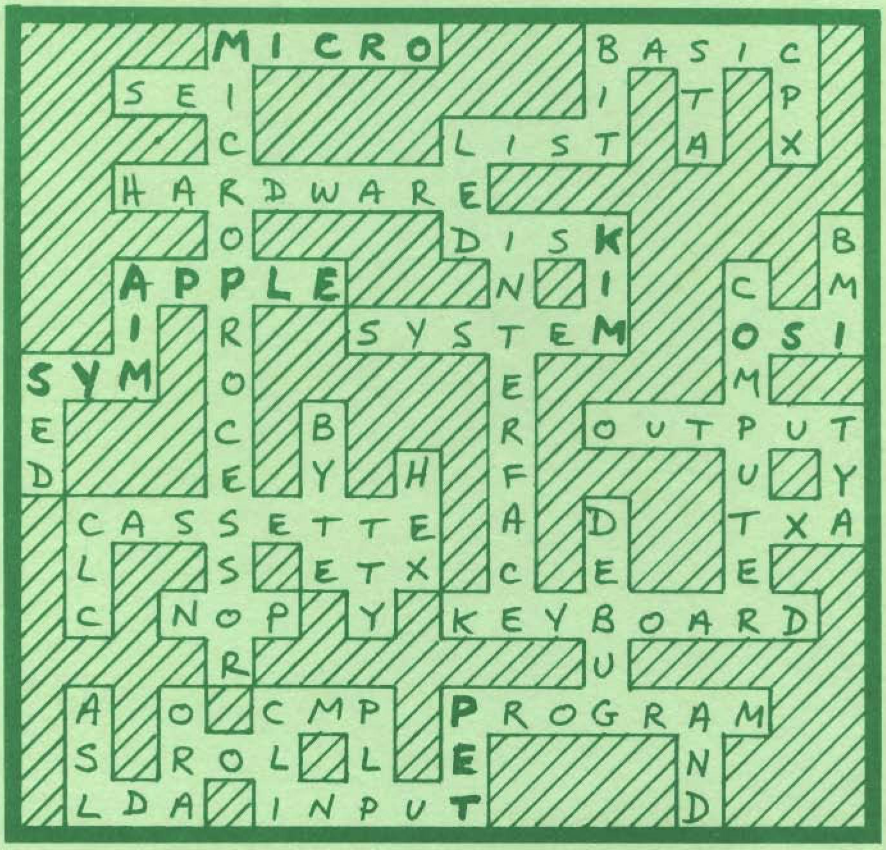
MICRO™

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



PUZZLED ABOUT THE 6502 ?

NO 9 February 79 $1.50

COMPUTER SHOP

288 NORFOLK ST. CAMBRIDGE, MASS. 02139
corner of Hampshire & Norfolk St. 617-661-2670

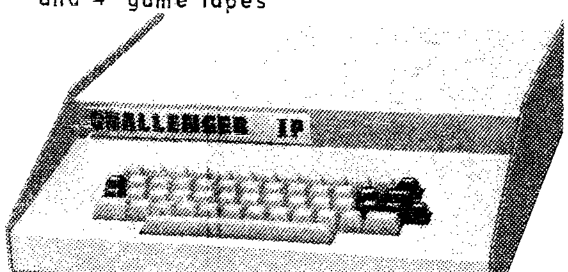
NOW WE HAVE OSI AND YOU CAN GET ONTO THE BUS

ALL-IN-ONE HOME FINANCIER, CALCULATOR, TEACHER, & VIDEO GAME MAKER AT THE WORLD'S LOWEST PRICE!

Ohio Scientific's Challenger 1P is the easy-to-use home computer. It does a lot more for a lot less! Just connect it to your TV* and a cassette player. That's all there is to it. There's nothing to build.

You can create your own personal programs. Or select from a whole library of programs on low-cost cassettes. Everything from teaching arithmetic to spectacular video games to balancing your checkbook. Something for every member of your family.

CHALLENGER 1P
SPECIAL ! ! ! $425.00 with 8K and 4 game tapes



SPECIAL ! ! ! $666.00 with 8K RAM and 4 game tapes

Why not switch to a complete 6502 System complete with an 8K ROM BASIC and 8K of RAM. It also has a keyboard and a Video Display with 30X64 display of upper and lower case alphanumerics and gaming graphics

C2-0 Board assmb. Serial I/O ROM Basic	298.00
C2-1 above with Cabinet & Pwr Supp.	429.00
C2-4P Challenger II P Kbd & Video	598.00
CM-1 4K 1MHz. Memory	125.00
CM-2 4K 2MHz. Memory	149.00
CM-3 16K 1.5 MHz. Ultra Low Pwr. Mem.	450.00
CA-7 I/O Board	299.00
480 Backplane..Motherboard	39.00
all above are OSI products available from us.	
VF8 4K Memory assembled & tested.	129.00
for low power RAM add	10.00
same in kit form	74.50
full set of sockets for Kit	10.00
VF8 Motherboard buffered for 4 Boards	65.00
Connector Assembly for KIM to VF8	20.00
8K S100 Memory Board with instructions.K	165.00
same but fully assembled and tested	199.00
CS100 Cabinet cut out for KIM	129.00
3 Connector S100 Motherboard Assembly	75.00
CGRS S100 TIM Kit	129.00
CGRS S100 6502 CPU Kit	179.00
CGRS S100 Front Panel Kit	129.00
XITEX Video Terminal Board 16X64K	155.00
XITEX Video Terminal Board Assembled	185.00
KIM-1	245.00
CS100 with CGRS, Xitex, 16KRAM, TV, KB	1529.00
Same but Assembled	1989.00
PS-5 Pwr Supp. 5V5A9V1A-12V1A6x6X2	75.00
PS-5 Assembled	90.00
Total of Order..Circle Items wanted.	$..............
Mass. Residents Sales Tax 5%	$..............
Shipping, 1%($2.00 min.)	$..............
Total Remittance or Charge	$..............

BAC, VISA, MC NO. ....

SIGNATURE ....

NAME ....

ADDRESS ....

CITY .... STATE ...... ZIP ......

MICRO™

FEBRUARY 1979
ISSUE NUMBER NINE

TABLE OF CONTENTS

MICRO

STAFF

Editor/Publisher
Robert M. Tripp

Business Manager
Donna M. Tripp

Administrative Assistant
Susan K. Lacombe

Circulation Manager
Maggie Fisher

Distribution
Eileen M. Enos

Micro-Systems Lab
James R. Witt
Steve Cahill

Gofer
Fred Davis

MICRO™ is published monthly by:

The COMPUTERIST, Inc.
P.O. Box 3
So. Chelmsford, MA 01824

Controlled Circulation postage paid at:

Chelmsford, MA 01824

Publication Number: COTR 395770

Subscription in US: $12.00/12 Issues

Advertiser's Index

Please address all correspondence, subscriptions and address changes to:

MICRO, P.O. Box 3, So. Chelmsford, MA 01824

IN THIS ISSUE ...

Dr. L. S. Reich, who last month challenged us with his "Computer-Determined Kinetic Parameters in Thermal Analysis", this month presents a program which may have more general use, a program to calculate "Long Distance Interstate Telephone Rates" (page 5). While this program is written in Applesoft BASIC II, it can be easily modified to work on a PET, OSI computer, and so forth. As the example shows, it comes up with pretty accurate results.

For those readers whose sleep is troubled at night because their mind is kept busy trying to discover all of the prime numbers, we present "The Sieve of Eratosthenes (page 8), by Gary J. Bullard. This BASIC program allows your PET to do the work while you get some rest. On the serious side, this fairly simple program may provide you some insight into solving similar problems.

Those Apple II owners who have the DOS, and those who are considering getting it, will find a lot of useful information about it in Andy Hertzfeld's "Exploring the Apple II DOS" (page 9). This article gets right to the heart of the system and describes the Command Table and the Important Address in a couple of tables.

In what may, at least for a while, be the last of his excellent series, Marvin L. De Jong shows how to interface "An ASCII Keyboard Input Port" (page 11) to a 6502 system. The programs show both polled and interrupt methods of servicing a device.

While most readers have "ready-to-go" systems, from the KIM-1 through the Apple II with disk, some hardy souls still insist that they would rather "do it myself". Even if you are not of this persuasion, you can learn a lot of techniques from these guys (and gals). Gary L. Tater is one of this breed, and presents "Two Short TIM Programs" (page 14).

While the KIM-1 is the uncontested "grand-daddy" of the 6502 family, it has been joined recently by two new members of the family who have a lot in common with it. The AIM 65 and the SYM-1 are similar to the KIM in many ways, but important differences do exist between them, some subtle and some not. "ASK the Doctor" (page 17) is the first in a series of articles by Robert M. Tripp which explore these three systems and detail the commonalities and differences.

Many Apple II users are quite content to do all of their programming in BASIC. If, however, you want to do assembly level programming with a full feature assembler, you have at least two choices at present. These are presented and evaluated by Allen Watson in "Two Apple II Assemblers: A Comparative Review" (page 19). In addition to discussing the two particular packages, this article provides a very good analysis of what features an assembler should provide. Both companies whose assemblers are being discussed were set copies of the article for their comment. S-C Software responded and described a disk based version of their package which will be available about the time this issue of MICRO "hits the streets". Microproducts did not have any response.

One of MICRO's most popular features, according to responses from our Reader Feedback Survey, is "The MICRO Software Catalog" (page 23). Finally the contributors are getting smart, and are submitting their material in the proper format. We have a policy of first using all of the material received in the proper format before even looking at other "news releases" and general descriptions.

Russell Rittimann really gets into his TIM system and makes it do what he wants it to, and then shows how to "Expand Your 6502-Based TIM Monitor" (page 26). Some very clever ideas are presented, so do not skip this just because it deals with a "homebrew" system.

William R. Dial continues to scan the growing volume of literature and provide us with the "6502 Bibliography, Part VIII" (page 29).

One of the first stumbling blocks encountered by the novice computerist is the Hex/Decimal stuff. Gary P. Sandberg asks "How Does 16 Get You 10" (page 32) and shows a couple of ways to make the conversions.

While it is nice to assume that your ROM, since it is a ROM, never changes any values, the possibility of a ROM going bad is real. Harvey B. Herman asks "How Goes Your ROM Today" (page 35) and provides programs and techniques for testing the KIM and PET ROMs. These methods can be readily adapted to other systems.

Having presented LIFE for the Apple and the PET in earlier issues of MICRO, we now present a version for the KIM-1 (and its relatives the AIM and SYM - with some modifications required) in "Life for the KIM-1 and an Xitex Video Board" (page 39) by Theodore E. Bridge.

With this issue, MICRO takes two giant steps forward. The first giant step is that from this issue on, MICRO will be published MONTHLY, instead of bi-monthly. Subscribers will receive as many issues as they have paid for, they will just come more often. The new annual subscription rate is $12.00 per year in the US. This step is being taken because we are receiving so much good material that a significant backlog has begun to develop. Also the monthly format will permit us to present timely announcements about clubs, courses, demonstrations, and the like which were not included earlier due to the two plus month lag between receipt of an item and the publication of the item.

The second giant step, is that MICRO is now being professionally typeset (except for minor items like the Table of Contents and In This Issue). This will, hopefully, accomplish two things: reduce the number of typographical errors and improve the overall readability of MICRO.

LONG DISTANCE INTERSTATE TELEPHONE RATES

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

This program estimates long distance USA interstate telephone rates (prior to taxes) with the exceptions of Alaska and Hawaii. These rates became effective as of Sept. 13, 1977. Because of rounding by the telephone company (Bell System), the rates arrived at in this program may be slightly more than the actual rates before taxes. Charges are based on the rate that is in effect at the place where the phone call originated.

Long distance interstate rates are based on several factors which are accounted for in this program. Thus, rates increase as the duration of the phone conversation increases and as the mileage between phones increases. Also, rates vary according to whether the dialing is direct or operator assisted. In the former case, weekday rates (Mon.-Fri., 8 AM-5 PM) are higher than evening rates (Sun.-Fri., 5 PM-11 PM) which, in turn, are higher than night or weekend rates (11 PM-8 AM or all day Sat. to 5 PM Sun.). In the latter case, station-to-station rates are much less than person-to-person rates.

If the program user lacks knowledge of the mileage between two cities involved in a phone conversation, this program allows the user to estimate the air line mileage between the two cities. Thus, three categories of cities are given, - cities in a westerly direction from Boston, cities in a southerly direction from Boston, and cities in a southwesterly direction from Boston (Boston is used as the base for mileage estimates). Obviously, all U.S. cities cannot be stored in memory. However, key cities are listed and can be used for mileage estimates. For example, suppose the distance (air line statute miles) between So. Chelmsford and Tucson is desired. The key cities one would now employ would be Boston and Phoenix to yield an estimate of about 2365 miles (a more accurate value is about 2320 miles).

The program requires about 7-8K bytes and is written in Applesoft Basic II. Explanatory REM statements are to be found in line #'s 24, 27, 35, 97, 98, 250, 410.

Editor's Note: This BASIC program should work on any system which supports a floating-point BASIC, the PET and most OSI systems for example, with at most only trivial modification required.

```
5    PRINT "THIS PROGRAM ESTIMATES LONG DISTANCE INTERSTATE TELEPHONE CHARGES(BE-
     FORE TAX).  ALASKA AND HAWAII ARE NOT INCLUDED.";
7    PRINT "THESE RATES ARE EFFECTIVE AS OF 9/13/77.": PRINT
10   DIM M (20),WD(20,3),E(20,3),N(20,3),SS(20),PP(20),DS$(24),DW$(38),SW$(24)
20   INPUT "GIVE THE LENGTH OF THE 'PHONE CONVERSATION (MIN.):";T
21   FOR J=1 TO 24: READ DS$(J): NEXT
22   FOR J=1 TO 36: READ DW$(J): NEXT
23   FOR J=1 TO 24: READ SW$(J): NEXT
24   REM #'S 21-23 YIELD 1 -DIMENSIONAL ARRAYS OF VARIOUS CITIES & CORRESPONDING
     MILEAGE WITH BOSTON AS BASE USING DATA #'S 420-480
25   PRINT: INPUT "IF U DON'T KNOW THE MILEAGE BETWEEN 'PHONES & WANT THE APPROX.
     DISTANCE, TYPE 'KNOW'; OTHERWISE, TYPE 'SKIP':";K$
27   REM #1000 ALLOWS DETN. OF DISTANCE BETWEEN 2 SELECTED CITIES
28   IF K$="KNOW" THEN GOSUB 1000
30   PRINT : INPUT "GIVE THE MILEAGE BETWEEN 'PHONES:"; M
35   REM #'S 40-95 DETN. MODE OF DIALING & WHEN CALL WAS MADE
40   PRINT: INPUT "WAS THE DIALING DIRECT (DD) OR WAS IT OPERATOR ASSISTED
     (OA):";D$
50   IF D$<>"DD" THEN 80
60   PRINT: PRINT "IF DIALING WAS DIRECT THEN DID IT OCCUR DURING A WEEKDAY (MON.-
     FRI., 8 AM-5 PM) (WD) OR DURING ";
63   PRINT "THE EVENING (SUN.-FRI.,5 PM-11 PM) (E) OR DURING THE NIGHT OR WEEKEND
     (11 PM-8 AM OR ALL DAY SAT. TO 5 PM SUN.) (N):";
65   INPUT W$
70   GOTO 100
80   PRINT: INPUT "IF DIALING WAS OPERATOR ASSISTED THEN DID IT OCCUR FROM STATION-
     STATION (SS) OR PERSON-PERSON (PP):";F$
85   PRINT: PRINT "DID OPERATOR ASSISTANCE OCCUR DURING A WEEKDAY (MON.-FRI.,8 AM-
     5PM) (WD) OR DURING ";
90   PRINT "THE EVENING (SUN.-FRI.,5 PM-11PM) (E) OR DURING THE NIGHT OR WEEKEND
     (11 PM-8 AM OR ALL DAY SAT. TO 5 PM SUN.)(N):";
95   INPUT W$
```

```
97  REM #'S 100-240 STORE IN ARRAYS THE FOLLOWING, RESP., MILEAGE, WEEKDAY RATES
    (1ST & ADDNL. MIN.),EVENG. RATES(1ST & ADDNL. MIN.), NIGHT RATES(1ST & ADDNL.
    MIN.), STATION-STATION RATES (1ST 3-MIN.),PERSON-PERSON (1ST 3-MIN.)
98  REM #'S 100-240 USE DATA STATEMENTS 500-550
100 FOR J=1 TO 14: READ M(J): NEXT
110 FOR J=1 TO 14
120 FOR K=1 TO 2: READ WD(J,K): NEXT K,J
140 FOR J=1 TO 14
150 FOR K=1 TO 2: READ E(J,K): NEXT K,J
170 FOR J=1 TO 14
180 FOR K=1 TO 2: READ N (J,K): NEXT K,J
200 FOR J=1 TO 14
210 READ SS(J): NEXT
230 FOR J=1 TO 14
240 READ PP(J): NEXT
250 REM #'S 260-350 DETN. MILEAGE RANGE, TYPE & TIME OF DAILING & CORRESPONDG.
    CHARGES (BEFORE TAX) FOR T-MIN.
260 FOR J=1 TO 14
270 IF M < = M(J) THEN 290
280 NEXT J
290 IF D$<>"DD" THEN 330
300 IF W$="WD" THEN PRINT: GOSUB 400: PRINT (WD(J,1)+INT(T-.1)*WD(J,2))/100: STOP
310 IF W$="E" THEN PRINT: GOSUB 400:PRINT (E(J,1)+INT(T-.1)*E(J,2))/100: STOP
320 IF W$="N" THEN PRINT: GOSUB 400, PRINT(N(J,1)+INT(T-.1)*N(J,2))/100: STOP
330 IF F$<>"SS" THEN 340
332 IF F$="SS" THEN PRINT
333 IF T<=3 THEN T=3
335 IF W$="WD" THEN SS2=WD(J,2)
336 IF W$="E" THEN SS2=E(J,2)
338 IF W$="N" THEN SS2=N(J,2)
339 GOSUB 400: PRINT (SS(J)+INT(T-2.1)*SS2)/100: STOP
340 IF T<=3 THEN T=3
342 IF W$="WD" THEN PP2=WD(J,2)
344 IF W$="E" THEN PP2=E(J,2)
346 IF W$="N" THEN PP2=N(J,2)
350 PRINT: GOSUB 400: PRINT (PP(J)+INT(T-2.1)*PP2)/100: STOP
400 PRINT "THE 'PHONE CHARGES (NO TAX)=$";: RETURN
410 REM #'S 420-480 ARE DATA STATEMENTS OF CITIES & CORRESPONDING MILEAGE (BOSTON=
    BASE)
420 DATA BOSTON, 0, N.Y.C., 188, PHILADELPHIA, 268, BALTIMORE, 358, WASHINGTON D.C.,
    392, RICHMOND, 471, NORFOLK, 467, ATLANTA, 933, BIRMINGHAM, 1052, NEW ORLEANS,
    1359, JACKSONVILLE, 1015, MIAMI, 1288
440 DATA BOSTON, 0, N.Y.C., 188, BUFFALO, 398, PITTSBURGH, 478, CLEVELAND 580, CIN-
    CINNATI, 767, DETROIT, 653, CHICAGO, 890, ST.LOUIS, 1066, KANSAS CITY, 1250,
    DES MOINES, 1200, OMAHA, 1310, FARGO, 1384, DENVER, 1806, SALT LAKE CITY, 2050,
    MINNEAPOLIS, 1185
460 DATA SAN FRANCISCO, 2760, INDIANAPOLIS, 837
480 DATA BOSTON, 0, N.Y.C. 188, LOUISVILLE, 843, NASHVILLE, 941, MEMPHIS, 1133,
    OKLAHOMA CITY, 1530, SHREVEPORT, 1410, DALLAS, 1551, ALBUQUERQUE, 2037, EL PASO,
    2100, PHOENIX, 2365, LOS ANGELES, 2660
500 DATA 10, 16, 22, 30, 40, 55, 70, 124, 196, 292, 430, 925, 1910, 3000
510 DATA 19, 9, 23, 12, 27, 14, 31, 18, 35, 21, 39, 25, 41, 27, 43, 29, 44, 30, 46,
    32, 48, 34, 50, 34, 52, 36, 54, 38
520 DATA 12, 6, 14, 8, 17, 10, 20, 12, 22, 14, 25, 17, 26, 18, 27, 19, 28, 20, 29,
    21, 31, 23, 32, 23, 33, 24, 35, 25
530 DATA 7, 4, 9, 5, 10, 6, 12, 8, 14, 9, 15, 10, 16, 11, 17, 12, 17 12, 18, 13, 19,
    14, 20, 14, 20, 15, 21, 16
540 DATA 45, 60, 80, 100, 110, 135, 160, 175, 185, 195, 200, 205, 215, 225
550 DATA 145, 160, 180, 200, 210, 235, 260, 275, 285, 295, 305, 315, 330, 355
```

```
1000 PRINT: PRINT "THE FOLLOWING CITIES ARE SOUTH OF BOSTON: N.Y.C., PHILA, BALT,
     WASH D.C., RICHMOND, NORFOLK, ATLANTA, NEW ORELEANS, BIRMINGHAM, JACKSONVILLE,
     MIAMI.  IF U ARE INTERESTED IN ANY 2 CITIES, NOTE THE CITIES & THE CODE 'DS'."
1050 PRINT: PRINT "THE FOLLOWING CITIES ARE WEST OF BOSTON: N.Y.C., BUFFALO, PITTS-
     BURGH, CLEVELAND, CINCINNATI, DETROIT, INDIANAPOLIS, CHICAGO, ST. LOUIS, KAN-
     SAS CITY, DES MOINES, OMAHA, FARGO, DENVER, SALT LAKE CITY, MINNEAPOLIS, SAN
     FRANCISCO.";
1100 PRINT" IF U ARE INTERESTED IN ANY 2 CITIES, NOTE THE CITIES & CODE 'DW'."
1150 PRINT: PRINT "THE FOLLOWING CITIES ARE SOUTHWEST OF BOSTON: N.Y.C., LOUISVILLE,
     NASHVILLE, MEMPHIS, OKLAHOMA CITY, SHREVEPORT, DALLAS, ALBUQUERQUE, EL PASO,
     PHOENIX, LOS ANGELES.";
1200 PRINT" IF U ARE INTERESTED IN ANY 2 OF THESE CITIES, NOTE THE CITIES & THE CODE
     'SW'."
1250 PRINT: INPUT "TYPE IN ORDER 2 CITIES & CODE (ABBREV. CITIES,-NO'.', EXCEPT
     N.Y.C.): "; C$(1), C$(2),CN$
1300 IF CN$="SW" THEN 1500
1305 IF CN$="DW" THEN 1400
1310 FOR J=1 TO 2
1315 FOR K=1 TO 23 STEP 2: IF C$(J)=MID$(DS$(K), 1, LEN(C$(J))) THEN CC=CC+1: CT(J)=
     VAL(MID$(DS$(K+1),1)): GOTO 1330
1320 NEXT K
1330 NEXT J
1335 GOTO 1900
1400 FOR J=1 TO 2
1415 FOR K= 1 TO 35 STEP 2: IF C$(J)=MID$(DW$(K), 1, LEN(C$(J))) THEN CC=CC+1:
     CT(J)=VAL(MID$(DW$(K+1), 1)): GOTO 1430
1420 NEXT K
1430 NEXT J
1435 GOTO 1900
1500 FOR J= 1 TO 2
1515 FOR K= 1 TO 23 STEP 2: IF C$(J)=MID$(SW$(K), 1, LEN(C$(J))) THEN CC=CC+1:
     CT(J)=VAL(MID$(SW$(K+1), 1)): GOTO 1530
1520 NEXT K
1530 NEXT J
1900 IF CC=2 THEN PRINT: PRINT "DISTANCE IN MILES=CA. "ABS(CT(1)-CT(2)): GOTO 2000
1950 PRINT: PRINT "THE 2 CITIES U CHOSE WEREN'T IN THE SAME CATEGORY LISTED --- TRY
     AGAIN!": PRINT: PRINT "PRESS 'CONT' TO CONTINUE!": END: CC=0: GOTO 1000
2000 RETURN
```

Program Example

A telephone call was made from W. Orange, NJ to San Francisco at 11:47 PM using operator assistance (station-to-station) and the conversation lasted 6 minutes. What is the charge (before tax)?

```
COMMAND:   RUN --> STATEMENTS 5, 7, AND "GIVE THE LENGTH OF THE 'PHONE CONVERSATION
           (MIN.):"
RESPONSE:  6--> "IF U DON'T KNOW THE MILEAGE BETWEEN 'PHONES & WANT THE APPROX.
           DISTANCE, TYPE 'KNOW'; OTHERWISE, TYPE 'SKIP':"
RESPONSE:  KNOW--> STATEMENTS 1000-1200, AND "TYPE IN ORDER 2 CITIES & CODE (ABBREV.
           CITIES,--NO '.', EXCEPT N.Y.C.):"
RESPONSE:  N.Y.C., SAN FRAN, DW --> "DISTANCE IN MILES =CA.2572  GIVE THE MILEAGE
           BETWEEN 'PHONES:"
RESPONSE:  2572 --> "WAS THE DIALING DIRECT (DD) OR WAS IT OPERATOR ASSISTED (OA):"
RESPONSE:  OA --> "IF DIALING WAS OPERATOR-ASSISTED THEN DID IT OCCUR FROM STATION-
           STATION (SS) OR PERSON-PERSON (PP):"
RESPONSE:  SS --> "DID OPERATOR ASSISTANCE OCCUR DURING A WEEKDAY (MON.-FRI., 8 AM-
           5PM) (WD) OR DURING THE EVENING (SUN.-FRI., 5PM- 11PM) (E) OR DURING THE
           NIGHT OR WEEKEND (11PM- 8AM OR ALL DAY SAT. TO 5PM SUN.) (N):?"
RESPONSE:  N --> "THE 'PHONE CHARGES (NO TAX) = $2.73"
```

The actual company charge (before tax) was $2.70.

THE SIEVE OF ERATOSTHENES

Gary J. Bullard
1722 S. Carson, #1502
Tulsa, OK 74119

Over 2000 years ago, a Greek geographer-astronomer named Eratosthenes devised a way of finding prime numbers that is still the most effective known. He simply started with the number 2 and crossed out all multiples of 2. Then he took the next number that had not yet been crossed out (3) and proceeded to cross out all multiples of it. And so on until he had found all the prime numbers he was interested in. This method of finding prime numbers is called a "sieve" because the prime numbers fall through the holes created by crossing out all the non-prime numbers.

So what? Well, this gives rise to an interesting program for the PET. Picture the 1000 character positions on your PET's screen as the numbers 1 to 1000. Now cross out all the positions that represent non-prime numbers. What you have left is a strange pattern that would make an interesting bathroom tile arrangement. It also shows the placement of the prime numbers occurring between 1 and 1000.

```
10 PRINT CHR$(147);
20 DIM A(200)
```

Line 10 simply clears the screen. PET users can use the CLR function rather than the CHR$(147). Line 20 reserves storage for the prime numbers we will extract later. (There are more prime numbers than you might think in the range of 1 to 1000.)

```
90 FOR N=2 TO 35
95 IF PEEK(N†32767)=102 THEN 130
100 FOR X=32767†(2*N) TO 33767 STEP N
110 POKE X,102
120 NEXT X
130 NEXT N
```

This double loop is the meat of our program. We only loop 34 times (2 to 35) because it is only necessary to test for multiples of primes up to the square root of your limit - in this case SQR(1000)=31†. (I added a couple for good measure). Line 95 checks the screen to see if our next potential prime has already been crossed out. Line 100 does the stepping across the screen, and line 110 does the "crossing out." Note that the PET's screen is actually addressable memory beginning at 32768(10).

```
200 N=1
210 FOR X=1 TO 1000
220 Z=PEEK(32767†X)
240 IF Z=32 THEN POKE(32767†X),81:A(N)=X:N=N†1
250 NEXT X
```

Now that we have crossed out all the non-primes, it is time to see what was left. This loop examines the screen to find the spaces. The index "X" will tell us what character we are looking at and the counter "N" will give us the next empty space in our table to store the prime number. Line 200 sets the table pointer to 1. Lines 210-250 is the loop that examines the screen. Line 220 looks at the current character position and puts its value in Z. In this case, the value will be 102 if it is a crossed out position, and 32 if it has not been crossed out. Line 240 then tests the value of Z and either ignores it if it has been crossed out or saves it in our table if it is prime.

```
300 GET A$:IF A$=" " THEN 300
```

This line simply causes the PET to pause while you admire its handiwork. When you are ready to see a list of the prime numbers, press any key.

```
400 PRINT CHR$(147);
410 FOR X=1 TO 200
420 IF A(X)=0 THEN STOP
430 PRINT A(X);
440 NEXT X
```

Line 400 clears the screen again. Lines 410-440 recovers our prime numbers from the table and prints them. When the table returns a zero, then we are finished, and the program will stop (line 420).

```
999 END
```

I hope you enjoyed this little bit of updated history. I'm sure old Eratosthenes would have been very happy to have had a PET to play with, but even 2000 years later he is not out of date.

EXPLORING THE APPLE II DOS

Andy Hertzfeld
2511 Hearst St. Apt. 204
Berkeley, CA 94709

To say that the documentation which comes with Apple's Disk II system is skimpy is being very kind. Only a terse description of each DOS command is provided and absolutely zilch is said about its memory usage or internal structure. Hopefully, Apple will soon remedy this situation but until that time hobbyists must rely on each other for the vital information. I have been exploring the internals of the DOS for the last few months; this article summarizes some of the interesting things I've found.

The DOS resides in the highest portion of your system's memory and is about 10K bytes long. Its exact size depends on how many file buffers you choose to allocate (one file buffer is needed for each simultaneously open file). Each file buffer is 595 bytes long and the system provides you with three to start with (you must have at least one).

The DOS communicates with the rest of the system via the input and output hooks CSW and KSW located at \$36 - \$39 (This article uses "\$" to indicate a hexadecimal number). Through these hooks it is given control every time a character is inputted or outputted. This is a nice scheme because it allows the DOS to be called from any environment (BASIC, Monitor, Mini-Assembler, etc.) but it has the drawback of activating the DOS when a command is typed as input to a user program, which is usually not what you want. Also, since the reset button resets the hooks, the DOS is disabled whenever the system is reset, which isn't so great.

The process of loading the DOS into memory for the first time is called "bootstrapping." Bootstrapping is initiated when control is transferred to the PROM on the disk controller card. Memory pages 3 and 8 are blown by a bootstrap. There are two different types of disks you can boot from: masters and slaves. The distinction is that a master disk can be used to bootstrap on a system of arbitrary memory size while a slave will only work properly on a system with the same memory size as that which created it. This is because since the DOS sits at the top of memory, its addresses (for JSRs, JMPs, etc.) will be different on systems with different memory sizes. A master disk cleverly solves this problem by loading into low memory first and then relocating itself up to where it belongs. Note that this means that a master bootstrap will blow alot of additional memory.

All addresses in this article are for a 48K system. If your system has memory size X, subtract 48K - X from the addresses that are given here.

A call to the routine at \$9DB9 will initialize or re-initialize the DOS. This routine should be called after every reset to restore the hooks. It is exactly like typing "3DO" "G" as Apple's documentation recommends but is a little bit safer since the \$3DO location is often destroyed by various programs.

Every diskette has a volume number from 1 to 254 associated with it. It is assigned when the diskette is initialized and there is currently no easy way to change it. The volume number of the current disk is stored at \$B7F6. Before most DOS commands the system checks to see if the current volume number matches the last volume number used. If it doesn't, a "volume mismatch" error is generated. While this "feature" may be nice for large business applications that don't want dumb operators inserting the wrong disks, it is very annoying to most average users, especially when you want to transfer a number of programs between two disks with different volume numbers. After much searching, I located the place where the volume check is performed and devised a patch to disable it. It's only two bytes long; just enter the monitor and type: "BDFE: A9 00". This will disable all volume checking until the next boostrap. It works by replacing the comparison instruction which performs the volume check with a " LDA #0 " instruction which sets the "equality" or Z flag, effectively forcing the match to succeed.

Binary files of arbitrary length can be saved on disk with the "BSAVE" command. Each BSAVEd file has an implicit starting address and length associated with it; when the file is BLOADed it is loaded at the starting address. Unfortunately, there is no way provided for a user to find out the starting address and length of a BSAVEd file; this makes copying files that you are not intimately familiar with very difficult.

Fortunately, when a file is BLOADed, the directory record of the file is always placed in a buffer in a fixed location. The buffer contains the starting address and length of the file as well as other useful information. The length is kept at memory locations \$A9A3 - \$A9A4 while the starting address is stored at \$A9B5 - \$A9B6 (with the least significant byte first, as usual). Thus to retrieve the starting address and length of a BSAVEd program you can simply BLOAD it and then peek at the above locations.

Some people might wish to alter the names of some of the DOS commands to suit their own, personal tastes (it is, after all, a personal computer). For example, I know many folks would like to abbreviate the "CATALOG" command to a simple "C". This is surprisingly easy to do; since the DOS lives in RAM the contents of its command table are easily changed. The command table is located from \$A7EO - \$A863. Each command name is represented as an ASCII string with the high bits off, except for the last character of the string, which has its high-order bit set. The strings are associated with the commands by their position in the command table (the first string corresponds to the INIT command, the second to the LOAD command, etc.). The position of every command is given below in Table 1.

Thus you can dream up your own names for the commands by storing new strings in the command table. For example to change the name of the INIT command to "DNEW" you would enter the monitor and type " A7EO: 44 4E 45 D7". However, some caution is required when you change the length of a command name; in general you will probably have to rewrite the entire command table to achieve the desired affect.

The error message table is stored at addresses \$A8CD - \$A980. By using the same techniques described for the command table, you can rewrite the error messages to be whatever you like.

TABLE 1: POSITION OF COMMANDS IN THE COMMAND TABLE

The position refers to which string in the command table is associated with the command. 1 means its the first string, etc.

Position	Command
1	INIT
2	LOAD
3	SAVE
4	RUN
5	CHAIN
6	DELETE
7	LOCK
8	UNLOCK
9	CLOSE
10	READ
11	EXEC
12	WRITE
13	POSITION
14	OPEN
15	APPEND
16	RENAME
17	CATALOG
18	MON
19	NOMON
20	PR#
21	IN#
22	MAXFILES
23	FP
24	INT
25	BSAVE
26	BLOAD
27	BRUN
28	VERIFY

It is hard to use the input and output hooks in conjunction with the DOS since you cannot simply change the hooks as they are the DOS' only contact with the rest of the system. Also, if you only change one of them, the DOS has the nasty habit of changing it back. Fortunately, the DOS has its own internal hooks it uses for keyboard input and video output. Its output hook is at $A996 - $A997 and the input hook immediately follows at $A998 $A999. If you change the contents of these addresses instead of the usual hooks at $36 - $39, everything should work just fine. For example, lets say you wanted to divert output to a line printer without disabling the DOS. If the line printer output routine is located at $300, all we would have to do is enter the monitor and type " A996: 00 03 ".

To execute a DOS command from a BASIC program, you simply print it, prefixing it with a "control-D". The prefix character is stored at memory location $A9F5, with its high-order bit set. Thus, if you don't like control-D and wish to use some other prefix character, all you have to do is store a different character value into $A9F5.

I am very curious to find out the primitive instructions the DOS uses to communicate with the disk controller, but without proper documentation it is very difficult to determine what does what (Can someone out there help me?). I have managed to find out the primitives that turn the drive on and off, though. If your controller card is in slot S, referencing memory location $C089 † $SO will power up the disk and start it spinning while referencing $C088 † $SO will turn it back off.

This article is merely the tip of the proverbial iceburg; most of the DOS's internals still remain a mystery to me. I hope Apple eventually distributes complete documentation but until then other curious users can use this article as a starting point for their own explorations and hopefully report back what they find. Table 2 (below) contains a summary of important addresses in the DOS for easy reference, including some not mentioned in the above commentary.

TABLE 2: IMPORTANT ADDRESSES IN THE APPLE II DOS

Address	Function
$B7F6	holds the volume number of the current diskette
$9DB9	routine to re-initialize the DOS
$A9E5	location of printing command character, initially set to control-D
$A9B5 - $A9B6	starting address of most recently loaded program, lsb first
$A9A3 - $A9A4	length of most recently loaded program
$A7E0 - $A863	the DOS command table
$A8CD - $A980	the DOS error message table
$A996 - $A997	the internal hook address to output a character
$A998 - $A999	the internal hook address to input a character
$C089 † $S0, S= slot no.*	address to power up the disk
$C088 † $S0, S= slot no.*	address to power down the disk
$9E4D	routine which handles the input hook
$9E7E	routine which handles the output hook
$BD00	routine which reads in the directory off the disk. It is called by virtually every DOS command

All addresses given (except those marked with an asterisk) refer to a system with 48K bytes of memory. If your system has memory size X, subtract (48K-X) from each address.

6502 INTERFACING FOR BEGINNERS: AN ASCII KEYBOARD INPUT PORT

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

Introduction

Many computer systems utilize a keyboard as an input device to get data or instructions from the outside world. KIM and TIM systems interface with teletype keyboards in which a 7-bit ASCII word is sent one bit at a time to the computer. This is called "serial input" and it is very common. Of course, the computer is capable of reading all 7 bits of an ASCII word in one byte. When operated in this way the keyboard input is just another location in memory, and the mode is sometimes referred to as "parallel." We will assume that the ASCII keyboard makes all 7 bits available at once and that it produces a positive strobe signal when the ASCII data is stable.

The following ingredients are necessary to implement a parallel keyboard input port.

1) A device select pulse $\overline{DS}$ for the memory location of the keyboard
2) Three-state buffer/driver connecting the keyboard to the data bus when the device select pulse occurs, but disabling it otherwise
3) A means for the keyboard to communicate with the computer; that is, the keyboard must inform the computer that a key has been depressed
4) A means to store the data until the computer reads it into the accumulator

Previous columns have dealt with the generation of $\overline{DS}$ pulse; it will be assumed that the appropriate circuitry is available. A single Intel 8212 Eight-Bit I/O Port will be used as ingredients 2), 3), and 4) above.

The 8212 I/O Port

A logic diagram for the 8212 is shown in Figure 1. The chip contains three subsystems; the control logic (including the $\overline{DS1}$, DS2, MD, STB, $\overline{CLR}$ inputs and the $\overline{INT}$ output), the data latch, and the three-state buffers. It all looks confusing but the situation can be simplified quickly. $\overline{CLR}$ will be tied to logic 1 to disable it. MD (for mode) is tied to logic O in the input mode. Examine the AND-OR control logic carefully to see that this last step in effect connects the strobe (STB) to the C inputs of the 8-bit data latch. The keyboard strobe will be connected to STB. When the STB is at logic 1 the Q outputs of the data latch follow the DI(1-7) inputs from the keyboard. The data is latched (stored at the Q outputs) on the trailing edge of the strobe. A single key depression results in the ASCII data being stored in the 8212, with one bit left over.

Note that the STB is also connected to the C input on the service request flip-flop. The trailing edge of the strobe latches a logic O into the Q output of the flip-flop because the D input is tied to logic O. The Q output is inverted, ORed, and inverted again to produce a logic O signal at $\overline{INT}$ whenever the strobe pulse occurs. The INT signal is used to communicate with the computer, telling it that data is available. Clearly it could be connected to the interrupt (IRQ or NMI) line on the 6502 to cause an interrupt. The interrupt vector would point to a routine to read the keyboard, and would have to include a LDA KYBD instruction.

The address of KYBD appears on the address bus during the third cycle of the LDA KYBD instruction. The address bus is decoded to produce a device select pulse $\overline{DS}$ for this address, and the device select goes to pin $\overline{DS1}$ on the 8212. At the same time DS2 is brought to logic 1 by the R/W line from the 6502. When $\overline{DS1}$ is low and DS2 is high the three-state buffers are enabled and the data from the keyboard is placed on the data bus to be read into the accumulator.

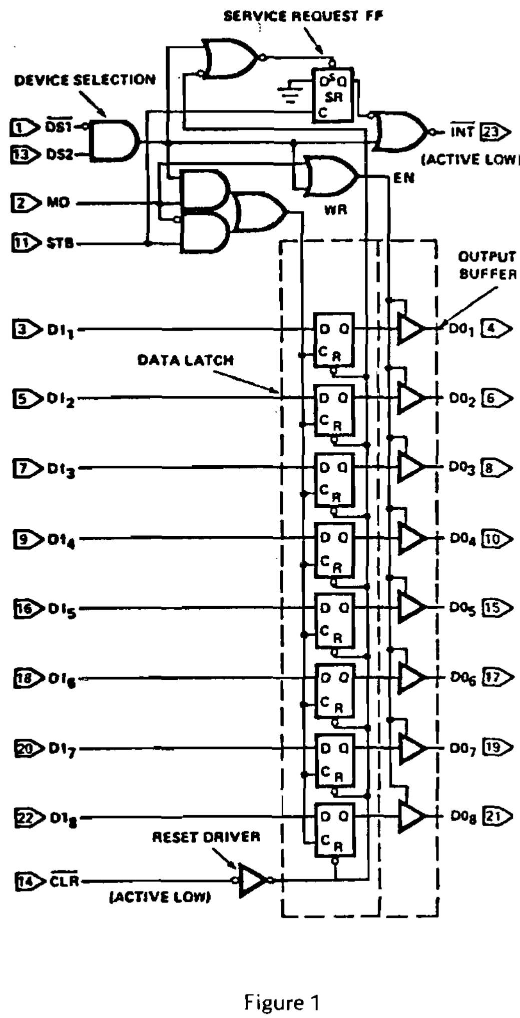


Figure 1
Logic diagram of the 8212 I/O Port.

Also observe that the $\overline{DS1}$.DS2 signal is connected to the "set" input on the service request flip-flop. This puts a logic 1 at the Q output which removes the interrupt request. The data has now been read, the interrupt cleared, and the computer is free to go on its way until another key is depressed and the entire process starts over.

Experiment with the 8212

A circuit to experiment with the 8212 is shown in Figure 2. You do not need an ASCII keyboard to construct this input port. The 74121 produces the necessary strobe signal. The data switches shown in Figure 2 can be jumper wires. For a device select I simply used the K1 select from the KIM-1, with a pull-up resistor added since the KIM-1 does not provide pull-ups for these selects. Any address decoding scheme to get a device select will do.

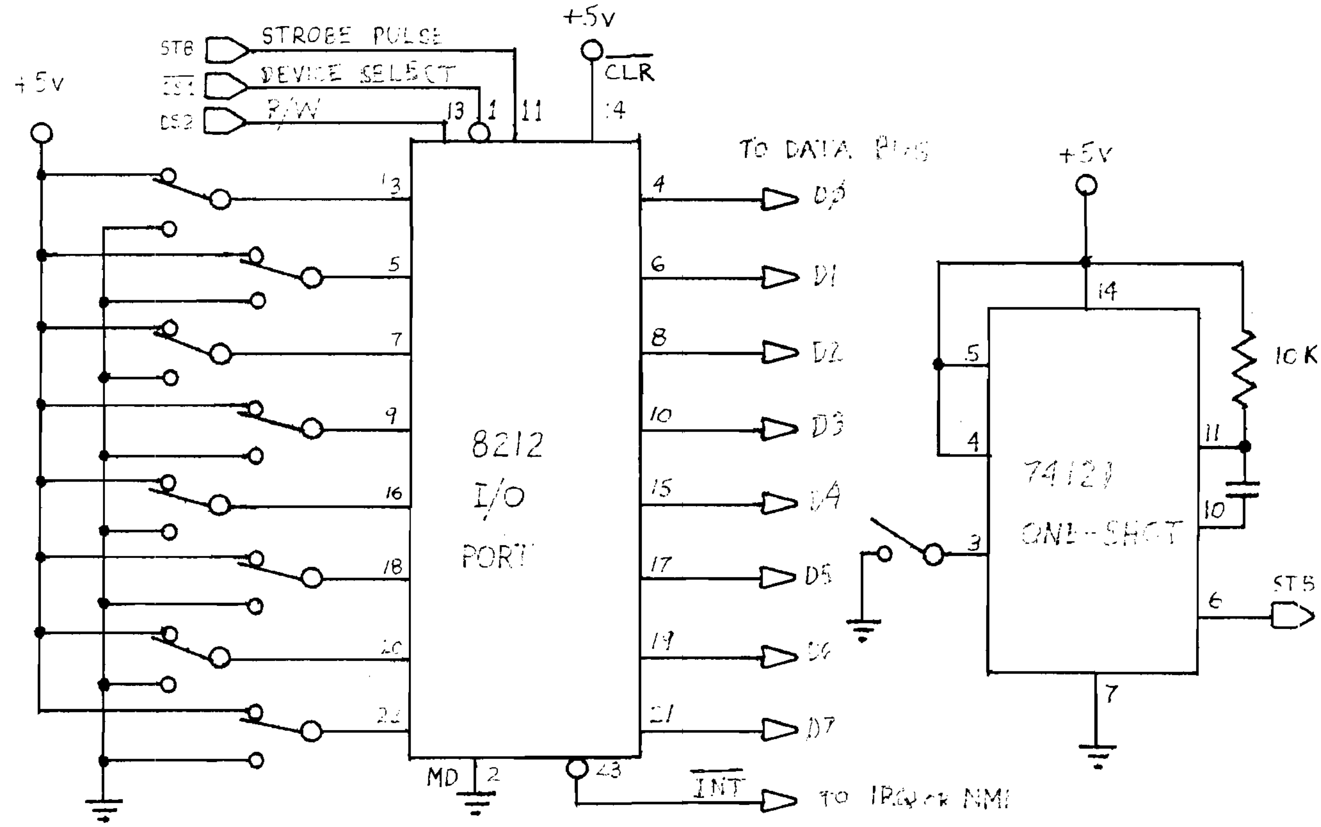


Figure 2.
8-Bit Input Port. The 74121 may be used to strobe the switch settings into the 8212. The power connections to the 8212 are pin 24 = +5V, pin 12 = GND.

Connect the data output pins to the data bus of the 6502, but leave the $\overline{INT}$ disconnected. Connect the strobe output of the 74121 to the STB pin on the 8212. Write a short program to read the 8212 and display the results on some output device. I used the following program for the KIM-1.

```
0000 AD 00 04  BEGIN  LDA    KYBD     K1 SELECT ON KIM USED
0003 85 FB            STAZ   DISP     PUT IN DISPLAY CELL
0005 20 1F 1F         JSR    SCANDS   JUMP TO KIM MONITOR
0008 4C 00 00         JMP    BEGIN    REPEAT
```

Load the program and run it. Set the switch settings for the data input to the 8212 to some value. Note that the switch settings have no effect on the displayed value. Now initiate the strobe pulse by closing the switch to the one-shot. This clocks the data into the 8212 and the computer will read it. Change the switch settings and initiate another strobe pulse. The data displayed should correspond to the switch settings. To initiate a strobe pulse the switch to the one-shot must first be opened, then closed.

Now connect the $\overline{INT}$ to the $\overline{IRQ}$ on your 6502. Run the following program:

```
0200 A2 00     BEGIN  LDXIM  $00    SET UP X AS COUNTER
0202 4C 02 02  HERE   JMP    HERE   WAIT FOR INTERRUPT

0000 AD 00 04  INT    LDA    KYBD   GET DATA FROM KYBD
0003 85 10            STAZ   MEM1   SAVE DATA
0005 E8               INX           BUMP COUNTER
0006 86 11            STXZ   MEM2   SAVE COUNTER
0008 40               RTI           RETURN FROM INTERRUPT
```

Be sure to set your interrupt vector to 0000, 17FE and 17FF on the KIM-1. Run the program starting at 0200. This is just an infinite loop which initializes the X register to zero. Now hit the strobe switch. Stop the program and examine the contents of 0010. It should be identical to the switch settings for the 8212 inputs. Examine 0011 where the X register was stored. Why doesn't it read 01 corresponding to the single interrupt we produced? Because the mechanical switch used to initiate the strobe pulse was not "debounced."

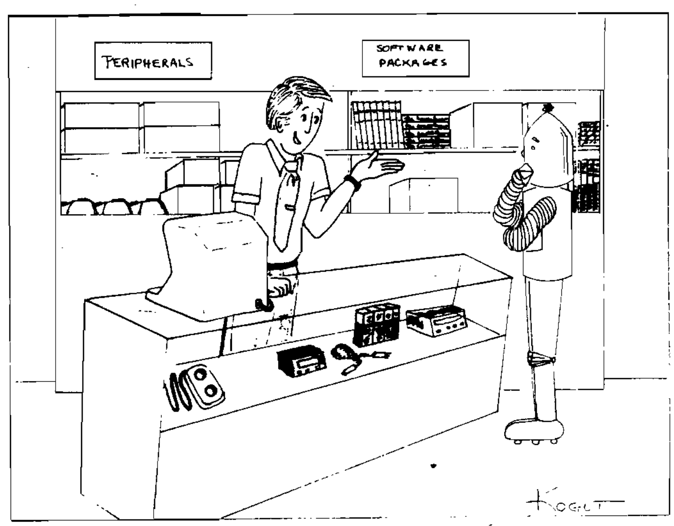
The program is very simple. The computer loops forever in the JMP HERE loop unless an interrupt occurs ($\overline{IRQ}$ pulled low by $\overline{INT}$). When the interrupt occurs the computer jumps to the interrupt routine which reads the 8212 and stores the result in 0010. X is also incremented and stored in 0011. This was done just to give you a feeling for keybounce. The program then returns to the infinite loop where you found it when you stopped the program. Change the switch settings on the 8212 then try the program again.

Disconnect the $\overline{INT}$ from the 6502 and connect it to the DI(8) input (pin 22) on the 8212. We will now **poll** the input port to see if any data is ready. If a strobe pulse has occurred, then bit seven will be low because $\overline{INT}$ is connected to this bit. Once the 8212 is read, $\overline{INT}$ goes high as does bit seven. Here is a program to demonstrate polled service.

```
0200 20 20 02  MAIN    JSR   INPUT   SIMULATES "MAIN PROGRAM"
0203 4C 00 02          JMP   MAIN

0220                   ORG   $0220

0220 20 1F 1F  INPUT   JSR   SCANDS  DISPLAY LAST INPUT DATA
0223 2C 00 04          BIT   KYBD    TEST BIT 7
0226 30 F8             BMI   INPUT   LOOP IF BIT 7 = 1
0228 AD 00 04          LDA   KYBD    ELSE, GET NEW DATA
022B 85 FB             STA   DISP    STORE IT
022D 60                RTS           RETURN TO MAIN PROGRAM
```

Play around with it changing switch settings and strobing data. Basically what it does is test bit-7 to see if any new data is available. MAIN is just a dummy program. It represents almost any program which uses a keyboard input. For example, my Micro-ADE assembler, disassembler, editor polls the keyboard for new data and my BASIC interpreter does the same thing. Both programs jump to subroutines which wait until new data has been entered from the keyboard, then return to the main program to process that information. I used JSR SCANDS in my INPUT subroutine so you could see the data on the KIM-1 display. Normally one would not use the KIM-1 display in an input routine. Rather he would "echo" the input with an output routine which would write the data on his CRT or teletype.

If you have an ASCII keyboard with a positive strobe you can do all of these same experiments but with an actual keyboard input.



May I show you something in a Ready to Ware?

by: Bertha B. Kogut

TWO SHORT TIM PROGRAMS

Gary L. Tater
7925 Nottingham Way
Ellicott City, MD 21043

A Fast Talking TIM

If you have used both KIM and TIM with a terminal, you know that TIM has many nice features. For instance you can enter eight bytes at a time with TIM, and TIM has many more subroutines you can call in your programs than KIM does. However, KIM can adapt to terminal frequencies up to 2400 baud whereas TIM was designed to work from 100 to 300 baud. This article describes a program which allows you to communicate with TIM at 1200 baud or higher.

After a reset TIM automatically measures the speed of your terminal and deposits the bit times representative of the baud rate in two zero page locations, OOEA and OOEB. To increase the baud rate above 300 baud, the procedure is to place the correct values into EA and EB and change your terminal to that speed.

```
0100 20 A4 73  NEWVAL JSR   $73A4  READ TWO BYTES VIA TIM MONITOR
0103 A5 EE            LDA   $00EE  PUT EE INTO EB
0105 85 EB            STA   $00EB
0107 A5 EF            LDA   $00EF  PUT EF INTO EA
0109 85 EA            STA   $00EA
010B 00               BRK
010C 4C 00 01         JMP   NEWVAL TYPE G FOR NEW VALUES
```

Figure 1
Program to Change OOEAand OOEB.
Type Major Value OOEA First

By using the short program of Figure 1, I was able to find the correct values for 600 and 1200 baud operation (See Table 1) for my CT-64 and CGRS CPU board which has a 6502 operating with a one megahertz crystal. For each baud rate there is a range of values that is acceptable for EB. I have attempted to find the center of the range for my system. You will probably need to experiment to find the best numbers for your computer.

Baud Rate	00EA	00EB
1200	01	50
600	03	13
300	06	3C

Table 1
Zero page memory values for three baud rates.

Using this basic information I wrote the program of Figure 2. The programs begins at 157E and asks:

SPEED 300 600 1200?

At this point you should type 3, 6, or 1 and change your terminal to the correct rate. The program determines what you have entered and stores the correct values in EA and EB. By inspection of the program, you should be able to expand it to 2400 baud if you have a faster terminal. For a one megahertz system typical values are 00 in EA and 75 in EB for 2400 baud.

A TIM Operating System Menu

If you have written a collection of utility programs, assemblers, disassemblers and application programs, you will need a directory program with which you can easily call your desired program. The short program in Figure 3 uses the alphabet to call 26 programs. When the programs finish, they should return to the beginning of the directory program at location 0100.

You may choose to keep the program in ROM as I do. Only locations 0116 and 011B need be changed to do this provided you start the program at the beginning of a page.

The program prints a prompting "-" so that you'll know its in command and not TIM. If you type a nonalphabetic character, it will restart. After you type a letter, say a C for compare or M for move, the program finds the appropriate starting address stored between 0122 and 0155. After the starting address is stored in 00F6 and 00F7, the program calls the "GO" subroutine in TIM which causes your program to be executed.

THIS PROGRAM IS RELOCATABLE AS LONG AS THE POINTER TO
THE TEXT MESSAGE IS CHANGED IN LINE "PRINT"

```
157E D8        START  CLD               CLEAR DECIMAL MODE
157F A0 00            LDYIM $00         INITIALIZE INDEX
1581 B9 B3 15  PRINT  LDAY  TEXT        GET ASCII CHARACTERS
1584 F0 06            BEQ   PDONE       DONE IF NULL CHARACTER
1586 20 C6 72         JSR   $72C6       PRINT VIA TIM OUTPUT ROUTINE
1589 C8               INY               BUMP POINTER
158A D0 F5            BNE   PRINT       UNCONDITIONAL BRANCH TO PRINT NEXT
158C 20 E9 72  PDONE  JSR   $72E9       READ CHOICE VIA MONITOR
158F C9 31            CMPIM '1          ASCII 1 ?
1591 F0 1A            BEQ   HIGH        1200 BAUD
1593 C9 36            CMPIM '6
1595 F0 10            BEQ   MEDIUM
1597 C9 33            CMPIM '3
1599 D0 E3            BNE   START       NOT VALID CHARACTER

159B A2 3C     LOW    LDXIM $3C         GET VALUES FOR 300 BAUD
159D A9 06            LDAIM $06

159F 85 EA     FIXIT  STA   $00EA       SAVE FOR TIM TIMING ROUTINES
15A1 86 EB            STX   $00EB       SAVE SECOND VALUE
15A3 00               BRK               RETURN TO MONITOR
15A4 18               CLC               CLEAR CARRY
15A5 B0 D7            BCS   START       UNCONDITIONAL BRANCH

15A7 A2 13     MEDIUM LDXIM $13         GET VALUES FOR 600 BAUD
15A9 A9 03            LDAIM $03
15AB D0 F2            BNE   FIXIT       UNCONDITIONAL BRANCH TO FIXIT

15AD A2 50     HIGH   LDXIM $50         GET VALUES FOR 1200 BAUD
15AF A9 01            LDAIM $01
15B1 D0 EC            BNE   FIXIT       UNCONDITIONAL BRANCH TO FIXIT

15B3 53        TEXT   =     'S            "SPEED  300 600 1200 ?"
15B4 50               =     'P
15B5 45               =     'E
15B6 45               =     'E
15B7 44               =     'D
15B8 20               =     '
15B9 20               =     '
15BA 33               =     '3
15BB 30               =     '0
15BC 30               =     '0
15BD 20               =     '
15BE 36               =     '6
15BF 30               =     '0
15C0 30               =     '0
15C1 20               =     '
15C2 31               =     '1
15C3 32               =     '2
15C4 30               =     '0
15C5 30               =     '0
15C6 20               =     '
15C7 3F               =     '?
15C8 20               =     '
15C9 00               =     $00
```

Figure 2
6502 Program to Change Speed

```
0100 20 8A 72  START  JSR   $728A  CRLF VIA TIM MONITOR
0103 A9 2D            LDAIM '-     PRINT "-"
0105 20 C6 72         JSR   $72C6  VIA TIM MONITOR
0108 20 EE 72         JSR   $72EE  READ A CHARACTER VIA TIM
010B C9 5B            CMPIM $5B    TEST FOR GREATER THAN Z
010D 10 F1            BPL   START  BRANCH IF TOO LARGE
010F 38               SEC          SET TO CONVERT ASCII TO INDEX
0110 E9 41            SBCIM 'A     BY SUBTRACTING VALUE OF ASCII A
0112 30 EC            BMI   START  IF MINUS, THEN CHARACTER LESS THAN A
0114 0A               ASLA         MULTIPLY BY TWO FOR INDEX
0115 AA               TAX          PUT CONVERTED VALUE INTO INDEX
0116 BD 24 01         LDAX  LOWADR GET START ADDRESS LOW
0119 85 F6            STA   $00F6  SAVE FOR TIM
011B BD 25 01         LDAX  HGHADR GET START ADDRESS HIGH
011E 85 F7            STA   $00F7  SAVE START ADDRESS HIGH
0120 20 5C 71         JSR   $715C  GO TO SUBROUTINE VIA TIM
0123 00               BRK

0124 00        LOWADR =     $00    LOW ADDRESS FOR A, FILLED IN BY USER
0125 00        HGHADR =     $00    HIGH ADDRESS FOR A, FILLED IN BY USER
0126 00               =     $00    LOW ADDRESS FOR B
0127 00               =     $00    HIGH ADDRESS FOR B
               AND SO FORTH THROUGH
               LOW AND HIGH PAIR FOR Z
```

Figure 3
A TIM Directory Program

ASK THE DOCTOR — PART I

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

The Rockwell International AIM 65, the Synertek SYM-1 and the Commodore KIM-1 form a closely knit family of microcomputers. Of course they all use the 6502 microprocessor, but the family resemblence is much deeper than that. A few of the features that make the three boards so similar are:

1. Each is a "bare" single board microcomputer without a case, built-in power supply, etc.

2. They have the same basic I/O support:
   A. 20 mA current loop TTY interface; and,
   B. Low Speed Audio Cassette interface. All three computers support the KIM-1 cassette tape format. This means that a cassette tape generated in the KIM-mode on any of the machines can be read on any other machine. This tape cassette compatibility is so complete that it is possible to directly interconnect a KIM to SYM, or KIM to AIM, or SYM to AIM via the the audio cassette interface - without the cassette! Simply take the **Audio Out HI** from one computer and connect it to the **Audio IN** of the other. Then run the Load KIM format cassette program on the second computer and the Write KIM format cassette program on the first computer.

3. They have a compatible bus structure. Each computer has two dual 22 pin edge connectors with essentially the same connections. The **Expansion connectors** have identical placement of all the **Address, Data, Control** and **Power** lines. The **Application connectors** have identical placement of most signals that are common on the three computers - **Port A** and **Port B I/O, Power** and **Ground, Audio Cassette I/O, TTY I/O** - plus some additional signals which are unique to each computer. This bus similarity is a very important component of the AIM/SYM/KIM (**ASK**) family compatibility.

4. The SYM intentionally "duplicates" many of the KIM Monitor routines, and has a similar **Hex Keypad** and **LED Display** on board. The reader is hereby warned to be careful when using SYM routines which proport to be 'the same as" the KIM routines. As will be shown in a later column, there are often minor, but important differences between two routines which at first appear identical. For example, in the KIM **PACKT** subroutine, a successful return is signaled by the **Zero Flag** being **Set**; an error return by the **Zero Flag** being **Cleared**. The similar SYM **PACKT** subroutine performs the same packing function, but signals a successful return with the **Carry** bit **Cleared**; an error return by the **Carry** bit **Set**. So, be careful.

An AIM/SYM/KIM Compatibility Example

One way to understand the nature of the similarities and differences between the ASK family members is to examine in detail a common situation which involves both hardware and software for the three systems. **MEMORY PLUS**(tm) is a multi-purpose board that was designed for the KIM-1 long before the SYM or AIM were even a gleam in their creators' eyes. It contains **8K RAM**, provision for up to **8K EPROM**, a **6522 Versatile Interface Adapter**, and an **EPROM Programmer**. Since it was designed to work on the KIM-1, it obviously is compatible with that computer. The question is: Is the MEMORY PLUS compatible with the SYM and AIM? The answer is Yes, No, and Maybe. Let's examine this seeming paradox in some detail.

YES

The **8K RAM** and the **8K EPROM** work directly with the KIM, SYM and AIM with no modification. In fact, the same connector cable may be used to connect the MEMORY PLUS to any one of the computers. This exact compatibility is due to the fact that all that MEMORY PLUS requires for operating the **RAM** and **EPROM** are the **Address, Data, Control** and **Power** lines, and these are all positioned identically on the **Expansion connector.**

NO

The addressing of the **6522 VIA I/O** was designed to use the **K5** *chip select that is generated by the KIM and which appears on the* **Application connector**. This same signal is generated by the SYM and makes the addressing of the **6522 VIA** identical to that of the KIM. The AIM does not generate this signal. Therefore, without some sort of modification, the AIM can not use the **6522 VIA**, and since this is the heart of the **EPROM Programmer**, can not program **EPROMs**. Fortunately, there are a couple of unused gates on the MEMORY PLUS and a minor wiring modification can be made so that the MEMORY PLUS will itself generate the equivalent of the **K5** signal and permit the AIM to use the **6522 VIA** and **EPROM Programmer.** This does point out a small, but significant difference, between the bus signals of the KIM, SYM and AIM. In general, the SYM made much more of an effort to be KIM compatible than the AIM did. This example where the KIM and SYM generate the **K1, K2, K3, K4,** and **K5** signals and the AIM does not, is probably the greatest difference in the hardware as seen on the **Application** and **Expansion busses.**

MAYBE

Since the KIM does not do all of the address decoding required for a system beyond the initial 8K used by the KIM on board, any additional memory device must generate a **DECODE** signal which enables the KIM memory at the proper times. The MEMORY PLUS board has circuitry to generate the **DECODE**. The SYM and the AIM do all of the required address decoding for their operation on-board, and do not therefore require this signal. The **DECODE** signal may be simply ignored in these two systems by not connecting it from the MEMORY PLUS to the SYM or AIM.

There are other addressing space differences between the three systems, which may or may not be important in a particular situation. All three have **RAM** in locations 0000 to 03FF. This includes the **Page Zero** and **Stack** locations. The KIM does not use 0400 to 16FF, but uses 1700 to 177F for **I/O** and **Timers**, 1780 to 17FF for **RAM**, and 1800 to 1FFF for the **ROM Monitor**. The AIM has 0400 to 0FFF available for on-board **RAM** expansion, 1000 to 9FFF are available for **User** expansion, A000 to AFFF is used for **I/O** and **System RAM**, and the remainder of the memory is allocated for various ROMs: B000 to CFFF for BASIC, D000 to DFFF for Assembler, and E000 to FFFF for Monitor. The SYM has 0400 to 0FFF for on board **RAM** expansion, 1000 to 7FFF for **User** expansion, 8000 to 8FFF for **Monitor ROM**, 9000 to 9FFF reserved for **Monitor** expansion, A000 to AFFF for **System RAM** and **I/O**, B000 to BFFF for **User** expansion, C000 to DFFF for **BASIC ROM**, E000 to FF7F reserved for **Assembler/Editor ROM**, and FF80 to FFFF for **SYSTEM RAM Echo** locations. The above listing of memory allocation should make it obvious that the three systems each have **I/O** and **Monitors** located in different places, so that software calling on the **I/O** or **Monitor** will have to be at least different in the addresses used. On the MEMORY PLUS this shows up when the host computer's **Port B** is used to generate three of the addresses required by the **EPROM Programmer**. While the three lines, **PB0, PB1,** and **PB2** are all mapped to the same Application connector locations, the address of the **I/O** device controlling the port is different. In fact, the **I/O** device on the KIM is a **6530** and the device on the SYM and AIM is a **6522**! All this does is require different addresses within the EPROM Programming program. Another memory mapping difference is in the location of the interrupt vectors. Each of the three computers uses different addresses to handle the interrupts. The MEMORY PLUS programmer uses the **IRQ interrupt**, and must therefore set up the **IRQ vector** in a different location on the KIM, SYM or AIM. Again, this is a minor problem, but is an incompatibility. Finally, since the Monitor is in a different location in each computer, a return to the Monitor at the end of the **EPROM program** will be to a different address for each. If the MEMORY PLUS used the on-board **Timers**, then it would again require some modifications to the software. In the case of the KIM, the **Timer** is of the **6530** variety; the SYM and AIM have **6522** types. This would require a different set of parameters as well as different addresses. As a matter of fact, MEMORY PLUS uses its own **6522 Timer**, and so this problem does not arise.

One final note of caution on the memory allocation of the three computers. Even though they all support **RAM** in locations 0000 to 03FF, the use of this **RAM**, especially the end of **Page Zero**, is quite different between them, both in the amount of **Page Zero RAM** used and the use of particular locations. In addition, while the KIM and the SYM do not use **Page One** for anything, in general, except as the **Stack**, the AIM makes extensive use of **Page One**. This variation in use of **Page Zero** and **Page One** will often require that existing programs undergo some re-definition of addresses and a re-assembly before they can be moved from one computer to another, even when the **Monitor** of the computer is not being used as part of the program.

SUMMARY

The AIM/SYM/KIM family of 6502 based microcomputers have a lot in common; but they also have some significant differences. In most cases these differences are not so great that they can not be overcome with some careful modification to existing hardware and/or software. But, significant differences do exist, and any user who plans to use a variety of these systems should be aware of the potential problems that exist. Subsequent columns will go into more detail on the similarities and differences between the ASK family members.

SYM Cassette Tape Problems

There are two problems with the SYM tape service that users should be aware of. The first is that the SYM hardware has a filter circuit that is used in shaping the input signal from the cassette recorder. This particular circuit is very sensitive and will not work reliably with all tape recorders. It apparently was optimized to a particular type of unit, possibly a SuperScope C-190; and is not very optimal for a large number of other units. Several suggestions have been made to improve this circuit. One is to replace the resistor R92 (see page 4-9 in the SYM Reference Manual for a circuit diagram) which is a 1K with a 3.3K. Another idea that has been used was to put a .01 MFD capacitor in parallel with C15 which is a .47 MFD. I have **NOT** had a chance to try either of these and do not guarantee that they either work or that they will not destroy your system. I am merely passing on a couple of suggestions which were given to me. I hope to be able to give a more complete and tested set of changes by next month.

The second tape problem has to do with reading KIM format tapes. As you probably know, the KIM format uses an ASCII "/" character to signal the end of data. This character has a hex value of 2F. The SYM Monitor has software to detect the end of data character which properly detects an ASCII "/" as it should. However, it also has software which erroneously thinks that an ASCII "2" followed by an ASCII "F" which when combined make a hex 2F data byte, is a terminator. This means that anytime your data has a 2F in it, as in

    4C 13 2F   JMP $2F13 (Jump to address 2F13)

it will mistake the legitimate 2F data as a "/" character and think that it has reached the end of the data. Since the following bytes of data will be considered to be the check digits, and will not be correct, the SYM will give you an error and stop loading. This can be very disheartening. Synertek is aware of the problem and is supposed to fix it, but no fix has been received here yet.

One way I have overcome this difficulty, with some difficulty, is to load my program into the KIM, change any 2F data to an FF, and then either make a cassette tape or dump the data directly into the SYM from the KIM via the Audio Out HI on the KIM to the Audio IN on the SYM. Then I have to go to the SYM and change all of the FF's which were substituted for the 2F's back to their original 2F value. This is cludgy, but it works. If you do not have a KIM handy, however, you are out of luck.

Coming Attractions

Future columns will cover all sorts of interesting information about the AIM, SYM, KIM (and maybe SUPERKIM). If you have discovered any useful bits of information about these machines, please drop me a line and I will try to include the info in a future column. In this way the material can be widely disseminated without your having to write a whole article about it.

Note: MEMORY PLUS(tm) is manufactured by The COMPUTERIST, Inc., P.O. Box 3, S. Chelmsford, MA 01824. It currently retails for $245.00.

TWO APPLE II ASSEMBLERS: A COMPARATIVE SOFTWARE REVIEW

Allen Watson
430 Lakeview Way
Redwood City, CA 94062

There are two assembler programs for the Apple II available from independent software vendors: the Microproducts Apple II Co-resident Assembler for $19.95 from Microproducts, 1024 17th Street, Hermosa Beach, CA 90254, and the S-C Assembler II for $25 from S-C Software, P.O. Box 5537, Richardson, TX 75080. The features and relative merits of these assemblers are the subject of this review.

Introduction: Software Tools

Some microcomputer owners hardly ever program, being satisfied to run programs written by other people. Others program only in BASIC or one of the compiler languages. Then there are those who write programs in machine language because the demands they make of their computers can be met in no other way. The assembler is a software tool which relieves them of much of the drudge-work involved in machine-language programming.

Software tools such as assemblers are much more important than their modest sizes might imply, since they are used over and over in the development of other programs. A poor tool is tiring to use and causes errors and frustration; a good tool requires minimum effort and soon seems like a natural extension of the user.

Built-In Assembler Features

The mini-assembler built into the Apple II sets it apart from conventional microcomputers. It will probably lead many Apple II owners to venture into machine-language programming for the first time.

The mini-assembler's primary function is instruction-code translation. Instead of remembering all the 6502 numeric opcodes, the programmer finds himself thinking in the 6502 mnemonics. The word **mnemonic** just means **easy to remember**; while letter combinations such as CMP and LDA may seem cryptic at first, it soon becomes second-nature to read CMP as **compare** and LDA as **load accumulator**.

The branch instructions in the 6502 use relative addresses. The address that is being branched to has to be converted into a one-byte offset value. Doing this by hand is so tedious and prone to error that there is even a small slide rule on the market to do the hexadecimal arithmetic. The Apple's mini-assembler and its companion disassembler take care of this automatically, so that the programmer can use the actual address values when he writes branch instructions.

The different addressing modes of the 6502 are handled very simply. Indexing is indicated by a comma and X or Y after the base address. Parentheses are used to delimit the address of the address in indirect-addressing mode, and indirect-indexed and indexed-indirect addressing are easily distinguished by this means.

The Apple's built-in assembler is very convenient, but the machine-language programmer soon finds himself wishing the machine could do more for him. Obviously, given the right program, it can. Enter the full-fledged assemblers, stage right.

More Assembler Features

Both of the assemblers described here have all the features of the Apple mini-assembler and several more besides. The two most important additional features are program editing and symbolic addressing. An editor is often a separate program, but since much of the value of an assembler would be lost without the ability to edit, both of these assemblers include editors and should properly be called editor-assemblers.

Once you face the necessity of re-entering most of a long program by hand in order to make room for additional instructions near the beginning of the program the need for an editor will be apparent. Some machines have editors that work directly on the machine code, but the editor portions of both of these assemblers manipulate the assembler input data or source file. They enable the programmer to add or delete instructions anywhere in the program without worrying about the consequences. (Well, almost; if the added instructions between a branch instruction and its destination increase the displacement to more than 128 bytes, the branch is no longer valid and must be replaced by a different branch and a jump.)

Symbolic addressing is one of the most important functions of an assembler. The older higher-level language BASIC and FORTRAN have symbolic addressing only for variables. The lack of symbolic addressing of instructions makes programs difficult to read.

Address references in assembler language are made by means of *symbols which are assigned their numeric values when the* program is assembled. The programmer needn't be concerned about the actual addresses except to make sure there is room for all of them. But symbolic addressing does more than just eliminate a lot of messy bookkeeping: since the symbols are entirely arbitrary, the programmer can choose them such that they serve as mnemonic labels for all of the important addresses in the program. For example, where a BASIC programmer would have to write something like GOTO 1275, an assembler-language programmer may write JMP DONE, where DONE is both a symbol which represents the required address and a label which is meaningful to the programmer.

The Microproducts Co-resident Assembler and the S-C Assembler II both qualify as full-fledge assemblers. They have several features in addition to those described above, including:

(1) loading and saving the assembler input file on tape;
(2) programmer specification of the starting address in memory of the assembled program;
(3) inclusion of ASCII character strings and hexadecimal numbers as part of the program; and
(4) the inclusion of comments, explanatory notes which are part of the input file but are ignored by the assembler.

What About Documentation?

A user's manual is provided with each of these assemblers. The Microproducts manual consists of seven pages and is barely adequate. It is poorly organized and there are a couple of errors in it. The manual for the S-C assembler is more substantial, with 17 pages of instructions giving complete information for the programmer. There are also 10 pages of appendices including a list of references and a listing of a printer-driver program. It is clear and candid, even pointing out a couple of weak places in the program.

Now For The Bad News

There are limits to how easy things can be made for the machine-language programmer. For one thing, both assemblers limit the length of symbols to not more than four characters, and special characters are not permitted: only letters and numbers. Another joy-killer is the strict formatting of the input statements. Labels must be in their specified columns, opcodes in theirs, and so on. If there is no label on a particular line, you must skip across to the correct column before typing in the operation mnemonic.

The S-C assembler ameliorates this problem by providing a tabulation feature: to skip a field, you just type in a TAB. Since the Apple II's keyboard doesn't have a TAB key, you have to use Control-I for this. The Microproducts assembler makes you count spaces, which is downright criminal. Computers can count without ever making a mistake, but programmers can't; therefore programmers should never be called upon to count when there is a computer available to do it for them.

Editing With Line Numbers

Both of these assemblers include editors that work like the BASIC editor by using line numbers. The programmer must type a line number at the beginning of every line, and the sequence of the numbers becomes the sequence of the lines. And woe be unto him who accidentally uses the same numbers twice: the lines entered earlier will be written over by the later ones having the same numbers. If you have never been so careless as to make this error, reading about it here will probably suggest it to your subconscious, so beware!

Now suppose that you have just typed in a program that is 250 lines long, dutifully numbering the lines in steps of 10, and you want to examine an earlier part of the program. What do you do? If you have a printer, you can list the whole thing and examine any part you want to. Both assemblers include commands for starting and stopping a printer. But short of listing the whole program, suppose you just want to display part of it on the TV screen.

Either assembler will enable you to start through the whole input file on the TV display and interrupt it when you reach the desired part, that is, if you have fast reactions. The S-C program is kinder: it has a SLOW mode for displaying. It also lets you specify range of line numbers to display, just as you do in BASIC.

The S-C assembler has another feature which should prove very useful: you can APPEND a source file saved on tape earlier onto the input file you are currently editing in memory and assemble the whole thing as a single program. This makes it possible to build yourself a library of standard routines which you can use in several different programs with a minimum of effort.

Shortcomings of the Microproducts Assembler

There aren't a great many nice things I can say about the Microproducts assembler. It simply doesn't do all the things it should to help the programmer. For example, error messages are output as number codes which you have to look up in the manual. If it were programmed to do so, the computer could look them up a lot faster and put them out in English. With the S-C assembler, it does.

In the Microproducts version, numeric expressions must include leading zeros. If you define a symbol as RATE .DL 5, RATE will be assembled as hexadecimal 5000, not 0005. But what's even more exasperating, once you get it defined as 0005, references to RATE will not assemble as zero-page addressing unless you prefix the symbol with an asterisk each time it is referenced. This is plain inexcuseable: the program should test for this and select the appropriate address mode automatically.

Are There Bugs in the Programs?

Nobody's perfect, not even the people who write assemblers. No matter how hard they try, debugging can't demonstrate the absence of bugs, only their presence. While I haven't tried out every feature of these assemblers yet, I have assembled the same program on both of them as a comparison. So far I have found only one bug in the S-C assembler. If you slip while typing an implied-operand instruction without a label and put the mnemonic in the label columns thus leaving the operation and operand fields blank the assembler will not detect the error but instead will repeat the previous instruction.

The Microproducts assembler has bugs, too. It permits a comment on an instruction line, but if the comment is long enough that the line exceeds 40 columns so that the display continues on a second line, the address and object code which normally appear at the left of the screen get written on the second line and obliterate the comment. Another bug appears whenever you interrupt a listing, which you can do by hitting any key. The Microproducts assembler fails to clear the keyboard strobe, causing the key you used to interrupt it to become the first character of the next command.

There is a curious error in the Microproducts manual where it states that the assembler is less than 3K bytes long, even though it loads from 2000 to 2CFF in memory, a total of 3,328 bytes. Just coincidentally, the S-C assembler loads from 1000 to 1BFF, making it exactly 3K bytes long.

Wouldn't It Be Nice If...?

While both of these assemblers are more powerful than the mini-assembler, some people are never satisfied. A couple of improvements occurred to me as soon as I started using these assemblers.

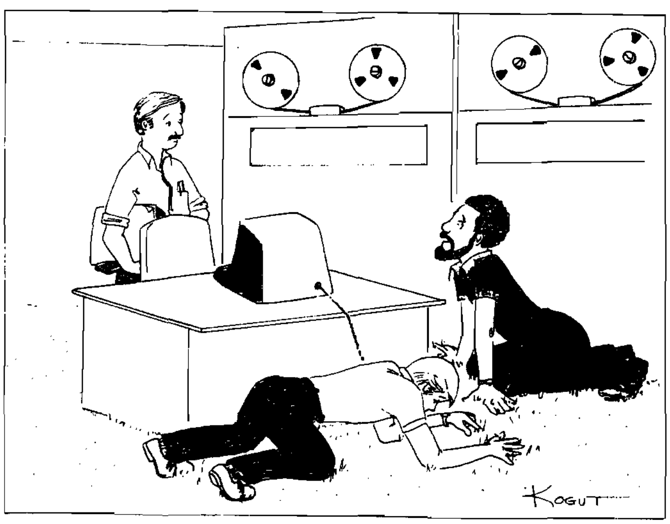
In a BASIC program, the line numbers are an innate part of the program, used as destinations for GOTOs and so on. Assembler language doesn't really use line numbers; these assemblers use them only because they make the editor simpler. It would be nice if the programmer didn't have to keep track of a lot of numbers; the computer is much better at it. If the editor has to have line numbers, an automatic line-number generator would be a nice option.

I'd like to see some kind of LOCATE function, too. Since the line numbers don't bear much relation to the program, especially after you've used the RENUMBER a time or two, the selective list feature of the S-C assembler isn't 100% effective for displaying a portion of the program. What if you don't remember the line number of the instruction you labelled SCAN? Wouldn't it be nice if you could type something like LOCATE "SCAN" and have the editor search for the line that has SCAN as its label? Some editors even have two different forms of this command: one which looks only at the beginning of each line, and another which searches all the way through each line to find the places where a label is used in an operand or in a comment.

Conclusion

It is interesting to note the similarities between these two assemblers. The programs are nearly the same size, about 3K bytes, and priced at $20-$25. They use similar input formats and both of them do their editing by means of BASIC-type line numbers.

Where they diverge the advantage is almost always with the S-C Assembler II. It has more features and a bigger manual, its error messages are output in English, and its format is a more logical extension of the Apple II mini-assembler. If you are the least bit interested in machine-language programming on the Apple II, I strongly recommend the purchase of a copy of the S-C Assembler II. And incidentally, I do mean purchase, not "obtain by fair means or foul." Sources of good programs should be encouraged, and the assembler will repay its purchase price many times over.



Johnson lost his microprocessor again

by: Bertha B. Kogut

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

Name: **Text Editor/Word Processor**
System: **Apple II**
Memory: **24K for cassette, 32K for Disk II**
Language: **Applesoft II**
Hardware: **Apple II, cassette tape recorder or Disk II and printer**
Description: Uses any width line, features upper and lower case using inverse video, justification by adding blanks, user set and cleared tabs in any column, automatically renumbers lines on insertion or deletion, usable with any printer interface by extremely slight program modifcation.
Copies: **100+**
Price: **$50. for cassette version, $60 for Disk version**
Includes: cassette or diskette and instructions. Source listing available by sending SASE with serial number
Author: **Craig Vaughn**
Available from: Local Apple dealers or:
Peripherals Unlimited
6012 Warwood Road
Lakewood, CA 90713

Name: **Mailing Label Package**
System: **Apple II**
Memory: **At least 32K**
Language: **Applesoft II**
Hardware: **Apple II, Disk II, and printer**
Description: Stores 3-line or 4-line addresses (may be mixed) plus phone # and a 15-character code field, any one record may be accessed by name or phone #, prints in zip code order, will print all records or select by code field with wild card, any number of labels horizontally, user formats spacing, may be used with any printer interface with very slight program modification. Five hundred records maximum on one diskette with 48K.
Copies: **20**
Price: **$40**
Includes: Diskette and instructions. Source listing available by sending SASE with serial number.
Author: **Claudia Vaughn**
Available from: Local Apple dealers or:
Peripherals Unlimited
6012 Warwood Road
Lakewood, CA 90713

Name: **APPLE PILOT**
System: **Apple II**
Memory: **16K tape I/O, 32K Disk I/O**
Language: **Interpreter in Applesoft II**
Hardware: **Apple II**
Description: A language to write games and school lessons with. Only 8 commands to learn plus special Apple graphics and tone commands.
Copies in circulation: **10**
Price: **$20.** Add $5 for a diskette.
Includes: Tape and manual and 1 year updates.
Author: **Earl Keyser**
Available from:
The Pilot Exchange
22 Clover Lane
Mason City, LA 50401

Name: **Programs for Indoor Advertising Applications**
System: **Apple II**
Memory: **16K**
Language: **Integer BASIC and Machine Language**
Hardware: **Standard Apple II**
Description: This Program allows the Apple to be used as an automated Advertising machine for stores, trade shows, etc.

HI-RES ALPHANUMERIC MESSAGES: 28 Characters per line, 4 lines, 3 pages of text. Features a right-side 'word-rap' plus instant 'page desolve', as one page ends and the next begins. Characters are crisp and can be Lavender or Green on a Black Background. They 'puff' on at reading speed.

GIANT-LETTER SEQUENCES: Brilliantly-colored letters, of full screen height appear one-at-a-time, in sequence, to spell out messages. The color of Successive Words progresses through the Apple rainbow. A running summary of letters appears in the bottom four screen lines, as the giant letters are presented.

THE SCROLLING WONDER: Allows user to enter up to four brief messages. They appear in Apple upper case by 'popping' onto the screen from below. Messages enter in random sequence, with random space between them. They have random horizontal placement and a random 50% sample of the messages 'flash'. A multiple-rainbow grand finale ends the program.

Copies: **All just released**
Prices: SCROLLING WONDER **$8.00**
GIANT-LETTER SEQUENCES **$8.00**
HI-RES ALPHANUMERIC MSG **$15.00**
ALL THREE PROGRAMS **$25.00**
Includes: Cassette only, with verbal instructions on reverse side of cassette and written instructions on screen.
Author: **Howard Rothman**
Available from:
Connecticut Information Systems Co.
218 Huntington Road
Bridgeport, CT 06608
203/579-0472

Name: **Hangman**
System: **Apple II**
Memory: **20K minimum**
Language: **BASIC**
Hardware: **Apple II, Disk II**
Description: This program is the old traditional Hangman we used to play with pencil and paper except that the computer will choose the word for you to guess. The disk comes with over 350 words and has routines accessed with 'ESC' to add or change words. Gallows is in lores and neck stretches when floor drops.
Copies: **Aprox. 25**
Price: **$14.00** post paid. Calif. residents add sales tax
Includes: Disk with program and over 350 words.
Order Info: Master Charge and Visa accepted.
Author: **Loy Spurlock**
Available from:
Computer Forum Company
14052 E. Firestone Blvd.
Santa Fe Springs, CA 90670

Name: **Feet and Inches Calculator**
System: **Apple II**
Memory: **16K**
Language: **Applesoft ROM**
Hardware: **Applesoft ROM**
Description: This program does calculations based on entries made in feet and inches. Functions include addition, subtraction, division, multiplication, roots, powers and decimal equivalents. Operating screen consists of three windows: one for entries, one lists functions, and the third reproduces the problem after entry. Performs calculations to 1/64". Has memory which allows recall of last answer for next problem.
Copies: **Just released**
Price: **$10.00**
Includes: Cassette tape
Author: **Dick Dickinson**
Available from:
Dick Dickinson
5400 Western Hills Drive
Austin, TX 78731

Name: **BLOCKADE**
Systems: **Challenger IIP**
Memory Required: **4K**
Language: **BASIC and assembly**
Hardware Required: **Challenger II or III**
Description: Two players are needed to play this challenging game in which the object is to block out your opponent before he blocks you out! Each play has four keys for NESW direction, which enable you to construct a wall, trying to block out the other player. The first person to run into the wall loses. Programmed for large characters, or small. Uses Assembly for fast clearing of the screen and printing of characters. Complete with scoring.
Copies: **Lots!**
Price: **$8.00** for listing, cassette, and instructions.
$4.00 for listing and instructions only.
Includes: Cassette at 300 Baud. ($8).
Author: **Bill Langford**
Available from:
Bill Langford
3823 Malec Circle
Sarasota, Fla. 33583

Name: **OSI Games**
System: **OSI Superboard II/Challenger 1P**
Memory: Not specified
Language: Not specified
Hardware: Not specified
Description: **Dodgem** - use strategy to get your pieces off the opposite side of the board (1 or 2 players). **Tank Attack** - seek and destroy enemy guns hidden among houses and trees before they get you (1 player). **Free-for-all** - airplane, destroyer, and submarine vie for each other (1 or 2 players). **Hidden Maze** - find your way through an invisible maze with one-way gates (1 or 2 players).
Copies: Not specified
Price: **$7.95** († 75 cents postage)
Includes: Tape cassette, instruction booklet.
Author: Not specified
Available from: A large number of dealers or:
Creative Computing Software
P.O. Box 789-M
Morristown, NJ 07960
201/540-0445

Name: **3D Graphics**
System: **Apple II**
Memory: **16K**
Language: **Floating Point BASIC**
Hardware: **Apple II** (Applesoft ROM for Load and Go option)
Description: Accurate 3D to 2D wire frame perspective transformations of your data bases. The standard software package contains the BASIC listing for transformation of 3D line endpoints (X,Y,Z coordinates) to perspective drawing endpoints in two dimensions (X,Y coordinates) for high-resolution plotting. User has control over location in space, direction of view, and viewing window (telephoto or wide angle). User must be able to run floating point BASIC and hi-res graphics simultaneously. Optional Load and Go version is specifically for Applesoft ROM and includes a sample data base and output-plotting interface. It is truly Load and Go.
Copies: **Over 200 sold**
Price: **$22** ($26 with Load and Go option)
Includes: 60 page manual and listing (Applesoft II cassette with Load and Go option)
Author: **Bruce Artwick** (option by **Jim Harter**)
Available from:
SubLOGIC
P.O. Box V
Savoy, IL 61874
217/367-0299

Name: **Program Catalog**
System: **Apple II**
Memory: **24K minimum**
Language: **BASIC**
Hardware: **Apple II, Disk II**
Description: This program will catalog all your disk programs by category on one disk. It will keep track of all your programs and which disks they are on as well as keeping notes about the program so you can be sure of the program before you hit the proper key to have this program load and run the program you want. It also contains numerous routines to manipulate the information.
Copies: **New, just released.**
Price: **$19.00** post paid. Calif. residents add sales tax.
Includes: Program on disk, documentation
Order Info: Master Charge and Visa accepted.
Author: **Loy Spurlock**
Available from:
Computer Forum Company
14052 E. Firestone Blvd.
Santa Fe Springs, CA 90670

Editor's Note: The **MICRO Software Catalog** was the most mentioned article in our recent reader survey. If you have software you would like to bring to the attention of the **MICRO** readers, simply type it up in the proper format and send it in. Please adhere to the format as strictly as possible, including UPPER and lower case, titles, and so forth. Since this material will be typeset someone has to get it into proper form. If you submit it in proper form, you increase your chances for early inclusion in **MICRO**. There is no charge for appearing in this catalog.

We are happy to see some programs for the OSI systems appearing.

Name: **DB/65**
System: **ANY 28 or 40 PIN 6500**
Hardware: **Power supply and terminal**
Power Requirements: 5V **at 3 AMPS. ±12, -12 at 20 Milliamps if RS232C terminal used.**
Description: DB/65 is a complete hardware/software debug system for any 6500 system. Command structure is identical to that of the ROCKWELL SYSTEM 65. Hardware breakpoint, scope syne, eight software breakpoints and any number of real-time breakpoints (via the BRK instruction) are supported. Object code and symbol table may be loaded from either serial or parallel port (compatable with SYSTEM 65 printer port). Symbolic disassembly is supported so programmer is always debugging at assembler level. In circuit emulation and 2K RAM are standard. RAM may be added for total of 8K if desired. User NMI and IRQ vectors and supported. System monitor resides in address range $7000 to $7FFF so user program may occupy high memory. 2MHZ option available.
Copies sold: **15**
Price: **$1450**
Includes: Manuals, In circuit emulation, 2K RAM shipping
Developed by: **COMPAS MICROSYSTEMS**
Available from:

COMPAS MICROSYSTEMS
224 SE 16th Street
P.O. Box 687
Ames, IA 50010
515/232-8181

Name: **Home Budget System**
System: **OSI** (Easily modified for PET or Apple II)
Memory: **4K**
Language: **MICROSOFT BASIC**
Hardware: **OSI Challenger IIP**
Description: A computerization of my own proven home budget system evolved over a 7 year period. Consists of interactive programs to add/update accounts, post budget and expenses and analyze status of accounts on detailed and summary basis. 4K RAM handles up to 15 accounts stored on cassette tape. Data stored for each account includes account number, description, budget amount, current month expenses, and year-to-date expenses. Requires posting only once per month. Helps balance checkbook, too!
Copies: **Just released**
Price: **$15**
Includes: Cassette (300 baud Kansas City std), user manual with complete BASIC listings, operating instructions, and sample runs.
Author: **Bruce Grayson**
Available from:

B. W. Grayson
905 Woodridge Drive
Savannah, Georgia 31410

EXPAND YOUR 6052-BASED TIM MONITOR

Russell Rittimann
2606 Willow Crest
San Antonio, TX 78247

This modification to TIM will expand its command set such that ROM resident programs or routines can be executed from within TIM. Since I had several programs in PROM (BASIC, assembler, etc.) that were used regularly, I wanted an easy way to execute them without the usual sequence of: displaying the registers; setting the program counter; and finally typing "G". Now my TIM monitor will recognize a "B" from the keyboard and immediately put me into BASIC, and similarly recognize other commands for the other programs.

The TIM manual from MOS TECHNOLOGY included a complete listing of the monitor program. The sequence for recognizing a command in TIM was: output the prompting "."; read the command; look the command up in a table; and then execute the command by indirectly jumping to the address of the routine that corresponded to the command. This sequence of instructions is located from 708F(16) to 70B4(16) in the TIM monitor. All I needed to do is intercept the command and check it against my own table before letting TIM have its turn at it, which presented a problem since the TIM program is in ROM and can't be changed.

What I did was to disable TIM for a "window" of 16 locations from 7090(16) to 709F(16) and enable a DM8578 32 x 8 PROM at these same locations. Figure 1 shows the schematic for the PROM and address decoding. Note, that the 3-input NAND gate connected to CS2 of TIM, limits the monitor to between 6000(16) and 7FFF(16). This was not shown in the TIM manual.

I programmed the first half of the 8578 identical to the 16 locations in TIM starting at 7090(16) except for locations 4, 5, 6 (corresponding to TIM's 7094(16) - 7096(16).) In TIM, these 3 locations are a jump to subroutine to read a character from the keyboard. Instead, I put a jump to location CC00(16) where I had a 2708 EPROM decoded. The program in the 8578 is shown in Figure 2.

Figure 3 shows the program in the 2708. This instruction sequence receives the command from the keyboard and checks it against its command table. If not found, program control is returned to TIM at location 7098(16) to check its commands. If the command is user-defined, then the program jumps indirectly to the routine

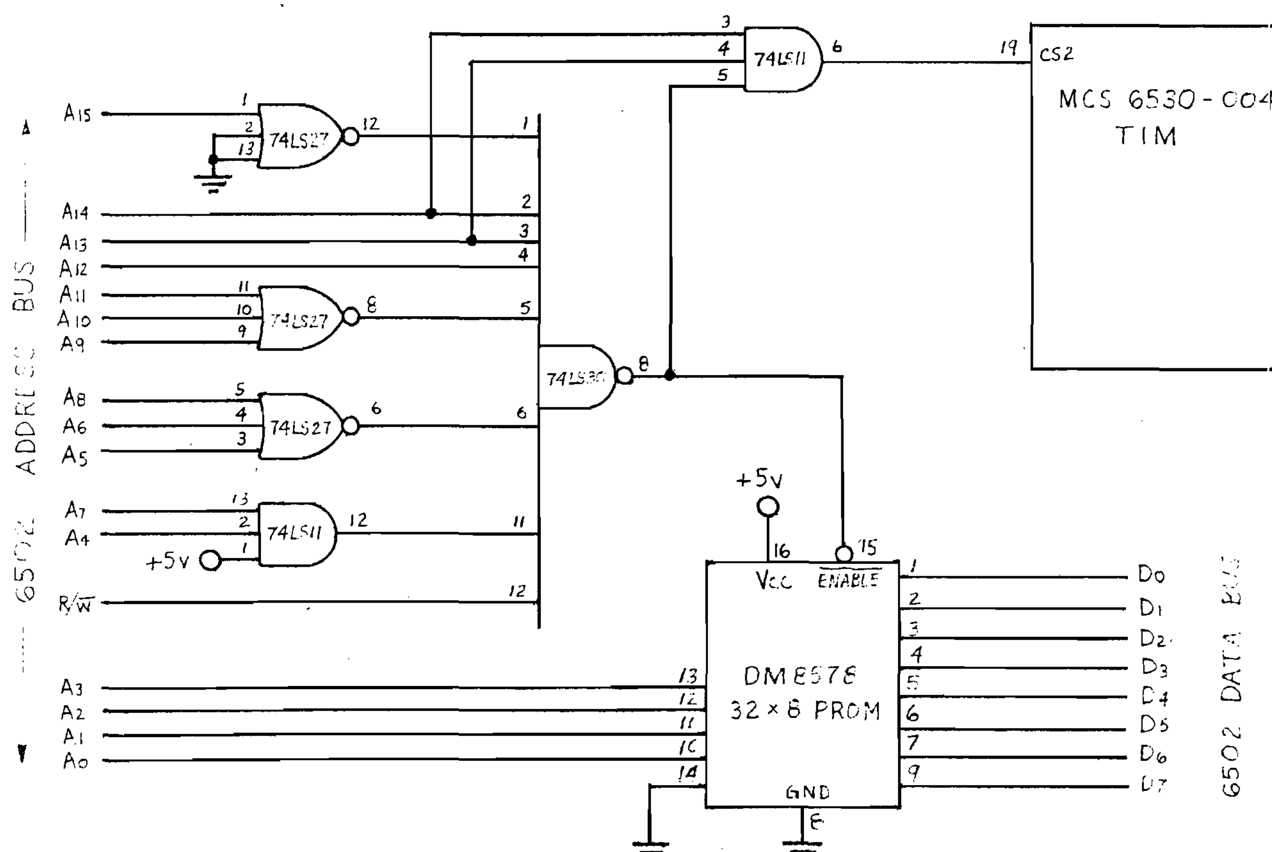


Figure 1
Schematic Diagram

```
LOC.    CONTENTS    INSTR.          COMMENTS

0000-   2E          .HS $2E         TIM PROMPTER "."
0001-   20 C6 72    JSR $72C6       OUTPUT PROMPTER USING TIM OUTPUT ROUTINE
0004-   4C 00 CC    JMP $CC00       JUMP INTO 2708 EPROM
0007-   A2 06       LDX #NCMDS-1    FOLLOWING INSTRUCTIONS AS IN TIM
0009-   DD 06 71    CMP CMDS,X
000C-   D0 19       BNE S2
000E-   A5 FD       LDA SAVX
```

Figure 2
Program in 8578 PROM

whose address is immediately following the command letter in the table. User-defined commands will have priority over TIM's commands. The format for each command in the table is as follows: command letter, low address of routine, high address of routine. Since the 2708 is erased to all 1's, I used FF(16) for the delimiter to signify the end of the table. Thus, the table can be added to at any time by programming 3 bytes.

Some final comments: I located the 2708 at CC00(16) but it can be located anywhere by changing the address in the 8578 and the address of the command table. At the end of each routine added, there must be a jump to 7086(16) to get back into TIM. The first byte of the 8578 is the TIM prompting character. If you want something other than the period, program any character you want into this location. Since the 8578 is an irreversible PROM, and I only used the first 16 locations, if you make a mistake in burning the PROM, the second half can be used by connecting the high address line, A4, to Vcc. Also, check the 2708 before the 8578 is wired since this modification won't work without all chips installed correctly.

This modification converts TIM into an adaptable operating system. Anytime I get more resident routines, I can add them to TIM by programming three locations into the command table in the 2708.

```
LOC.    CONTENTS          INSTR.              COMMENTS

CC00-   20 E9 72          JSR $72E9           GET COMMAND USING TIM INPUT ROUTINE
CC03-   A2 00             LDX #$00            X IS INDEX INTO TABLE
CC05-   BC 28 CC    LOOP  LDY TABL,X          CHECK COMMAND LETTER IN TABLE FOR DEFAULT
                                                  DELIMITER
CC08-   C0 FF             CPY #$FF
CC0A-   D0 03             BNE CHEK            IF NOT DELIMITER, COMPARE COMMAND FROM KEYBOARD
CC0C-   4C 97 70          JMP $7097           OTHERWISE, JUMP BACK INTO TIM
CC0F-   DD 28 CC    CHEK  CMP TABL,X          CHECK KEYBOARD COMMAND AGAINST TABLE
CC12-   D0 0F             BNE NEXT            IF NOT COMMAND, CHECK NEXT IN TABLE
CC14-   E8                INX                 FOUND COMMAND
CC15-   BD 28 CC          LDA TABL,X          GET LOW ADDRESS OF ROUTINE
CC18-   85 EC             STA $EC
CC1A-   E8                INX
CC1B-   BD 28 CC          LDA TABL,X          GET HIGH ADDRESS OF ROUTINE
CC1E-   85 ED             STA $ED
CC20-   6C EC 00          JMP ($00EC)         JUMP INDIRECT TO ROUTINE
CC23-   E8          NEXT  INX                 INCREMENT POINTER TO NEXT COMMAND
CC24-   E8                INX
CC25-   E8                INX
CC26-   D0 DD             BNE LOOP            GO BACK AND CHECK REST OF COMMANDS
CC28-   2A          TABL  .HS $2A             COMMAND LETTER "*"
CC29-   92                .HS $92             LOW ADDRESS OF ROUTINE #1
CC2A-   CC                .HS $CC             HIGH ADDRESS OF ROUTINE #1
CC2B-   42                .HS $42             COMMAND LETTER "B" FOR BASIC PROGRAM
CC2C-   A1                .HS $A1             LOW ADDRESS OF BASIC PROGRAM
CC2D-   CC                .HS $CC             HIGH ADDRESS OF BASIC PROGRAM
CC2E-   FF                .HS $FF             END OF TABLE DELIMITER
```

Figure 3
Program in 2708 EPROM

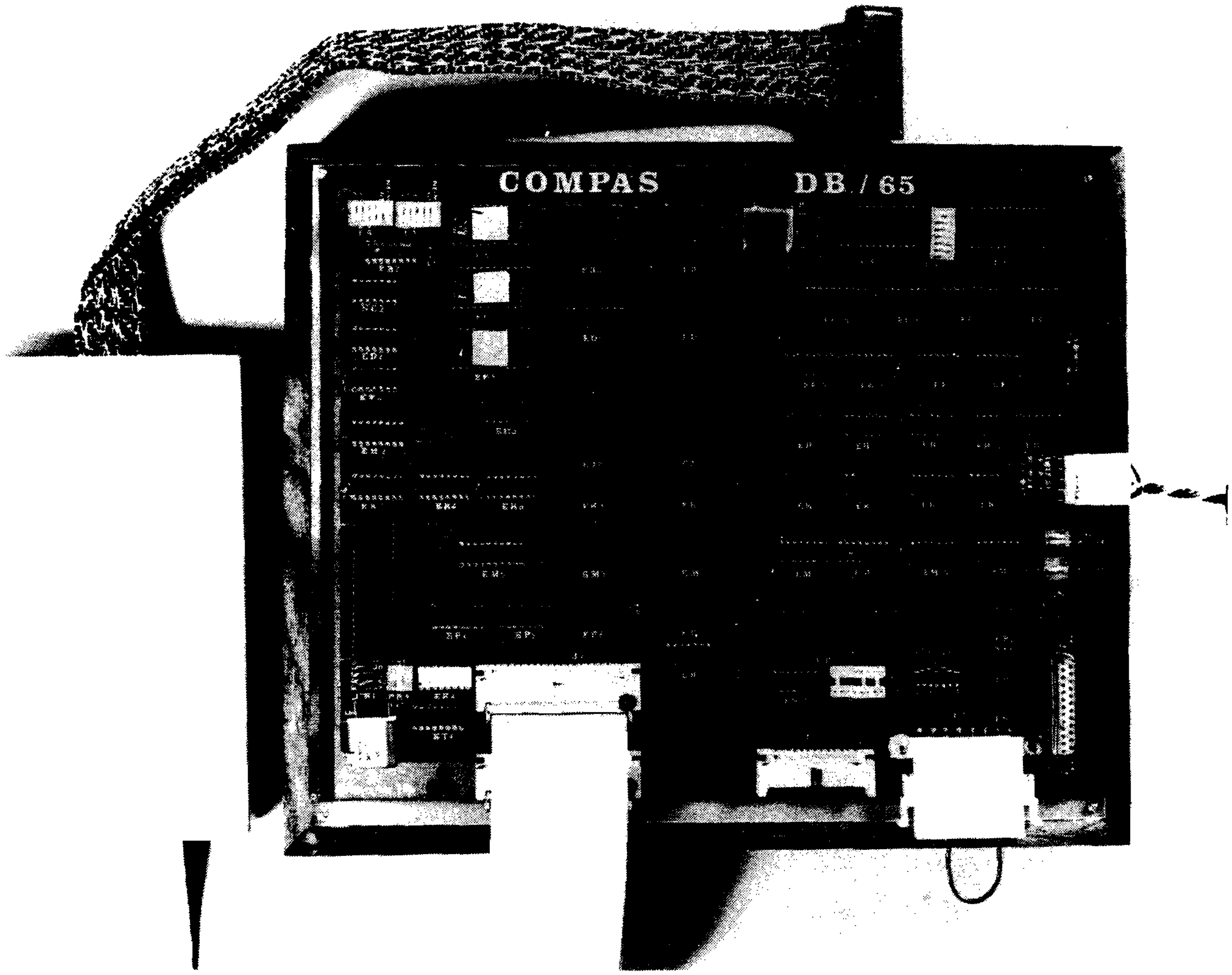
compas microsystems

P.O. Box 687 224 S.E. 16th Street Ames, Iowa 50010

515/232-8187

DB/65

A complete hardware/software debug system for the Rockwell, Synertek, MOS/Technology 6500 microprocessor family.



Features

* Standard in-circuit emulator
* Hardware breakpoints
* Single step mode
* Eight software breakpoints
* Real-time software breakpoints
* RS 232C or current loop terminals
* Symbolic disassembly of user program
* Serial/parallel load capability
* Program trace of instructions and registers
* Prom resident debug monitor
* Software history of instruction addresses
* 2K ram standard with sockets for additional 6K if required
* Scope sync output
* User NMI and IRQ vectors supported
* Write protect
* User program may reside in high memory

SINGLE QUANTITY PRICE = $1450

6502 BIBLIOGRAPHY
PART VIII

William R. Dial
438 Roslyn Avenue
Akron, OH 44320

397. Babcock, Robert E. "1C Tester Using the KIM-1"
Ham Radio 11 No 11 pg 74-76 (Nov., 1978)
Test the 7400 series IC's using the KIM-1, a minimum of hardware and tables of parameters tucked away in memory.

398. Purser, Robert "Software List"
Robert Purser's Reference List of Computer Cassettes, Edition 3, August, 1978 (P.O. Box 466, El Dorado, CA 95623
A very complete listing of available Cassette software for the PET and Apple II.

399. Lilie, Paul A. "Look What Followed Me Home!"
73 Magazine No 218 pg 142-147 (Nov., 1978)
A description of the PET.

400. Creason, Sam "The Micro Maestro!"
73 Magazine No. 218 pg 150-166 (Nov., 1978)
Sound generation and waveform control with the 6502.

401. Akingbehin, Kiumi "LEDIP, A KIM/6502 Test Editor"
Dr. Dobb's Journal 3 Issue 9 No 29 pg 4-12 (Oct., 1978)
Here is an expandable program for creating text and source code.

402. Tepperman, Barry "Comments on KIM Cassette Program"
Dr. Dobb's Journal 3 Issue 9 No 29 pg 41 (Oct., 1978)
Points out that the relatively slow speed of the KIM cassette program has led to the publication of several high-speed load/dump programs.

403. Firebaugh, Morris; Johnson, Luther and Stone, William "A Feast of Microcomputers"
Personal Computing 2 No 11 pg 60-70 (Nov., 1978)
The Authors evaluated a wide range of microcomputers to pick the best ones for teaching science students. Includes several 6502 micros.

404. Creative Computing 4 No 6 [Nov./Dec., 1978]
Foote, Gary A. "Apple Speed"
A comparison of several programs for sorting a group of 1000 words showed several BASIC programs to require 600 to 650 seconds on the Apple II while using the same sort in Sweet-16 required only 158 seconds. The same sort in 6502 assembler required only 3 seconds.

Ahl, David "Random Ramblings"
Commodore plans to make an electronic chess game based on the 6504 chip of MOS Technology.

Yob, Gregory "Personal Electronic Transactions"
A column on the PET with software references, hints on operating, etc.

Milewski, Richard A. "Apple-Cart"
A column on the Apple II with software reviews.

Butterfield, Jim "Games—Not Just For Fun"
The author urges micro users to have fun with their computers; don't be ashamed of games and recreational programs. Creating programs is in itself a highly instructive experience.

405. Dinnell, Rob C. "Graphics Program"
Interface Age 3 Issue 11 pg 14 (Nov., 1978)
Graphics program for the Apple II.

406. Schumacher, Ernst "Sweets for KIM Spurned"
Byte 3 No 11 pg 146 (Nov., 1978)
A fix for a bug in the Sweets for KIM program, Byte Feb., 1978 pg 62.

407. Creative Computing 4 No 5 [Sept./Oct., 1978]

Ahl, D.H. "Personal Computing: The size of the Market"

Out of a total market for personal computers sold in the first three years of 150,000 units, PET is said to account for 15000, TRS-80 for 8000 to 20000 and Apple for 25000 units. All others together account for 75000 to 100000.

Ahl, D.H. "The Home Computer: A Tool Not A Toy"

An interview with Mike Scott, President, Apple Computer.

Ahl, D.H. "Home Computers: The Name of the Game is Peripherals"

An interview with Chuck Peddle, designer of the Commodore PET.

Ahl, D.H. "Reliability and Mass Production"

The most frequent computer problems over all manufacturers including the 6502 types fall into two categories: (1) Cassette recorder, mostly head misalignment and (2) overheating errors after running a while.

North, Steve "PET Cassettes from Peninsula School"

A review of software available from the Peninsula School of Monlo Park, CA.

408. Anon, "12-Test Benchmark Study Results Show How Three Microprocessors Stack Up"

EDN 22 No 21 pg 19 (Nov. 20, 1978)

Once again the 6502 is shown to be substantially faster than the 8080 or 6800, as well as using less memory.

409. Anon, "Project Indecomp–EDN Builds a computer System"

EDN 22 No 21 pg 221-233 (Nov. 20, 1978)

Outlines the beginning of this project that was to provide material for a number of articles to follow, principally on methods of interfacing to a tape deck.

410 Schreirer, Paul G. "Low-Cost System Requirements Multiply Interface Headaches"

EDN 23 No 3 pg 39-44 (Feb. 5, 1978)

Interconnecting a cassette system to Indecomp proved tough due to strong chip-discrimination against the 6502 in this 8080/Z80 world.

411 Call - Apple 1 No 10 [Nov./Dec., 1978]

Anon, "Use of Apple II Color Graphics in Assembly Language"

Tutorial article on graphics

Jackson, Gene "Checkbook Changes for Disk"

Modifications for this popular program for the Apple Disk.

Paulson, Steve, "Using Game-Paddle Buttons"

How to change keyboard control over to the paddle buttons.

Anon, "& Now, the Further Adventures of the Mysterious Ampersand."

Continued from last month—more on the functions of the character "&", in Applesoft routines.

Wigginton, R. "Simple Tones–A Demonstration for Extensions to Applesoft II."

Simple tone program for Applesoft II inside the Applesoft Program.

Finn, Jeffrey K. "Apple-Sharing" Part I of II

Part I of a tutorial article on time sharing and the Apple.

Thyng, Mike "Apple Mash"

This issue discusses how and why the DIMensioned statement works, Alpha String arrays, Integar and Floating Point Arrays, etc.

Anon, "Peeks, Pokes and Calls"

A discussion of the utility of these very useful tools.

Thyng, Mike "Apple Source"

Question and answer session with Mike Scott, President of Apple Computer and Randy Wigginton of Apple.

Golding, Val J. "Identifying Binary Disk Programs"
Ways to help you save and identify machine language programs on disk.

Anon, "Resurecting a Dead FP Program."
Methods to help you retrieve an Applesoft II program that has blown up while you were working on it.

412. Southeastern Software Newsletter Issue No 4 [Nov., 1978]

Anon, "Hires Graphics"
Examples of how to program in Hires machine language. Also includes a program in Applesoft II called Random Walk

Anon, "How to Use "Quotation" Marks in a Print Statement."
Tricky in Applesoft II to make the quote marks print.

Hartley, Tim "How Memory is interpreted in Integar Basic"
A program to list the tokens used in Integar Basic.

Banks, Guil "Programs for Disk"
Two programs are given. EXEC GEN and READ FILE.

Anon, "Applesoft in Firmware"
A discussion of the use of the Applesoft II ROM card.

413. Carpenter, C.R. [Chuck] "Pilot for the Apple"
People's Computers 7 No 3 pg 4 (Nov./Dec., 1978)
An extended version of PILOT for the Apple Disc II is being written.

414 Cole, Phyllis "SPOT"
People's Computers 7 No 3 pg 48-51 (Nov./Dec., 1978)
Hints on using the Commodore PET include tips for loading balky tapes from the cassette, adding an auxilliary keyboard, and review of new software.

415. Greenberg, Gary "Phone Directory"
Personal Computing 2 No 12 pg 34-35 (December, 1978)
A PET program provides rapid access to a phone number without a random access filing system.

416. Zimmermann, Mark "Assembler for the PET"
Personal Computing 2 No 12 pg 42-45 (December, 1978)
This BASIC program lets you write in Assembly Language.

417. Gable, G.H. "Zapper--A Computer Driven EROM Programmer"
Byte 3 no 12 pg 100-106 (December, 1978)
The Zapper is a Erom programmer using a KIM-1 as driver for the Zapper.

418. Watson, Allen, III 430 Lakeview Way, Redwood City, CA 94062
Byte 3 No 12 pg 208 (December, 1978)
Notes on minimizing TV interference by the Apple II.

419. Lantz, Kim H. "RTTY with the KIM"
73 Magazine Issue 219 pg 170-173 (December, 1978)
This article goes a step further and uses the KIM to deliver the RTTY to the HAL terminal.

420. Anon. "Bringing up the New Disk"
Southeastern Software Newsletter Issue No 5, Pg 2 (Dec., 1978)
Hints and Kinks on putting that newly delivered Apple Disk to work. Making duplicate masters, creating random files, reading back files, transferring programs from one disk to another for backup, etc.

HOW DOES 16 GET YOU 10?

Gary P. Sandberg
1144 Amber Ridge Drive
Lilburn, GA 30247

In order to PEEK, POKE, figure CALL numbers, etc. effectively a knowledge of Hexadecimal / Decimal conversion is a necessity. My experience during the past ten years, working with computer systems and data processing equipment did not include anything that required hexadecimal addressing and coding. When I started using my Apple II, I was completely lost and confused with base 16 math. I began looking for a way to work with hexadecimal effectively. The following conversion table was the answer.

HEXADECIMAL / DECIMAL CONVERSION TABLE

	16^3	16^2	16^1	16^0
0	0	0	0	0
1	4,096	256	16	1
2	8,192	512	32	2
3	12,288	768	48	3
4	16,384	1,024	64	4
5	20,480	1,280	80	5
6	24,576	1,536	96	6
7	28,672	1,792	112	7
8	32,768	2,048	128	8
9	36,864	2,304	144	9
A	40,960	2,560	160	10
B	45,056	2,816	176	11
C	49,152	3,072	192	12
D	53,248	3,328	208	13
E	57,344	3,584	224	14
F	61,440	3,840	240	15

To convert a number from hexadecimal to decimal;

1. in each column of the table, find the decimal equivalent for the hexadecimal digit in that position.
2. add the decimal equivalents, found in step #1, to obtain the decimal number.

Hopefully the following examples will help you master the use of the conversion table.

To convert a number from decimal to hexadecimal;

1. In the table find the largest decimal value that will fit into the decimal number to be converted.
2. note its **column position** and hexadecimal equivalent.
3. find the decimal remainder (subtract)
4. repeat steps 1, 2, & 3 for each remainder. When a hexadecimal equivalent has been found in the **right most column**, the conversion is done.

Convert Hex to Decimal using the conversion table.

16^3	16^2	16^1	16^0		Convert from left to right.	
			E_{16}	=	14	
		5		=	80	list and
	E			=	3584	ADD TOGETHER
F				=	61440	
		$FE5E_{16}$		=	65118_{10}	

Convert Decimal to Hex using the conversion table.

```
                              16^3  16^2  16^1  16^0

  65118
 -61440  from table =   F
   3678
  -3584  from table =         E
     94
    -80  from table =               5
     14  from table =                     E

  65118(10)          =      FE5E(16)
```

Remember the Apple II's system monitor can help you with some of your hexadecimal problems. The monitor will do hexadecimal addition and subtraction, as shown on page 70 of the Apple II reference manual.

The Apple II's PEEK function also can be helpful. In BASIC key in PRINT PEEK (2), the Apple II will display on the screen the decimal value of decimal memory location 2.

Use the POKE statement to change memory location 2, In BASIC key in POKE 2,255, then Return. Then PRINT PEEK (2), Return. The Apple will display 255.

Then CALL -151, or hit Reset. The Apple II is now in the System Monitor. Key in 0002 or 2, Return, and the Apple II displays 0002-FF. Why?, because we put the decimal value 255 into memory location 2 with the POKE statement, 255(10) is equal to FF(16), get the idea?

For some conversions from hexadecimal to decimal or back the other way, you can use the POKE and PEEK method, but for most conversions use the table.

Here are two more examples that don't use a conversion table: same numbers different method of conversion:

Convert Hex to Decimal without using the conversion table.

```
First digit is      * 1       E * 1    =    14
Second digit is     * 16      5 * 16   =    80
Third digit is      * 256     E * 256  =  3584
Fourth digit is     * 4096    F * 4096 = 61440
                                         -----
                              FE5E(16) = 65118(10)
```

Convert Decimal to Hex without using the conversion table.

```
65118 / 16 = 4069.875 ->   .875 * 16 = 14  =         E
 4069 / 16 =  254.3125 -> .3125 * 16 =  5  =       5
  254 / 16 =   15.875      .875 * 16 = 14  =     E
   15 / 16 =     .9375    .9375 * 16 = 15  =   F
           65118(10)          =              FE5E(16)
```

Use either method to convert from one number system to the other, and with a little practice you will be converting numbers with speed and accuracy.

Several readers, and the author, pointed out a small bug in "The SYM-1 Tape Directory" by John Gieryic (MICRO 8:35). In the subroutine DELAY, line 02BB should read 20 06 89 not 20 0B 89.

A few errors occurred in the "Inside PET BASIC" programs by Jim Butterfield (MICRO 8:39):

Line 9000 X=PEEK(1029) FOR should be
X=PEEK(1029):FOR
Line 9005 FI should be IF
Line 60010 T0= should be T=0
Line 60120 ?"GO";"L" should be
?"GO";V;"L"
60240 C<9 should be C<=9
60250 S+44 should be S=44

Notes

Harvey B. Herman's "Peeking at PET's BASIC" (MICRO 7:47) prompted two notes - one from the author and one from Commodore.

John Feagans of Commodore wrote:

"Mr. Harvey B. Herman's comments about peeking at PET BASIC were misinformed, and I would like to set the record straight. Microsoft Co. inserted the code to protect their copyright on 6502 BASIC. Commodore is only maintaining a contractual obligation not to reveal the ROM contents. I personally believe the protection on peek is ineffective since a machine language PEEK program can easily be written. However, to rip off the BASIC and alter it requires symbols, and most hobbyist disassemblers do not generate symbols so code may be reassembled elsewhere. I originally wrote the published PET peek machine language program exactly as reprinted for a PET bulletin."

Harvey B. Herman writes:

"I recently found out some information which at the same time , partially blunts and reinforces criticism of Commodore in my recent article. The East Bay PET User's Group (Sphinx) publishes a very useful newsletter. In one issue (Vol 0, No 2) is a reprint of a Commodore Bulletin containing a description of a program much like the one in my article. This program in conjunction with the User function allows PEEKing at any memory location. If as I requested, Commodore had sent me their bulletins (I have only received ads for nonexistant software) or if Commodore had sent a complete set to my PET owner colleague (he has received some early ones but not that one) I could have saved lots of work and aggravation. Developing the program independently did teach me a lot about the workings of the PET but I still would have preferred to spend my time on other things. Apparently Commodore would like to be helpful but their bulletins are not getting out into the field. Magazines like MICRO and newsletters like Sphinx's are helping to fill in the PET information gap."

Announcements

A new Apple II users group has been formed in the Denver area. We call ouselves Apple Pi. We meet at 7:30 the first Thursday of each month in room 271 of the Green Center on the Colorado School of Mines Campus in Golden. Contact:

Austin R. Brown, Jr.
407 Peery Parkway
Golden, CO 80401
303/279-5388 (home)
303/279-0300, x2434 (work)

An Apple II users group is forming in the Boston area under the direction of Richard Suitor (who has had several excellent articles published in MCIRO). Contact him for information.

Richard Suitor
166 Tremont Street
Newton, MA 02158

PET users in the Boston area should contact Jim Yost who is directing a users group as part of the Boston Computer Society. Call him at:
617/625-4295

Johnson Computer does have a PROMable version of BASIC available. Questions were raised in an earlier issue of MICRO about whether or not the BASIC could be put into PROM. Contact Johnson for details on their PROMable version. It is not the same as the standard version offered.

HOW GOES YOUR ROM TODAY?

Harvey B. Herman
Chemistry Department
University of North Carolina-Greensboro
Greensboro, North Carolina 27412

Everytime I turn on my KIM-system or PET Personal Computer I keep my fingers crossed that everything works. So far I have been "lucky" and the few failures were patently obvious. However, I have been concerned about the possibility of subtle errors appearing which, while not obvious, will still cause programs to print garbage out without my having inputted garbage. To ease my troubled mind, I wrote an assembly language program which computes a checksum byte from the data in a specified area of memory. The 6502 programs, which I named CHECK, can be used to check data in both ROMs and RAMs for erroneous bits.

The program for a KIM system is shown in Figure 1. It can be entered into memory with the KIM monitor program or an assembler. With a few minor changes, which I believe are obvious by looking at the code, it can be placed practically anywhere in memory. The program requires four zero page locations to be initialized to the starting and ending locations of the specified area. I used locations hex E1, E2 and E3, E4 respectively (low byte first) as these were the first free page zero locations in Microsoft 8K BASIC. The reader may wish to change these locations if it interfers with other programs that are frequently used. The KIM CHECK program ends with a BRK (break) instruction and will not operate properly unless two locations, hex 17FE, 17FF, are initialized to 00, 1C, respectively. The BRK instruction, when executed will then jump to the start of the KIM monitor and among other things, print the value saved in location hex 31D - the calculated checksum. Initialization and executation of this program can be done with the KIM monitor. The checksum bytes which I calculated for two different KIM system ROMs are shown in Table 1.

Several changes are necessary that allow a similar program to work on Commodore's PET computer. The modified program is shown in figure 2 and is a listing from a cross assembly done on the KIM system. The values could be placed in memory with a monitor program, if available, or as I did, poked into memory from a BASIC program. The latter approach requires a conversion from hex to decimal before using the POKE command. Again, as before, four locations in page zero need to be initialized. Part of the area reserved for the second cassette buffer was used for the program (hex 33A-371) and four locations (hex 53-56) in the keyboard buffer were used for the page zero locations representing the starting and ending locations of the area to be checked. The PET CHECK program is designed to be run from BASIC. A call to the USR (user) function, ?USR(0), jumps to the checksum program and returns the checksum value. The program has two entry points. It can be used to calculate checksums (see Table 1) for the BASIC interpreter and/or the operating system (both are in ROM) or BASIC programs which have just been loaded or saved. The latter use somewhat obviates the need to use the VERIFY tape command after a load. This can save considerable time particularly if long programs are loaded. Alternate entry points are specified by POKEing locations 1 and 2 to decimal 58 and 3 for program checks and to decimal 82 and 3 for ROM checks, respectively. The starting and ending locations in page zero are automatically set by the program for program checks but must be specified for ROM checks.

Further details on the use of each program is shown in Table 2. The checksums calculated are the exclusive OR of all the bytes between the starting and ending addresses, inclusively. Changing as little as one bit in the sequence will give a different value for the checksum. There is a finite probability that when extensive errors are encountered the checksum calculated would fortuitously be the same, since only 256 different 8 bit checksums are possible. However, in that case the errors would probably not be subtle and you would not be fooled. Whenever the checksums for the ROMs change it would be prudent also to run a diagnostic test on the 6502 MPU before blaming the ROM. Since programs like that are sadly lacking I will leave it as an exercise for the reader. A program and article to that effect would be greatly appreciated by the author for one, and I believe most of 6502 personal computing fraternity.

KIM ROMs (Serial numbers 1988 and 6931)

Locations (Hex)	Checksum (Hex)
1800-1BFF	F5
1C00-1FFF	F8
1800-1FFF	0D

KIM CHECK Program. Example for 1800-1FFF. After placing program from Figure 1 into memory

```
KIM
17FE  0.
17FF  1C.            0300  AD  G
E1    0.             KIM
E2    18.            031D  (CHECKSUM)
E3    FF.
E4    1F.
```

PET ROMs (Serial numbers 10252 & 20549)

Locations (Hex)	Loc.(Dec., Inv.)	Check
C000-CFFF	0,192-255,207	189
D000-DFFF	0,208-255,223	87
E000-E777	0,224-119,231	26
F000-FFFF	0,240-255,255	92

PET CHECK Program. After poking program from Figure 2 into memory

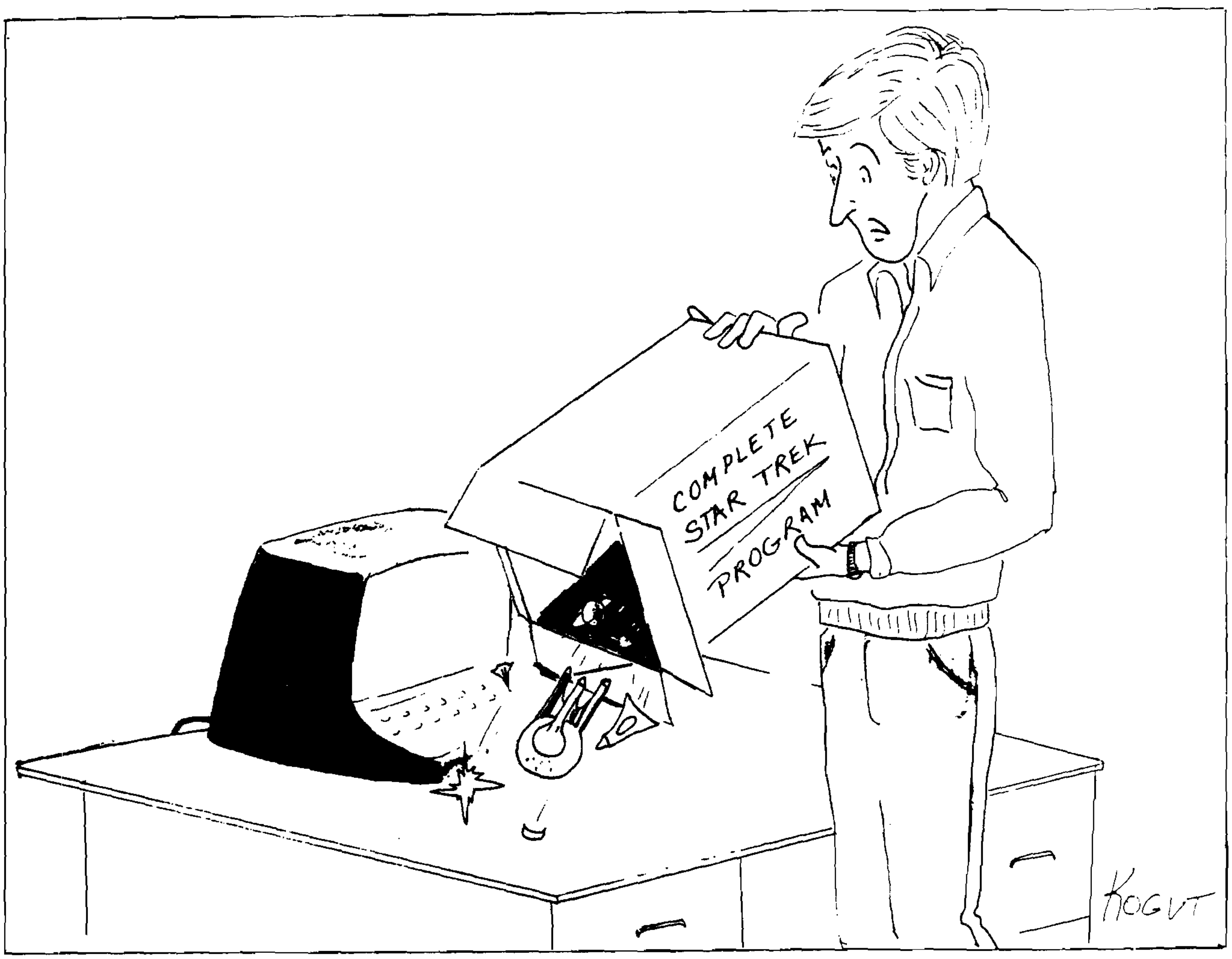
Program Checks	ROM Checks
POKE 1,58	(Example for C000-CFFF)
POKE 2,3	
LOAD "program name"	POKE 1,82
or	POKE 2,3
SAVE "program name"	POKE 83,0
?USR (0)	POKE 84,192
(checksum returned	POKE 85,255
depends on program)	POKE 86,207
	?USR (0)
	189 (Checksum returned)

```
033A                ;       KIM CHECKSUM PRØGRAM
033A                ;       HARVEY B. HERMAN
033A                ;       INITIALIZE $17FE/FF
033A                ;       TØ 0/1C SØ BRK WØRKS.
00E1                        *=$E1
00E1 0000             START .WØRD 0
00E3 0000             END   .WØRD 0
0300                        *=$300
0300                ;       ENTER HERE FØR
0300                ;       CALCULATIØN ØF
0300                ;       CHECKSUM BETWEEN
0300                ;       START AND END.
0300                ;       ANS DISPLAYED LØC 315
0300 A000                   LDY #0
0302 B1E1                   LDA (START),Y
0304 E6E1             LØØP  INC START
0306 D002                   BNE CHECK
0308 E6E2                   INC START+1
030A 51E1             CHECK EØR (START),Y
030C A6E4                   LDX END+1
030E E4E2                   CPX START+1
0310 D0F2                   BNE LØØP
0312 A6E3                   LDX END
0314 E4E1                   CPX START
0316 D0EC                   BNE LØØP
0318 8D1D03                 STA *+5
031B 00                     BRK
031C                        .END
```

Figure 1
KIM Checksum Program.

```
033A                ;      PET CHECKSUM PRØGRAM
033A                ;      HARVEY B. HERMAN
0053                       START=$53
0055                       END=$55
033A                .      *=$33A
033A                ;      ENTER HERE TØ CHECK
033A                ;      BASIC PRØGRAMS AFTER
033A                ;      LØAD ØR SAVE.
033A A900           PRØG   LDA #0
033C 8553                  STA START
033E A904                  LDA #4
0340 8554                  STA START+1
0342 A5E6                  LDA $E6
0344 8556                  STA END+1
0346 A5E5                  LDA $E5
0348 38                    SEC
0349 ED7103                SBC TWØ
034C B002                  BCS SKIP
034E C656                  DEC END+1
0350 8555           SKIP   STA END
0352                ;      ENTER HERE TØ CHECK
0352                ;      ANY LØCATIØNS IN
0352                ;      MEMØRY. INITIALIZE
0352                ;      $53-$56 FIRST.
0352 A000           RØM    LDY #0
0354 B153                  LDA (START),Y
0356 E653           LØØP   INC START
0358 D002                  BNE CHECK
035A E654                  INC START+1
035C 5153           CHECK  EØR (START),Y
035E A656                  LDX END+1
0360 E454                  CPX START+1
0362 D0F2                  BNE LØØP
0364 A655                  LDX END
0366 E453                  CPX START
0368 D0EC                  BNE LØØP
036A A8                    TAY
036B A900                  LDA #0
036D 2078D2                JSR $D278
0370 60                    RTS
0371 02             TWØ    .BYTE 2
0372                       .END
```

Figure 2
PET Checksum Program


COMPLETE
STAR TREK
PROGRAM
KOGUT

LIFE FOR THE KIM-1 AND AN XITEX VIDEO BOARD

Theodore E. Bridge
54 Williamsburg Drive
Springfield, MA 01108

I have been very interested in the game of LIFE ever since I read Martin Gardiner's "Recreational Mathematics" section in the Scientific American - Oct. Nov., 1970. Naturally, I was very much interested in Dr. Frank Covitz' excellent article that appeared on page 5:5 pf the June-July issue of MICRO, 1978.

Just as soon as I got my XITEX video board working on my KIM-1 (16 K on a KIMSI mother board), I attempted to put the Covitz program on my machine. Because the display feature of the XITEX video board is so different from the PET, I thought it was necessary to write a completely new program. I think there may be other KIM-1 users who would like to try my version of this fascinating game.

John Conway invented the game of LIFE. I like to think of it as a simulation of a virus growing on the surface of a POND of DNA. Therefore, I call the work area in which births and deaths are recorded, the POND. I have a routine SHOALL that will display the POND on the screen. I have another routine DISPLY that will add a cell to the screen when a new one is born, and will remove one that is about to die. The POND is updated after each generation in UPDATE. The routine NBRS will record the number of neighbors for a given cell in variable NN. In the pond, zero represents a nonliving cel; (1) represents a living cell; (-1) represents a cell that is about to be born; and (2) represents one that is about to die.

It would take about a second to sweep the entire POND looking for births and deaths, but it takes 1/6 seconds to process a birth or a death. The POND is a matrix 16 x 64. In the routine EDGE, the POND is edged with zeroes to prevent WRAP-AROUND that would destroy symetry in a life form. According to Conway's rules:
1 A new cell is born in an empty cell having 3 neighbors.

2 Any living cell having less than two, or more than three neighbors will die.

3 All deaths and births occur at the same time. A new cell will not be counted as a neighbor until after all cells have been processed.

The POND may be relocated on another page by putting the page number at address $2004. Sixty four ($40) bytes must be reserved immediately before and after the POND for edging with zeroes.

START THE PROGRAM AT $2000

The routine PLANT will put a live cell in the center of the screen, and ask for coordinates V , H for other cells, measured from the center. V is the line number († is down and - is up). H is the column number († is right and - is left). Both V and H must be in the range: minus 7 to plus 7. The sign must follow the digit entered, but a space may be substituted for the plus sign. The following entries will establish a blinker in mid screen.

```
ENTER V,H ? 1-,0†        0
ENTER V,H ? 1†,0†        0
ENTER V,H ? /            0
```

The slash (/) above will terminate the data and start the program.

A generation count is displayed in the upper left corner of the screen. The computer will enter a break if there are no births and no deaths in any generation. To return to the monitor, you will need to insert $1000 in the IRQ vector. – 17FE 00, and in 17FF 1C.

If your video board uses different commands for positioning the cursor, you will need to change the routine DISPLY. The XITEX board uses the following commands.

```
Key   Hex
      Code
ESC   $1B   invokes coordiante mode
'=    $3D   invokes absolute addressing
"V"   BINARY ROW NUMBER - from top
"H"   BINARY COLUMN NUMBER - from left
           (add $40 if less than $20)
'0    $30   will display a zero
'     $20   will overwrite a cell with
            a space
```

If you have a highspeed video board, you might wish to reform the entire display after each generation with this patch:

```
change Address $204F from EC  to  E9
change Address $2271 from 48  to  60
```

An article by David J. Buckingham in the Dec 1978 issue of BYTE, on page 54 gives a great many life forms that you might like to try with this program.

For practice on inputting data, you might like to try the following life forms given by John Gardner in the Oct.-Nov. 1970 issue of the SCIENTIFIC AMERICAN.

```
000          0+    1+
0            0+    2+
             1+    0
```

Beehive

This fellow lives for four generations and becomes stable in a form called a beehive.

```
000        0+   1+
 0         0+   1-
           1+   0+
```

Traffic Light

After 10 generations, this fellow becomes a blinking traffic light.

```
000        0+   1+
0          0+   2+
 0         1+   0+
           2+   1+
```

Glider

This glider floats up the pond. When he hits the ceiling, he turns into a stable block of four living cells.

```
0000       0+   1+
0  0       0+   2+
 0         0+   3+
           1+   4+
           1+   0+
           2+   0+
           3+   1+
```

Spaceship

This spaceship travels across the pond colliding with the left edge after 10 generations. He then shoots a glider down.

```
0  0       2-   2-
0  0       2-   2+
 000       1-   2-
0  0       1-   2+
0  0       0+   1-
           0+   1+
           1+   2-
           1+   2+
           2+   2-
           2+   2+
```

Spaceman

This life form was first tried by Bob Borg. See figures 1 and 2 for the history of this interesting life form.

If we turn spaceman sideways, he bumps the ceiling after 13 generations losing partial symmetry. He regains symmetry after generation 94. After generation 111, he turns into 2 beehives and four blinkers.

```
       000   000
      0  0   0  0
      00 0   0 00
      0  00 00  0
         00 00

0     0   0 0   0
00   0      0  00
0    0 0  0 0   0

         00 00
      0  00 00  0
      00 0   0 00
      0  0   0  0
       000   000
```

Figure 1

This is SPACEMAN after 18 generations. He will soon bump his head on the ceiling just before his feet touch the floor. This will throw him out of symmetry. After generation 33, he will begin to contract to the form displayed in figure 2.

```
  00           00
    0         0
0  0        0  0
 000          000
```

Figure 2

This is SPACEMAN after 75 generations. This is his minimum size. He will now grow and then later contract again. I have only followed his history through 150 generations.

CONWAY'S GAME OF LIFE

```
2000              LIFE   ORG   $2000
2000 4C 2E 20            JMP   START
2003 00           DATA   =     $00
2004 23                  =     $23      FIRST ADDRESS IN POND
                   ALLOW $40 BYTES BEFORE AND
                   AFTER POND FOR WRAP-AROUND.
                   POND IS 1 K  BYTES LONG.
2005 C0                  =     $C0      PON
2006 51                  =     $51
2007 00                  =     $00      LAS
2008 56                  =     $56
2009 00                  =     $00       UL OFFSET
200A 01                  =     $01       UP
200B 02                  =     $02       UR
200C 40                  =     $40       LEFT
200D 42                  =     $42
200E 80                  =     $80       LL
200F 81                  =     $81       DOWN
2010 82                  =     $82       LR
2011              PONDL  *     $001C     FIRST ADDRESS IN POND
2011              PONDH  *     $001D
2011              PON    *     $001E
2011              LAS    *     $0020
2011              OFFSET *     $0022     DATA WILL BE MOVED HERE
2011              LAST   *     $002A     POINTS TO LAST ADDR. IN POND
2011              ADR    *     $002C     (POINT-POND)=($40*V + H)
2011              V      *     $002E     VERTICAL ORDINATE
2011              H      *     $002F
2011              CNT    *     $0030     COUNT
2011              NN     *     $0031     NUMBER OF NEIGHBORS
2011              LFLAG  *     $0032     LIFE FLAG
2011              SAVY   *     $0033
2011              POINTL *     $0034
2011              POINTH *     $0035
2011              POINT  *     $0036
2011              GL     *     $0038
2011              GH     *     $0039

                   KIM ROUTINES

2011 4C 3B 1E     PRTBYT JMP   $1E3B
2014 84 33        GETCH  STY   SAVY
2016 20 5A 1E            JSR   $1E5A
2019 A4 33               LDY   SAVY
201B 60                  RTS
201C A9 0D        CRLF   LDAIM $0D
201E 20 23 20            JSR   OUTCH
2021 A9 0A               LDAIM $0A
2023 84 33        OUTCH  STY   SAVY
2025 20 A0 1E            JSR   $1EA0
2028 A4 33               LDY   SAVY
202A 60                  RTS
```

```
                    BEGIN HERE

202B A0 00       START  LDYIM $00
202D 84 38              STY   GL
202F 84 39              STY   GH
2031 20 53 20           JSR   MOVZ    MOVE DATA TO ZERO PAGE
2034 20 D7 21           JSR   CLEAR
2037 20 2E 21           JSR   PLANT    SEED IN POND
203A 20 A5 21           JSR   SHOALL   OF POND ON TUBE
203D 20 39 22    STAR   JSR   INCG    INCRE. GENER. COUNT
2040 A0 00              LDYIM $00
2042 84 32              STY   LFLAG    ZERO LIVING FLAG
2044 20 11 22           JSR   EDGE      POND WITH ZEROES
2047 20 AF 20           JSR   POST     BIRTHS & DEATHS
204A 20 F1 21           JSR   UPDATE   THE POND
204D A5 32              LDA   LFLAG
204F D0 EC              BNE   STAR     YES. CHECK NEXT GENERATION
2051 00                 BRK
2052 00                 BRK
2053 A2 0D       MOVZ   LDXIM $0D
2055 BD 03 20           LDAAX DATA     GET A DATA WORD
2058 95 1C              STAZX PONDL    PUT IN PAGE ZERO
205A CA                 DEX
205B 10 F8              BPL   MOVZ   +02
205D 18                 CLC
205E A5 1C              LDA   PONDL    POND - $40
2060 69 C0              ADCIM $C0      POND     K      A
2062 85 2A              STA   LAST              R     E
2064 A5 1D              LDA   PONDH            O     R
2066 69 03              ADCIM $03       LAST W      A
2068 85 2B              STA   LAST   +01   LAST +40
206A A5 1D              LDA   PONDH
206C 85 1F              STA   PON    +01
206E C6 1F              DEC   PON    +01
2070 A5 2B              LDA   LAST   +01
2072 85 21              STA   LAS    +01
2074 E6 21              INC   LAS    +01
2076 60                 RTS

                 CALC V & H  FROM ADDRESS IN  ADR

2077 A6 2D       CALCVH LDX   ADR    +01
2079 A5 2C              LDA   ADR
207B 4C 80 20           JMP   CAL
207E E6 2E              INC   V
2080 38          CAL    SEC
2081 E9 40              SBCIM $40
2083 B0 F9              BCS   CAL    -02
2085 CA                 DEX
2086 10 F6              BPL   CAL    -02
2088 85 2F              STA   H        REMAINDER IN  H
208A 60                 RTS

                 CALC  ADR = POINT - POND

208B 38          CLCADR SEC
```

```
208C A5 34              LDA    POINTL
208E E5 1C              SEC    PONDL
2090 85 2C              STA    ADR
2092 A5 35              LDA    POINTH
2094 E5 1D              SBC    PONDH
2096 85 2D              STA    ADR     +01
2098 60                 RTS

                   SET NN = NO. OF NEIGHBORS FOR CELL
                   AT POINT.

2099 20 5F 22  NBRS     JSR    MOV
209C A2 07              LDXIM  $07
209E E5 22     NBR      LDAAX  OFFSET
20A0 A8                 TAY
20A1 E1 36              LDAIY  POINT
20A3 F0 04              BEQ    NE         NOT A NEIGHBOR
20A5 30 02              BMI    NB         CONTINUE
20A7 E6 31              INC    NN
20A9 CA        NB       DEX
20AA 10 F2              BPL    NBR
20AC A0 00              LDYIM  $00
20AE 60                 RTS

                   POST BIRTHS & DEATHS

20AF 20 CC 21  POST     JSR    MOVE       BIRTH = -1
20B2 20 99 20           JSR    NBRS
20B5 A5 31              LDA    NN         ALIVE =+1
20B7 C9 02              CMPIM  $02        WILL DIE
20B9 30 13              BMI    DEATH      IF < 2
20BB C9 03              CMPIM  $03
20BD F0 1C              BEQ    BIRTH      IF = 3
20BF 10 0D              BPL    DEATH      IF > 3
20C1 20 58 22  POSTA    JSR    INCPT      INCREMENT POINT
20C4 38                 SEC
20C5 A5 35              LDA    POINTH
20C7 E5 1D              SBC    PONDH
20C9 C9 04              CMPIM  $04
20CB 30 E5              BMI    POST    +03 NOT YET DONE WITH THIS CELL
20CD 60                 RTS             NOW WE ARE DONE WITH IT
20CE E1 34     DEATH    LDAIY  POINTL
20D0 F0 EF              BEQ    POSTA
20D2 A9 02              LDAIM  $02
20D4 91 34              STAIY  POINTL
20D6 A9 20              LDAIM  $20
20D8 4C E5 20           JMP    BIRTHS
20DB B1 34     BIRTH    LDAIY  POINTL
20DD D0 E2              BNE    POSTA
20DF A9 FF              LDAIM  $FF
20E1 91 34              STAIY  POINTL
20E3 A9 30              LDAIM  '0
20E5 20 71 22  BIRTHS   JSR    DISPLY
20E8 E6 32              INC    LFLAG
20EA 4C C1 20           JMP    POSTA
```

```
20ED 18         CONVI   CLC
20EE 65 2F              ADC    H
20F0 85 2C              STA    ADR
20F2 90 02              BCC    CONVH   -01
20F4 E6 2D              INC    ADR     +01
20F6 60                 RTS

                 CONVERT H & V TO EQUIV. ADDR.

20F7 A6 2E      CONVH   LDX    V
20F9 A0 00              LDYIM  $00
20FB 84 2C              STY    ADR
20FD 84 2D              STY    ADR     +01    CLEAR ADR
20FF CA         CONV    DEX
2100 30 EE              BMI    CONVI
2102 18                 CLC
2103 A9 40              LDAIM  $40
2105 65 2C              ADC    ADR
2107 85 2C              STA    ADR
2109 90 F4              BCC    CONV
210B E6 2D              INC    ADR     +01
210D 4C FF 20           JMP    CONV

                 ASK FOR  V,H

2110 20 1C 20   ENTRVH  JSR    CRLF
2113 A2 0B              LDXIM  $0B
2115 BD 1F 21           LDAAX  ENT
2118 20 23 20           JSR    OUTCH
211B CA                 DEX
211C 10 F7              BPL    ENTRVH +05
211E 60                 RTS
211F 20         ENT     =      '
2120 3F                 =      '?
2121 20                 =      '
2122 48                 =      'H
2123 2C                 =      ',
2124 56                 =      'V
2125 20                 =      '
2126 52                 =      'R
2127 45                 =      'E
2128 54                 =      'T
2129 4E                 =      'N
212A 45                 =      'E

                 PLANT THE SEED

212B A0 00              LDYIM  $00
212D 60                 RTS
212E A9 07      PLANT   LDAIM  $07            SET FOR MIDSCREEN
2130 85 2E              STA    V
2132 A9 1F              LDAIM  $1F
2134 85 2F      BACK    STA    H
2136 20 F7 20           JSR    CONVH
2139 18                 CLC
213A A5 2C              LDA    ADR
```

```
2130 65 1C              ADC    PONDL
213E 85 34              STA    POINTL
2140 A5 2D              LDA    ADR     +01
2142 65 1D              ADC    PONDH
2144 85 35              STA    POINTH
2146 A9 01              LDAIM  $01
2148 91 34              STAIY  POINTL
214A 20 10 21   BASK    JSR    ENTRVH
214D 20 9E 21           JSR    GET
2150 F0 F8              BEQ    BASK
2152 C9 30              CMPIM  '0
2154 30 D5              BMI    PLANT   -03
2156 29 07              ANDIM  $07
2158 85 2E              STA    V
215A 20 9B 21           JSR    GET
215D F0 EE              BEQ    BASK
215F C9 2D              CMPIM  '-
2161 D0 07              BNE    PLAN
2163 38                 SEC
2164 A9 00              LDAIM  $00
2166 E5 2E              SBC    V
2168 85 2E              STA    V
216A A9 2C      PLAN    LDAIM  ',
216C 20 23 20           JSR    OUTCH
216F 18                 CLC
2170 A5 2E              LDA    V
2172 69 07              ADCIM  $07
2174 85 2E              STA    V
2176 20 9B 21           JSR    GET
2179 F0 CF              BEQ    BASK
217B C9 30              CMPIM  '0
217D 30 AC              BMI    PLANT   -03
217F 29 07              ANDIM  $07
2181 85 2F              STA    H
2183 20 9B 21           JSR    GET
2186 F0 C2              BEQ    BASK
2188 C9 2D              CMPIM  '-
218A D0 07              BNE    PLANTB
218C 38                 SEC
218D A9 00              LDAIM  $00
218F E5 2F              SBC    H
2191 85 2F              STA    H
2193 A5 2F      PLANTB  LDA    H
2195 18                 CLC
2196 69 1F              ADCIM  $1F          MEAGURE TO CENTER
2198 4C 34 21           JMP    BACK

                 GET A COORDINATE

219B 20 14 20   GET     JSR    GETCH
219E C9 38              CMPIM  '8
21A0 30 02              BMI    BAD
21A2 A9 00              LDAIM  $00
21A4 60         BAD     RTS

                 DISPLAY ALL OF POND
```

```
21A5 20 CC 21   SHOALL JSR   MOVE
21A8 A9 0F             LDAIM $0F
21AA 85 2E             STA   V
21AC A9 3F      SHOAL  LDAIM $3F
21AE 85 2F             STA   H
21B0 20 1C 20          JSR   CRLF
21B3 B1 34      SHOA   LDAIY POINTL
21B5 F0 04             BEQ   SHO
21B7 A9 30             LDAIM '0
21B9 10 02             BPL   SHO     +02
21BB A9 20      SHO    LDAIM $20
21BD 20 23 20          JSR   OUTCH
21C0 20 58 22          JSR   INCPT
21C3 C6 2F             DEC   H
21C5 10 EC             BPL   SHOA
21C7 C6 2E             DEC   V
21C9 10 E1             BPL   SHOAL
21CB 60                RTS

                MOVE POND TO POINT

21CC A5 1C      MOVE   LDA   PONDL
21CE 85 34             STA   POINTL
21D0 A5 1D             LDA   PONDH
21D2 85 35             STA   POINTH
21D4 A0 00             LDYIM $00
21D6 60                RTS

                  CLEAR POND

21D7 20 CC 21   CLEAR  JSR   MOVE
21DA A9 0F             LDAIM $0F
21DC 85 30             STA   CNT
21DE A2 3F             LDXIM $3F
21E0 98                TYA
21E1 91 34      CLEA   STAIY POINTL
21E3 20 58 22          JSR   INCPT
21E6 CA                DEX
21E7 10 F8             BPL   CLEA
21E9 C6 30             DEC   CNT
21EB 10 F1             BPL   CLEA    -03
21ED 20 CC 21          JSR   MOVE
21F0 60                RTS

                 BURY THE DEAD AND RAISE THE CHILDREN

21F1 20 CC 21   UPDATE JSR   MOVE
21F4 B1 34             LDAIY POINTL
21F6 30 08             BMI   POSTIT -02
21F8 C9 02             CMPIM $02
21FA 30 08             BMI   POSTIT +02
21FC A9 00             LDAIM $00
21FE F0 02             BEQ   POSTIT
2200 A9 01             LDAIM $01
2202 91 34      POSTIT STAIY POINTL
```

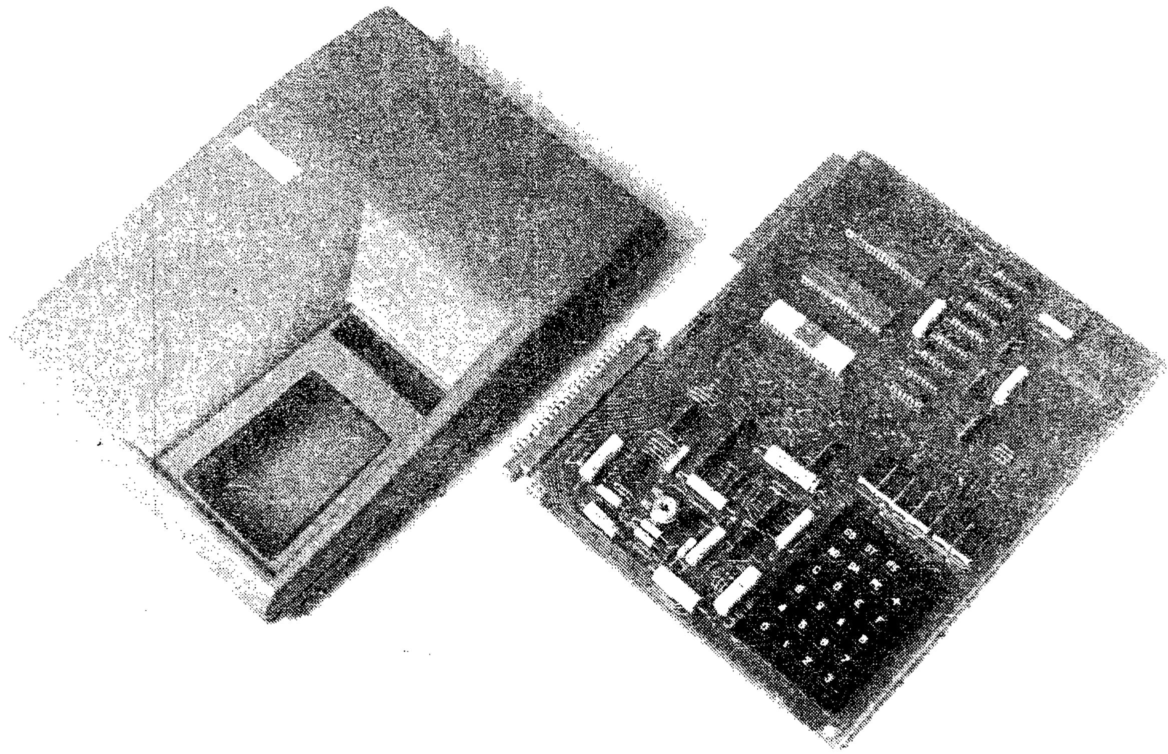
```
2204 20 58 22          JSR   INCPT
2207 A5 35             LDA   POINTH
2209 C5 21             CMP   LAS    +01
220B 30 E7             BMI   UPDATE +03
220D 20 CC 21          JSR   MOVE
2210 60                RTS

                       EDGE POND WITH ZEROES
                       TO PREVENT WRAP-AROUND

2211 20 CC 21   EDGE   JSR   MOVE
2214 A0 3F             LDYIM $3F
2216 A9 00             LDAIM $00
2218 91 1E             STAIY PON
221A 91 20             STAIY LAS
221C 88                DEY
221D 10 F9             BPL   EDGE   +07
221F A0 00             LDYIM $00
2221 A5 34      WRA    LDA   POINTL
2223 18                CLC
2224 69 40             ADCIM $40
2226 85 34             STA   POINTL
2228 A9 00             LDAIM $00
222A 65 35             ADC   POINTH
222C 85 35             STA   POINTH
222E C5 21             CMP   LAS    +01
2230 B0 DE             BCS   EDGE   -01
2232 A9 00             LDAIM $00
2234 91 34             STAIY POINTL
2236 4C 21 22          JMP   WRA

                       INCREMENT AND DISPLAY
                       THE GENERATION COUNT

2239 18         INCG   CLC
223A F8                SED
223B A9 01             LDAIM $01
223D 65 38             ADC   GL
223F 85 38             STA   GL
2241 A9 00             LDAIM $00
2243 65 39             ADC   GH
2245 85 39             STA   GH
2247 D8                CLD
2248 A9 04      NCG    LDAIM $04
224A 20 23 20          JSR   OUTCH
224D A5 39             LDA   GH
224F 20 11 20          JSR   PRTBYT
2252 A5 38             LDA   GL
2254 20 11 20          JSR   PRTBYT
2257 60                RTS
2258 E6 34      INCPT  INC   POINTL
225A D0 02             BNE   INCPT  +06
225C E6 35             INC   POINTH
225E 60                RTS
225F 38         MOV    SEC
2260 A5 34             LDA   POINTL
```

```
2262 E9 41          SBCIM $41
2264 85 36          STA   POINT
2266 A5 35          LDA   POINTH
2268 E9 00          SBCIM $00
226A 85 37          STA   POINT  +01
226C A0 00          LDYIM $00
226E 84 31          STY   NN
2270 60             RTS

                  DISPLAY  THE CHARACTER IN THE ACC.
                  AT THE -- POINT -- ADDRESS ON TUBE

2271 48        DISPLY PHA              SAVE ACC
2272 20 8E 20         JSR   CLCADR
2275 84 2E            STY   V
2277 20 77 20         JSR   CALCVH     CALC V,H
227A A9 1B            LDAIM $1B        PRINT ESCAPE
227C 20 23 20         JSR   OUTCH      TO MOVE CURSOR
227F A9 3D            LDAIM '=         ABS ADDRESS
2281 20 23 20         JSR   OUTCH
2284 A5 2E            LDA   V
2286 09 40            ORAIM $40        ADJSUT  V
2288 20 23 20         JSR   OUTCH
228B A5 2F            LDA   H          ADJUST  H
228D C9 20            CMPIM $20
228F 10 02            BPL   DISP
2291 09 40            ORAIM $40
2293 20 23 20  DISP   JSR   OUTCH
2296 68               PLA              GET ACC
2297 20 23 20         JSR   OUTCH      PRINT IT
229A 60               RTS
```

SYMBOL TABLE

		ADR	002C	BACK	2134	BAD	21A4
BASK	214A	BIRTH	20DB	BIRTHS	20E5	CALCVH	2077
CAL	2080	CLCADR	208E	CLEA	21E1	CLEAR	21D7
CNT	0030	CONV	20FF	CONVH	20F7	CONVI	20ED
CRLF	201C	DATA	2003	DEATH	20CE	DISP	2293
DISPLY	2271	EDGE	2211	ENTRVH	2110	ENT	211F
GETCH	2014	GET	219E	GH	0039	GL	0038
H	002F	INCG	2239	INCPT	2258	LAST	002A
LAS	0020	LFLAG	0032	LIFE	2000	MOVE	21CC
MOVZ	2053	MOV	225F	NB	20A9	NBRS	2099
NBR	209E	NCG	2248	NN	0031	OFFSET	0022
OUTCH	2023	PLAN	216A	PLANT	212E	PLANTE	2193
POINT	0036	POINTH	0035	POINTL	0034	PONDH	001D
PONDL	001C	PON	001E	POST	20AF	POSTA	20C1
POSTIT	2202	PRTBYT	2011	SAVY	0033	SHOA	21B3
SHOAL	21AC	SHOALL	21A5	SHO	21BE	STAR	203D
START	202B	UPDATE	21F1	V	002E	WRA	2221

QUICK CHANGE ARTISTRY



ENGINEERED SPECIFICALLY FOR THE KIM-1 MICRO COMPUTER
- Protection of Chips and Other Components
- Viewing Angle of Readout Enhanced
- Improved Keyboard Position for Easier Operation

EASILY ASSEMBLED
- Absolutely No Alteration of KIM-1 Required
- All Fasteners Provided
- Goes Together in Minutes with a Small Screwdriver

ATTRACTIVE FUNCTIONAL PACKAGE
- Professional Appearance
- Four Color Combinations
- Improves Man/Machine Interface

MADE OF HIGH IMPACT STRENGTH THERMOFORMED PLASTIC
- Kydex 100 *
- Durable
- Molded-In Color
- Non-Conductive

AVAILABLE FROM STOCK
- Allow Two to Three Weeks for Processing and Delivery
- No COD's Please
- Dealer Inquiries Invited

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

NAME ______________________

STREET ______________________

CITY ______________________

STATE ____________ ZIP ____________

Please Ship Prepaid ______ SKE 1-1(s)
@ $23.50 Each
California Residents please pay
$25.03 (Includes Sales Tax)

the enclosures group

55 stevenson, san francisco 94105

Color Desired blue ☐ beige ☐
black ☐ white ☐

* TM Rohm & Haas

Patent Applied For