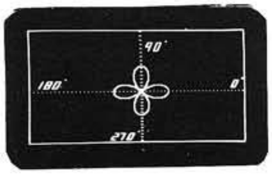
PROGRESSIVE SOFTWARE

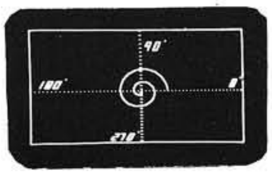
Presents
Software and Hardware for your APPLE

Missile—Anti—Missile (Aplsft) **$9.95**



Polar Coordinator Plot **$9.95**





by TD Moteles

Tables Generator forms shape tables with ease from directional vectors-starting address, length and position of each shape. Program saves shape tables anywhere in usable memory.
Murray Summers **$9.95**

Sales Forecast provides the best forecast using the four most popular forecasting techniques: linear regression, log trend, power curve trend, and exponential smoothing.Neil D. Lipson **$9.95**

Single Drive Copy is a special utility program, written by Vince Corsetti in Integer BASIC, that will copy a diskette using only one drive. It is supplied on tape and should be loaded on to a diskette. It automatically adjusts for APPLE memory size and should be used with DOS 3.2 **$19.95**

Curve Fit accepts any number of data points, distributed in any fashion, and fits a curve to the set fo points using log curve fit, exponential curve fit, least squares, or a power curve fit. It will compute the best fit, or employ a specific type of fit, and display a graph of the result. By Dave Garson.**$9.95**

Touch Typing Tutor teaches typing. Indicates speed and errors made. Finger Bldrs, Gen. Typing, Basic Language and User Supplied. Diskette. Written by Wm. A. Massena. **$19.95**

Apple Menu Cookbook index-accessed data storage/retrieval program. Recipes stored, umlimited lines per entry. Easy editing. Formulated after N.Y. Times Cookbook. Other useful features included. Written by Wm. Merlino, M.D. **$19.95**

Mailing List Program maintains complete record of name, address, phone no., mailing lables accommodated parallel card or built-in printer driver,easy data entry
Diskette. 32K. **$19.95**

POSTAGE AND HANDLING

Please add $1.25 for the first item and $.75 for each additional item.

- **Programs accepted for publication**
- **Highest royalty paid**

Utility Pack combines four versatile programs by Vince Corsetti, for any memory configuration.

- **Integer to Applesoft conversion:** Encounter only those syntax errors unique to Applesoft after using this program to convert any Integer BASIC source.
- **Disk Append:** Merge any two Integer BASIC sources into a single program on disk.
- **Interger BASIC copy:** Replicate an Integer BASIC program from one disk to another,as often as required, with a single keystroke.
- **Applesoft Update:** Modify Applesoft on the disk to eliminate the heading always produced when it is first run.
- **Binary Copy:** Automatically determines the length and starting address of a program while copying its binary file from one disk to another in response to a single keystroke. **$9.95**

Solitare — Old European peg game, played by one (similar to Chinese checkers). Object — to finish with last peg in center.
Charles B. Smith **$9.95**

Water The Flowers — Math (add., subt., mult., div,.) grades 1-6. (disk)
Judy Pegg **$19.95**

Catch The Pig — teaches the cartesian coordinate system.Educ. game, upper grade school level, 1-4 students, many levels of play. Disk or tape.
Judy Pegg **$16.95** (disk) **$9.95** (tape)

Blockade lets two players compete by building walls to obstruct each other. An exciting game written in Integer BASIC by Vince Corsetti. **$9.95**

Saucer Invasion, Space Maze, Starwars, Rocket Pilot:
Written by Bob Bishop **Each $9.95**

Hardware

Light Pen with seven supporting routines. The light meter takes intensity readings every fraction of a second from 0 to 588. The light graph generates a display of light intensity on the screen. The light pen connects points that have been drawn on the screen, in low or high resolution, and displays their coordinates. A special utility displays any number of points on the screen, for use in menu selection or games, and selects a point when the light pen touches it. The package includes a light pen calculator and light pen TIC TAC TOE. Neil D. Lipson's programs use artificial intelligence and are not confused by outside light. The HIRES light pen, only requires 48K and ROM card **$34.95**

TO ORDER

Send Check or Money Order to:

P.O Box 273
Plymouth Meeting, PA 19462

PA residents add 6% sales tax.

U.S. and foreign dealer and distributor inquires invited
All programs require 16K memory unless specified

6502 Bibliography: Part XX

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

594. MICRO 17 (October 1979)

Peck, Robert A., "SYM-1 6532 Programmable Timer," pgs. 55-56.
The 6532 programmmable timer is useful as a backup timer or as a loop controller.

Kintz, Robert T., "A Real-Time Clock for OSI Disk Systems," pgs. 59-60.
Dial, William R., "6502 Bibliography: Part XIII," pgs. 61-62.
About 70 more references on the 6502.

595. Applesauce 1, No. 1 (March 1979)

Ohrbach, Jeffrey,"The 6502 is Stacked," pgs. 5-6.
Discussion toward understanding those OP-Codes affecting the machine stack of the Apple.

Wigginton, Randy, "Apple's Bootleg Assembler," pgs. 11-14.
Documentation for the program 'Randy's Text Editor and Weekend Assembler (Ted II)."

596. Applesauce 1, No. 2 (April 1979)

Hyde, Randall, "Loaded and Gone," pgs. 5-6.
How to make your own "load and go" tapes for the Apple.

Emrich, Dick, "Simple Sort," pg. 8.
A simple bubble sort for the Apple.

Curtis, Frank, "Numerical Control Programming," pgs. 11-12.
Related to the control of machine tools.

Baker, Dwight, "The Assembly Line," pgs. 15-16.
A tutorial using Randy's Assembler.

Anon., "Docu-Prog: Lazer's Text Editing System," pgs. 16-18.
Documentation for the Lazer Text Editor.

597. Applesauce 1, No. 3 (May, 1979)

Walls, William, "Yes, Virginia, You CAN Save Data," pgs. 4-6.
A tutorial on Data Storage Methods.

Hyde, Randy, "The Apple Monitor—Part 1," pgs. 8-9.
Dealing with the many uses of the Apple Monitor.

Irvine, Al, "Docu-Prog: Program Compare," pgs. 10-11.
Documentation for PROGRAM COMPARE which compares the listings of two programs.

Paymar, Dan, "A Disc Write Protect/Enable Override Switch," pg. 12.
A simple hardware modification for your Apple Disc.

Anon., "Charts," pgs. 14-16.
Apple II ASCII CHART; Apple Integer Basic HEX Representation-Numerical Order; Hex Representation Chart-Alphabetical; Vector Table Address Chart.

598. Applesauce 1, No. 4 (June 1979)

Curtis, Frank, "Dr. Memory—A New Text Processor," pgs. 5-6.
A review.

Hyde, Randall, "Single Disk Text File Transfer," pg. 8.
Transfer a text file with only one disk drive.

Anon., "LISA: What is an Interactive Assembler Anyway?" pgs. 8-9.
A review of LISA.

Lu, Ron, "Docu-Prog: Beneath Apple Manor," pgs. 10-11.
Documentation and Review of "Beneath Apple Manor," an adventure type game.

Anon., "Apple II Mini-assembler F666G," pg. 12.
How to use the mini-assembler.

Haefner, Mike S., "Processor Status Register 'P'," pgs. 13-16.
Discussion of the processor status register and chart of flags.

Spurlock, Loy, "Hex Representation Chart," pg. 13.
Discussion of the Apple Token Chart.

Anon., "Zero Page Chart," pg. 14.
Discussion of the contents of the Apple page zero.

Anon., "Apple II 6502 OP CODE Chart," pg. 15.
Chart showing the codes for different types of addressing on the 6502.

Anon., "Integer Basic Internals," pg. 20.
List of variables used by basic; integer basic routines.

Anon., "Useful Basic Pointers," pg. 22.
Discussion and list of addresses.

Tognazzini, Bruce, "Apple Basic Interpreter Instruction Set," pg. 22.
Names and addresses.

Anon., "Apple DOS Symbol Table," pg. 27.
Names and addresses.

599. Applesauce 1, No. 5 (July/August 1979)

Lemens, Vernon Jr., "Superchip Eccentricities," pgs. 8-13.
A tutorial on the Superchip for the Apple.

Anon., "New Product Releases," pgs. 10-11.
Description of the Dan Paymar lowe case chip, the Universal Parallel card, the Apple bulletin board system, etc.

Anon., "Original Apple Corps' Program Library," pgs. 16-17.
Program library.

Lenz, John and Spurlock, Loy, "From the Forum," pgs. 20-22.
A few Calls and JSR's for the Apple.

Diay, Robert, "The Assembly Line: Page List Program," pgs. 23-24.
Page List is a machine language routine to list twenty lines at a time.

Amromin, Joel L., "Apple II Slowlist," pgs. 24-25.
Paddle-controlled slowlist program.

Hyde, Randy, "UCSD PASCAL," pg. 29.
A short tutorial on PASCAL for the Apple.

600. Byte 4, No. 11 (November 1979)

Partyka, Dave, "Shape Table Conversion for the Apple II," pg. 63.
Hints for using shape tables.

Govind, P.K., "Interfacing the PET to a Line Printer," pgs. 98-102.
Interfacing the 8-bit user port on the PET 2001, including software program "PRINTSCREEN."

601. Creative Computing 5, No. 11 (November 1979)

Heuer, Randy, "PET Software from Creative Software," pg. 46.
Five programs are reviewed.

Daro, Paul, "A Home Control System," pgs. 54-59.
Description of the use of the Introl X/10 system marketed by Mountain Hardware.

Yob, Gregory, "Personal Electronic Transactions," pgs. 183-185.
This month's column discusses Software in ROM; some graphics gizmos; animation routines, etc.

602. Stems from Apple 2, Iss. 10 (October 1979)

Hoggatt, Ken, "Ken's Korner," pg. 3.
A short tutorial on Pascal with sample routines.

Keyes, Pat, "ROM Applesoft II Vector Chart," pgs. 4-5.
Machine language subroutines and keywords.

Newman, Will II, "File Handler," pgs. 6-9.
Routine for handling files.

603. MICRO, No. 18 (November 1979)

Wells, George, "Dual Tape Drive for SYM-1 Basic," pgs. 5-7.
Make your SYM-1 Basic work with two tape recorders and manage tape cassette files.

Murphy, S. R., "Some Useful Memory Locations and Subroutines for OSI Basic in ROM," pgs. 9-10.
Where some important subroutines reside in OSI Basic.

Hawthorne, Alan R., "A Tape Indexing System for the PET," pgs. 11-13.
Rapid indexing of the PET cassette.

Swanson, Mark, "Subroutine Parameter Passing," pgs. 14-15.
A technique to facilitate passing parameters to subroutines.

Bishop, Bob, "Apple II Hires Picture Compression," pgs. 17-24.
How to put several times as many hires slides on a single disk as previously done. Used in Bob Bishop's Super Slide Show.

Floeter, Alan D., "Assembly Language Applesoft Renumber," pgs. 27-29.
A very fast renumber program.

Bruey, Alfred J., "Performing Math Functions in Machine Language," pgs. 30-31.
Math for the KIM-1.

Hooper, Philip K., "TSAR: A Time Sharing Administrative Routine for the KIM-1," pgs. 35-41.
Tsar is a super monitor which supports time-sharing for the KIM.

Dennis, Jim, "Interfacing the CI-812 to the KIM," pgs. 43-44.
The Percom CI-812 contains a full-duplex data terminal interface and a fast cassette (2400 baud).

Hill, Alan G., "Ampersort," pg. 45.
A corrected listing for this fast sort routine.

Leary, Richard A., "SYM-1 Baudot TTY Interface," pgs. 49-54.
Teletype with your SYM-1.

Rowe, Mike (staff), "The MICRO Software Catalog: XIV," pgs. 55-56.
Six new programs for 6502 devices are reviewed.

Irwin, Paul, "Alarming Apple," pgs. 59-60.
Teach your Apple to respond to errors with an alarm and keyboard lockout.

Dial, William R., "6502 Bibliography: Part XIV," pgs. 61-62.
Fifty-five more references in the 6502 literature.

604. Control Engineering (October 1979)

Faust, Gregory, "Programmable Controller Offers Fiber Optic Data Link for Remote I/O," pgs. 53-54.
Both the fiber optic interface and the remote I/O use a 6502 microprocessor for communication purposes.

605. Rainbow 1, Iss. 9 (October 1979)

Fleming, Jim, "Updating the ARESCO DMS," pgs. 7-16.
An improved file management system.

Vermehr, Jay, "An Apple Recorder Box," pgs. 19-20.
A very useful box to use between the Apple and the tape-deck.

Anstis, Stuart M., "Rocket Lander in Lo-Res Graphics," pgs. 21-24.
A game in Apple II Applesoft.

606. The Paper 2, Iss. 8 (October 1979)

Wachtel, A., and Szepesi, Z., "Pythagorean Triplets Revisited," pgs. 12-17.
A program for the PET.

Costarakis, Dennis A., "A Screen Dump Subroutine for Use with the 2022/2023 Printers," pgs. 23-24.
Routine for the PET.

607.Personal Computing 3, No 12 (December 1979)

Forbes, John L., "Applesoft Conversions," pg. 10
Conversion of a program in PT Basic to Applesoft Basic.

Whack, Margaret, "Create Your Own Periodical Guide," pgs 69-70.
Catalog your favorite computer magazine articles.

608. Recreational Computing 8, No 3 (Nov/Dec 1979)

Carpenter, Chuck, "Apple II's Three M's," pgs. 28-31.
More on Memory, Monitor and Machine language.

Bruey, Alfred J.,"Making Music on the PET," pgs. 49-51.
Software and hardware for a musical PET.

609. Kilobaud Microcomputing No 36 (December 1979)

Lindsay, Len, "Pet-Pourri," pg. 12.
Discusses Commodore Word Processor and Printer, Games, Graphics programs.

Lancaster, Don, "Lowercase for Your Apple II," pgs. 34-42.
Now for the software to complete this two-part article.

Haehn, Lou, "Chess I for the Apple II," pgs. 46-52.
Writing a chess program for the Apple II.

Lary, Richard A.,"Reverse Video from OSI's 540 Board," pgs. 128-129.
Black on white to enchance graphics displays.

David, D. J.,"PET's Machine Language Monitor," pgs. 134-140.
A review and analysis of the PET Monitor.

610. Call Apple 2, No 8 (October 1979)

Cahill, Gerald, "Auto Number," pgs. 5-8.
A program to automatically number Applesoft programs as they are input.

Foote, Gary A., "Multiple Disk Catalog," pg. 11.
A special program to read catalogs off Apple disks for entry into File Cabinet.

Merchant, Fred, "Instructions for Running Multiple Disk Catalog," pg. 13.
Detailed instructions for running Diskcat.

Barnes, Keith Allen, "File Cabinet Improvements," pgs. 13-14.
Fixes to make this program work better on the Apple.

Golding, Val J., "So Who Needs Applesoft (Revisited)," pgs. 15-16.
A tutorial for the Apple including INT BASIC VAL FN SUBROUTINE.

Corsetti, Vince, "Self Writer," pg. 18.
This program performs the equivalent of an exec file without using DOS.

Reynolds, Lee, "Logical Variables in Apple II Basic," pgs. 21-22.
A tutorial.

Golding, Val J., "Exec File Shortcut," pg. 24.
A short program to create a text file that when EXEC'd would BLOAD a binary program, set HIMEM, and Run a basic program.

Golding, Val J., "Flash Card Modifications," pgs. 24-25.
How to provide different data sets from a disk to run the Flash Card Program.

Krantz, Bill, "Write -Apple," pgs. 26-27.
All about using the High Speed Serial Card with Integral Data IP 125/225 printer.

Hockenhull, James L., "Simulating a Two-Dimensional Matrix from Integer Basic," pgs. 30-31.
A new way of handling Hi-Res programs.

Alex, Steve, "Applesoft II Firmware Card Hi-Res Routines," pgs. 33-34.
How to use the hi-res graphics routines available on the Applesoft II firmware card from assembly programs.

Garson, David B., "Poor Man's PRINT—USING Program," pg. 34.
A useful utility routine.

611. Fort Worth Apple Users Group 1, No. 4, (October 15, 1979).

Meador, Lee, "Interrupt Daisy Chain," pg. 2.
An explanation of daisy chain interrupts.

Meador, Lee, "Disassembly of Dos 3.2," pg. 4-14.
Disassembly program.

612. Stems from Apple 2, Issue 11 (Nov. 1979)

Dolema, Nels, "Worm," pg. 4.
A new version of the "WORM" with paddle control.

613. Abacus II Newsletter I, Iss. 11 (Nov. 1979)

Anon, "DOS 3.2 R.H.D.," pg. 2.
Addresses and functions of DOS 3.2 for Apple.

Howard, Clifton M., "Hex/Dec Conversion Chart," pg. 5.
Poor Man's Ti programmer.

Howard, Clifton M., "Diablo Character Values,"pg. 6.
Useful table for Apple owners using the Diable printer.

Howard, Clifton M., "Interpretation of Memory," pg. 7.
Apple II Basic Integer Memory Interpretation.

Freeland, Bruce, "Plotting Algorithm," pg. 8.
Apple II page I layout and map.

Yee, David R., "Stop that Blinking Cursor,;; pg. 12.
Now you can jazz up your own programs with a custom Cursor.

Anon, "Disk Access Update," pg. 12.
A fix for a previous routine.

Staff, "Control Character Show," pg. 12.
Show up those hidden control characters.

614. The Target, (Nov/Dec 1979)

Anon, "Basic Short Cut," pg. 2-4.
This routine automatically inputs characters to the Basic interpreter thus freeing the user of some drudgery of entering programs, into the AIM 65.

Butterfield, Jim, "Inside Basic-Tokens," pg. 4-8.
Key addresses, zero page usage, ROM organization, etc. for the AIM 65.

Bressen, Steve, "Roll," pg. 9.
Program to scroll a message onto the AIM-65 display.

615. Southeastern Software Newsletter, No. 14, (Nov. 1979)

Anon, "Length and Starting Address of Binary Files," pg. 4-5.
How to find binary program addresses in your Apple.

Anon, "El Cheapo Pascal Lower/UPPER CaseWriter," pg. 5-7.
A tutorial Pascal program.

Carpenter, Chuck, "Game Paddle I/O Applications," pg. 7-9.
An informative "How to" article.

The Apple Shoppe 1, No. 4, (Nov. 1979).

Van Winkle, Don, "Hi-Res Artillery Game," pg. 4-8.
A HiRes game of artillery, shooting over random terrains at the enemy gun.

Welman, Chuck, "Easywriter—Text Editor Review," pg. 10-11.
A rather favorable review of this text editor.

Anon, "Plotting Functions," pg. 12-14.
A tutorial for the Apple.

Welman, Chuck, "A Review of 'APPLE DOC'," pg. 15.
Useful Utility to aid programmers.

Anon, "String Magic," pg. 16.
All about strings.

Smith, David E., "Language Lab; Apple Pascal," pg. 17-22.
A tutorial on Pascal with program examples.

Crouch, Bill, "Apple Text Editors,;; pg. 23.
Reviews Applecations Unlimited Version 2.2, with Apple Pie Text Editor and The Word Weaver.

Crouch, Bill, "Apple II Text Processing System," pg. 27-28.
A review of this text editor and assembler.

617. Appleseed Newsletter 1 No. 12 (Dec. 15, 1979)

Hyde, Bill, "International Apple Core," pg. 1.
The International Apple Core is being formed as an organization to provide software, information and other services to the Apple user. Officers include Val Golding, Ken Silverman, Dave Gordon, Neil Lipson, etc.

618. Apple Bits (Dec. 1979)

Geier, Bob, "Basic Errors?," pg. 3.
How to rewrite your disk commands on the Apple.

Anon, "Rumor Mill," pg. 3.
The new Radio Shack TRS-90 will probabally be based on the new Motorola 6809 microprocessor. The Apple III may choose this chip or a new chip by MOS Technology. There is an unconfirmed rumor that Heath will discontinue production of the h-8.

619. Byte 4, No. 11 (Nov. 1979)

Anon, "Free Newsletter," (Nov. 1979)
Hands On! is a free newsletter published three times a year for science and technology educators and the initial issue contains an article "A Biased Introduction to the World of the 6502 Microprocessor."

620. Fort Worth Apple User Group Newsletter, No. 5 (Nov. 15, 1979)

Cahill, Gerald, "Auto Number," pg. 1.

Meador, Lee, "DOS Disassembly," pg. 2-16.
Listing of the assembly language for the DOS 3.2. Also a detailed listing and explanation of the RWTS Routine.

Meador, Lee, "Drawer-for Hi-Res Pictures," pg. 17-19.

Hoyt, Jim, "Special Subroutine," pg. 19.
Subroutine to allow prohibited characters like commas, etc. in string inputs.

621. Contact No. 6 (Oct. 1979)

Anon, "Invisible Writing," pg. 5.
How to plot one page of graphics while displaying the other.

Anon, "DOS Update for Dual Drive Users," pg. 6.
Improve the DOS 3.2 by updating to 3.2.1 on the Apple.

Anon, "Dollars and Cents," pg. 8-9.
Formats numeric output on the Apple to a dollar and cents format.

Anon, "Restore to Line Number," pg. 9-10.
A demo of how to do a RESTORE statement to a particular line number, on the Apple II.

Anon, "What Interface Card is in this System, Anyhow?", pg. 10.
With this program, CONFIG, it is possible to tell just what interface is in a particular slot.

622. Rainbow 1, Iss. 10 (Nov. 1979)

Deardon, Dr. Hinkley W., "From the Pits," pg. 15-16.
A two bit serial interface for the Selectric typewriter.

Wachtel, A. and Szepesi, Z., "The Development of a Basic-Program," pg. 9-11.
Illustrating the many ways in which a seemingly simple programming task on the Apple can be improved.

Laudereau, Terry L., "Pokeing Machine Language from Basic," pg. 17-18.
A tutorial article on entering machine language into the Apple II.

Wagner, Roger, "Fast Moves in Applesoft," pg. 19-20.
How to use the MOVE routine present in the Monitor by calling from Applesoft using a special routine.

Wagner, Roger, "One Less Error," pg. 19-20.
How to keep Apple's Integer Basic programs from bombing when dealing with an address greater than 32767.

623. Apple Peelings 1, No. 3, (Nov. 1979)

Anon, "November DOM (Disk of the Month)," pg. 3.
A dozen good programs plus Library List 10-79, the latest listing of the SF Apple Core's complete library as of the last of October, 1979.

Johnson, Allen, "What Is It?," pg. 4.
A short routine to identify the DOS 3.1 or 3.2 on Apple diskettes.

Fisher, Frank E., "Slot ÷s as Variables," pg. 4.
Specifying I/O slots in the Apple as variables and using in programs.

624. The Paper 1, No. 3 (Nov. 1979)

Lee, Arnie, "Clocks and Timers," pg. 4.
How to use the microprocessor clock in the PET to use in a timer or real time clock.

Anon, "Screen Display and Cursor Positioning," pg. 7-9.
A short tutorial for the PET.

Anon, "To Write a Character String to the Screen," pg. 9-10.
Short tutorials on this and a number of other short PET routines.

Anon, "PET USER GROUPS," pg. 15.
A list of about 40 user groups for the PET.

Szepesi, Zoltan, "Using the Monitor Subroutines," pg. 16-21.
A tutorial to show how the PET actually runs programs.

625. Apple-Com-Post No. 4 (Nov. 1979)

Anon, "Software Tups and Tricks," (in German), pg. 7.
All about handling numbers on the Apple.

626. Call-Apple 2, No. 9 (Nov./Dec., 1979)

Greenfarb, Sandy, "Internal Structure of Integer B asic," pg. 5-10.
A tutorial on Integer Basic.

Golding, Val, "Why Variables," pgs. 11-12.
Why replace numeric constants with variables in both Apple Integer and Applesoft.

Hyde, Randall, "The Assembly Line," pgs. 14-17.
Benchmarking Sweet 16 with 6502 Assembly Language. 6502 Machine code runs 5-7 times as fast as Sweet 16 but Sweet 16 code requires only about half as much memory to perform an equivalent function, on the Apple.

Sedgewick, Dick, "Sedgewick Plays it Straight," pgs. 19-20.
A tutorial dealing with the consequences of tokenizing, etc.

Golding, Val, "Basic Memory Move," pg. 23.
A hex memory move program for the Apple, and a corresponding one in Decimal.

Cox, Ross E., "Life with an Apple," pgs. 30-34.
All about how to make Life more enjoyable by improving The Game of Life on the Apple.

Hilger, Jim, "Apple Gaming: Playing Card Generation," pgs. 39-45.
General purpose routines that will generate hi-res images of playing cards.

627. Stems from Apple 2, Iss. 12 (December 1979)

Reinhardt, John, "President's Message," pg. 1.
An announcement about the newly formed "International Apple Core Club" to which individual User Groups may subscribe.

Stein, Greg, "Circles," pg. 3.
A short circle drawing program.

Anon., "The Twelve days of Christmas," pg. 3.
A program which prints the traditional Crhristmas song.

Ward, Dennis, "Do Your Words Runneth Over?," pg. 4.
How to avoid split words, bad spacing, etc.

Doeleman, Nels, "The Appl-Ogical Way to Arrange Numbers in DEcending Order," pg. 5.
Program to arrange numbers.

Announcing The Micro LIMERICK CONTEST

You could win a free 1-year subscription of MICRO if your limerick is judged to be the best by our staff.

The American Heritage Dictionary defines a limerick as: "A light humerous or nonsensical verse of five lines with the rhyme scheme *aabba*". For those of you who may not have been English majors, here is an example:

There was a young princess from Niger,
Who smiled as she rode on a tiger.
They returned from the ride
With the princess inside
And the smile on the face of the tiger.

To win our limerick contest, all you have to do is come up with an original 5-line verse about MICRO.

Shouldn't be too tough, right? Winners will be judged on creativity, originality, and above all, humor. Remember to follow the rules of a limerick, and no cheating by using the same word to end the first, second and/or fifth lines! And please, keep them clean.

The winning limerick will be published in the September issue, along with some honorable mention runners-up.

Please send all entries to:

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

MICRO™

Index, Volume Number 3

(Issues 13-24)
(June 1979-May 1980)

PET

APPLE

OSI

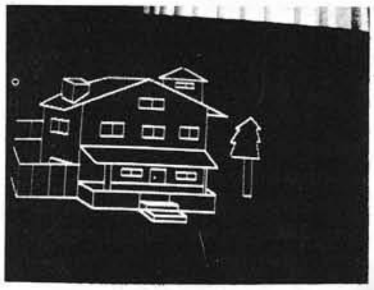
Editor's Note: Use this index as a guide to material which is of interest to you. Remember that most of the articles and features under the 'General' heading apply to 6502 systems in general. Also, many articles and programs which are classified for one system may be readily modified and/or adapted to another system.

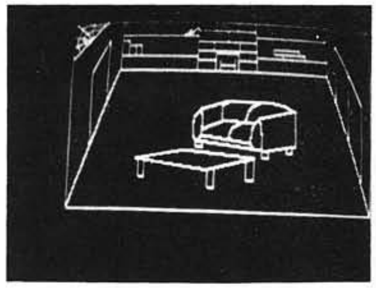
NEW APPLE II / APPLE II PLUS SOFTWARE FROM ON— LINE SYSTEMS

TIRED OF BUYING GAMES THAT BECOME BORING AFTER A FEW HOURS OF PLAY? ON—LINE SYTEMS IS DEDICATED TO DELIVERING SERIOUS SOFTWARE FOR THE DISCRIMINATING GAMESMAN. THESE PRODUCTS HAVE BEEN SIX MONTHS IN DEVELOPMENT AND PROVIDE THE QUALITY AND SPEED POSSIBLE ONLY THROUGH MACHINE LANGUAGE!

ALL NEW

HI-RES ADVENTURE (''MYSTERY HOUSE'')





What is an adventure game? According to the dictionary, an adventure is a hazardous or daring enterprise; an exciting experience; to risk, hazard, to venture on. One who goes on an adventure is a venturer. A seeker of fortune in daring enterprises; a speculator. In essence, an adventure game is a fantasy world where you are transported, via your own computer. You are the key character of the fantasy as you travel through a land the likes of which you will find in books that take you, through your imagination, to the world it is creating.

Through the use of over a hundred Hi-Res pictures you play and see your adventure. You communicate with HI-RES ADVENTURE in plain english (it understands over 300 words!) All rooms of this spooky old house appear in full Hi-Res Graphics complete with objects you can get, carry, throw, drop, or ?.

In this particular HI-RES ADVENTURE game, you are transported to the front yard of a large, old victorian house. When you enter the house you are pulled into the mystery, murder, and intrigue and can not leave until you solve the puzzles. Your friends are being murdered one by one. You must find out why, and who the killer is. Be careful, because the killer may find you! As you explore the house there are puzzles to be solved and hazards to overcome. The secret passage-way may lead you to the answer.

ALSO NEW FROM ON—LINE SYSTEMS

SKEET/TRAP have become Olympic shooting sports and and obsession among Scatter-gunners all over the world. These games are the All-American although they have become international.

SKEETSHOOT allows one to five shotgunners to test their marksmanship as they fire from the eight prescribed positions on an official NSSA skeet firing range. Each position provides a new perspective of the field with the pigeons travelling at different angles. At each position a pigeon is launched from one side of the field and then the other. At certain positions, pigeons are launched from both sides of the field simultaneously. This is a true game of skill, simulating skeet shooting down to the last detail.

TRAPSHOOT allows one to five shotgunners to test their markmanship. The trap firing range has five positions where the one to five players shoot from. Each player is at a different location on the field. The challenge is to shoot pigeons out of the sky which launch at random trajectories. The challenge is to hit the pigeons while they are still in gun range.

SKEETSHOOT and TRAPSHOOT both allow you to control the size and speed of the pigeons and the width of your shotgun spray. Realistic sound-effects and HI-RES animation combine to make this simulation unparrelled for the APPLE.

DEALER INQUIRIES INVITED!

ALL SOFTWARE SHIPPED SAME DAY.
PHONE ORDER: (805) 522-8772

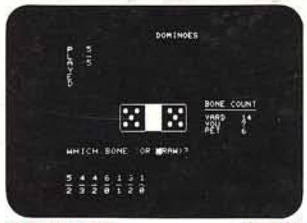
ON—LINE SYSTEMS, 772 NO HOLBROOK, SIMI, CA 93065

SEND TO: ______________________

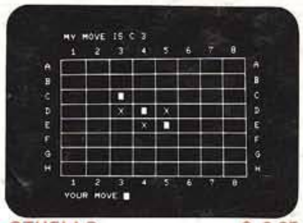
____

		QUANTITY	TOTAL
Hi-Res Adventure/Disk	$24.95		
Skeetshoot/Cassette	$14.95		
Trapshoot/Cassette	$ 9.95		
Skeetshoot/Disk	$19.95		
Trapshoot/Disk	$14.95		
Hi-Res Adventures & Skeetshoot/Disk	$37.50		
		Subtotal	
Payment: Check ______		6% tax (Calif.)	
Master Chg/Visa # ______		Shipping	$1.00
Expires: ______		Total	

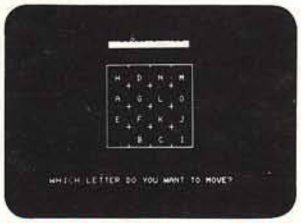
Software for the PET



DOMINOES $ 6.95

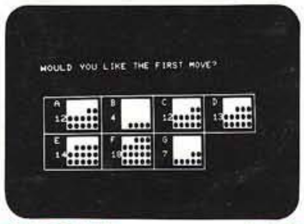


OTHELLO $ 9.95

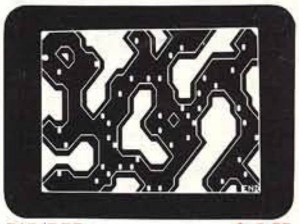


LETTER SQUARES $ 6.95

RPN MATHPACK $19.95



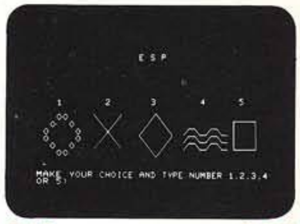
SUPER NIM $ 6.95



DIR/REF $ 6.95

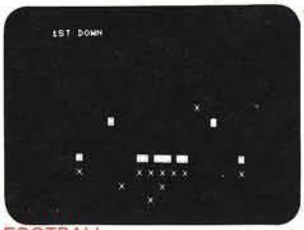
SPACE WARS $ 9.95

GRAND PRIX $ 6.95

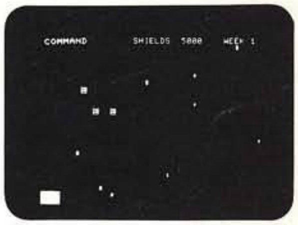


E.S.P. $ 9.95

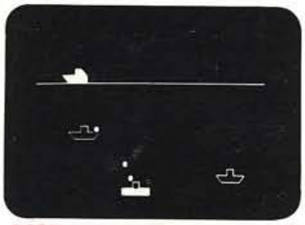
FORECAST $ 9.95



FOOTBALL $ 6.95

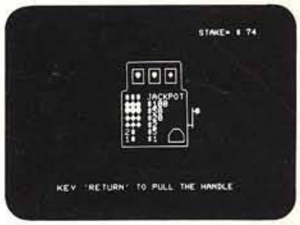


BLOCKADE $ 9.95



DEPTH CHARGE $ 9.95

BASKETBALL $ 9.95

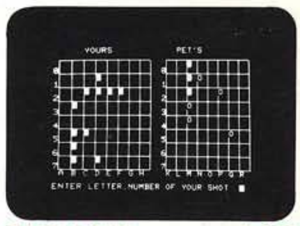


SLOT MACHINE $ 6.95

HOME ACCOUNTING $ 9.95

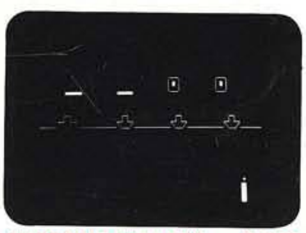
All orders include 3% postage and handling with a minimum of $1.00. California residents include 6% Sales Tax.

VISA MASTERCHARGE



BATTLE SHIP $ 9.95

PET IS A TRADEMARK OF COMMODORE BUSINESS MACHINES, INC.



SHOOTING GALLERY $ 9.95

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

Dealer Inquiries Invited

ZAP $ 6.95

PROGRAMMA Software Program Products