MICRO

THE 6502 JOURNAL

PET Keysort

NO. 23 APRIL 1980 $2.00

PET-AIM
KIM-SYM

Look To MTU
For 6502
System Expansion

MTU

Micro Technology Unlimited
P.O. Box 4596, 841 Galaxy Way
Manchester, N.H. 03108
603-627-1464

Call Or Write For Our Full Line Catalog

PET BASIC BREAKTHROUGH

Softside Software presents

SYMBOLIC/STRUCTURED BASIC

At last, **Symbolic/Structured Basic** is available for your PET 8-32K personal computer! S-Basic is a pre-compiler that enhances the PET's built-in basic monitor with the addition of extra-control statements found only in the most sophisticated computers. **WHILE** ... **GOSUB** ... calls a subroutine as long as a condition is true. **UNTIL** ... **GOSUB** ... jumps to a subroutine unless the condition is true. The **IF** ... **Then** ... **ELSE** statement allows the programmer to command the computer to execute instructions if the normal IF condition is not met.

Forget about line numbers, S-Basic allows you to program naturally only naming (numerically or alphabetically!) statements that you will need to refer to, for example: LOOP/PRINT "HI": GO TO LOOP. S-Basic program **lines can be up to 255 characters long,** two-and-one-half times as long as on standard Basic. S-Basic does not compromise any of PET Basic's existing features. All PET Basic commands can be used.

S-Basic includes an **editor** with full text capabilities, a **translator/pre-compiler** with its own error messages, and the **S-Basic loader**. These programs are recommended for disk-based PETs. A printer is optional but suggested. Cassette copies are available and require two cassette drives. Comprehensive instructions are included. Symbolic-/Structured Basic package is available complete for an **introductory price** of **$35.95**.

- WHILE....GOTO....
- IF....THEN....ELSE
- 255 CHAR. LINES

A PET PROGRAMMING BREAKTHROUGH

- UNTIL....GOTO....
- SYMBOLIC OPTIONAL
- LINE NUMBERS

305 RIVERSIDE DRIVE, NEW YORK, NEW YORK 10025

PERFECT AIM

ATTRACTIVE FUNCTIONAL PACKAGING FOR YOUR AIM-65 MICROCOMPUTER

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

ENGINEERED SPECIFICALLY FOR THE ROCKWELL AIM-65

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

EASILY ASSEMBLED

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

MADE OF HIGH IMPACT STRENGTH THERMOFORMED PLASTIC

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

AVAILABLE FROM STOCK

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

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

NAME ______________________________

STREET ______________________________

CITY ______________________________

STATE ____________________ ZIP __________

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

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

enclosures group

771 bush street
san francisco, california 94108

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

MICRO™

April 1980
Issue Number 23

Table of Contents

Staff

Editor/Publisher
Robert M. Tripp
Associate Editor
Mary Ann Curtis
Business Manager
Maggie E. Fisher
Circulation Manager
Carol A. Stark
Art/Advertising Coordinator
Terry Spillane
Comptroller
Donna M. Tripp
Production Assistant
L. Catherine Bland
MICRO Lab
R. Keith Beal

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

Advertiser's Index

Applesoft II Shorthand

If you want to make Applesoft a little easier to use, try this program which permits entire commands to be input with a single control key. Since the command lookup is table driven, you can select the keys to conform to your own preferences. The techniques used provide a valuable understanding of how to add your own modifications.

Allen J. Lacy
1921 W. Oglethorpe
Albany, GA 31707

This routine allows a programmer to type in an entire Applesoft command with the use of one control key.

Overview

The routine Shorthand ties into the input hooks at $38 and $39 (56 and 57 decimal) and uses a table inside the RAM version of Applesoft II. In Applesoft's table, each command is represented as an ASCII string with the high bit off except for the last character of the string which has the high bit set. The routine also uses a monitor routine to read a key. If it is a control character, shorthand gets an address from its internal table. If the high byte of the address is 0, the routine passes the control character back. If the address is not 0 shorthand passes the command stored at that location back.

Step 1 turns DOS off. Step 2 turns Shorthand on. Step 3 turns DOS back on. But DOS will not be on at the same time as shorthand.

To use with ROM version.

Shorthand could be adapted to run with the ROM version of Applesoft II. The addresses in Shorthand would have to be changed. I do not have access to a ROM card and so do not know the addresses. But if the ROM version is just a relocated RAM version, the addresses in Shorthand and table just need $C800 added to them.

Shorthand does not use all of the control keys because some have special functions. These functions are shown in Table 1. If you do not mind losing these functions, these keys can be used also. The choices for which command is tied to which key is shown in the program listing. If you do not like my choices, you can change the command addresses stored in Table 2. The addresses are for the RAM version and will not work for the ROM version.

Use Of Shorthand

Shorthand is relocatable and can be placed anywhere in memory. I normally load it at $300—$3AE, which is where I assembled it. But it can be placed anywhere. Applesoft's HIMEM: can be used to protect some upper memory.

Example:
A 32K system without DOS can have Shorthand loaded at $7F51-7FFF and then HIMEM: can be set to 32593.
So to bring up Shorthand use the following steps:

1. LOAD and RUN the Applesoft TAPE
2. Enter the monitor by pressing RESET or do a CALL—151
3. Type
   300.3AER
   or type
   7F51.7FFFR
4. Start tape with Shorthand on it and press RETURN, stop the tape when it has loaded
5. Type
   OG
   Press Return
6. Type
   POKE 1144,0
   Press RETURN
7. If Shorthand is at $300—$3AE type
   POKE 56,0; POKE 57,3
   If Shorthand is at $7F51—$7FFF type
   POKE 56,81: POKE 57,127
8. Press RETURN
9. If Shorthand is at 7F51 type
   HIMIM: 32593
   Press RETURN

Another good place to store Shorthand is between Applesoft II and your program. The problem is that Applesoft's LOMEM: does not set the lowest memory used by Applesoft, but sets the point at which Applesoft will start storing variables. But the monitor can be used to set pointers. To do this the following steps are used:

1. LOAD and RUN the Applesoft II tape
2. Enter the monitor by pressing RESET or do a CALL—151
3. Type
   3000.30AER
4. Start the tape with Shorthand on it and press RETURN
   When it has loaded stop the tape.
5. Type
   67:B0 30
   Press RETURN
6. Type 30AF:0
   30AF:0
   Press RETURN
7. Type
   0G

```
              1000 ***********************************
              1010 *                                 *
              1020 *     APPLESOFT II SHORTHAND      *
              1030 *                                 *
              1040 *       BY ALLEN J LACY           *
              1050 *         SEPT 1979               *
              1060 *                                 *
              1070 ***********************************
              1080 *
              1090 ZP    .EQ $1E        R15 OF SWEET16
              1100 *
              1110 ***********************************
              1120 *                                 *
              1130 * LOCATIONS 478-47F NOT USED BY   *
              1140 *  SCREEN DISPLAY SEE MICRO # 8   *
              1150 *                                 *
              1160 ***********************************
              1170 *
              1180 SW    .EQ $0478      SWITCH
              1190 CT    .EQ $0479      CHAR COUNT
              1200 XSAV  .EQ $047A
              1210 YSAV  .EQ $047B
              1220 POIN  .EQ $047C      POINTER
              1230 ZPS   .EQ $047E
              1240 *
              1250 RKEY  .EQ $FD1B      KEY READ CODE
              1260 SW16  .EQ $F689      SWEET16
              1270       .OR $300
              1280 *
              1290 ***********************************
              1300 *                                 *
              1310 * START LOCATION OF SHORTHAND     *
              1320 *                                 *
              1330 ***********************************
              1340 *
300- 8E 7A 04 1350 SH    STX XSAV       SAVE X REG
303- 8C 7B 04 1360       STY YSAV       SAVE Y REG
306- 48       1370       PHA            SAVE ACC
307- A5 1E    1380       LDA ZP         SAVE ZERO PAGE
309- 8D 7E 04 1390       STA ZPS         LOCATIONS
30C- A5 1F    1400       LDA ZP+1
30E- 8D 7F 04 1410       STA ZPS+1
              1420 *
              1430 ***********************************
              1440 *                                 *
              1450 * SWEET16 IS USED TO STORE KP     *
              1460 * PROGRAM COUNTER IN $1E $1F      *
              1470 * AND THIS IS USED TO FIND        *
              1480 * THE LOCATION OF THE TABLE IN    *
              1490 * SHORTHAND                       *
              1500 *                                 *
              1510 ***********************************
              1520 *
311- 20 89 F6 1530       JSR SW16       ENTER SWEET16
314- 00       1540       .HS 00         LEAVE SWEET16
              1550 KP    .EQ *          KNOWN POINT
315- AD 78 04 1560       LDA SW
318- D0 3B    1570       BNE NBYT       CHECK SW
31A- 68       1580       PLA            RESTORE ACC
31B- 20 1B FD 1590       JSR RKEY       READ A KEY
31E- 48       1600       PHA            STORE KEY VAL
31F- C9 9B    1610       CMP #$9B       CONTROL KEY?
321- 90 14    1620       BCC CTR
              1630 *
              1640 ***********************************
              1650 *                                 *
              1660 * IF NOT A CONTROL JUST RETURN    *
              1670 *                                 *
              1680 ***********************************
              1690 *
323- 68       1700 RET   PLA            RESTORE KEY
324- AE 7A 04 1710 RT    LDX XSAV       RESTORE X REG
327- AC 7B 04 1720       LDY YSAV       RESTORE Y REG
32A- 48       1730       PHA
32B- AD 7E 04 1740       LDA ZPS        RESTORE ZERO
32E- 85 1E    1750       STA ZP          PAGE LOCATIONS
330- AD 7F 04 1760       LDA ZPS+1
333- 85 1F    1770       STA ZP+1
335- 68       1780       PLA
336- 60       1790       RTS
337- 29 7F    1800 CTR   AND #$7F       WHICH KEY
339- 0A       1810       ASL            TIMES 2
33A- 69 64    1820       ADC #TAB-KP    OFFSET FROM KP
33C- A8       1830       TAY
33D- C8       1840       INY
33E- B1 1E    1850       LDA (ZP),Y     LOAD ENTRY
              1860 *
              1870 ***********************************
              1880 *                                 *
              1890 * IF VALUE OF THE HIGH BYTE IN    *
              1900 *  TABLE IS 0 THEN RETURN THE     *
              1910 *  CONTROL CHAR ELSE SET UP TO    *
              1920 *  RETURN THE CHARACTERS FROM     *
              1930 *  APPLESOFT'S INTERNAL TABLE     *
              1940 *                                 *
              1950 ***********************************
              1960 *
340- F0 E1    1970       BEQ RET        IF 0 RETURN
342- 8D 7D 04 1980       STA POIN+1     STORE IN POIN
345- 88       1990       DEY
346- B1 1E    2000       LDA (ZP),Y
348- 8D 7C 04 2010       STA POIN
34B- A9 FF    2020       LDA #$FF       SET SW
```

```
34D- 8D 78 04          STA SW
350- A9 00             LDA #0          SET CT TO 0
352- 8D 79 04          STA CT
                  *
                  *********************************
                  *                               *
                  * NBYT IS USED TO PASS THE      *
                  *  CHARACTERS FROM THE TABLE IN *
                  *  APPLESOFT AS IF THEY WERE    *
                  *  TYPED IN                     *
                  *                               *
                  *********************************
                  *
355- 68           NBYT PLA
356- AC 79 04          LDY CT          LOAD CHAR CT
359- AD 7C 04          LDA POIN        STORE POIN IN
35C- 85 1E             STA ZP           ZERO PAGE
35E- AD 7D 04          LDA POIN+1
361- 85 1F             STA ZP+1
363- B1 1E             LDA (ZP),Y      LOAD NEXT CHAR
365- C9 80             CMP #$80        LAST CHAR?
367- B0 07             BCS END
369- 09 80             ORA #$80
36B- EE 79 04          INC CT          INCREMENT CT
36E- D0 B4             BNE RT          RETURN CHAR
370- 48           END  PHA             SAVE CHAR
371- A9 00             LDA #0          RESET SW
373- 8D 78 04          STA SW
376- 68                PLA             RESTORE CHAR
377- D0 AB             BNE RT          RETURN CHAR
                  *
                  *********************************
                  *                               *
                  *  TABLE TO STORE ADDRESSES OF  *
                  *   COMMANDS IN APPLESOFT II    *
                  *                               *
                  *  WILL HAVE TO BE CHANGED FOR  *
                  *   ROM VERSION                 *
                  *                               *
                  *********************************
379- 00 00        TAB  .DA $000        @
37B- F9 08             .DA $8F9        A CALL
37D- 3B 0A             .DA $A3B        B PEEK
37F- 00 00             .DA $000        C
381- 00 00             .DA $000        D
383- EF 08             .DA $8EF        E TEXT
385- D3 08             .DA $8D3        F FOR
387- 00 00             .DA $000        G
389- 00 00             .DA $000        H
38B- DE 08             .DA $8DE        I INPUT
38D- 00 00             .DA $000        J
38F- D0 09             .DA $9D0        K CONT
391- D4 09             .DA $9D4        L LIST
393- 00 00             .DA $000        M
395- D6 08             .DA $8D6        N NEXT
397- EF 09             .DA $9EF        O THEN
399- FD 08             .DA $8FD        P PLOT
39B- 01 09             .DA $901        Q HLIN
39D- 5B 09             .DA $95B        R COLOR=
39F- A4 09             .DA $9A4        S GOSUB
3A1- 93 09             .DA $993        T GOTO
3A3- 00 00             .DA $000        U
3A5- 64 09             .DA $964        V VTAB
3A7- 05 09             .DA $905        W VLIN
3A9- 00 00             .DA $000        X
3AB- 25 09             .DA $925        Y HTAB
3AD- C7 09             .DA $9C7        Z POKE
                  LS   .EN
```

Symbol Table

ZP	001E	SW	0478	CT	0479
XSAVE	047A	YSAV	047B	POIN	047C
ZPS	047E	RKEY	FD1B	SW16	F689
SH	0300	KP	0315	RET	0323
RT	0324	CTR	0337	NBYT	0355
END	0370	TAB	0379	LS	03AF

Table 1

Control U	->
Control H	<-
Control M	RETURN
Control J	Line feed
Control G	BELL
Control X	Kill input line
Control C	Stops a running program
Control D	Is used by DOS

Press RETURN

8. Type
   NEW
   Press RETURN
9. Type
   POKE 1144,0
   Press RETURN

10. Type
    POKE 56,0:POKE 57,48
    Press Return

Shorthand will now be tied in.

Step 5 sets the pointer which tells Applesoft II where to start storing a program to $30B0. Step 6 sets the byte just below the start point to 0, I do not know why Applesoft wants this, but it will bomb if it is not done. Step 8 causes Applesoft to reset the rest of its pointers to reflect the new start point.

Now every time you want to type one of the commands stored in the table just press the control key and another key at the same time.

Example:
To enter INPUT *press the control key at the same time as the* I.

I have made labels for my keyboard showing which command is under which key. To return full control to the key board, use the command IN 0. To turn Shorthand back on just POKE the correct values back into 56 and 57. Shorthand does not have to be turned off when you are finished programing and want to run a program, unless the program wants for input one of the control keys which Shorthand uses. I normally set the hooks when I bring up Applesoft and leave them set.

The routine should work with DOS. I do not have DOS so these techniques are not tested. Since DOS communicates with the rest of the system via the input and output hooks at $36—39, you can not set the hooks to tie in shorthand without turning off DOS. But DOS has its own internal hooks. Unfortunately the hooks are at different places for different memory sizes. In a 48K system the input hook is at $A998, $A999 (22120, 22119 decimal). For smaller systems subtract 48K—X from the numbers, where x is the memory size. The above information came from Exploring the APPLE II DOS by Andy Hertzfeld in MICRO 9. So POKE the address of Shorthand in the DOS hooks

Another way that should work is to turn DOS off by the use of the following steps.

1. After bringing up Applesoft and loading Shorthand type
   PR O:IN O
   Press RETURN

2. Use POKEs to set 56 and 57 if Shorthand is at $300
   POKE 56,0:POKE57,3

3. When you are finished type
   CALL 976
   Press RETURN

Step 1 turns DOS off. Step 2 turns shorthand back on. DOS will not be on at the same time as Shorthand.

Table 2

8D0	END	8D3	FOR	8D6	NEXT	8DA	DATA
8DE	INPUT	8E3	DEL	8E6	DIM	8E9	READ
8ED	GR	8EF	TEXT	901	HLIN	905	VLIN
909	HGR2	90D	HGR	910	HCOLOR=	917	HPLOT
91C	DRAW	920	XDRAW	925	HTAB	929	HOME
92D	ROT=	931	SCALE=	937	SHLOAD	93D	TRACE
942	NOTRACE	949	NORMAL	94F	INVERSE	956	FLASH
95B	COLOR=	961	POP	964	VTAB	968	HIMEM:
96E	LOMEM:	974	ONERR	979	RESUME	97F	RECALL
985	STORE	98A	SPEED=	990	LET	993	GOTO
997	RUN	99A	IF	99C	RESTORE	9A3	&
9A4	GOSUB	9A9	RETURN	9AF	REM	9B2	STOP
9B6	IN	9B8	WAIT	9BC	LOAD	9D0	CONT
9D4	LIST	9D8	CLEAR	9DD	GET	9E0	NEW
9E3	TAB(	9E7	TO	9E9	FN	9EB	SPC(
9EF	THEN	9F3	AT	9F5	NOT	9F8	STEP
9FC	+	9FD	-	9FE	*	9FF	/
A00	↑	A01	AND	A04	OR	A06	>
A07	=	A08	>	A09	SGN	A0C	INT
A0F	ABS	A12	USR	A15	FRE	A18	SCRN(
A1D	PDL	A20	POS	A23	SQR	A26	RND
A29	LOG	A2C	EXP	A2F	COS	A32	SIN
A35	TAN	A38	ATN	A3B	PEEK	A3F	LEN
A42	STR$	A46	VAL	A49	ASC	A4C	CHR$
A50	LEFT$	A55	RIGHT$	A5B	MID$		

The Value of 16 Bits

Several years ago, the guest speaker at the local computer club, a gentleman from Texas Instruments, talked about the importance of the size of a microprocessor. Using all kinds of charts, tables, and various rather logical sounding arguments, he determined that 8 bit micros did not make any sense and would never find much popularity or application! A 4 bit micro is all that is required in most process control situations, and anyone wanting to do real computer type stuff -number crunching, assembling, text processing - would much prefer a 16 bit micro. Conclusion: the 8 bit micro was doomed. Well, hundreds of thousands of 8 bit microcomputers later, it is obvious that there is a market for the 8 bit micro. Isn't 20/20 hindsight wonderful!

Actually, I did not buy this thesis at the time it was presented. I had worked on a number of projects with either minis or a precursor of the micros, and had discovered a number of instances in which an 8 bit processor was superior to its bigger brother. Does this seem strange. Let's examine the details.

One obvious type of application, in which we all participate to some degree, is any form of word processing. How many bits does it normally take to represent the normal alphanumerics and special symbols that we use in everyday writing, BASIC, assembler programming, and so forth? ASCII defines 128 characters, including a bunch of specialized control codes, and that seems to be enough for most applications. Even if you want to add special sets, such as greek for APL, the total number of unique codes required is normally going to be less than 256 decimal. Can you imagine a keyboard to generate more than 256 characters? Since 8 bits can be used to represent 256 unique values, it is adequate for this work. In fact, it is ideal. A 16 bit machine either must ignore half of each byte, which is of course wasteful and essentially reduces it to an 8 bit machine, or must pack two 8 bit bytes into each 16 bit word. And then it must, of course, unpack the two bytes for processing, repack them again, and so forth. Therefore, the 8 bit micro is perfect for most word processing based applications. Since this single application category must account for a large percentage of the systems being purchased today, the strength of the 8 bit micro should not be surprizing.

Another application I worked on used a high speed photo scanner to digitize material for use in newspaper production - halftones and text. The scanner produced 8 bit chunks of data. The minicomputer was 16 bit based, and a lot of overhead was spent in packing and unpacking data, making records come out to an integral number of words, and other such nonsense. While the fact that 8 bits were appropriate to this particular application may have been pure serendipity, I am sure that there are numerous process control types of application which have a similar data range and which could best be served by the 8 bit micro.

Okay, how about number processing. Surely the 16 bit micro is better at performing math functions than the 8 bit micro. True, there is some advantage to a 16 bit micro if your application requires a lot of number crunching. 16 bit math operations can handle twice as much data as 8 bit ones. But, the savings may be minimal. In many numeric calculations, the amount of code and time spent actually performing math functions may be insignificant relative to the amounts required to do all of the other programming steps required -the set up, testing one bit, branching, subroutine jumps, and so forth. So, while there will probably be a time improvement with a 16 bit micro in heavy math programs, the savings may not be as great as initially imagined.

Where does the 16 bit computer excel then? I am not sure that, in general, it does. Given the generally higher cost of the micro, the higher cost and complexity of a 16 bit data bus, and so forth, the 16 bit must justify itself for a particular application. It is not a generally "better" solution. There are some features of a typical 16 bit micro that would be nice to have in the 8 bit as well. This is particularly true in improved addressing capabilities. Since the address space of most 8 bit micros is actually 16 bits, it would make sense in many instances to be able to handle the full range of address space with 16 bit registers. In the 6502, a number of 16 bit addressing modes are already supported. The two main places where the 8 bit limit is restrictive are in the relative branches and in the indexed instructions. The "proposed" 6516 discussed by Randall Hyde in this issue shows how the benefits of a 16 bit micro can be combined with the strengths of the 8 bit micro to form a superior computer. It is interesting to note, however, that many of the improvements are **not** based on 16 bits, but are independent enhancements. My latest intelligence suggests that the initial statement in the referenced article -"Synertek is almost ready to ship the SY6516" - is a bit optimistic. But, if we all call and ask our Synertek Reps about this superior product, maybe we can get some action!

Robert M. Tripp

The APPLE Stripper

One of the classic dilemmas in BASIC has to do with REMarks. If you use them, they take up space and time. If you do not use them, the code is hard to understand. This program resolves the problem. It permits you to generously REMark your program for documentation purposes and then remove the REMarks for the run-time version.

Bill Crouch
P.O. Box 926
Long Beach, CA 90801

As a writer of custom business software for the APPLE computer,I kept running into the same conflict; good programming style insisted that I document my programs with frequent REMark statements. My customers would have a hard time understanding or changing my programs if I did not.

On the other hand, large business programs use a great deal of memory and every byte is precious. The Applesoft manual tells us that the statement: 130 THIS IS A COMMENT uses up 24 bytes of memory. In a large program, a lot of memory will be taken by REMs, leaving less for arrays and program operation. It also means more frequent waits while the machine "housecleans" its string space.

The answer is obvious; write the program with REMarks and then remove them in the final working version. If changes are needed, make them on the version with REMarks and then remove the REMs again after the bugs have been corrected.

Removing REMs by hand took too long so I wrote a simple program to do it for me. It is disk based and will work on any APPLE with a disk drive.

Program Requirements

To use this program you need only observe a couple of simple rules. First, NEVER GOTO or GOSUB to a REmark. Always GOTO or GOSUB to the first line of code after the REMark.

Secondly, for maximum benefits, put your REMarks on a separate line rather than at the end of a line of code. This program only eliminates those lines where a REM is the first thing in the line.

```
10  REM

        REM KILLER

20  REM BY BILL CROUCH
30  REM PO BOX 926
40  REM LONG BEACH CA 90801

50  PRINT  CHR$ (4);"MON I,O,C"
60  DIM ARRAY(1000)
70  ONERR  GOTO 240
80 X = 0
90  REM

        READ TEXT FILE

100  HOME : REM CLEAR SCREEN
110  PRINT  CHR$ (4);"OPEN PROG.FILE"
120  PRINT  CHR$ (4);"READ PROG.FILE"
130  INPUT L$: REM GET A LINE FROM DISK
140  IF  LEFT$ (L$,5) = "63000" GOTO 250: REM CHECK FOR END OF TEXT
150  IF L$ = "" GOTO 130: REM ELIMINATE NULL STRINGS
160 LN =  VAL (L$):LN =  INT (LN): REM SAVE LINE NUMBER
170  IF  LEFT$ (L$,1) = "" THEN L$ =  RIGHT$ (L$,( LEN (L$) - 1)): GOTO 1
     70
180  IF  LEN (L$) < 2 GOTO 130: REM IF LINE USED UP GET ANOTHER
190  IF  ASC (L$) < 65 THEN L$ =  RIGHT$ (L$,( LEN (L$) - 1)): GOTO 170
200  IF  LEFT$ (L$,3) = "REM" THEN X = X + 1:ARRAY(X) = LN: REM KEEP TRAC
     K OF REMS
210  IF X > 995 GOTO 250: REM STAY WITHIN ARRAY
220  GOTO 130: REM DO IT ALL AGAIN
230  REM

         WRITE STRIP FILE

240  IF  PEEK (222) <  > 5 GOTO 130: REM CHECK FOR OUT OF DATA ERROR
250  PRINT  CHR$ (4);"CLOSE"
260  POKE 216,0: REM CLEAR ONERR GOTO FLAG
270  IF X = 0 GOTO 340: REM NO REMS IN PROGRAM
280  PRINT  CHR$ (4);"OPEN STRIP.FILE"
290  PRINT  CHR$ (4);"WRITE STRIP.FILE"
300  FOR Y = 1 TO X
310  PRINT ARRAY(Y): REM SAVE LINE # OF REM
320  NEXT Y
330  PRINT  CHR$ (4);"CLOSE"
340  END

]

]PR#0
```

How to Use the Programs

There are two separate programs. The first, XFILE.MAKER, must be appended to the end of your program. You could type it in yourself or, better still, use the merge routine on the DOS 3.2 Master. The only requirement is that line 63000 be after the last line of your program. It tells the next program that it is done.

You start the process with the command "RUN 63000"

You should have both programs on their own diskette with plenty of space for their text files. If REM KILLER is not on the same diskette with XFILE.MAKER, remove line #63130.

XFILE.MAKER will convert your program into a text fie and then run REM KILLER. REM KILLER then reads the text file, makes a list of REMs and then writes them off as STRIP.FILE.

By the way, certain characters in your program will cause the computer to say EXTRA IGNORED during the running of REM KILLER. You can ignore it too.

When it is done, load your original program and EXEC STRIP.FILE. Every line which is a REMark will be removed. Then save the stripped program.

Of course also save a copy of your original program. The first program I used this on was part of a trucking company package. It saved me over 2400 bytes.

How it Works

XFILE.MAKER clears the screen with line 63050 and squashes the listing to suppress extra carriage returns with line 63060.

The rest of the program writes your program to the disk as a text file. Line 63130 calls REM killer.
(Note: CHR$(4) is the same as CTRL D and is required before every APPLE disk command.)

REM KILLER: Line 60 sets up an array in which REMs are saved. It now allows for 1000 REMs which probanly is too many. If you have memory limitations, you may reduce this number and the corresponding one on line 210.

Line 140 checks for the end of your file and is the reason line 63000 is required in XFILE.MAKER.

Lines 150-190 get rid of null lines and all non-alpha characters. Line 200 then sees if the first alpha string is REM. If so, it saves the number in the array.

Lines 240-340 save the approximate line numbers as a text file called STRIP.FILE.

When you EXEC STRIP.FILE, the line numbers are prnted just as if you had typed them yourself. And the REMark lines are eliminated.

XFILE.MAKER

```
63010  REM
             BY BILL CROUCH

63020  REM PO BOX 926
63030  REM LONG BEACH CA 90801


63040  REM APPEND TO END OF PROGRAM
63050  CALL  - 936
63060  POKE 33,33: REM FORMAT LISTING
63070  PRINT  CHR$ (4);"MON I,O,C": REM LET US SEE IT WORK
63080  PRINT  CHR$ (4);"OPEN PROG.FILE"
63090  PRINT  CHR$ (4);"WRITE PROG.FILE"
63100  LIST 0,63000
63110  PRINT  CHR$ (4);"CLOSE"
63120  TEXT
63130  PRINT  CHR$ (4);"RUN REM KILLER"
63140  END
63150  REM

            CHANGE CHR$(4) TO CRTL D FOR INTEGER PROGRAMS
```

INTRODUCING . . . NIBBLE
THE REFERENCE
FOR APPLE COMPUTING

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

NIBBLE IS:
A REFERENCE GUIDE to new Programming Methods.

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

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

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

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

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

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

Try a NIBBLE

nibble
BOX 325 Lincoln, Mass. 01773

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

Name ______________________________

Address ____________________________

City ______________________________

State ____________________ Zip __________

"As manager of Personal Computers, here at Computerland of San Francisco, evaluating new software products is part of my job. With all the word processors on the market today, I choose EasyWriter for my business and personal use."

—Karen Dexter Weiss

EasyWriter T.M.

80 COLUMNS OF WORD PROCESSING POWER
FOR YOUR APPLE II COMPUTER

Finally . . . INFORMATION UNLIMITED SOFTWARE is able to bring you a complete word processing system for the Apple II. The new EasyWriter system gives you ***80 columns of upper and lower case characters*** for your Apple's video display, using the new SUP'R'TERM 1 board!

A long time ago, we decided to bring you the best simple-to-use and understand tools for your computer system. Today we've taken another stride in that same direction. It took some doing, in both hardware and software, but we think you'll agree that for the buck, no one can touch us.

Check it out:

- 80 Columns on the Screen!
- Upper & Lower Case!
- Global Search & Replace!
- Underlining!
- Bidirectional Printing!
- Incremental Spacing!
- File Appending!
- 50 Pages of Text Per Disk!

You can purchase the new EasyWriter 80 column word processing system as a complete hardware and software package directly from our new office in California, or from your local computer dealer.

Information Unlimited Software™ Inc.

Information Unlimited Software, Inc.
793 Vincente St.
Berkeley, CA 94707
(415) 525-4046

- EasyWriter is a TM of Cap'n Software, Inc.
- Apple II is a TM of Apple Computers, Inc.

See The System at the 5th West Coast Computer Faire.

Graphics and the Challenger C1P, Part 4

This continuing series on Graphics and the Challenger shows how to apply the material to create pictures and demonstrates how this may be used in Computer Aided Instruction.

William L. Taylor
246 Flora Road
Leavittsburg, OH 44430

Computers are well suited for use in an educational environment, whether this is in a class room at a local high school, college, or in an industrial training seminar. The computer can aid the instructor or can be used as an individual instructor. With the introduction of the micro processor and the number of low cost personal computers that are owned and used by individuals as a hobby, the computer must be considered as a training tool for use in the home environment.

Children seem fascinated by computers and are equally fascinated by any device that has a keyboard. If the computer has any form of graphics display, either animated or still, they seem even more delighted to experiment with the device. This leads to the point that if children are drawn to the computer, then the computer, if programmed to be a teaching aid, can be a valuable tool in their education.

With this evidence I decided to try to develope a program that combines the elements that have the most attraction for children. Also, through this method, the program will at the same time be an educational tool.

The program, which I will call "Picture" was developed to be a teaching aid in the developement and spelling of English words. The program uses Graphics to draw a picture of several objects. Then the child is asked to spell the different parts of the picture that have been displayed. The child tries to spell the names of the objects displayed, and the computer displays the answer "Right" or "Wrong" on the screen in large letters.

In Part 3 of this series ("Graphics and the Challenger C1P"), we described the features of the C1P. We developed some programs using Basic and Machine Language, in combination, to further explore the Graphics capabilities of the C1P. Many techniques were discussed and many Basic functions and statements were used in our example programs. This time let's continue with our graphics developmenmt and try a new programming approach.

This article has a two-fold purpose. First to continue our discussion of how to use the Graphics of the OSI Challenger C1P, and to secondly present a working program using the Graphics techniques in a Computer Assisted Instruction program (CAI). The program in this part will be used as a CAI tool and will be treated as an example program. *This program, by no means, is complete.* That is, it can be expanded by the user. The program simply is a pure example of how to develop graphic plots: get these characters out to the monitor screen. Combining these Graphics with a program is a useful tool in the hands of the enterprising programmer. From the techniques that are presented in this example, the user will more fully understand how to develop such programs of his own.

Program Description

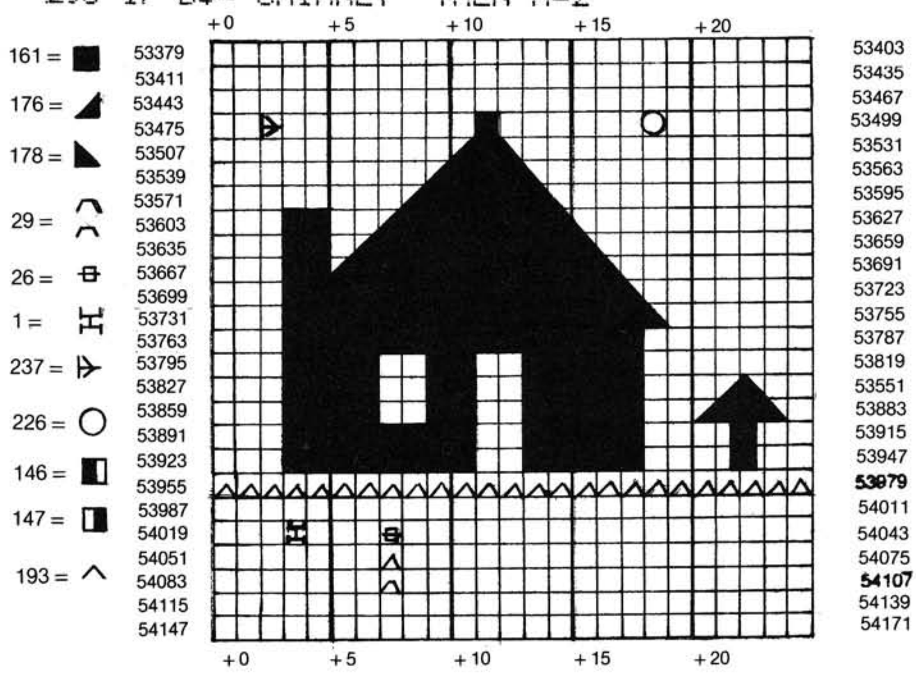
Let's start with a description of the "picture" program. First, what the program does is to generate a picture on the monitor screen. This picture is shown in the video memory map plotting chart in Figure 1. Notice that we have developed routines in the program that will POKE characters from the Graphics Set in the Character Generator ROM out to the monitor screen at the locations shown on the chart. In part 3 of this series, I gave a similar video memory chart. This time we will use the chart as in Figure 1. Notice that in the chart, we have drawn the picture that we wish to POKE out to the screen. All the memory locations can now easily be found, and routines written to accomplish the end task. Such a routine is located in the program between lines 10000 and 10420. This routine is used to draw the House, the Airplane, the Sun, the Man and the Car in the picture. All the parts of the picture were built from the Graphics elements in the Character Generator ROM. A list of these character elements appears in the upper left corner of Figure 1.

Please examine the program listing, starting at line 10000. Take the value in the statement line, For A = 53606 To 53926 Step 32. From these statement values find the corresponding value on the video memory map chart in Figure 1. It will be found that when the statement line at 10000 is compared to the memory map, you will be able to see just what the For-Next loop does. The Characters will be poke'd to these locations. Examine the program completely from line 10000 to 10420 to see how each unit works compared to the map. This example should give you a clear understanding of how to use the memory chart so that you can develop routines for your own program.

The program "Picture" contains two other Graphics Routines. These routines are used with the program to inform the user (or student) if he has identified and correctly spelled and element on the picture, or if he has identified and incorrectly spelled the object. These routines display the words: Right and Wrong, respectively. These words are in large graphic format at the top of the C1P's monitor screen. A video memory location map of these elements are in Figures 2 and 3. Please review these two figures for the memory locations. The subroutines for these graphic displays are located beginning at line 20000 for the word "Right" and at line 5000 for the word "Wrong".

These two subroutines were developed in the same manner as the one for the picture. That is, the video memory locations were plotted on the video memory plotting chart. Next, the graphics elements that were needed to generate the characters were selected from the list of graphics elements and finally, routines were written to do the task of POKEing the elements out to the screen. Analyze

```
OK
LIST

 1 GOSUB 8000
 35 PRINT"**** PICTURE ****":PRINT
 50 PRINT
 60 PRINT" HELLO IM A COMPUTER   MY NANE IS
  CHALLENGER"
 70 PRINT:PRINT
 80 PRINT" WHAT IS YOUR NAME?"
 90 INPUT A$
 100 PRINT:PRINT"HELLO "A$;"  GLAD TO MEET YOU"
 110 PRINT:PRINT" THIS IS A SPELLING GAME";A$
 120 PRINT:PRINT" I WILL SHOW YOU A"
 130 PRINT" PICTURE.  YOU ARE TO"
 140 PRINT"TELL ME THE PARTS"
 150 PRINT" THAT YOU KNOW"
 155 FOR Q=1 TO 10000:NEXT Q
 160 POKE 11,232:POKE 12,15
 170 X=USR(X)
 180 GOSUB 10000
 190 PRINT:PRINT" DID YOU SEE THE PICTURE?"
 200 PRINT" TELL ME THE PARTS":PRINT
 210 PRINT:PRINT" SPELL THE PARTS THAT MAKE
   THE PICTURE"
 220 INPUT B$
 230 IF B$="ROOF" THEN A=2
 240 IF B$="CHIMNEY" THEN A=2
 250 IF B$="WINDOW" THEN A=2
 260 IF B$="DOOR" THEN A=2
 270 IF B$="YARD" THEN A=2
 280 IF B$="ROOF" THEN A=2
 290 IF B$="CHIMNEY" THEN A=2
```



C1P Video Memory Map in Decimal

```
300 IF B$="SUN" THEN A=2
310 IF B$="PLANE" THEN A=2
330 IF A<>2  THEN GOSUB 5000
335 IF A=2 THEN GOSUB 20000
500 GOTO 200
5000 FOR A=53541 TO 53637 STEP 32
5010 POKE A,161:NEXT A
5020 FOR A=53544 TO 53640 STEP 32
5030 POKE A,161:NEXT A
5040 POKE 53638,175:POKE53639,
     177:POKE 53606,176:POKE 53607,178
5050 FOR A=53546TO53642STEP32
5060 POKEA,161:NEXTA
5070 POKE 53547,151:POKE 53548,161:
     POKE 53579,150
5080 POKE 53580,175:POKE 53611,177:
     POKE 53612,178
5090 POKE 53644,161
5100 FOR A=53550 TO 53646 STEP 32
5110 POKE A,161:NEXT A
5120 FOR A=53552 TO 53648 STEP 32
5130 POKE A,161:NEXT A
5140 POKE 53551,161:POKE 53647,161
5150 FORA=53554 TO 53650 STEP32
5160 POKE A,161:NEXT A
5170 FOR A=53556 TO 53652 STEP 32
5180 POKE A,161:NEXT A
5190 POKE 53587,178:POKE 53619,177
5200 FOR A=53590 TO 53654 STEP 32
5210 POKE A,161:NEXT A
5220 FOR A=53592 TO 53720 STEP 32
5230 POKEA,161:NEXT A
5240 FOR A=53591 TO 53719 STEP 64
5250 POKE A,161:NEXT A
5260 FOR T=1 TO 500:NEXT T
5270 X=USR(X)
5280 RETURN
8000 FOR Q=4072 TO 4095
8010 READ F:POKE Q,F
8020 NEXT Q
8030 DATA 169,32,160,8,162,0,157,0
8040 DATA 208,232,208,250,238,240
8050 DATA 15,136,208,244,169,208
8060 DATA 141,240,15,96
8070 RETURN
10000 FOR A=53606 TO 53926 STEP 32
10010 POKE A,161:NEXT A
10020 FOR A=53607 TO 53927 STEP 32
10030 POKE A,161:NEXT A
10040 FOR A=53517 TO 53672 STEP 31
10050 POKE A,176:NEXT A —
10060 FOR A=53519 TO 53716 STEP 33
10070 POKE A,178:NEXT A
10080 FOR A=53673 TO 53683
10090 POKE A,161:NEXT A
10100 FOR A=53642 TO 53650
10110 POKE A,161:NEXT A
10120 FOR A=53611 TO 53617
10130 POKE A,161:NEXT A
10140 FOR A=53580 TO 53584
10150 POKE A,161:NEXT A
10160 FOR A=53549 TO 53551
10170 POKE A,161:NEXT A
10180 POKE 53518,161
10190 POKE 53486,171
10191 FOR A=53704 TO 53716
10192 POKE A,161:NEXT A
10193 FOR A=53736 TO 53748
10194 POKE A,161:NEXT A
10195 FOR A=53768 TO 53780
10196 POKE A,161:NEXT A
10200 FOR A=53804 TO 53805
10205 POKEA,161:NEXT A
10207 FOR A=53836 TO 53837
10240 POKE A,161:NEXT A
10250 FOR A=53896 TO 53901
10260 POKE A,161:NEXT A
10270 FOR A=53928 TO 53933
10280 POKE A,161:NEXT A
10290 FOR A=53937 TO 53940
10300 POKE A,161:NEXT A
10310 FOR A=53905 TO 53908
10320 POKE A,161:NEXT A
10330 FOR A=53873 TO 53876
10340 POKE A,161:NEXT A
10350 FOR A=53841 TO 53844
10360 POKE A,161:NEXT A
10370 FOR A=53809 TO 53812
10380 POKE A,161:NEXT A
10390 FOR A=53800 TO 53801
10400 POKE A,161:NEXT A
10402 POKE53477,237
10405 POKE 53493,226
10410 FOR A=53955 TO 53979
10420 POKE A,193:NEXT A
10430 FOR D=1 TO 5000:NEXT D
10440 X=USR(X)
10450 RETURN
20000 FOR A=53509 TO 53637
      STEP 32
20010 POKE A,161:NEXT A
20020 FOR A=53511 TO 53575
      STEP 32
20030 POKE A,161:NEXT A
20040 POKE 53607,178:POKE53639,
177:POKE 53574,161:POKE 53510,161
```

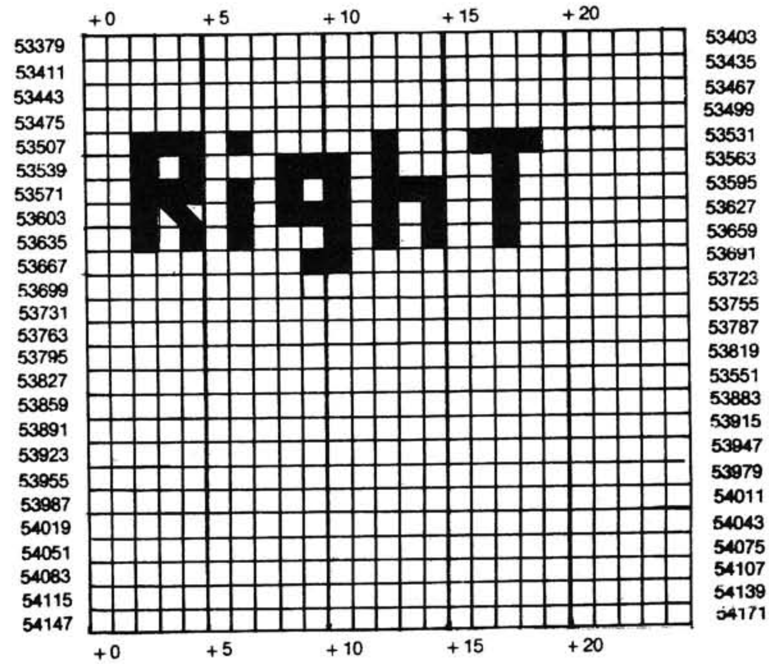
the video memory plotting charts of Figure 2 and Figure 3 along with the subroutines at lines 5000 and 20000 to see how the routines were plotted, written and used in the program.

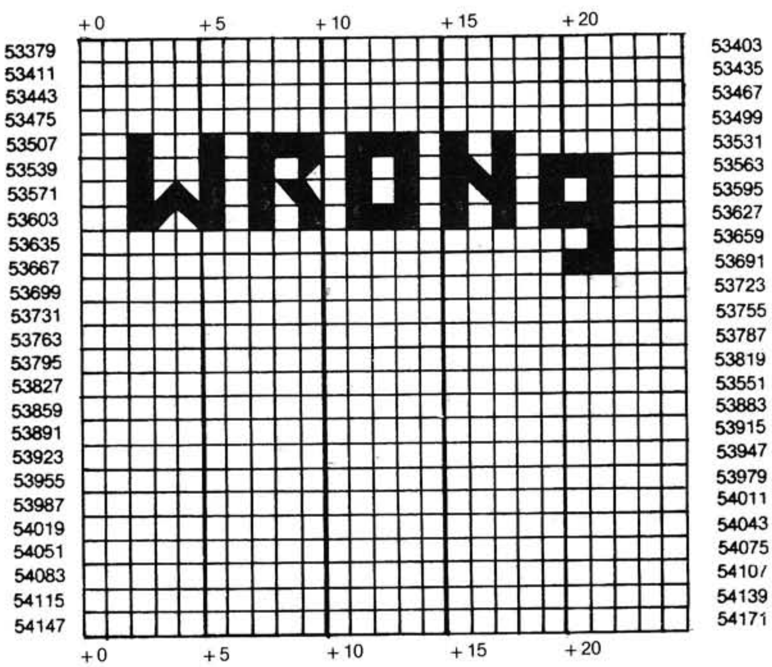
The subroutine located between lines 8000 and 8070 is for loading the machine code routine into user memory. This routine is for the Fast Screen Erase routine used by the program to clear the screen whenever called. This subroutine will be called at line 1 in the Main Line Basic program. The routine for the fast screen erase has been included in the previous parts of this series. The reader should review these parts for a complete description fo this routine.

Now that I have described the Graphics generating routines and how they were developed, let's continue with the Mainline BASIC program that uses the subroutines. The program from line 35 through 500 forms the BASIC CAI user program. This program is a demonstration of how to develope programs in which the user can be taught such things as spelling which is the purpose of this program, combined with the graphics presentation.

The For-Next loop at line 155 is used to give the user time to read the screen text just displayed. Statement line 160 sets the USR Vector to point to the Fast Screen Erase Machine Language routine located at OFE8 hex or 8168 decimal. Line 170 causes the program to jump to the machine language routine at OFE8 ot 8168 decimal where the screen will be erased.

Input from the user is accepted at statement line 220. This data is stored in a string variable, labeled B$. The input string (B$) is then compared in the string looking up table for a string match. If a match between the input and a table content is found, the information is then passed to the variable A as a decimal value. This value is then compared at Line 330 and Line 335 to check for a correct answer from the user. If a correct match was found in the string table, the A variable will force a GOSUB to Line 20000 where the answer word "Right" will be displayed for the user's answer. If a match did not occur in the string table, at line 330 a GOSUB to line 5000 will cause the answer word "WRONG" to be displayed informing the user that the answer was not correct or was not an element in the picture. At line 500 a return to the beginning of the BASIC Main-Line program will cause a new pass through the program.





This program, as stated before, is not really complete, but an example to show how such a program can be constructed. This program can be expanded or modified by the reader. If you should desire to expand the picture display to include more ojects, then these object names should be included in the string table.

The complete program including the Graphics routines and the fast screen erase routine is located at the top of the first 4K of user memory. If you have more memory and wish to expand the program, you will have to relocate this routine. Listing 2, shows the modifications to the program which will allow the routine to be relocated to begin at 1FE8 Hex or 8168 decimal. The user must set memory size to 4050 decimal for a system with 4K of memory, and 8167 for a 8 K system.

```
160 POKE 11,232 : POKE 12, 31

8000 FOR Q = 8168 TO 8191

8050 DATA 31,136,208,244,169,208

8060 DATA 141,240,31,96
```

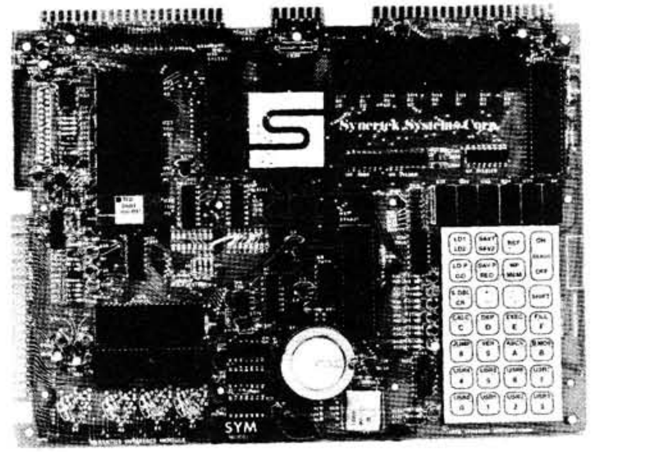
```
20045 POKE 53575,175:POKE 53606,162:
      POKE 53574,154
20050 FOR A=53513 TO 53641 STEP 32
20060 POKE A,161:NEXT A
20065 POKE 53545,32
20070 FOR A=53579 TO 53643 STEP 32
20080 POKE A,161:NEXT A
20090 FOR A=53581 TO 53709 STEP 32
20100 POKE A,161:NEXT A
20110 POKE 53580,161:POKE 53644,161:
      POKE 53708,161
20140 FOR A=53519 TO 53647 STEP 32
20150 POKE A,161:NEXT A
20160 FOR A=53585 TO 53649 STEP 32
20170 POKE A,161:NEXT A
20180 POKE 53584,161
20190 FOR A=53524 TO 53652 STEP 32
20200 POKEA,161:NEXT A
20210 POKE 53523,161:POKE 53525,161
20230 FOR E=1 TO 500:NEXT E
20240 X=USR(X)
20250 RETURN
OK
```

SYM-1, 6502-BASED MICROCOMPUTER

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

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

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



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

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

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

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

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

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

**We do honor Synertek discount coupons*

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

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

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

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

VAK-1 8-SLOT MOTHERBOARD

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

VAK-1 Motherboard **$139.00**

VAK-2/4 16K STATIC RAM BOARD

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

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

VAK-5 2708 EPROM PROGRAMMER

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

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

VAK-6 EPROM BOARD

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

VAK-6 EPROM Board **$119.00**

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

VAK-8 PROTYPING BOARD

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

VAK-8 Protyping Board **$39.00**

POWER SUPPLIES

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

FULL SYSTEM POWER SUPPLY

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

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

VAK-EPS Power Supply **$119.00**

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

KCP-1 Power Supply **$39.00**

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

VCP-1 Power Supply **$39.00**

*KIM is a product of MOS Technology

RNB ENTERPRISES INCORPORATED

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

master charge VISA

SYMple BASIC Data Files

The SYM-1 has a Microsoft BASIC available in ROM. Data Save and Data Load via the cassette are NOT supported by this version. The routines required to implement these two important functions are presented here.

John M. Blalock
3054 West Evans Drive
Phoenix, AZ 85023

If you've read "A SYMple Memory Expansion" in the August 1979 issue of MICRO and "Another KIM Expansion" in the September 1979 issue of *Kilobaud Microcomputing*, then you know that I like Micro-Z's BASIC for the KIM. You will also know that I have the Synertek BAS-1 BASIC for the SYM. Both versions were written by Microsoft, have 9-digit decimal accuracy, etc. but differ in some of their functions.

Comparing the Micro-Z Synertek BASICS

Synertek BASIC has a more convenient USR function and a &"hex" function that are definite improvements over the original BASIC. Their ROM version has no GET function like Micro-Z's. Another difference is that a response of a carriage return only to an INPUT statement will cause a break in program execution with Synertek's BASIC. Micro-Z has supplied a patch to defeat this break. The Synertek ROM does not include any trig functions, but they have recently released Technical Note #53-SSC that gives you full trig capability using only 313 bytes of RAM.

The main difference between the two BASICs, then, is the data save/data load feature added to his version by Bob Kurtz of Micro-Z. This is a very valuable feature that Microsoft left out. BASIC can not be used to maintain any types of files such as mailing lists, inventory records, or financial records without this feature. Perhpaps you could enter the data via DATA statements, but that would be a very trying task indeed! This feature is the major reason that I have preferred Micro-Z's BASIC over Synertek's.

Data Save/Data Load for Synertek BASIC

Listings 1, 2, and 3 are my first attempts to provide the same data save/data load functionality for the SYM with BAS-1. Listing 1 is just BASIC initialization, program loading, and a LIST of the program. All terminal input has been underlined for clarity. The little crooked arrows represent a carriage return typed in.

Listing 2 is a RUN of the program showing the means used to save the data. Three separate records are saved; the page zero pointers, the numeric data and string pointers, and the string data itself. To reload this data, BASIC must be initialized with the same memory size and the program can not have been modified.

Listing 3 is another RUN of the program after memory was cleared and the program reloaded. The data saved in listing 2 was restored, as can be seen. No, it is not as convenient as Bob Kurtz's method, but it works! Bob packs all the data together with a machine language subroutine and save it as one record. Another subroutine loads the combined record and then unpacks it, moving the data back to its original locations.

Machine Language Version of Data Save/Data Load

Listing 4 is a machine language subroutine that will save and load BASIC data files without having to turn control over to the SYM monitor. The data is still saved in three separate records, but they are recorded/loaded one right after another by the routine. An extra few seconds for each save or load (for sync, etc.) shouldn't hurt anyone, should it?

Listing 5 is a VERIFY dump of the subroutine. Load it in, VERIFY between the same addresses, and if you check sums match mine then you keyed it in correctly. Now we know why Synertek put those check sums on the VERIFY dumps! The rest of listing 5 shows BASIC initialization and the loading of the revised BASIC program.

Listing 6 is just a LIST of the revised program. Note the memory size was specified to allow room for the machine language subroutine which is called by statements 100 and 400. With either of the two methods, put the call to the load routine after any DIM statements and before the main program body. The call to the save routine should be at the very end of the program, as shown. Any changes to the program that increases the memory size needed for it will prevent data saved by a prior version from being loaded correctly.

Listing 7 is a RUN of the revised program wherein the data that is entered is saved at the end of the RUN. Listing 8 shows memory being cleared, BASIC initialization identical to that used in listing 7, and then the BASIC program being reloaded. The RUN of the program loads the data saved in listing 7.

If you plan on saving and loading data files very often, dedicating 148 bytes of memory to this subroutine should pay for itself in convenience over the method given earlier.

SYMple Memory Expansion Update

Regular readers of MICRO will recognize from the listings that my SYMple memory expansion board is still work-

Listing 4

```
                   ;
                   ;    ***  SAVER  ***
                   ;
                   ;
                   ; Routine to save and load SYM BASIC data tables.
                   ;
                   ; Initialize BASIC with MEMORY SIZE? = 8043 for an 8K SYM.
                   ; Call data load routine with 'Q = USR(8136,384)' after
                   ; any DIM statements, but before any processing.
                   ; Call data save routine with 'Q = USR(8044,384)' after
                   ; all data processing in the program has been done.
                   ; Always initialize BASIC with the same values and don't
                   ; alter the program or the data will not load properly.
                   ; The code is completely relocatable, only the MEMORY SIZE
                   ; and USR addresses must be changed for other locations.
                   ;
                   ;
                   ; Written by John M. Blalock, August 24, 1979
                   ;
                   ;
                   ; Routine and pointer addresses
                   ;
                   SAD:      equ     $A64C     ; tape starting address
                   EAD:      equ     $A64A     ; tape ending address + 1
                   ID:       equ     $A64E     ; tape record identifier
                   BOD:      epz     $7D       ; beginning of data
                   EOD:      epz     $81       ; end of data + 1
                   BOS:      epz     $83       ; beginning of strings
                   EOS:      epz     $87       ; end of strings + 1
                   NXL:      epz     $D3       ; next line pointer
                   TMP:      epz     $EE       ; temporary data
                   DUMPT:    equ     $8E87     ; SYM tape save routine
                   LOADT:    equ     $8C78     ; SYM tape load routine
                   SAVER:    equ     $8188     ; SYM register save routine
                   RESALL:   equ     $81C4     ; restores registers & returns
                   OUTCHR:   equ     $8A47     ; SYM terminal output routine
                   ;
                   ; Data save routine
                   ;
                             org     $1F6C     ; 8044 decimal
1F6C:20 88 81      SAVE:     JSR     SAVER     ; save all registers
1F6F:8D 4E A6                STA     ID        ; ID passed in A
1F72:A9 00                   LDA     #00       ; clear accum
1F74:8D 4D A6                STA     SAD+1     ; SA and EA are on
1F77:8D 4B A6                STA     EAD+1     ;  page zero
1F7A:A9 65                   LDA     #$65      ; will start at $0065
1F7C:8D 4C A6                STA     SAD       ;  and record
1F7F:A9 EA                   LDA     #$EA      ;   all pointers, etc.
1F81:8D 4A A6                STA     EAD       ;    thru $00E9
1F84:20 87 8E                JSR     DUMPT     ; save it, mode passed in Y
1F87:20 FA 1F                JSR     DOT       ; send asterisk to terminal
1F8A:A0 80                   LDY     #$80      ; mode = HS
1F8C:A5 7D                   LDA     BOD       ; set up tape
1F8E:8D 4C A6                STA     SAD       ;  addresses
1F91:A5 7E                   LDA     BOD+1     ;
1F93:8D 4D A6                STA     SAD+1     ;
1F96:A5 81                   LDA     EOD       ;
1F98:8D 4A A6                STA     EAD       ;
1F9B:A5 82                   LDA     EOD+1     ;
1F9D:8D 4B A6                STA     EAD+1     ;
1FA0:EE 4E A6                INC     ID        ; second record
1FA3:20 87 8E                JSR     DUMPT     ; save it
1FA6:20 FA 1F                JSR     DOT       ; print another asterisk
1FA9:A0 80                   LDY     #$80      ; mode = HS
1FAB:A5 83                   LDA     BOS       ; set up tape addresses
1FAD:8D 4C A6                STA     SAD       ;  for string data
1FB0:A5 84                   LDA     BOS+1     ;
1FB2:8D 4D A6                STA     SAD+1     ;
1FB5:A5 87                   LDA     EOS       ;
1FB7:8D 4A A6                STA     EAD       ;
1FBA:A5 88                   LDA     EOS+1     ;
1FBC:8D 4B A6                STA     EAD+1     ;
1FBF:EE 4E A6                INC     ID        ; third record
1FC2:20 87 8E                JSR     DUMPT     ; save it
1FC5:4C C4 81                JMP     RESALL    ; restore regs & return
                   ;
                   ; Data load routine
                   ;
1FC8:20 88 81      LOAD:     JSR     SAVER     ; save all registers
1FCB:8D 4E A6                STA     ID        ; ID passed in A, mode in Y
1FCE:A5 D3                   LDA     NXL       ; put next line pointers
1FD0:85 EE                   STA     TMP       ;  in a spare location
1FD2:A5 D4                   LDA     NXL+1     ;   that won't be altered -
1FD4:85 EF                   STA     TMP+1     ;    will need to continue.
1FD6:20 78 8C                JSR     LOADT     ; load first record
1FD9:20 FA 1F                JSR     DOT       ; send asterisk to terminal
1FDC:A0 80                   LDY     #$80      ; mode = HS
1FDE:EE 4E A6                INC     ID        ; second record
1FE1:20 78 8C                JSR     LOADT     ; load it
1FE4:20 FA 1F                JSR     DOT       ; print another asterisk
1FE7:A0 80                   LDY     #$80      ; mode = HS
1FE9:EE 4E A6                INC     ID        ; third record
1FEC:20 78 8C                JSR     LOADT     ; load it
1FEF:A5 EE                   LDA     TMP       ; restore next line
1FF1:85 D3                   STA     NXL       ;  pointers so that
1FF3:A5 EF                   LDA     TMP+1     ;   BASIC can continue
1FF5:85 D4                   STA     NXL+1     ;    from where it was.
1FF7:4C C4 81                JMP     RESALL    ; restore regs & return
                   ;
                   ; Subroutine to print an asterisk on the terminal so that
                   ; the operator doesn't doze off, or think the SYM has.
                   ;
1FFA:A9 2A         DOT:      LDA     #$2A      ; ASCII asterisk
1FFC:20 47 8A                JSR     OUTCHR    ; send to terminal
1FFF:60                      RTS               ; and return
```

Listing 5

```
.V 1F6C-1FFF
1F6C 20 88 81 8D 4E A6 A9 00,53
1F74 8D 4D A6 8D 4B A6 A9 65,5F
1F7C 8D 4C A6 A9 EA 8D 4A A6,EE
1F84 20 87 8E 20 FA 1F A0 80,7C
1F8C A5 7D 8D 4C A6 A5 7E 8D,CD
1F94 4D A6 A5 81 8D 4A A6 A5,08
1F9C 82 8D 4B A6 EE 4E A6 20,0A
1FA4 87 8E 20 FA 1F A0 80 A5,1D
1FAC 83 8D 4C A6 A5 84 8D 4D,22
1FB4 A6 A5 87 8D 4A A6 A5 88,9E
1FBC 8D 4B A6 EE 4E A6 20 87,A5
1FC4 8E 4C C4 81 20 88 81 8D,7A
1FCC 4E A6 A5 D3 85 EE A5 D4,D2
1FD4 85 EF 20 78 8C 20 FA 1F,A3
1FDC A0 80 EE 4E A6 20 78 8C,C9
1FE4 20 FA 1F A0 80 EE 4E A6,04
1FEC 20 78 8C A5 EE 85 D3 A5,B8
1FF4 EF 85 D4 4C C4 81 A9 2A,64
1FFC 20 47 8A 60,B5
 4CB5
```

Listing 6

```
LIST

 10 REM   SYM DATA SAVE/LOAD DEMO PROGRAM
 20 REM      JOHN BLALOCK        AUGUST 24, 1979
 30 DIM A(100), B$(100):N = 0
 40 PRINT CHR$(12);:FOR I = 1 TO 200:NEXT I
 50 PRINTTAB(30)"DATA SAVE/LOAD DEMO":PRINT:PRINT:PRINT
 60 INPUT"DO YOU WANT TO RESTORE PRIOR SAVED DATA? ";Q$
 80 IF LEFT$(Q$,1) <> "Y" THEN 200
 90 PRINT:PRINT"START RECORDER ON PLAYBACK.":PRINT
 100 Q = USR(8136,384)
 110 PRINT:PRINT"DATA LOADED.":PRINT
 120 FOR I = 1 TO 1000:NEXT I
 200 PRINT CHR$(12);:FOR I = 1 TO 200:NEXT I:PRINT
 210 PRINT"ENTER PAY NUMBER AND NAME, SEPARATED BY A COMMA:"
 220 PRINT "FOR EXAMPLE; '12345,JOHN SMITH'"
 230 PRINT"ENTER A NEGATIVE NUMBER (-1) AS A PAY NUMBER TO END
 240 PRINT                                                ENTRY."
 250 FOR I = N+1 TO 100
 260 INPUT A(I),B$(I)
 270 IF A(I) < 0 THEN N = I-1 :GOTO 300
 280 NEXT I:PRINT"TABLE IS FULL!"
 290 N = I
 300 PRINT"THE TABLE NOW CONTAINS THE FOLLOWING DATA:"
 310 PRINT" #          PAY NUMBER        NAME"
 320 FOR I = 1 TO N
 330 PRINT I TAB(11) A(I) TAB(27) B$(I)
 340 NEXT I
 350 PRINT:PRINT:PRINT
 360 INPUT"DO YOU WANT TO SAVE THIS DATA? ";Q$
 370 IF LEFT$(Q$,1) <> "Y" THEN PRINT:PRINT"PROGRAM ENDS.":END
 380 PRINT:PRINT:PRINT
 390 INPUT"START RECORDER ON RECORD, THEN ENTER 'G CR'.   ";Q$
 400 Q = USR(8044,384)
 410 PRINT:PRINT"DATA SAVED.":PRINT:PRINT"PROGRAM ENDS.":END
OK
```

Listing 7

```
RUN
                              DATA SAVE/LOAD DEMO
DO YOU WANT TO RESTORE PRIOR SAVED DATA? NO

ENTER PAY NUMBER AND NAME, SEPARATED BY A COMMA:
FOR EXAMPLE; '12345,JOHN SMITH'
ENTER A NEGATIVE NUMBER (-1) AS A PAY NUMBER TO END ENTRY.

? 12345-JOHN SMITH
? 23456,JANE JONES
? 34567,MARY JOHNSON
? -1.
THE TABLE NOW CONTAINS THE FOLLOWING DATA:
 #         PAY NUMBER         NAME
 1           12345          JOHN SMITH
 2           23456          JANE JONES
 3           34567          MARY JOHNSON

DO YOU WANT TO SAVE THIS DATA? YES
START RECORDER ON RECORD, THEN ENTER 'G CR'.   G
**
DATA SAVED.

PROGRAM ENDS.


.F 00-00-F0
.F 00-200-1FFF
.L2 01
.J 0
MEMORY SIZE? 8043
WIDTH?

 7530 BYTES FREE

BASIC V1.1
COPYRIGHT 1978 SYNERTEK SYSTEMS CORP.

OK
LOADT
LOADED
OK
RUN
                              DATA SAVE/LOAD DEMO
DO YOU WANT TO RESTORE PRIOR SAVED DATA? YES

START RECORDER ON PLAYBACK.

**
DATA LOADED.
ENTER PAY NUMBER AND NAME, SEPARATED BY A COMMA:
FOR EXAMPLE; '12345,JOHN SMITH'
ENTER A NEGATIVE NUMBER (-1) AS A PAY NUMBER TO END ENTRY.

? 44444,JAMES BOND
? 44444,ALLEN SMITH
? -1,
THE TABLE NOW CONTAINS THE FOLLOWING DATA:
 #         PAY NUMBER         NAME
 1           12345          JOHN SMITH
 2           23456          JANE JONES
 3           34567          MARY JOHNSON
 4           44444          JAMES BOND
 5           44444          ALLEN SMITH

DO YOU WANT TO SAVE THIS DATA? NO

PROGRAM ENDS.

OK
```

A Perpetual Calendar Printer for the AIM

If you know the proper tricks, a Perpetual Calander is quite easy to program. Here it is presented for the AIM 65. In addition to being an interesting demonstration, it points out a few programming tricks required when using integer numbers in BASIC.

Mel Evans
1027 Redeemer
Ann Arbor, MI 48103

Another calendar printer? Yes, but with a couple of new twists. First, it puts out to the AIM printer. So the next time someone asks, "Okay, but what can it actually do?," you can give him an answer he can put in his pocket and take home with him.

Second, it has a built-in perpetual-calander algorithm that finds the starting day-of-the-week for any month of any year from 1583 AD (the start of the Gregorian calendar) to 999999999 AD (or until we change the calendar, or until the world ends, whichever comes first.) The algorithm is fairly simple, but the results can be impressive. For example:

```
RUN
HOW MANY MONTHS? 1
MONTH #? 7
YEAR? 1776


***** JULY  1776 ***
 S  M  T  W  T  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

"So, Independence Day happened on a Thursday."

"You mean it figured out all those leap years clear back to 1776?"

"Well, the equivalent of that, yes."

"How do I know it's right?"

"You don't."

"Okay, print me December, 1941. I know what day Pearl Harbor happened on."

```
RUN
HOW MANY MONTHS? 1
MONTH #? 12
YEAR? 1941
** DECEMBER 1941 ***
 S  M  T  W  T  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

"So December 7th was a Sunday."

"Hey, that's right! Okay, print me the start of year 2000."

```
RUN
HOW MANY MONTHS? 2
FIRST MONTH #? 1
YEAR? 2000
*** JANUARY 2000 ***
 S  M  T  W  T  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
** FEBRUARY 2000 ***
 S  M  T  W  T  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29
```

"How about that! It got February right. Century years aren't normally leap years, but every fourth century is, and there it is."

"Right. Want a calendar of this month, and may;be the rest of the year?"

"Sure, but make it through next February. Why do all calendars end at December?"

"I don't know, but this one won't."

```
RUN
HOW MANY MONTHS? 5
FIRST MONTH #? 10
YEAR? 1979
*** OCTOBER 1979 ***
 S  M  T  W  T  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
** NOVEMBER 1979 ***
 S  M  T  W  T  F  S
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
** DECEMBER 1979 ***
 S  M  T  W  T  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

```
*** JANUARY 1980 ***
 S  M  T  W  T  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

** FEBRUARY 1980 ***
 S  M  T  W  T  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
```

The day-of-the-week algorithm appeared in **BYTE** (Day of Week and Elapsed Time Programs," W. B. Agocs, **BYTE**, September, 1979, p. 126.). I read it, thought "That's neat," and forgot it. Then a calendar printing program for Teletype came out in **Kilobaud** ("Calendar Program," Steve Tabler, **Kilobaud Microcomputing**, October 1979, p. 102.). Can the AIM do that on its printer? Sure it can! Can I build in that day-of-week algorithm so that it doesn't need starting instructions? Sure I can! The resulting AIM BASIC program is listed in Figure 1.

The starting day-of-week algorithm is in lines 85 through 150. It uses "Zeller's congruence," as explained in Agoc's article. Zeller first does some juggling of month and year numbers before getting down to the main computation of the day-of-week (variable DW in line 150).

The algorithm packs more power than I needed here; it works for any year, month, and day-of-month (day-of-month is variable DM in line 130). Since I only needed the beginning day-of-week of each month to be printed, I set DM = 1 in line 129. To restore the algorithm to its full power, just delete that one statement, and use DM as an input.

AIM BASIC (like most BASICs) does not allow much format flexibility in printing numbers, so to squeeze those date-lines onto the 20-column printer, a string variable, L$, is used to build each line before printing. L$ is first nulled (e.g., line 290), and is then built up, character by character, as in line 350:

L$ = L$ + CHR$(48 + D2)

This statement adds D2, the second (units) digit of a two-digit date number, to line L$. As shown in Appendix E of the AIM BASIC manual, CHR$(48) is ASCII "0" (zero), and the other digits follow. So, if D2 = 5, say, ASCII "5" is added to the string. After the last character has been added, the line is printed (e.g., line 380).

If you are fussy about format, the above technique gives you total control over each column of each line. If numbers don't print to suit you; don't print numbers, print characters.

AIM BASIC has one quirk which I haven't noticed in others (but if you're running a different BASIC, you might like to check it out). If X evaluates internally as less than an integer, but is sufficiently close to that integer, it will print as the integer, but INT(X) will truncate down to the next-lower integer; e.g., if X = 4.99999...,you get:

```
PRINT X
5
PRINT INT (X)
4
```

Don't believe it? Try this:

```
LIST
 20 X=5
 30 Y=X/3
 40 Z=Y*X/3
 50 X=3*3*Z/X
 60 PRINT "X=";X;" I
NT(X)=";INT(X)
 70 END

RUN
X= 5  INT(X)= 4
```

To prevent this from happening, add a dab to X before doing INT(X). How much is a dab? Anything less than the smallest meaningful increment in X. The first equation in line 258, for example, is computing the century from the year:

C = INT(Y/100 + .005)

If year Y increases by 1, Y/100 increases by .01, so the added dab is half that. This assures that it will work for the year 2000, and is small enough so it will also work for 1999.

Another example is on Line 262:

INT(YC/4 + .1).

When YC increments by one, YC/4 increases by .25, and the added dab is less than half that. The previous .005 would work fine here, too, but .1 costs fewer bytes.

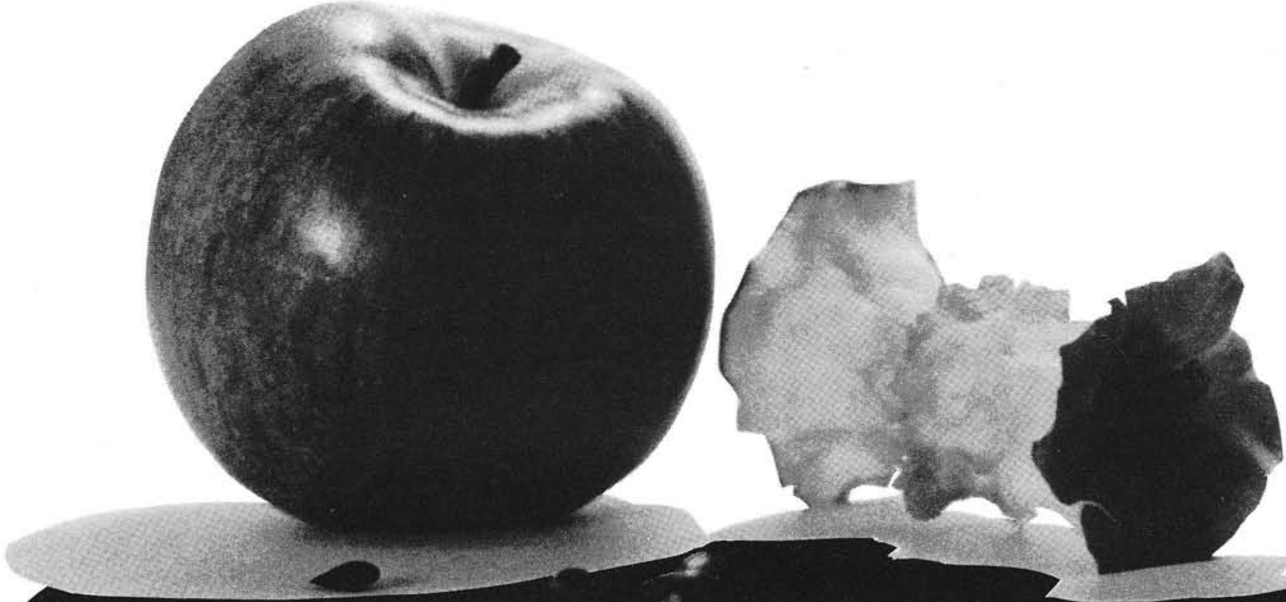
A final note of minor interest. Line 80 sends two line-feeds to the printer before starting the calendar, and line 430 sends it five line-feeds, so you can tear off the finished calendar without having to pump th "LF" key. And PRINT TAB (100) is sure neater than a string of five PRINT statements, isn't it?

```
LIST
 4 REM
 5 REM   PERPETUAL-
   CALENDAR PRINTER
 6 REM
 10 DIM A(12),R$(12)
 20 FOR I=1 TO 12:RE
AD A(I):NEXT I
 30 FOR I=1 TO 12:RE
AD R$(I):NEXT I
 40 INPUT "HOW MANY
MONTHS";N
 50 IF N=1 THEN INPU
T "MONTH #";M
 60 IF N>1 THEN INPU
T "FIRST MONTH #";M
 70 INPUT "YEAR";Y
 80 PRINT TAB(40)
 85 REM  CONVERT TO
 ZELLER MONTH & YEAR
 90 MZ=M-2:YZ=Y
 100 IF M=1 THEN MZ=
11:YZ=Y-1
 110 IF M=2 THEN MZ=
12:YZ=Y-1
 115 REM   FIND
STARTING DAY-OF-WEEK
 120 CZ=INT(YZ/100+.
005):YZ=YZ-100*CZ:DM
=1
 130 D1=INT(2.6*MZ-.
1)+DM+YZ
 140 D1=D1+INT(YZ/4+
.1)+INT(CZ/4+.1)-2*C
Z
 150 DW=D1-7*INT(D1/
7+.01)+1
 155 REM PRINT HEADE
R
 160 PRINT R$(M);:PR
INT Y;:PRINT"***"
 170 PRINT" S  M  T
  W  T  F  S"
 175 REM BUILD FIRST
   DATE-LINE & PRINT
 180 L$="":D1=DW-.5
 190 FOR I=1 TO 7
 200 DT=I-DW+1
 210 IF I<D1 THEN L$
=L$+"   "
 220 IF I>D1 THEN L$
=L$+"  "+CHR$(48+DT)
 230 IF I<6.5 THEN L
$=L$+" "
```

```
 240 NEXT I
 250 PRINT L$
 255 REM  CHECK FOR
      LEAP-YEAR
 258 C=INT(Y/100+.00
5):YC=Y-100*C
 260 A(2)=28
 262 IF YC=4*INT(YC/
4+.1) THEN A(2)=29
 264 IF YC<.5 THEN A
(2)=28
 270 IF YC<.5 AND C=
4*INT(C/4+.1) THEN A
(2)=29
 275 REM   BUILD
REMAINING DATE-LINES
       AND PRINT
 280 EN=0
 290 L$=""
 300 FOR I=1 TO 7
```

```
 310 DT=DT+1:IF DT>A
(M)+.5 THEN EN=1:GOT
O 380
 320 D1=INT(DT/10+.0
5):D2=DT-10*D1
 330 IF D1<.5 THEN L
$=L$+" "
 340 IF D1>.5 THEN L
$=L$+CHR$(48+D1)
 350 L$=L$+CHR$(48+D
2)
 360 IF I<6.5 THEN L
$=L$+" "
 370 NEXT I
 380 PRINT L$
 390 IF EN<.5 THEN 2
90
 400 PRINT" "
 405 REM   DO AGAIN
       FOR NEXT MONTH
```

```
 410 M=M+1:IF M>12.5
 THEN M=1:Y=Y+1
 420 N=N-1:IF N>.5 T
HEN 90
 430 PRINT TAB(100)
 440 END
 450 REM DATA: MONTH
   LENGTHS AND NAMES
 460 DATA 31,28,31,3
0,31,30,31,31,30,31,
30,31
 470 DATA *** JANUAR
Y,** FEBRUARY,**** M
ARCH
 480 DATA **** APRIL
 ,***** MAY  ,*****
JUNE
 490 DATA ***** JULY
,**** AUGUST,* SEPT
EMBER
 500 DATA *** OCTOBE
R,** NOVEMBER,** DEC
EMBER
```



Bi-Directional Scrolling

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

Roger Wagner
SW Data Systems
P.O. Box 582
Santee, CA 92071

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

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

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

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

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

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

If you do not want new data brought onto the screen, then 6,7 and 8,9 will have

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

```
*******************************
*                             *
*   APPLE SCROLLING ROUTINE   *
*                             *
*             BY              *
*        ROGER WAGNER         *
*                             *
*  THIS WILL LET EITHER PAGE  *
* SCROLL IN EITHER DIRECTION. *
*   IT IS PRIMARILY DESIGNED  *
* TO FEED NEW SCREEN DATA IN  *
* FROM A GIVEN RANGE OF RAM.  *
*                             *
*******************************
*
*
*
              OBJ   $300
              ORG   $300
WNDLFT   EQU  $20
WNDWDTH  EQU  $21
WNDTOP   EQU  $22
WNDBTM   EQU  $23
CH       EQU  $24
CV       EQU  $25
BASL     EQU  $28
BASH     EQU  $29
BAS2L    EQU  $2A
BAS2H    EQU  $2B
PAGE     EQU  $05
* FOR APPLESOFT USE PAGE EQU $1F
* PAGE MUST HOLD $04 FOR PG. 1,
* $08 FOR PG. 2
SCRNTP   EQU  $06
* $06,$07 = LO/HI BYTES
* OF START OF LINE JUST BEFORE
* TOP LINE
SCRNBTM  EQU  $08
* $08,$09= LO/HI BYTES
* OF START OF LINE JUST AFTER
* BOTTOM LINE
*
*

0300 A5 22     SCROLL   LDA  WNDTOP
0302 48                 PHA
0303 20 9E 03           JSR  VTABZ
0306 A5 28     NXTLN    LDA  BASL
0308 85 2A              STA  BAS2L
030A A5 29              LDA  BASH
030C 85 2B              STA  BAS2H
030E A4 21              LDY  WNDWDTH
0310 88                 DEY
0311 68                 PLA
0312 69 01              ADC  #$01
0314 C5 23              CMP  WNDBTM
0316 B0 0D              BCS  LDBTM
0318 48                 PHA
0319 20 9E 03           JSR  VTABZ
031C B1 28     NXTCHR   LDA  (BASL),Y
031E 91 2A              STA  (BAS2L),Y
0320 88                 DEY
0321 10 F9              BPL  NXTCHR
0323 30 E1              BMI  NXTLN
0325 A0 00     LDBTM    LDY  #00
0327 B1 08     LD2      LDA  (SCRNBTM),Y
0329 91 28              STA  (BASL),Y
032B C8                 INY
032C C4 21              CPY  WNDWDTH
032E 90 F7              BCC  LD2
0330 18        CRRCT    CLC
0331 A5 06              LDA  SCRNTP
0333 65 21              ADC  WNDWDTH
0335 85 06              STA  SCRNTP
0337 A5 07              LDA  SCRNTP+1
0339 69 00              ADC  #00
033B 85 07              STA  SCRNTP+1
033D 18                 CLC
033E A5 08              LDA  SCRNBTM
0340 65 21              ADC  WNDWDTH
0342 85 08              STA  SCRNBTM
0344 A5 09              LDA  SCRNBTM+1
0346 69 00              ADC  #00
0348 85 09              STA  SCRNBTM+1
034A 4C 9C 03           JMP  VTAB
               *
               *
034D 38        SCROLLDN SEC
034E A5 23              LDA  WNDBTM
0350 E9 01              SBC  #$01
0352 48                 PHA
0353 20 9E 03           JSR  VTABZ
0356 A5 28     NXTLN2   LDA  BASL
0358 85 2A              STA  BAS2L
035A A5 29              LDA  BASH
035C 85 2B              STA  BAS2H
035E A4 21              LDY  WNDWDTH
0360 88                 DEY
0361 68                 PLA
0362 E9 00              SBC  #$00
0364 C5 22              CMP  WNDTOP
0366 30 0D              BMI  LDTOP
0368 48                 PHA
0369 20 9E 03           JSR  VTABZ
036C B1 28     NXTCHR2  LDA  (BASL),Y
036E 91 2A              STA  (BAS2L),Y
0370 88                 DEY
0371 10 F9              BPL  NXTCHR2
0373 30 E1              BMI  NXTLN2
0375 A0 00     LDTOP    LDY  #$00
0377 B1 06     LT2      LDA  (SCRNTP),Y
0379 91 28              STA  (BASL),Y
037B C8                 INY
037C C4 21              CPY  WNDWDTH
037E 90 F7              BCC  LT2
0380 38        CRRCT2   SEC
0381 A5 06              LDA  SCRNTP
0383 E5 21              SBC  WNDWDTH
0385 85 06              STA  SCRNTP
0387 A5 07              LDA  SCRNTP+1
0389 E9 00              SBC  #00
038B 85 07              STA  SCRNTP+1
038D 38                 SEC
038E A5 08              LDA  SCRNBTM
0390 E5 21              SBC  WNDWDTH
0392 85 08              STA  SCRNBTM
0394 A5 09              LDA  SCRNBTM+1
0396 E9 00              SBC  #00
0398 85 09              STA  SCRNBTM+1
039A 60                 RTS
039B 00                 BRK
               *
               *
039C A5 25     VTAB     LDA  CV
039E 20 A6 03  VTABZ    JSR  BASCALC
03A1 65 20              ADC  WNDLFT
03A3 85 28              STA  BASL
03A5 60                 RTS
               *
               *
03A6 48        BASCALC  PHA
03A7 4A                 LSR
03A8 29 03              AND  #$03
03AA 05 05              ORA  PAGE
03AC 85 29              STA  BASH
03AE 68                 PLA
03AF 29 18              AND  #$18
03B1 90 02              BCC  BSCLC2
03B3 69 7F              ADC  #$7F
03B5 85 28     BSCLC2   STA  BASL
03B7 0A                 ASL
03B8 0A                 ASL
03B9 05 28              ORA  BASL
03BB 85 28              STA  BASL
03BD 60        END      RTS
--- END ASSEMBLY ---
TOTAL ERRORS: 00
```

to point to a part of memory that contains 40 blank space characters. One way to do this is to freeze on blank line on either page 1 or 2, and then set 6, 7 and 8, 9 must be reset to that value each time the scroll is done. This is because normally the scroll routine updates 6,7 and 8,9 by thee screen width so as to remain synchronized with the screen display another technique is to just clear the top or bottom line to blanks each time a scroll is done.

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

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

Special Notes:

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

```
*300.3BF
0300- A5 22 48 20 9E 03 A5 28
0308- 85 2A A5 29 85 2B A4 21
0310- 88 68 69 01 C5 23 B0 0D
0318- 48 20 9E 03 B1 28 91 2A
0320- 88 10 F9 30 E1 A0 00 B1
0328- 08 91 28 C8 C4 21 90 F7
0330- 18 A5 06 65 21 85 06 A5
0338- 07 69 00 85 07 18 A5 08
0340- 65 21 85 08 A5 09 69 00
0348- 85 09 4C 9C 03 38 A5 23
0350- E9 01 48 20 9E 03 A5 28
0358- 85 2A A5 29 85 2B A4 21
0360- 88 68 E9 00 C5 22 30 0D
0368- 48 20 9E 03 B1 28 91 2A
0370- 88 10 F9 30 E1 A0 00 B1
0378- 06 91 28 C8 C4 21 90 F7
0380- 38 A5 06 E5 21 85 06 A5
0388- 07 E9 00 85 07 38 A5 08
0390- E5 21 85 08 A5 09 E9 00
0398- 85 09 60 00 A5 25 20 A6
03A0- 03 65 20 85 28 60 48 4A
03A8- 29 03 05 05 85 29 68 29
03B0- 18 90 02 69 7F 85 28 0A
03B8- 0A 05 28 85 28 60 FF FF
```

Symbol Table

WNDLFT	0020
WNDWDTH	0021
WNDTOP	0022
WNDBTM	0023
CH	0024
CV	0025
BASL	0028
BASH	0029
BAS2L	002A
BAS2H	002B
PAGE	0005
SCRNTP	0006
SCRNBTM	0008
SCROLL	0300
NXTLN	0306
NXTCHR	031C
LDBTM	0325
LD2	0327
CRRCT	0330
SCROLLDN	034D
NXTLN2	0356
NXTCHR2	036C
LDTOP	0375
LT2	0377
CRRCT2	0380
VTAB	039C
VTABZ	039E
BASCALC	03A6
BSCLC2	03B5
END	03BD

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

The real rub is that Applesoft programs normally begin in memory at $800 (hex) which conflicts with page 2 use. The way around this is to do a 'POKE 104, 12: POKE 3072, 0' before loading your program. After loading do a 'CALL 54514' (unnecessary with DOS 32.). Unless you do a 'RESET', 'Control-B' other programs. Unfortunately, use of page 2 with the RAM version of Applesoft is to my knowledge impossible. (Sorry...)

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

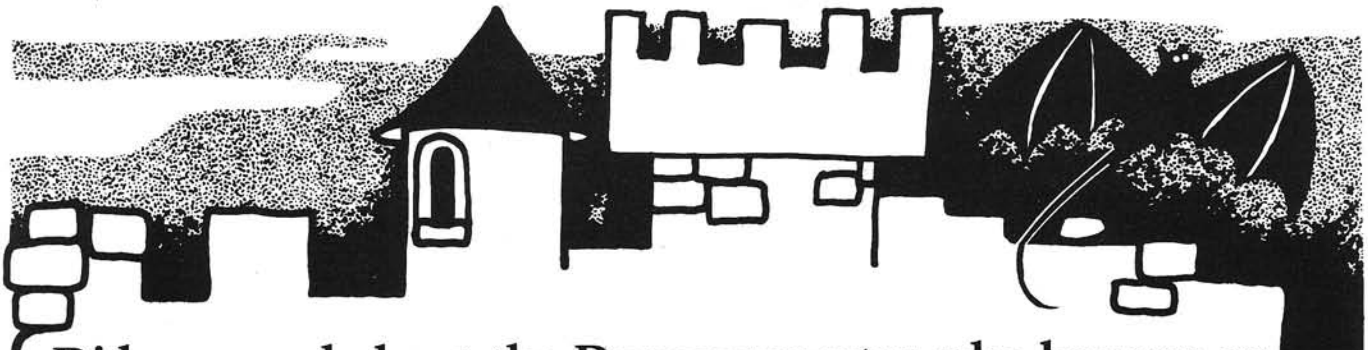
For further enlightenment, see the sample Integer Basic program which makes use of the scrolling routine. Have Fun!

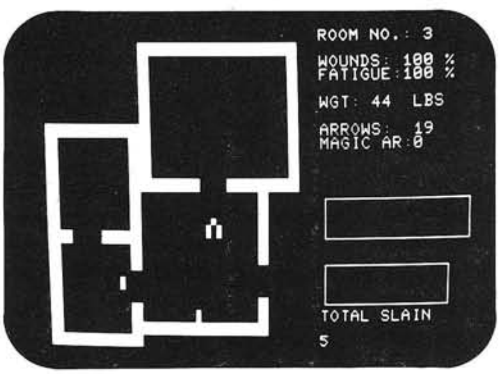
Location dependent:

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

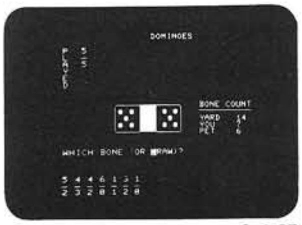
If page 2 of TEXT/GR is to be used, it must be protected by a 'LOMEM:3072' for integer BASIC, or a 'special Load' (as described in article) when using Applesoft.

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

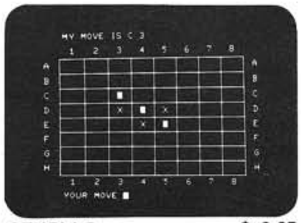



ROOM NO.: 3
WOUNDS: 100 %
FATIGUE: 100 %
WGT: 44 LBS
ARROWS: 19
MAGIC AR: 0
TOTAL SLAIN
5

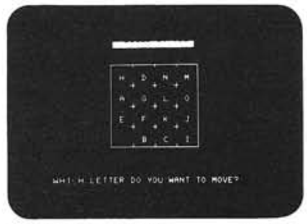
Software for the PET



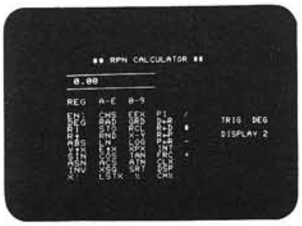
DOMINOES $ 6.95



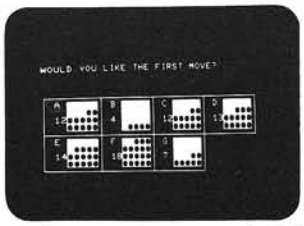
OTHELLO $ 9.95



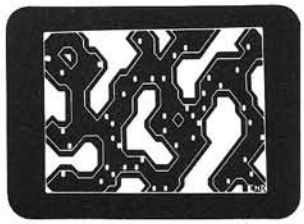
LETTER SQUARES $ 6.95



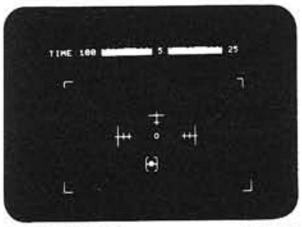
RPN MATHPACK $19.95



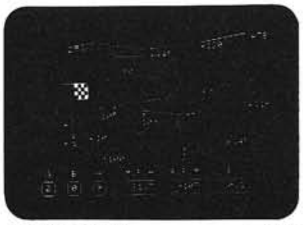
SUPER NIM $ 6.95



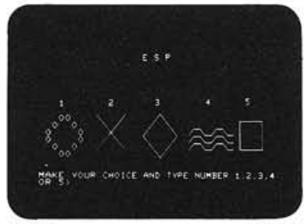
DIR/REF $ 6.95



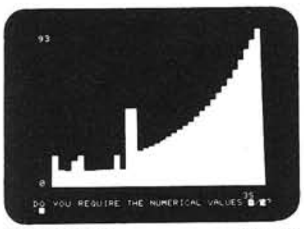
SPACE WARS $ 9.95



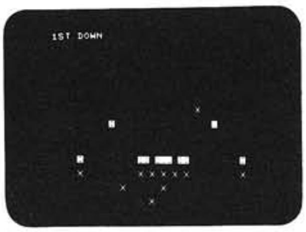
GRAND PRIX $ 6.95



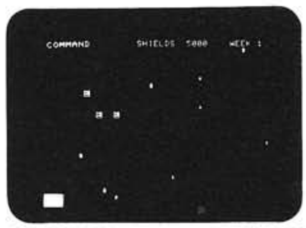
E.S.P. $ 9.95



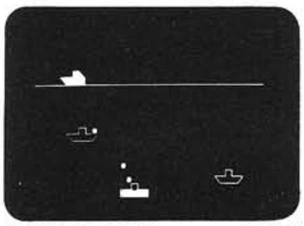
FORECAST $ 9.95



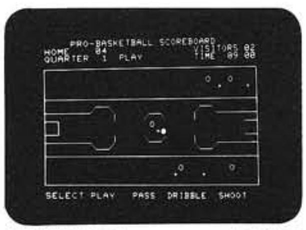
$ 6.95



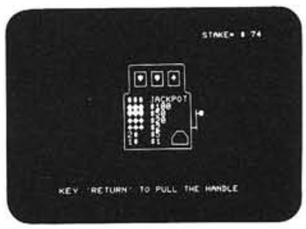
BLOCKADE $ 9.95



DEPTH CHARGE $ 9.95



BASKETBALL $ 9.95



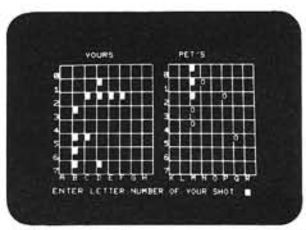
SLOT MACHINE $ 6.95

HOME ACCOUNTING $ 9.95

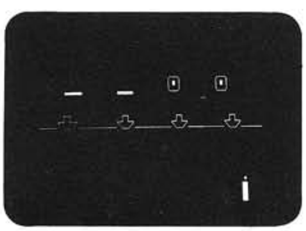
All orders include 3% postage and handling with a minimum of $1.00. California residents include 6% Sales Tax.

VISA MASTERCHARGE

PET IS A TRADEMARK OF COMMODORE BUSINESS MACHINES, INC.



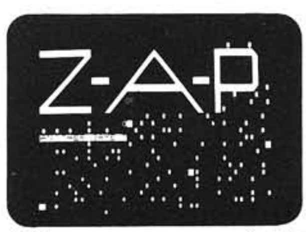
BATTLE SHIP $ 9.95



SHOOTING GALLERY $ 9.95

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

Dealer Inquiries Invited



ZAP $ 6.95

PROGRAMMA Software Program Products

The SY6516 Pseudo-16 Bit Processor

While the 6502 is a great microprocessor as it stands, advances are being considered to make it even better. One of the approaches is to add some new capabilities such as some 16 bit operations, improved addressing, and more.

Randall Hyde
12804 Magnolia
Chino, CA 91710

For those of you who may have wondered what the 6502 equivalent of the MC6809 would be, wonder no longer. Synertek is almost ready to ship the SY6516.

Synertek announced the 6516 almost a year ago, but due to production problems, it never quite made it. The 6516 was designed by Atari Inc. (back then it was to be called the 6509) for use with the Atari 400 and 800 computer systems. Unfortunately, Synertek was unable to deliver the chip in time for Atari to use it in their computers.

What is a Pseudo 16-bit Computer?

A pseudo 16-bit computer uses an internal 16-bit register arrangement, but externally it uses an eight bit bus. Sixteen bit data is multiplexed in, much like the Alpha Micro computer on the S-100 bus. In addition to the new 16-bit instructions, the 6516 maintains all of the 8-bit instructions of the 6502. You may reassemble your source files currently on the 6502 and run them directly on the 6516. All the information that I have recieved says that the 6516 is SOURCE code compatable with the 6502 and that it is OBJECT code incompatable with the 6502. I have heard rumors that Synertek is attempting to make the 6516 object code compatable, but quite honestly, I don't believe there is much chance of it happening.

Unlike the Motorola MC6809, which has a distinct set of 8-bit instructions and a distinct set of 16-bit instructions, the SY6516 contains a special register (the "Q" register) which toggles the system back and forth between 8-bit operation and 16-bit operation. In addition, all registers in the 6516 (A, X, Y, and SP) are now 16-bits wide. The "Q" register contains four bits which may be programmed to put the accumulator in the 16-bit mode, the X-register in the 16-bit mode, the Y-register in the 16-bit mode, and memory in the 16-bit mode (for use with INC, DEC, ASL, ROL, ROR, LSR, etc.). If the accumulator is programmed to be in the 16-bit mode, then LDA will load the accumulator with 16-bits, the low order byte coming from the specified address and the high order byte coming from the specified address plus one. If the accumulator is in the 8-bit mode, then the LDA instruction behaves identically to the LDA on the 6502. The other registers (X, Y, and Memory) behave identically.

It does not take twice as long to perform a 16-bit instruction compared to the equivalent 8-bit instruction, as you might expect. Usually only one additional clock cycle is required. This means that 6516 code will run as much as 3 times faster than 6502 code performing the same operation.

In addition, several instructions have been "speeded up" over the 6502 equivalent. For instance, implied instructions now only require one cycle for complete execution (the 6502 requires 2). Several other instructions have been speeded up as well (see Table One).

Variety of addressing modes is what makes the 6502 as flexible as it is. The 6516 includes many more addressing modes in its instruction set. In particular, indirect addressing (without the indexed by Y or preindexed by X), 16-bit relative addressing (there is now a jump relative, so your code can be relocatable), and direct page addressing.

Direct page addressing is something really special. It is available on the 6502 in a restricted form; on the 6502 it is called zero page addressing. Direct page addressing is different , in that any of the 256 pages in the 6516 address space may be used. The particular page is selected by the 8-bit direct page register "Z". The direct page facility should clear up many problems associated with zero page conflicts occuring in the 6502.

The New Instructions

The 6516 has a total of 114 instructions (compared to the 6502's 56). This gives a total of 255 different opcodes. Some of the new instructions are listed on the next page.

The User Flag

Bit 5 of the P register has been undefined to this point in the 6502. The 6516 utilizes this bit as a user defined flag. Included in the instruction set are instructions to set and clear this flag, as well as branch if set, and branch if clear. This user defined flag will prove to be a great help to users who are writing a boolean function. Up till now, the 6502 programmer had to use the carry or overflow flag. The user defined flag will help allieviate problems associated with the use of the aforementioned flags.

The 6516 instruction set was defined to allow maximum capability with the minimum number of instructions possible. For those of you who would really like to have seen an instruction of the form:

```
JMP (LBL,X)
```

you may simulate this by:

```
LDY LBL,X
YPC
```

The instruction sequence still requires only 3 bytes (assuming LBL is a direct page reference) and the timing is 7 cycles which is only two cycles more than a straight jump indirect. This would execute just as fast as a JMP (LBL,X) instruction were it included directly in the instruction set.

For those of you who would like to have seen the auto-increment and auto-decrement instructions of the MC6809, once again they can be simulated by the 6516. For instance, the sequence LAX, INX simulates a post increment and INX, LAX simulates a pre-increment. These instructions require two bytes (the same as the 6809) and execute in 3 to 4 cycles (depending on whether you are in the eight-bit or 16-bit mode). This speed is comparable to the 6809.

The only advantage of the 6809 over the 6516 is the 6809 multiply instruction. However, a software multiply on the 6516 should execute fast enough so that it won't make that big a difference.

The addition of two stacks in the 6809 is no real advantage since you can simulate 2, 3 or even n stacks with one 16-bit stack pointer. Those of you writing machine interpreters (such as the UCSD Pascal Pcode interpreter) will be able to simulate a stack machine quite easily on the 6516.

In my opinion, Synertek has taken everything wrong with the 6502 and fixed it, in addition to adding several features which I had not even previously considered. The 6516 is easily the most powerful 8-bit processor available (with due respects to the Intel 8088 which I would rate "almost there"). This opinion, incidently, is not just my own. EDN rated the 6516 above all the 8-bit processors and even some 16-bit processors, several months ago. If Synertek does indeed make the 6516 processor object code compatable with the 6502, it will definitely make the 6516 something you shouldn't scoff at. Why? Because oncethis happen, 50,000 APPLE II computers will be upgradeable directly to a 16-bit processor and maintain software compatability with existing software. Likewise, the 70,000 or so PETs will be upgradeable and the OSI, and the KIM, and of course, the SYM, etc. etc.

The only fault I find with the 6516 is the assembly language mnemonics chosen by Synertek. They should have followed the example laid down by Motorola and used mneomics which specify the action, leaving the decision of where the data is coming from to the operand field.

I am currently writing a version of LISA (an interactive 6502 assembler for the APPLE II) for the 6516. I will maintain Synertek's syntax, however I will add several extensions to the syntax and instruction set to allow a much more regular syntax. This should prove to be a little more pleasant to the die-hard computer scientist.

The New Instructions

```
LDS   M->S          (LOAD STACK POINTER FROM MEMORY)
LHA   M->AH         (LOAD HIGH ORDER ACC FROM MEMORY)
LHX   M->XH         (LOAD HIGH ORDER X-REG FROM MEMORY)
LHY   M->YH         (LOAD HIGH ORDER Y-REG FROM MEMORY)
LAX   M(X)->A       (LOAD ACC INDIRECT THROUGH X REG)
LAY   M(Y)->A       (LOAD ACC INDIRECT THROUGH Y REG)
SAY   A->M(Y)       (STORE ACC INDIRECT THROUGH Y REG)

ADD   A+M->A        (ADD W/O CARRY)
SUB   A-M->A        (SUBTRACT W/O CARRY)
AXA   A+X->A        (ADD X REG TO ACC)
AYA   A+Y->A        (ADD Y REG TO ACC)
AAX   A+X->X        (ADD ACC TO X REG)
AAY   A+Y->Y        (ADD ACC TO Y REG)
AMX   X+M->X        (ADD MEMORY TO X REG)
AMY   Y+M->Y        (ADD MEMORY TO Y REG)
NEG   NEG(A)->A     (2'S COMPLIMENT ACC)

RLT                 (ROTATE LEFT ACC)
RRT                 (ROTATE RIGHT ACC)
ASR                 (ARITHIMETIC SHIFT RIGHT ACC)
RHL                 (ROTATE AH LEFT THROUGH CARRY)
RHR                 (ROTATE AH RIGHT THROUGH CARRY)
RXL                 (ROTATE X REG LEFT THROUGH CARRY)
RXR                 (ROTATE X REG RIGHT THROUGH CARRY)
RYL                 (ROTATE Y REG LEFT THROUGH CARRY)
RYR                 (ROTATE Y REG RIGHT THROUGH CARRY)

TZA   Z->AL         (TRANSFER Z TO ACC LOW)
YPC   Y->PC         (TRANSFER Y REG TO PC)
PCY   PC->Y         (TRANSFER PC TO Y REG)
XHA   AL<->AH       (EXCHANGE ACC BYTES)
XHY   YL<->YH       (EXCHANGE Y REG BYTES)
XHX   XL<->XH       (EXCHANGE X REG BYTES)
XXY   X<->Y         (EXCHANCE X WITH Y REGISTAR)
      Qx<->Qy

SEF   1->F          (SET USER DEFINABLE FLAG)
CLF   0->F          (CLEAR USER DEFINABLE FLAG)
LDQ   M->Q          (LOAD Q REGISTAR FROM MEMORY)
SEV   1->V          (SET OVERFLOW FLAG)

BFS                 (BRANCH IF FLAG SET)
BFC                 (BRANCH IF FLAG CLEAR)
JNE                 (JUMP IF NOT EQUAL TO ZERO 16-BIT RELATIVE)
JEQ                 (JUMP IF EQUAL TO ZERO, 16-BIT RELATIVE)

PHD   A->(S)        (16-BIT ACC PUSH)
PLD   (S)->A        (16-BIT ACC PULL)
PHX   X->(S)        (16-BIT X REG PUSH)
PLX   (S)->X        (16-BIT X REG PULL)
PHY   Y->(S)        (16-BIT Y REG PUSH)
PLY   (S)->Y        (16-BIT Y REG PULL)
PHZ   Z->(S),       (PUSH Z REG ONTO STACK,
      Q->(S)         PUSH Q REG ONTO STACK)
PLZ   (S)->Q        (PULL Q FROM STACK,
      (S)->Z         PULL Z FROM STACK )
PHR                 (COMBINATION OF PHD, PHX, PHY, AND PHZ)
PLR                 (COMBINATION OF PLD, PLX, PLY, AND PLZ)

BR1                 (PERFORMS A  JSR ($FFF0) )
BR2                 (PERFORMS A  JSR ($FFF2) )
BR3                 (PERFORMS A  JSR ($FFF4) )
BR4                 (PERFORMS A  JSR ($FFF6) )
BR5                 (PERFORMS A  JSR ($FFF8) )
```

GREAT PET SOFTWARE

"Precise, humanized, well documented an excellent value" are the applauds now being given to United Software's line of software. These are sophisticated programs designed to meet the most stringent needs of individuals and business professionals. Every package is fully documented and includes easy to understand operator instructions.

DATABASE MANAGEMENT SYSTEM - A comprehensive, interactive system like those run on mainframes! Six modules comprising 42K of programming allow you to; create, edit, delete, display, print, sort, merge, etc., etc. - databases of up to 10,000 records. Printer routines automatically generate reports and labels on demand. **60** pages of concise documentation are included. Requirements - 16-32K PET and 2040 Dual Disk (printer optional). . . .**Cost $125**

ACCOUNTS RECEIVABLE/PAYABLE - A complete, yet simple to use accounting system designed with the small businessman in mind. The United Software system generates and tracks purchase orders and invoices all the way through posting "controlled" accounts payable and accounts receivable subsystems.
Keyed Random Access file methods makes data access almost instantaneous. The low-cost solution for the first time computer user with up to 500 active accounts. Requirements - 32K PET, Dual Disk, any 80-column printer. . . .**Cost $175**

CASH RECEIPTS & DISBURSEMENTS - Makes it a breeze to track all outgoing payments made by any type of business operation. Checks are tracked by number and categorized by type of expense. Sorting, summary, and audit trails make it easy to post to general ledger. This system also categorizes incoming receipts. Uses KRAM file access method. Requirements - 32K PET, Dual Disk (printer optional). . . .**Cost $99.95**

KRAM - Keyed Random Access Method - The new, ultra-fast access method for the PET Disk, provides keyed retrieval/storage of data, in either direct or sequential mode, by either full or partial key values. Written by United Software in 6502 machine code, and designed with the PET in mind, it exploits all the benefits of the PET Disk, allowing full optimization of your system. Eliminates the need for "Sort" routines! KRAM provides flexibility never seen on a micro before. KRAM is modeled after a very powerful access method used on large-scale IBM Virtual Storage mainframes. So "KRAM" all you can into your PET - it will love you for it. . . .**Cost $79.95**

(Sublicenses available to software houses.)

PROGRAMS FOR ENTERTAINMENT

Space Intruders ("Best Game of 1979")	$19.95
Jury/Hostage	12.50
Kentucky Derby/Roulette	9.95
Alien I.Q./Tank	9.95
Tunnelvision/Maze Chase	14.95
Submarine Attack	9.95
Battle of Midway	7.95
Laser Tank Battle	9.95
Swarm	14.95
Super Startrek	14.95
PET Music Box	29.95

UNITED SOFTWARE PROGRAMS FOR BUSINESS

Checkbook	$15.95
Mortgage	15.95
Finance	12.95
Bonds	12.95
Stock Analyzer	22.95
Stock Options	24.95
6502 Macro Assembler	49.95

Look for the RED-WHITE-BLUE United Software Display at your local computer dealer, or send check or moneyorder, plus $1.00 shipping to:

UNITED SOFTWARE OF AMERICA
750 Third Ave.
New York, N.Y. 10017

Dealer inquiries invited

DAKIN5 UTILITIES

```
***** DAKIN5 PROGRAMMING AIDS MENU *****
     TODAY'S DATE 09/18/79
  1. THE LISTER
  2. THE PEEKER
  3. THE CRUNCHER
  4. THE TEXT FILE COPY
  5. THE PROMPTER
  6. THE CALCULATOR
  7. THE DISKETTE COPY
  8. CHANGE TODAY'S DATE
  ENTER YOUR SELECTION -> .
```

```
** DAKIN5 PROGRAMMING AIDS II MENU ***
     TODAY'S DATE 12/17/79
  1. THE SCREEN PRINTER
  2. THE ARRAY EDITOR
  3. THE COPIER
  4. THE PATCHER
  5. THE LINE CROSS REFERENCE
  6. THE VARIABLE CROSS REFERENCE
  7. THE CALCULATOR II
  8. CHANGE TODAY'S DATE
  ENTER YOUR SELECTION -> .
```

The Cruncher removes REM statements and compresses code in Applesoft programs. The Prompter is a powerful data entry subroutine that can handle both string and numeric data. Options include using commas, decimal points, and leading zeros with right-justified numerics. A maximum field length can be specified to prevent overflow in both numeric and alphanumeric fields. The Diskette Copy formats an output disk, copies each track, and verifies that the output matches the input. PLUS FOUR MORE UTILITIES TO AID YOUR OWN PROGRAMMING. Suggested Retail Price for Dakin5 Programming Aids is $39.95.

The Copier copies absolutely any kind of file or program from one diskette to another. The Variable Cross Reference produces a cross-reference for all variable names used in an Applesoft BASIC program. The Line Cross Reference creates a cross-reference for an Applesoft BASIC program, showing where a given line is referenced by GOTO, GOSUB, THEN, or LIST statements. The Patcher allows you to display any sector of a diskette, and then to update any data within that sector. PLUS THREE MORE UTILITIES TO FACILITATE YOUR OWN PROGRAMMING. Suggested Retail Price for Dakin5 Programming Aids II is $49.95.

Each utility package includes a program diskette and very complete documentation. The hardware requirements are an Apple II, 48K of memory, 2 Disk II's, and a printer. Languages are Applesoft/Assembler.

See your Apple dealer or contact Dakin5 Corporation, P.O. Box 21187, Denver, Colorado 80221. Telephone: (303) 426-6090. Dakin5 developed the business application software The Controller™ for Apple Computer Inc.

DISK DRIVE WOES? PRINTER INTERACTION? MEMORY LOSS? ERRATIC OPERATION? DON'T BLAME THE SOFTWARE!

ISO-1

ISO-2

Power Line Spikes, Surges & Hash could be the culprit!
Floppies, printers, memory & processor often interact!
Our unique ISOLATORS eliminate equipment interaction AND curb damaging Power Line Spikes, Surges and Hash.

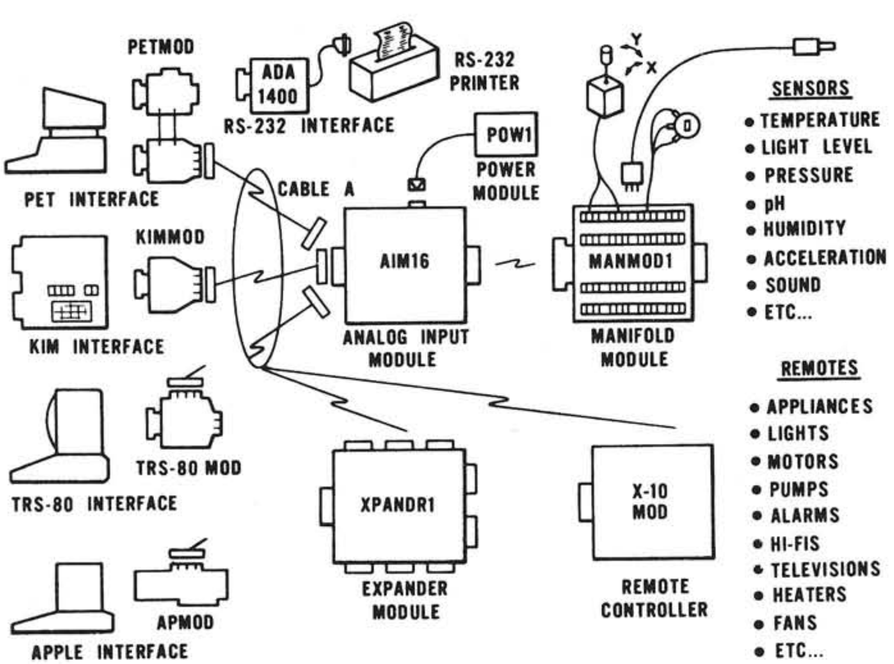
*ISOLATOR (ISO-1A) 3 filter isolated 3-prong sockets; integral Surge/Spike Suppression; 1875 W Maximum load, 1 KW load any socket	$54.95
*ISOLATOR (ISO-2) 2 filter isolated 3-prong socket banks; (6 sockets total); integral Spike/Surge Suppression; 1875 W Max load, 1 KW either bank	$54.95
*SUPER ISOLATOR (ISO-3), similar to ISO-1A except double filtering & Suppression	$79.95
*ISOLATOR (ISO-4), similar to ISO-1A except unit has 6 individually filtered sockets	$93.95
*ISOLATOR (ISO-5), similar to ISO-2 except unit has 3 socket banks, 9 sockets total	$76.95
*CIRCUIT BREAKER, any model (add-CB)	Add $ 6.00
*CKT BRKR/SWITCH/PILOT any model (-CBS)	Add $11.00

PHONE ORDERS 1-617-655-1532

Electronic Specialists, Inc.
171 South Main Street, Natick, Mass. 01760

Dept. mi

MICROCOMPUTER MEASUREMENT and



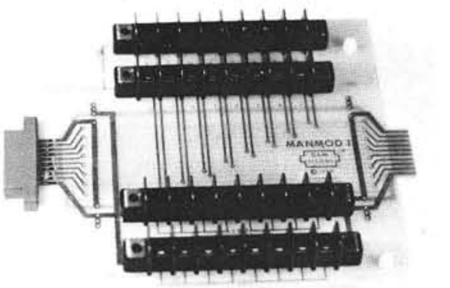
The world we live in is full of variables we want to measure. These include weight, temperature, pressure, humidity, speed and fluid level. These variables are continuous and their values may be represented by a voltage. This voltage is the analog of the physical variable. A device which converts a physical, mechanical or chemical quantity to a voltage is called a sensor.

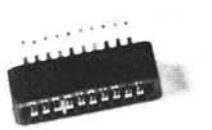
Computers do not understand voltages: They understand bits. Bits are digital signals. A device which converts voltages to bits is an analog-to-digital converter.

Our AIM 16 (Analog Input Module) is a 16 input analog-to-digital converter.

The goal of Connecticut microComputer in designing the uMAC SYSTEMS is to produce easy to use, low cost data acquisition and control modules for small computers. These acquisition and control modules will include digital input sensing (e.g. switches), analog input sensing (e.g. temperature, humidity), digital output control (e.g. lamps, motors, alarms), and analog output control (e.g. X-Y plotters, or oscilloscopes).

Connectors

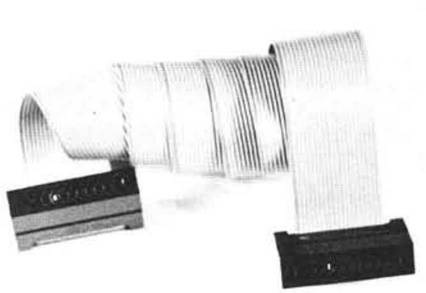




The AIM 16 requires connections to its input port (analog inputs) and its output port (computer interface). The ICON (Input CONnector) is a 20 pin, solder eyelet, edge connector for connecting inputs to each of the AIM16's 16 channels. The OCON (Output CONnector) is a 20 pin, solder eyelet edge connector for connecting the computer's input and output ports to the AIM16.

The MANMOD1 (MANifold MODule) replaces the ICON. It has screw terminals and barrier strips for all 16 inputs for connecting pots, joysticks, voltage sources, etc.

CABLE A24 (24 inch interconnect cable) has an interface connector on one end and an OCON equivalent on the other. This cable provides connections between the uMACSYSTEMS computer interfaces and the AIM 16 or XPANDR1 and between the XPANDR1 and up to eight AIM 16s.



Analog Input Module

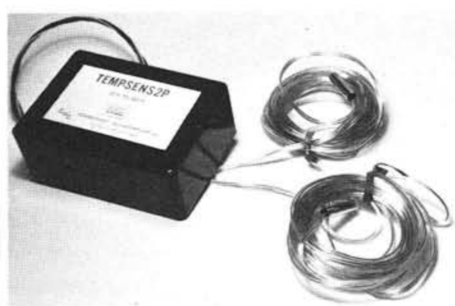
The AIM 16 is a 16 channel analog to digital converter designed to work with most microcomputers. The AIM16 is connected to the host computer through the computer's 8 bit input port and 8 bit output port, or through one of the uMAC SYSTEMS special interfaces.

The input voltage range is 0 to 5.12 volts. The input voltage is converted to a count between 0 and 255 (00 and FF hex). Resolution is 20 millivolts per count. Accuracy is 0.5% ± 1 bit. Conversion time is less than 100 microseconds per channel. All 16 channels can be scanned in less than 1.5 milliseconds.

Power requirements are 12 volts DC at 60 ma.

The POW1 is the power module for the AIM16. One POW1 supplies enough power for one AIM16, one MANMOD1, sixteen sensors, one XPANDR1 and one computer interface. The POW1 comes in an American version (POW1a) for 110 VAC and in a European version (POW1e) for 230 VAC.

TEMPSENS

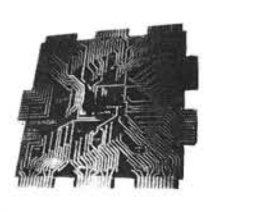


This module provides two temperature probes for use by the AIM16. This module should be used with the MANMOD1 for ease of hookup. The MANMOD1 will support up to 16 probes (eight TEMPSENS modules).

Resolution for each probe is 1°F.

XPANDR1

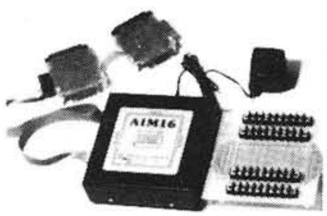
The XPANDR1 allows up to eight Input/Output modules to be connected to a computer at one time. The XPANDR1 is connected to the computer in place of the AIM16. Up to eight AIM16 modules are then connected to each of the eight ports provided using a CABLE A24 for each module. Power for the XPANDR1 is derived from the AIM16 connected to the first port.

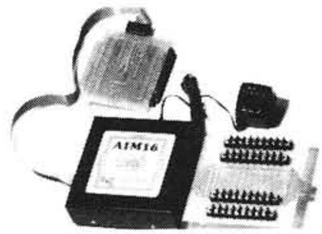


CONTROL for PET, Apple, KIM, and AIM

µMAC SYSTEMS

Computer Interfaces and Sets





For your convenience the AIM16 comes as part of a number of sets. The minimum configuration for a usable system is the AIM16, one POW1, one ICON and one OCON. The AIM16 Starter Set 2 includes a MANMOD1 in place of the ICON. Both of these sets require that you have a hardware knowledge of your computer and of computer interfacing.

For simple plug compatible systems we also offer computer interfaces and sets for several home computers.

INTRODUCING
SUPER X-10 MODULE

Open a door or window and turn on a light, tape recorder, alarm!

Control lab equipment. CLOSE THE LOOP on the real world.

AN INEXPENSIVE CONTROL SOLUTION FOR

HOME SECURITY
ENERGY CONSERVATION
GREENHOUSES
ENVIRONMENTAL CONTROL
INDUSTRIAL CONTROL
LABORATORIES

PLUS:
CLOCK, CALENDAR,
REMOTE SEQUENCE TRIGGERS

SUPER X-10 MOD SPECS

1. Remote controller
   Controls up to 256 different remote devices by sending signals over the house wiring to remote modules. Uses BSR remote modules available all over the USA (Sears, Radio Shack, etc.).
   Does not require BSR control module.
   Does not use sonic link.
2. Clock/calendar
   Time of day - hours, minutes, seconds
   Date - month, day - automatically corrects for 28,29,30 and 31 day months.
   Day of the week.
3. Digital inputs/outputs
   8 inputs - TTL levels or switch closures. Can be used as triggers for stored sequences.
   8 outputs - TTL levels
4. Computer interfaces
   S-100: Requires one 8-bit input port and one 8-bit output port.
   Requires cable assembly.
   PET, APPLE, TRS-80, KIM, SYM, AIM65: Plug-in sets available - no cable assembly required.
   Other: same as S-100
5. Self-contained module in metal case with its own power supply. Physical size approximately 5X6X2.

Price (until April 30, 1980): $199.00 (S-100), $249.00 (other)

All prices and specifications subject to change without notice. Our 30-day money back guarantee applies.

AIM16 (16 channel-8 bit Analog Input Module)	**179.00**
POW1a (POWer module-110 VAC)	**14.95**
POW1e (POWer module-230 VAC)	**24.95**
ICON (Input CONnector)	**9.95**
OCON (Output CONnector)	**9.95**
MANMOD1 (MANifold MODule)	**59.95**
CABLE A24 (24 inch interconnect cable)	**19.95**
XPANDR1 (allows up to 8 Input or Output modules to be connected to a computer at one time)	**59.95**
TEMPSENS2P1 (two temperature probes, -10°F to 160°F)	**49.95**
LIGHTSENS1P1 (light level probe)	**59.95**

The following sets include one AIM16, one POW1, one OCON and one ICON.

AIM16 Starter Set 1a (110 VAC)	**189.00**
AIM16 Starter Set 1e (230 VAC)	**199.00**

The following sets include one AIM16, one POW1, one OCON and one MANMOD1.

AIM16 Starter Set 2a (110 VAC)	**239.00**
AIM16 Starter Set 2e (230 VAC)	**249.00**

The following modules plug into their respective computers and, when used with a CABLE A24, eliminate the need for custom wiring of the computer interface.

PETMOD (Commodore PET)	**49.95**
KIMMOD (KIM, SYM, AIM65)	**39.95**
APMOD (APPLE II)	**59.95**
TRS-80 MOD (Radio Shack TRS-80)	**59.95**

µMAC SYSTEMS

Order Form

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659 TWX: 710-456-0052

The following sets include one AIM16, one POW1, one MANMOD1, one CABLE A24 and one computer interface module

PETSET1a (Commodore PET - 110 VAC)	**295.00**
PETSET1e (Commodore PET - 230 VAC)	**305.00**
KIMSET1a (KIM, SYM, AIM65 - 110 VAC)	**285.00**
KIMSET1e (KIM, SYM, AIM65 - 230 VAC)	**295.00**
APSET1a (APPLE II - 110 VAC)	**295.00**
APSET1e (APPLE II - 230 VAC)	**305.00**
TRS-80 SET1a (Radio Shack TRS-80 - 110 VAC)	**295.00**
TRS-80 SET1e (Radio Shack TRS-80 - 230 VAC)	**305.00**

QUANTITY	DESCRIPTION	PRICE	TOTAL
SUBTOTAL			
Handling and shipping — add per order			$3.00
Foreign orders add 10% for AIR postage			
add 7% sales tax			
TOTAL ENCLOSED			

NAME ______

COMPANY ______

ADDRESS ______

______

CITY ______

STATE ______ ZIP ______

VISA ○ M/C ○ Expiration date ______

Card number ______

PET Keysort

One of the most useful operations to perform on a real data base is a keysort. On the PET, due to some problems in the 'garbage collection' procedures, sorting string arrays can become very time consumming. A complete, general purpose keysorting program is presented which has many useful features and is efficient.

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

One of the most needed features of any business database program is a good sort routine. On the PET computer, there is also a real need for a way to sort string arrays without changing the strings. This is due to a quirk in the PET's "garbage collection" routine. PET was designed so every time a string is changed, a new string is created. Old versions are erased only after memory is filled. Then it deletes all the unneeded strings at once. As string space increases, collection time increases dramatically. With 24K of strings in memory, it can take several minutes.

Until future ROM's speed this process, it is best to avoid unneeded string manipulations. This makes a different sort program essential. For example, in an attendance program I developed, a heapsort is used. The heapsort itself takes about 20 minutes to sort 500 records. However, garbage collection adds another 2 hours! Clearly this is unacceptable.

One solution would be to define another array of integer string pointers, and sort that array. This would avoid moving strings entirely. As it happens. BASIC already stores its strings that way. Each string array is a table or pointers to another array of the actual strings. The pointers are above the program in memory, at the end of the variables. The strings are usually at the top of memory, though they may be anywhere.

I wrote a 'pointer sort' using the pointer table. It worked, but took too much memory, and had to be part of each program using it. I decided to put it in machine language instead. In the final form, it uses just under 1K of memory, at the top of memory. It resets BASIC's top of memory pointer to protect itself from BASIC, and saves a copy of PET's zerobase to protect basic from the program. The other main features of KEYSORT, are as follows:

1. extreme speed
2. simple operation
3. has defaults for all options
4. works with BASIC arrays
5. Remains until PET is reset
6. accepts any number of fields within a string
7. sorts any specified string array in memory
8. accepts any character as a field marker
9. both strings and fields may individually vary in length
10. extensive error ckecking

The two BASIC demonstration programs will illustrate these features. Listing #1 creates an array of random strings to sort. It does 3000 names in 28 seconds. Once you create an array to sort, merely enter 'sys(31841)' to sort it, either directly or from a program. Later, when you are ready to sort on an array other than the first in memory, try out listing #2. It uses all of KEYSORT's options at once. First, it selects the 'a$' array as the one to sort, ignoring all other arrays. Second, it selects the '>' character as the marker between fields. Using a marker allows one string to hold about 128 separate fields at once. The array may be instantly resorted on any of these fields, as shown in sample run #2, which sorts on field #4, actually the fifth field, since there is a field #0.) You may sort by name one minute, by birthdate the next and by zip code after that.

There is no need for strings to have a fixed length. Nor is there any need for fields within strings to be any special length. This avoids any waste of array space. KEYSORT's default field marker is the [tab] character, chr$(9). This is easily changed, as shown in listing #2. Also there need not be any end of field marker unless you select one. Listing #1 works fine without fields. If time is very important to you. Note that using fields doubles the sort time. In return, it allows you to maintain a single data base, for several programs, and sort only the fields needed by the particular program currently in use. That saves a lot of typing time.

When you study the asssembly source listing of KEYSORT, you will note a subroutine called 'spg'. This is a routine any 6502 owner can use to save up to half of zero base. By placing it at the end of the normal program flow, it only has to be called once, and its ending 'rts' then returns to BASIC.

After you assemble and save a copy of KEYSORT, call it without any arrays in memory. You will immediately see:

?array error
ready

This is KEYSORT's error message. Here it means no array was found. However, in the process, it reset Himem to protect itself from BASIC. You should do this each time you load KEYSORT, before defining strings. Otherwise they will overwrite the program. Note that if another program has already moved Himem lower than KEYSORT needs, the program leaves it alone.

If you see the '?array error' message at other times, one of several things has gone wrong. Perhaps there is no array to sort, ie. you cleared the variables. Or maybe the array has more than one dimension—only one is allowed. (Unsorted arrays may have all the dimensions you wish.

Then again, you may have erred in poking in KEYSORT: that becomes the default for future sorts. Note that at the end of listing #2, the seven command locations are reset to zero. Unless the next sort uses the same KEYSORT features or more, you will need to zero those functions not desired in the next sort.

Both the assembly listing and the hex dump of KEYSORT here are for a 32K PET. However, the program is easily relocatable. There is no data in the body of the program, and the program does not change itself. To relocate it, merely change all of the high order bytes of 3 byte instructions, except for the one that jumps to $C357 at $7 of 8. This is a call to the new ROM's error message printer. Table #1 shows all the locations to change for relocation at the top of all PET model's memory. If you have an 'old ROM' PET, (8K '79 or earlier vintage), you will need to make the changes listed in table -2. You will also be limited to 256 element arrays, as the old ROM's couldn't handle more elements than that at once.

Other 6502 users with Microsoft may be able to adapt KEYSORT to their needs. My local 6502 group is converting it to the Apple, which uses a similar memory structure. It may help you to know how PET stores arrays. Each array starts with 7 housekeeping bytes. The first byte of the first array's housekeeping is addressed by 'aras' in BASIC ($2c-2d,) low and high.) The last aray ends just before the address in 'eara', ($2e-2f). The first 2 housekeeping bytes in each array contain its name. If it is a string array, $80 will be added to the second character of the name as a flag. Even if there is no second character,byte 2 will contain $80. Bytes 3 and 4 are the low and high bytes of the offset from the start of the current array to the start of the next one. Byte 5 is the number of dimensions in the array, 1-3. Bytes 6 and 7 are the HIGH and low bytes respectively of the number of elements in the array. (This is backwards from the usual 6502 format.) There will be 1 more element than in the DIM statement, as the 0th element counts too. The 0th element begins immediately after the housekeeping bytes. Each element consists of 3 bytes. The first is the length of the string. The other 2 are the low and high bytes long. Also, when first dimensioned, all the length bytes and address bytes are set to zero.

I won't try to fully explain the BASIC and assembly listings of KEYSORT; they are fully commented. The only unusual feature in the BASIC programs is the use of PET's built-in 60th of a second jiffy clock, TI. When entering the assembly source, save $3500 for the text file and $0200 for labels. If you have less room available, delete some comments.

If you have questions about KEYSORT, or need help, write me at the above address. Please include a stamped reply envelope. If you want a custom tape copy of KEYSORT, please send along $5 for my time. Also, specify the starting or ending address you wish, and which ROM set you have.

***Table 1:* Locations to change on relocation**

$7C is found at:		$7C62
	7EFF	7F3A
$7D is found at:		$7C75
	7CF5	7D33
	7EAD	7EDC
$7E is found at:		$7DF7
	7E48	7E87
$7F is found at:		$7D44
	7D8F	7DA4
	7DAA	7DC7
	7E0C	7E68
	7E9A	7EB8
		7ECB

To relocate for:

PET 4K, change 7s to 0s
PET 8K, change 7s to 1s
PET 16K, change 7s to 3s

Code will reside at Himem.

***Table 2:* Changes for using old ROMs**
Source Changes:

Line 430 ARAS .DE $7E
Start of array space [650 & 670]
Line 440 EARA .DE $80
End of array space [1080 & 1120]
Line 450 HIM .DE $86
End of memory [560, 590, 610, & 630]
Line 460 ARER .DE $85
Offset into error table [1320]
Line 470 ERRP .DE $C359
Error msg. and stop [1330]

Object Code Changes

$7C77 = $7E	$7C7B = $7F
$7CC7 = $81	$7CCF = $80
$7C64 = $87	$7C6A = $87
$7C6E = $86	$7C72 = $86
$7CF7 = $85	$7CF9 = $59

```
 100 REM> SORT DEMO #1
 110 PRINT"SAMPLE RUN FOR LISTING #1":PRINT
 120 SZ=10:REM> ARRAY SIZE
 130 DIM A$(SZ)
 140 REM> MAKE UP STRINGS TO SORT
 150 FOR I=0 TO SZ
 160 A$=""
 170 : FOR J=1 TO 10*RND(0)+1
 180 : : A$=A$+CHR$(65+26*RND(0))
 190 : NEXT
 200 : A$(I)=A$
 210 : PRINT I,A$
 220 NEXT
 230 T1=TI:REM> ZERO THE CLOCK
 240 SYS(31845):REM> SORT
 250 T2=TI:REM> STOP THE CLOCK
 260 PRINT:PRINT"ORDER AFTER SORTING":PRINT
 270 REM> PRINT THE SORTED STRINGS
 280 FOR I=0 TO SZ
 290 : PRINT I,A$(I)
 300 NEXT
 310 REM> BRAG ABOUT THE TIME REQUIRED
 320 PRINT:PRINT"TIME TO SORT="(T2-T1)/60"SECONDS
READY.
```

```
100 REM> KEYSORT DEMO #2
110 PRINT"SAMPLE RUN FOR LISTING #2":PRINT
120 SZ=10:REM> ARRAY SIZE
130 F1=4:REM> FIELD # TO SORT BY
140 D1=ASC(">"):REM> FIELD DELIMITER
150 S$="A$":REM> SORT ARRAY NAME
160 ZC=32731:REM> START OF Z.P. COPY
170 NMFL=ZC+2:REM> FLAGS GIVEN ARRAY
180 DFLG=ZC+3:REM> FLAGS NEW DELIM.
190 DLIM=ZC+4:REM> STORES DELIMITER
200 FDFL=ZC+5:REM> FLAGS KEY FIELD
210 FLDS=ZC+6:REM> STORES KEY FIELD #
220 DIM B$(10,2):REM> GARBAGE
230 DIM C%(10)
240 DIM D(10)
250 DIM A$(SZ):REM> ACTUAL SORT ARRAY
260 REM> MAKE UP STRINGS TO SORT
270 FOR I=0 TO SZ
280 : A$=""
290 : FOR K=1 TO 5:REM> # OF FIELDS
300 : : FOR J=1 TO 10*RND(0)+1
310 : : : A$=A$+CHR$(65+26*RND(0))
320 : : NEXT
330 : : REM> FIELD DELIMITER
340 : : IF K<5 THEN A$=A$+CHR$(D1)
350 : NEXT
360 : A$(I)=A$
370 : PRINT I,A$
380 NEXT
390 REM> TELL SORT FIELD # IS GIVEN
400 POKE FDFL,ASC("#")
410 REM> TELL SORT WHICH FIELD TO USE
420 POKE FLDS,F1
430 REM> GIVE SORT NEW DELIMITER
440 POKE DLIM,D1
450 REM> TELL SORT TO CHANGE DELIMITERS
460 POKE DFLG,ASC("%")
470 REM> CHANGE SORT ARRAY NAME TO BASIC
480 REM> TELL SORT SETTING NAME
490 POKE NMFL,ASC("$")
500 POKE ZC,ASC(S$):REM> CHARACTER #1
510 S2=ASC(MID$(S$,2)):REM> & #2
520 IF S2=ASC("$") THEN S2=128
530 POKE ZC+1,S2
540 T1=TI:REM> ZERO THE CLOCK
550 SYS(31841):REM> SORT
560 T2=TI:REM> STOP THE CLOCK
570 REM> CANCEL SPECIAL OPTIONS
580 FOR I=ZC TO ZC+6
590 : POKE I,0
600 NEXT
610 PRINT:PRINT"SORTED ON FIELD #"F1:PRINT
620 REM> PRINT THE SORTED STRINGS
630 FOR I=0 TO SZ
640 : PRINT I,A$(I)
650 NEXT
660 REM> BRAG ABOUT THE TIME REQUIRED
670 PRINT:PRINT"TIME TO SORT="(T2-T1)/60"SECONDS
READY.
```

Sample Run for Listing 2

```
 0 BCHRFE>AKKYTGFCH>DG>XT>NTUTHO
 1 YKNJZBKT>NJVSMVI>UOF>NYCCINWGVG>YSCF
 2 IHRQ>WAGQWNEX>X>MP>QGCORK
 3 QJ>HQMWUE>UKQC>GVKPMF>VZ
 4 I>GZTWM>RLH>XSBP>MLWDPWO
 5 E>FL>Q>DRDT>SIVR
 6 F>WO>ZNBZOHCJG>PTQDLIO>ZVGLAH
 7 G>TFZDLKPN>NPCHNZSXMV>GSKC>BWIDSZ
 8 NSDNDTKBO>LPRJWBO>VLCD>FI>OXU
 9 SQPIFSBR>GKVSJCH>WDHCUQ>WQODPC>ZSIGN
10  BLDQENO>AY>Z>L>VAOKJHQR
```

Sorted on Field 4

```
 0 G>TFZDLKPN>NPCHNZSXMV>GSKC>BWIDSZ
 1 I>GZTWM>RLH>XSBP>MLWDPWO
 2 BCHRFE>AKKYTGFCH>DG>XT>NTUTHO
 3 NSDNDTKBO>LPRJWBO>VLCD>FI>OXU
 4 IHRQ>WAGQWNEX>X>MP>QGCORK
 5 E>FL>Q>DRDT>SIVR
 6 BLDQENO>AY>Z>L>VAOKJHQR
 7 QJ>HQMWUE>UKQC>GVKPMF>VZ
 8 YKNJZBKT>NJVSMVI>UOF>NYCCINWGVG>YSCF
 9 SQPIFSBR>GKVSJCH>WDHCUQ>WQODPC>ZSIGN
10  F>WO>ZNBZOHCJG>PTQDLIO>ZVGLAH

TIME TO SORT= .0833333333 SECONDS
READY.
```

END OF MAE PASS!

```
;              pet keysort
;
; a multi-key sort for pet basic arrays
;
;
;          by rev. james strama
;             120 w. king st.
;             decatur, il. 62521
;
;          as of feb. 14, 1980
;
;
sart        .ba $7c61             ;sys(31841)
;
;
;first 5 var.s poked from basic
arnm        .de $00               ;stores array name
nmfl        .de $02               ;array selected flag
dflg        .de $03               ;delimiter set flag
dlim        .de $04               ;delim. char.
fdfl        .de $05               ;key field set flag
flds        .de $06               ;sort field #
;
;
;most var.s as in basic heapsort
i           .de $07
j           .de $09
k           .de $0b
l           .de $0d
ln          .de $0f               ;1 by-$ lengths
ln1         .de $10               ;"
ln2         .de $11               ;"
n           .de $12               ;elements in array
r1          .de $14               ;3 by-temp. registers
r2          .de $17               ;"
r3          .de $1a               ;"
r4          .de $1d               ;"
s           .de $20               ;"
v1          .de $23               ;pointer start-3
;
;
;rom-dependent var.s
```

```
                 0430 aras         .de $2c        ;start of array space
                 0440 eara         .de $2e        ;end of array space
                 0450 him          .de $34        ;end of memory
                 0460 arer         .de $80        ;offset w/i error table
                 0470 errp         .de $c357      ;error mss. & stop
                 0480 ;
                 0490 ;
                 0500 ;other labels
                 0510 dch          .de $09        ;tab char.
                 0520 locs         .de $24        ;# of locations to flip
                 0530 ;
                 0540 ;
7C61- A9 7C      0550              lda #h,sart    ;lower himem
7C63- C5 35      0560              cmp *him+1     ;unless already lower
7C65- F0 04      0570              beq hok
7C67- B0 0A      0580              bcs sav
7C69- 85 35      0590              sta *him+1
7C6B- A9 61      0600 hok          lda #l,sart    ;hi, then lo
7C6D- C5 34      0610              cmp *him
7C6F- B0 02      0620              bcs sav
7C71- 85 34      0630              sta *him
7C73- 20 A0 7D   0640 sav          jsr spg        ;save z.p.
7C76- A5 2C      0650              lda *aras      ;set current array ptr.
7C78- 85 15      0660              sta *r1+1      ;curr. ar. st.
7C7A- A5 2D      0670              lda *aras+1
7C7C- 85 16      0680              sta *r1+2
7C7E- A5 02      0690              lda *nmfl
7C80- C9 24      0700              cmp #'$        ;flags array name
7C82- F0 06      0710              beq ckna       ;find name to match
7C84- A9 80      0720              lda #$80       ;basic's $ array flag
7C86- 85 00      0730              sta *arnm      ;use any $ array
7C88- 85 01      0740              sta *arnm+1
7C8A- A0 00      0750 ckna         ldy #0
7C8C- B1 15      0760              lda (r1+1),y
7C8E- C5 00      0770              cmp *arnm
7C90- F0 06      0780              beq lok        ;1st. char. ok
7C92- A9 80      0790              lda #$80
7C94- C5 00      0800              cmp *arnm
7C96- D0 08      0810              bne wrnm       ;not right array
7C98- C8         0820 lok          iny
7C99- B1 15      0830              lda (r1+1),y
7C9B- C5 01      0840              cmp *arnm+1    ;now 2nd. char.
7C9D- 30 01      0850              bmi wrnm       ;>=$80 if $ array
7C9F- C8         0860              iny            ;y>1=right
7CA0- 98         0870 wrnm         tya
7CA1- AA         0880              tax            ;name flag to x
7CA2- A5 03      0890              lda *dflg      ;flag to change delim.
7CA4- C9 25      0900              cmp #'%        ;unless=, use [tab]
7CA6- F0 04      0910              beq flc2       ;delimiter entered
7CA8- A9 09      0920              lda #dch       ;standard char.
7CAA- 85 04      0930              sta *dlim
7CAC- A5 05      0940 flc2         lda *fdfl      ;see if fields>0
7CAE- C9 23      0950              cmp #'#        ;flag set?
7CB0- F0 04      0960              beq flch       ;yes
7CB2- A9 00      0970              lda #$00       ;no, set 0 fields
7CB4- 85 06      0980              sta *flds
7CB6- A0 02      0990 flch         ldy #2         ;offset to next array
7CB8- B1 15      1000              lda (r1+1),y   ;lo
```

```
7CBA- 18              clc
7CBB- 65 15           adc *r1+1       ;add to current's start
7CBD- 85 18           sta *r2+1       ;to next array ptr.
7CBF- C8              iny
7CC0- B1 15           lda (r1+1),y    ;hi
7CC2- 65 16           adc *r1+2
7CC4- 85 19           sta *r2+2
7CC6- A5 2F           lda *eara+1     ;last array?
7CC8- C5 19           cmp *r2+        ;next ar. st.
7CCA- F0 02           beq ehfd        ;meybe
7CCC- B0 0E           bcs nare        ;no
7CCE- A5 2E     ehfd  lda *eara       ;check lo
7CD0- C5 18           cmp *r2+1
7CD2- F0 02           beq efnd        ;end found
7CD4- B0 06           bcs nare        ;not end
7CD6- E0 02     efnd  cpx #2          ;found array?
7CD8- B0 11           bs fara         ;yes
7CDA- 90 17           bcc oops        ;no
7CDC- E0 02     nare  cpx #2          ;found it?
7CDE- B0 0B           bcs fara        ;yes
7CE0- A5 18           lda *r2+1       ;no, next=current
7CE2- 85 15           sta *r1+1
7CE4- A5 19           lda *r2+2
7CE6- 85 16           sta *r1+2
7CE8- 18              clc
7CE9- 90 9F           bcc okna        ;jump
7CEB- A0 04     fara  ldy #4          ;1 dimension allowed
7CED- B1 15           lda (r1+1),y
7CEF- C9 01           cmp #1
7CF1- F0 08           beq fsiz        ;ok
7CF3- 20 A0 7D  oops  jsr spg         ;restore basic
7CF6- A2 80           ldx #arer
7CF8- 4C 57 C3        jmp errp        ;print error & abort
7CFB- A0 06     fsiz  ldy #6          ;# of elements
7CFD- B1 15           lda (r1+1),y    ; lo
7CFF- 85 12           sta *n
7D01- 88              dey
7D02- B1 15           lda (r1+1),y    ;hi
7D04- 85 13           sta *n+1
7D06- 18              clc             ;find mid element
7D07- 6A              ror a
7D08- 85 0E           sta *l+1
7D0A- A5 12           lda *n
7D0C- 6A              ror a
7D0D- 18              clc             ;make % & +1
7D0E- 69 01           adc #1
7D10- 85 0D           sta *l
7D12- A5 0E           lda *l+1
7D14- 69 00           adc #0
7D16- 85 0E           sta *l+1
7D18- A5 15           lda *r1+1       ;current=element#0-3
7D1A- 18              clc
7D1B- 69 04           adc #4
7D1D- 85 23           sta *v1
7D1F- A5 16           lda *r1+2
7D21- 69 00           adc #0
7D23- 85 24           sta *v1+1
```

```
                      ;k=n
7D25- A5 12                     lda *n
7D27- 85 0B                     sta *k
7D29- A5 13                     lda *n+1
7D2B- 85 0C                     sta *k+1
                      ;if l<>1 goto l=l-1
7D2D- A5 0E           main      lda *l+1
7D2F- F0 03                     beq ndec
7D31- 4C B2 7D        dec2      jmp decl
7D34- A5 0D           ndec      lda *l
7D36- C9 01                     cmp #1
7D38- D0 F7                     bne dec2
                      ;r1=k
7D3A- A5 0B                     lda *k              ;get k
7D3C- 85 1B                     sta *r3+1
7D3E- A5 0C                     lda *k+1
7D40- 85 1C                     sta *r3+2
7D42- 20 B3 7F                  jsr conv            ;ele. # to ptr. addr.
                      ;s=v(k)
7D45- A0 00                     ldy #0              ;r(1) to s
7D47- B1 15                     lda (r1+1),y
7D49- 85 20                     sta *s
7D4B- C8                        iny
7D4C- B1 15                     lda (r1+1),y
7D4E- 85 21                     sta *s+1
7D50- C8                        iny
7D51- B1 15                     lda (r1+1),y
7D53- 85 22                     sta *s+2
                      ;v(k)=v(1)
7D55- A5 23                     lda *v1             ;r(2)=v1+3
7D57- 18                        clc
7D58- 69 03                     adc #$03
7D5A- 85 18                     sta *r2+1
7D5C- A5 24                     lda *v1+1
7D5E- 69 00                     adc #$00
7D60- 85 19                     sta *r2+2
7D62- B1 18                     lda (r2+1),y        ;(r(1))=(r(2))
7D64- 91 15                     sta (r1+1),y
7D66- 88                        dey
7D67- B1 18                     lda (r2+1),y
7D69- 91 15                     sta (r1+1),y
7D6B- 88                        dey
7D6C- B1 18                     lda (r2+1),y
7D6E- 91 15                     sta (r1+1),y
                      ;k=k-1
7D70- 38                        sec
7D71- A5 0B                     lda *k
7D73- E9 01                     sbc #1              ;subtract with borrow
7D75- 85 0B                     sta *k
7D77- A5 0C                     lda *k+1
7D79- E9 00                     sbc #0
7D7B- 85 0C                     sta *k+1
                      ;if k<>1 goto jeql
7D7D- C9 00                     cmp #0
7D7F- D0 57                     bne jeql
7D81- A5 0B                     lda *k
7D83- D0 53                     bne jeql
                      ;r1=i
```

```
7D85- A5 07              lda *i          ;converted i to r(1)
7D87- 85 1B              sta *r3+1
7D89- A5 08              lda *i+1
7D8B- 85 1C              sta *r3+2
7D8D- 20 B3 7F           jsr conv
            ;v(i)=s
7D90- A5 20              lda *s
7D92- A0 00              ldy #0
7D94- 91 15              sta (r1+1),y    ;s to (r(1))
7D96- C8                 iny
7D97- A5 21              lda *s+1
7D99- 91 15              sta (r1+1),y
7D9B- C8                 iny
7D9C- A5 22              lda *s+2
7D9E- 91 15              sta (r1+1),y
            ;exchange z.p. s/r
7DA0- A2 24    spg       ldx #locs       ;flip z.p. locations
7DA2- BD DB 7F slop      lda cpg,x
7DA5- 48                 pha
7DA6- B5 00              lda *0,x
7DA8- 9D DB 7F           sta cpg,x
7DAB- 68                 pla
7DAC- 95 00              sta *0,x
7DAE- CA                 dex
7DAF- 10 F1              bpl slop        ;$7f max
7DB1- 60                 rts             ;end or return
            ; l=l-1
7DB2- 38       decl      sec
7DB3- A5 0D              lda *l
7DB5- E9 01              sbc #1          ;-1
7DB7- 85 0D              sta *l
7DB9- A5 0E              lda *l+1
7DBB- E9 00              sbc #0
7DBD- 85 0E              sta *l+1
            ;r1=l
7DBF- 85 1C              sta *r3+2
7DC1- A5 0D              lda *l          ;conv. l to r(1)
7DC3- 85 1B              sta *r3+1
7DC5- 20 B3 7F           jsr conv
            ;s=v(l)
7DC8- A0 00              ldy #0          ;(r(1)) to s
7DCA- B1 15              lda (r1+1),y
7DCC- 85 20              sta *s
7DCE- C8                 iny
7DCF- B1 15              lda (r1+1),y
7DD1- 85 21              sta *s+1
7DD3- C8                 iny
7DD4- B1 15              lda (r1+1),y
7DD6- 85 22              sta *s+2
            ;j=l
7DD8- A5 0D    jeql      lda *l
7DDA- 85 09              sta *j
7DDC- A5 0E              lda *l+1
7DDE- 85 0A              sta *j+1
            ;i=j
7DE0- A5 09    ieqj      lda *j
7DE2- 85 07              sta *i
7DE4- A5 0A              lda *j+1
```

```
7DE6- 85 08                 sta *i+1
           ;j=j+j
7DE8- 18                    clc
7DE9- 26 09                 rol *j              ;double j
7DEB- 26 0A                 rol *j+1
           ;compare j & k
7DED- A5 0A                 lda *j+1            ;hi first
7DEF- C5 0C                 cmp *k+1
7DF1- F0 05                 beq heq
7DF3- 90 0D                 bcc j<k
7DF5- 4C 90 7E   toj>       jmp j>k
7DF8- A5 09      heq        lda *j              ;if hi=, then ck. lo
7DFA- C5 0B                 cmp *k
7DFC- 90 04                 bcc j<k
7DFE- F0 5E                 beq jeqk
7E00- B0 F3                 bcs toj>
           ;if j<k then r1=j
7E02- A5 09      j<k        lda *j              ;j to r(1)
7E04- 85 1B                 sta *r3+1
7E06- A5 0A                 lda *j+1
7E08- 85 1C                 sta *r3+2
7E0A- 20 B3 7F              jsr conv
           ;r2=v(j)
7E0D- A0 00                 ldy #0              ;(r(1)) to r(2)
7E0F- B1 15                 lda (r1+1),y
7E11- 85 17                 sta *r2
7E13- C8                    iny
7E14- B1 15                 lda (r1+1),y
7E16- 85 18                 sta *r2+1
7E18- C8                    iny
7E19- B1 15                 lda (r1+1),y
7E1B- 85 19                 sta *r2+2
           ;r1=v(j+1)
7E1D- 18                    clc
7E1E- A5 15                 lda *r1+1
7E20- 69 03                 adc #3              ;3 by. betw. ptrs.
7E22- 85 15                 sta *r1+1           ;up (r(1)) by 1 ele.
7E24- A5 16                 lda *r1+2
7E26- 69 00                 adc #0
7E28- 85 16                 sta *r1+2
           ;compare v(j+1) & v(j)
7E2A- A0 02                 ldy #2              ;copy to r(3) & r(4)
7E2C- B1 15                 lda (r1+1),y        ;v(j+1)
7E2E- 85 1C                 sta *r3+2
7E30- 88                    dey
7E31- B1 15                 lda (r1+1),y
7E33- 85 1B                 sta *r3+1
7E35- 88                    dey
7E36-B1 15                  lda (r1+1),y
7E38- 85 1A                 sta *r3
7E3A- A5 19                 lda *r2+2           ;v(j)
7E3C- 85 1F                 sta *r4+2
7E3E- A5 18                 lda *r2+1
7E40- 85 1E                 sta *r4+1
7E42- A5 17                 lda *r2
7E44- 85 1D                 sta *r4
7E46- 20 DF 7E              jsr cmpr            ;compare actual $ data
           ;if v(j)>=v(j+1) goto jeqk
```

```
7E49- A5 14     3330            lda *r1         ;deciding char.s
7E4B- C5 17     3340            cmp *r2
7E4D- 90 0F     3350            bcc jeqk        ;r(2)>=r(1)
7E4F- F0 0D     3360            beq jeqk
                3370 ;j=j+1
7E51- 18        3380            clc             ;+1
7E52- A5 09     3390            lda *j
7E54- 69 01     3400            adc #1
7E56- 85 09     3410            sta *j
7E58- A5 0A     3420            lda *j+1
7E5A- 69 00     3430            adc #0
7E5C- 85 0A     3440            sta *j+1
                3450 ;r1=j
7E5E- A5 09     3460 jeqk       lda *j          ;j =ed k
7E60- 85 1B     3470            sta *r3+1       ;conv. j to r(1)
7E62- A5 0A     3480            lda *j+1
7E64- 85 1C     3490            sta *r3+2
7E66- 20 B3 7F  3500            jsr conv
                3510 ;compare v(j) & s
7E69- A0 02     3520            ldy #2          ;(r(1)) =ed v(j)
7E6B- B1 15     3530            lda (r1+1),y
7E6D- 85 1C     3540            sta *r3+2       ;copy for s/r
7E6F- 88        3550            dey
7E70- B1 15     3560            lda (r1+1),y    ;v(j)
7E72- 85 1B     3570            sta *r3+1
7E74- 88        3580            dey
7E75- B1 15     3590            lda (r1+1),y
7E77- 85 1A     3600            sta *r3
7E79- A5 22     3610            lda *s+2        ;s
7E7B- 85 1F     3620            sta *r4+2
7E7D- A5 21     3630            lda *s+1
7E7F- 85 1E     3640            sta *r4+1
7E81- A5 20     3650            lda *s
7E83- 85 1D     3660            sta *r4
7E85- 20 DF 7E  3670            jsr cmpr        ;compare $s
                3680 ;if s<v(j) goto s<vj
7E88- A5 14     3690            lda *r1         ;results here
7E8A- C5 17     3700            cmp *r2         ;r(3)'s in r(2)
7E8C- F0 02     3710            beq j>k         ;if=
7E8E- B0 1E     3720            bcs s<vj        ;if r(1)>r(2)
                3730 ;r1=i
7E90- A5 07     3740 j>k        lda *i          ;v(j)'s<=s's
7E92- 85 1B     3750            sta *r3+1
7E94- A5 08     3760            lda *i+1        ;conv. i to r(1)
7E96- 85 1C     3770            sta *r3+2
7E98- 20 B3 7F  3780            jsr conv
                3790 ;v(i)=s
7E9B- A0 00     3800            ldy #0          ;s to (r(1))
7E9D- A5 20     3810            lda *s
7E9F- 91 15     3820            sta (r1+1),y
7EA1- C8        3830            iny
7EA2- A5 21     3840            lda *s+1
7EA4- 91 15     3850            sta (r1+1),y
7EA6- C8        3860            iny
7EA7- A5 22     3870            lda *s+2
7EA9- 91 15     3880            sta (r1+1),y
7EAB- 4C 2D 7D  3890            jmp main        ;to top of main loop
                3900 ;r2=i
```

```
7EAE- A5 07     s(vj          lda *i             ;s's(v(j)'s
7EB0- 85 1B                   sta *r3+1
7EB2- A5 08                   lda *i+1           ;conv. i to r(2)
7EB4- 85 1C                   sta *r3+2
7EB6- 20 B3 7F                jsr conv
7EB9- A5 15                   lda *r1+1          ;move to r(2)
7EBB- 85 18                   sta *r2+1
7EBD- A5 16                   lda *r1+2
7EBF- 85 19                   sta *r2+2
                ;r1=j
7EC1- A5 09                   lda *j             ;conv. j to r(1)
7EC3- 85 1B                   sta *r3+1
7EC5- A5 0A                   lda *j+1
7EC7- 85 1C                   sta *r3+2
7EC9- 20 B3 7F                jsr conv
                ;v(i)=v(j)
7ECC- A0 00                   ldy #0             ;j's indirect to i's
7ECE- B1 15                   lda (r1+1),y
7ED0- 91 18                   sta (r2+1),y
7ED2- C8                      iny
7ED3- B1 15                   lda (r1+1),y
7ED5- 91 18                   sta (r2+1),y
7ED7- C8                      iny
7ED8- B1 15                   lda (r1+1),y
7EDA- 91 18                   sta (r2+1),y
7EDC- 4C E0 7D                jmp ieqj           ;back to middle
                ;cmpr $s s/r
7EDF- A0 00     cmpr          ldy #0
7EE1- 84 0F                   sty *ln
7EE3- A6 06                   ldx *flds
7EE5- D0 0B                   bne notz
7EE7- A5 1D                   lda *r4            ;sort on field#0
7EE9- 85 10                   sta *ln1
7EEB- A5 1A                   lda *r3            ;1st. var. in r(3)
7EED- 85 11                   sta *ln2           ;2nd. in r(4)
7EEF- 18                      clc
7EF0- 90 70                   bcc fsh            ;find shorter $
7EF2- A5 04     notz          lda *dlim          ;field delimiter
7EF4- D1 1E     cont          cmp (r4+1),y       ;cmp flag to $ char.
7EF6- F0 08                   beq fndd           ;found delim.
7EF8- C8        cnt3          iny
7EF9- C4 1D                   cpy *r4            ;end of $?
7EFB- 90 F7                   bcc cont           ;no
7EFD- 4C F3 7C                jmp oops           ;on error
7F00- 84 0F     fndd          sty *ln            ;mark current offset
7F02- CA                      dex
7F03- F0 02                   beq sfld           ;sort field
7F05- B0 F1                   bcs cnt3           ;count on
7F07- C8        sfld          iny                ;@ sort field
7F08- D1 1E                   cmp (r4+1),y       ;next char.
7F0A- F0 05                   beq fnef           ;field beyond sort
7F0C- C4 1D                   cpy *r4            ;end of $?
7F0E- 90 F7                   bcc sfld
7F10- C8                      iny
7F11- 88        fnef          dey                ;end or next field
7F12- 98                      tya
7F13- 38                      sec
7F14- E5 0F                   sbc *ln            ;current-start of "
```

```
7F16- 85 10      4490           sta *ln1         ;len sort field
7F18- E6 0F      4500           inc *ln          ;skip delim.
7F1A- A5 0F      4510           lda *ln          ;offset from start
7F1C- 18         4520           clc
7F1D- 65 1E      4530           adc *r4+1        ;start of sort field
7F1F- 85 1E      4540           sta *r4+1
7F21- A5 1F      4550           lda *r4+2
7F23- 69 00      4560           adc #$00
7F25- 85 1F      4570           sta *r4+2        ;r(4)'s done
7F27- A0 00      4580           ldy #0           ;now other $
7F29- 84 0F      4590           sty *ln
7F2B- A6 06      4600           ldx *flds
7F2D- A5 04      4610           lda *dlim        ;only reg.s differ
7F2F- D1 1B      4620 cnt2      cmp (r3+1),y
7F31- F0 08      4630           beq fnd2
7F33- C8         4640 cnt4      iny
7F34- C4 1A      4650           cpy *r3
7F36- 90 F7      4660           bcc cnt2
7F38- 4C F3 7C   4670           jmp oops
7F3B- 84 0F      4680 fnd2      sty *ln
7F3D- CA         4690           dex
7F3E- F0 02      4700           beq sfd2
7F40- B0 F1      4710           bcs cnt4
7F42- C8         4720 sfd2      iny
7F43- D1 1B      4730           cmp (r3+1),y
7F45- F0 05      4740           beq fne2
7F47- C4 1A      4750           cpy *r3
7F49- 90 F7      4760           bcc sfd2
7F4B- C8         4770           iny
7F4C- 88         4780 fne2      dey
7F4D- 98         4790           tya
7F4E- 38         4800           sec
7F4F- E5 0F      4810           sbc *ln
7F51- 85 11      4820           sta *ln2
7F53- E6 0F      4830           inc *ln
7F55- A5 0F      4840           lda *ln
7F57- 18         4850           clc
7F58- 65 1B      4860           adc *r3+1
7F5A- 85 1B      4870           sta *r3+1
7F5C- A5 1C      4880           lda *r3+2
7F5E- 69 00      4890           adc #$00
7F60- 85 1C      4900           sta *r3+2
7F62- A5 10      4910 fsh       lda *ln1         ;found shorter $
7F64- C5 11      4920           cmp *ln2         ;r(4)'s in ln1
7F66- F0 08      4930           beq eq
7F68- B0 0C      4940           bcs two<         ;2nd. shorter?
                 4950 ;which longer?
7F6A- 85 0F      4960           sta *ln          ;store least
7F6C- A2 01      4970           ldx #1           ;1st. shorter
7F6E- D0 0C      4980           bne begs         ;jump
7F70- 85 0F      4990 eq        sta *ln
7F72- A2 00      5000           ldx #0           ;same
7F74- F0 06      5010           beq begs         ;jump
7F76- A5 11      5020 two<      lda *ln2
7F78- 85 0F      5030           sta *ln
7F7A- A2 02      5040           ldx #2           ;2nd. shorter
                 5050 ;init. $ ctr.
7F7C- C9 00      5060 begs      cmp #0           ;ck. if $ is null
```

```
7F7E- F0 0D                  beq null
7F80- A0 00                  ldy #0
            ;cmpr next char.
7F82- B1 1B      nex         lda (r3+1),y
7F84- D1 1E                  cmp (r4+1),y
7F86- D0 24                  bne dif            ;char.s differ?
7F88- C8                     iny                ;no
            ;beyond last char.?
7F89- C4 0F                  cpy *ln
7F8B- 90 F5                  bcc nex            ;<len
            ;if so,which $ is longer?
7F8D- E0 01      null        cpx #1
7F8F- F0 09                  beq one<<          ;1st.?
7F91- 10 10                  bpl two<<          ;no, 2nd.?
            ;same
7F93- A9 00                  lda #0             ;no
7F95- 85 14                  sta *r1            ;1 rts below selected
7F97- 85 17                  sta *r2            ;from 4 options
7F99- 60                     rts
            ;one is <
7F9A- B1 1E      one<<       lda (r4+1),y
7F9C- 85 14                  sta *r1
7F9E- A9 00                  lda #0
7FA0- 85 17                  sta *r2
7FA2- 60                     rts
            ;two is <
7FA3- A9 00      two<<       lda #0
7FA5- 85 14                  sta *r1
7FA7- B1 1B                  lda (r3+1),y
7FA9- 85 17                  sta *r2
7FAB- 60                     rts
            ;found a difference
7FAC- 85 14      dif         sta *r1
7FAE- B1 1E                  lda (r4+1),y
7FB0- 85 17                  sta *r2
7FB2- 60                     rts
            ;conversion from # to address w/i pointer array s/r
7FB3- A5 1B      conv        lda *r3+1
7FB5- 85 1E                  sta *r4+1
7FB7- A5 1C                  lda *r3+2
7FB9- 85 1F                  sta *r4+2
7FBB- 18                     clc
7FBC- 26 1B                  rol *r3+1          ;double it
7FBE- 26 1C                  rol *r3+2
7FC0- A5 1B                  lda *r3+1          ;+t1=t*3
7FC2- 18                     clc
7FC3- 65 1E                  adc *r4+1
7FC5- 85 1B                  sta *r3+1
7FC7- A5 1C                  lda *r3+2
7FC9- 65 1F                  adc *r4+2
7FCB- 85 1C                  sta *r3+2
7FCD- A5 1B                  lda *r3+1          ;distance from ar. start
7FCF- 18                     clc
7FD0- 65 23                  adc *v1
7FD2- 85 15                  sta *r1+1          ;result in r(1)
7FD4- A5 1C                  lda *r3+2
7FD6- 65 24                  adc *v1+1
7FD8- 85 16                  sta *r1+2
7FDA- 60                     rts
7FDB-            cpg         .ds locs+1         ;save z.p. here
                             .en
```

Label File

```
aras =002C      arer =0080      arnm =0000
begs =7F7C      ckna =7C8A      cmpr =7EDF
cnt2 =7F2F      cnt3 =7EF8      cnt4 =7F33
cont =7EF4      conv =7FB3      cps =7FDB
dch =0009       dec2 =7D31      decl =7DB2
dflg =0003      dif =7FAC       dlim =0004
eara =002E      efnd =7CD6      ehfd =7CCE
eq =7F70        errp =C357      fara =7CEB
fdfl =0005      flc2 =7CAC      flch =7CB6
flds =0006      fnd2 =7F3B      fndd =7F00
fne2 =7F4C      fnef =7F11      fsh =7F62
fsiz =7CFB      heq =7DF8       him =0034
hok =7C6B       i =0007         ieqj =7DE0
j =0009         j<k =7E02       j>k =7E90
jeqk =7E5E      jeql =7DD8      k =000B
l =000D         ln =000F        ln1 =0010
ln2 =0011       locs =0024      lok =7C98
main =7D2D      n =0012         nare =7CDC
ndec =7D34      nex =7F82       nmfl =0002
notz =7EF2      null =7F8D      one<< =7F9A
oops =7CF3      r1 =0014        r2 =0017
r3 =001A        r4 =001D        s =0020
s<vj =7EAE      sart =7C61      sav =7C73
sfd2 =7F42      sfld =7F07      slop =7DA2
spg =7DA0       toj> =7DF5      two< =7F76
two<< =7FA3     v1 =0023        wrnm =7CA0

//0000,8000,8000
]
```

KIM Scorekeeper

Always on the lookout for new applications for the basic KIM-1, a general purpose, multi-player scorekeeper is presented. The techniques can be readily modified for use on a SYM-1 or AIM 65, and the scorekeeping function can be included as part of larger game programs.

Joel Swank
4655 SW 142nd, 186
Beaverton, OR 97005

Ever have a problem getting someone to keep score for your friendly game of Hearts? Well KIM would like to be a volunteer. KIM will keep up to nine separate scores for you to display and update from the keyboard. Each player can have from 0 to 9999 points, sufficient for most card games or other games needing a scorekeeper. Bridge fans can drop the low order zero from their scores (150 points for a grand slam??). I must credit the idea to a hardware project in October *Popular Electronics* by Joseph Fortuna. He used decade counters and 7-segment LED drivers to two-digit scores. A telephone dial was used to increment to counters. I immediately saw a job that KIM could do with software. Naturally with all the power of KIM available I had to improve and expand the idea.

The KIM SCOREKEEPER uses nine 2-byte memory registers to save the players' scores. Normally one of the players' scores is displayed continuously in the KIM display. The high order digit of the display is the player number, 1 to 9. The next digit is blank and the four low order digits contain that player's score. To display another player's score the PC (Player Change) key is pushed and the display goes blank. Then a number from 1 to 9 is pushed to get that player's score in the display. After a player is selected, the score can be updated. A player's score can be increased by entering the number to be added to the score and pushing the 'E' (Enter) key. Up to four digits can be entered. During entry of a number, the display shows the number being entered in the four low order digits with the two high order digits blank. Digits are shifted through the display as they are entered. If more that four digits are entered, the high order digits are shifted out and lost as in the KIM monitor.

The player's score can be decreased by pushing the 'D' (Decrease) key to set subtract mode. When the subtract mode is in effect, any number entered wil be subtracted from the player's score when the 'E' key is pushed. The high order digit of the display will show a minus sign when the number being entered is to be subtracted. Subtract mode stays in effect until the '+' key is pushed to reset the program to add mode. The '+' and 'D' keys are effective anytime except when performing the player change function. If any key except 0 to 9, '+' or 'D' is entered during the update operation the display returns to the current player. The 'C' (Clear) key may be used to zero the current player's score.

As shown by the programs, SCOREKEEPER has two main display loops. One displays the current player and his score while waiting for a command from the keyboard. The other displays the number being entered while inputting digits from the keyboard. The code is divided into subroutines for the sake of modularity and readability. The KIM subroutine GETKEY is used for communication from the keyboard, and the HEX to 7-segment conversion table in the KIM ROM is used to generate characters. The display is driven directly by the subroutine DISSEG. DISSEG is more flexible than the KIM subroutine SCANDS since it allows individual control of each segment of the KIM display. Thus any pattern can be displayed. DISSEG reads data from memory at SEGBUF and dumps it directly to the KIM display high order digit first. This subroutine could be used in a wide variety of games for KIM.

KIM SCOREKEEPER is an example of KIM's ability to replace and improve a hardware gadget. There is nothing I like more than finding a hardware function that KIM can replace with software. Someday I will calculate the weight of the hardware that my KIM has displaced.

```
                *****************************************
                *                                       *
                *           KIM SCOREKEEPER             *
                *      VERSION 1 SEPTEMBER 1979         *
                *                                       *
                *****************************************

0200           SCORER ORG    $0200

                ZERO PAGE STORAGE

0200           PLAYER *      $0080  PLAYER SCORE TABLE
0200           MODE   *      $0094  0=ADD ELSE SUBTRACT
0200           CURPLA *      $0095  INDEX TO CURRENT PLAYER
0200           CURKEY *      $0096  LAST KEY ENTERED
0200           TEMP   *      $0097  REGISTER SAVE AREA
0200           INDEX  *      $0098  REGISTER SAVE AREA
0200           SEGBUF *      $0099  DISPLAY BUFFER
0200           NUMBUF *      $009F  NUMBER INPUT BUFFER

0200           ZERO   *      $0000
```

```

                         LABELS

ØØØØ                     GETKEY  *      $1F6A  READ THE KEYBOARD
Ø2ØØ                     TABLE   *      $1FE7  HEX TO 7-SEG TABLE
Ø2ØØ                     PADD    *      $1741  DIRECTION REGISTER
Ø2ØØ                     SBD     *      $1742  DIGIT SELECT PORT
Ø2ØØ                     SAD     *      $174Ø  SEGMENTS OUTPUT PORT


                         ****************************************
                         *                                      *
                         *           INITIALIZATION             *
                         *                                      *
                         ****************************************
Ø2ØØ AØ Ø2                       LDYIM $Ø2
Ø2Ø2 A9 ØØ                       LDAIM ZERO   INIT ALL SCORES TO Ø
Ø2Ø4 99 8Ø ØØ            CLRLUP  STAAY PLAYER
Ø2Ø7 C8                          INY
Ø2Ø8 CØ 1A                       CPYIM $1A
Ø2ØA 9Ø F8                       BCC   CLRLUP
Ø2ØC 85 96                       STA   CURKEY INIT CURRENT KEY
Ø2ØE 85 94                       STA   MODE   SET MODE TO ADD
Ø21Ø 85 9A                       STA   SEGBUF +Ø1 SECOND DIGIT BLANK
Ø212 A9 Ø2                       LDAIM $Ø2    CURRENT PLAYER IS #1
Ø214 85 95                       STA   CURPLA

                         ****************************************
                         *                                      *
                         *              MAINLINE                *
                         *                                      *
                         ****************************************
Ø216 2Ø Ø6 Ø3            GETLUP  JSR   DISGET DISPLAY SCORE & GET KEY
Ø219 C5 96                       CMP   CURKEY DEBOUNCE
Ø21B FØ F9                       BEQ   GETLUP
Ø21D 85 96                       STA   CURKEY

Ø21F C9 12                       CMPIM $12    PLUS KEY?
Ø221 DØ Ø6                       BNE   NOPLUS NO
Ø223 A9 ØØ                       LDAIM ZERO   YES, SET ADD MODE
Ø225 85 94                       STA   MODE
Ø227 FØ ED                       BEQ   GETLUP

Ø229 C9 ØD               NOPLUS  CMPIM $ØD    D KEY?
Ø22B DØ Ø4                       BNE   NOMNUS NO
Ø22D 85 94                       STA   MODE   YES SET SUBTRACT MODE
Ø22F FØ E5                       BEQ   GETLUP

Ø231 C9 ØC               NOMNUS  CMPIM $ØC    C KEY?
Ø233 DØ Ø6                       BNE   NOCLR  NO
Ø235 2Ø 4F Ø2                    JSR   CLRCUR YES, CLEAR CURRENT PLAYER
Ø238 4C 16 Ø2                    JMP   GETLUP

Ø23B C9 14               NOCLR   CMPIM $14    PC KEY?
Ø23D DØ Ø6                       BNE   NOPC   NO
Ø23F 2Ø 59 Ø2                    JSR   UPPLAY YES, CHANGE CURRENT PLAYER
Ø242 4C 16 Ø2                    JMP   GETLUP

Ø245 C9 ØA               NOPC    CMPIM $ØA    NUMERIC KEY?
Ø247 BØ CD                       BCS   GETLUP NO, THEN IGNORE IT
Ø249 2Ø 6E Ø2                    JSR   UPDATE YES, UPDATE SCORE
Ø24C 4C 16 Ø2                    JMP   GETLUP


                         ****************************************
                         *                                      *
                         *  CLRCUR : CLEAR SCORE OF CURRENT     *
                         *                PLAYER                *
                         *                                      *
                         ****************************************
Ø24F A6 95               CLRCUR  LDX   CURPLA GET PLAYER INDEX
Ø251 A9 ØØ                       LDAIM ZERO
Ø253 95 8Ø                       STAZX PLAYER ZERO BOTH BYTES
Ø255 E8                          INX
Ø256 95 8Ø                       STAZX PLAYER OF SCORE
Ø258 6Ø                          RTS


                         ****************************************
                         *                                      *
                         *  UPPLAY : GET NUMBER OF NEW CURRENT  *
                         *           PLAYER FROM KEYBOARD       *
                         *                                      *
                         ****************************************
Ø259 2Ø 6A 1F            UPPLAY  JSR   GETKEY READ KEYBOARD
Ø25C C5 96                       CMP   CURKEY WAIT FOR LAST KEY TO LIFT
Ø25E FØ F9                       BEQ   UPPLAY
Ø26Ø 85 96                       STA   CURKEY
Ø262 C9 Ø1                       CMPIM $Ø1    IGNORE LESS THAN 1
Ø264 9Ø F3                       BCC   UPPLAY
Ø266 C9 ØA                       CMPIM $ØA    AND MORE THAN 9
Ø268 BØ EF                       BCS   UPPLAY
Ø26A ØA                          ASLA         MULT BY 2 FOR INDEX
Ø26B 85 95                       STA   CURPLA SAVE IT
Ø26D 6Ø                          RTS


                         ****************************************
                         *                                      *
                         *  UPDATE : INPUT NUMBER AND UPDATE    *
                         *           SCORE OF CURRENT PLAYER    *
                         *                                      *
                         ****************************************
Ø26E 85 97               UPDATE  STA   TEMP   SAVE FIRST KEY
Ø27Ø A9 ØØ                       LDAIM ZERO
Ø272 85 9F                       STA   NUMBUF CLEAR INPUT BUFFER
Ø274 85 AØ                       STA   NUMBUF +Ø1
Ø276 A5 97                       LDA   TEMP   RESTORE KEY

Ø278 2Ø 59 Ø3            UPLUP   JSR   SHFKEY SHIFT KEY INTO BUFFER
Ø27B 2Ø CB Ø2            UDLOOP  JSR   DISNUM DISPLAY BUFFER
Ø27E 2Ø 6A 1F                    JSR   GETKEY READ KEYBOARD
Ø281 C5 96                       CMP   CURKEY DON'T READ SAME KEY TWICE
Ø283 FØ F6                       BEQ   UDLOOP
Ø285 85 96                       STA   CURKEY
Ø287 C9 15                       CMPIM $15    IGNORE 'NO KEY'
Ø289 FØ FØ                       BEQ   UDLOOP

Ø28B C9 ØE                       CMPIM $ØE    ENTER KEY?
Ø28D FØ 17                       BEQ   ADDUM  YES GO CALCULATE
```

```
028F C9 12          CMPIM $12     PLUS KEY?
0291 D0 06          BNE   CKD     NO
0293 A9 00          LDAIM ZERO    YES, SET ADD MODE
0295 85 94          STA   MODE
0297 F0 E2          BEQ   UDLOOP

0299 C9 0D     CKD  CMPIM $0D     D KEY?
029B D0 04          BNE   CKNUM   NO
029D 85 94          STA   MODE    YES, SET SUBTRACT MODE
029F F0 DA          BEQ   UDLOOP

02A1 C9 0A   CKNUM  CMPIM $0A     NUMERIC KEY?
02A3 90 D3          BCC   UPLUP   YES, PUT IN BUFFER
02A5 60             RTS           NO, EXIT

02A6 A6 95   ADDUM  LDX   CURPLA GET CURRENT PLAYER
02A8 F8             SED           DECIMAL MODE FOR HUMANS
02A9 B5 80          LDAZX PLAYER GET LO BYTE OF SCORE
02AB A4 94          LDY   MODE    ADD OR SUBTRACT?
02AD D0 0E          BNE   SUBTRK SUBTRACT

              ADD BUFFER TO CURRENT SCORE

02AF 18             CLC
02B0 65 9F          ADC   NUMBUF ADD LO BYTE
02B2 95 80          STAZX PLAYER & SAVE
02B4 E8             INX
02B5 B5 80          LDAZX PLAYER GET HI BYTE
02B7 65 A0          ADC   NUMBUF +01 ADD HI BYTE OF #
02B9 95 80          STAZX PLAYER AND SAVE
02BB D8             CLD           BACK TO BINARY
02BC 60             RTS

              SUBTRACT BUFFER FROM CURRENT SCORE

02BD 38      SUBTRK SEC
02BE E5 9F          SBC   NUMBUF SUBTRACT LO BYTE OF #
02C0 95 80          STAZX PLAYER AND SAVE
02C2 E8             INX
02C3 B5 80          LDAZX PLAYER GET HI BYTE
02C5 E5 A0          SBC   NUMBUF +01 SUBTRACT HI BYTE OF #
02C7 95 80          STAZX PLAYER AND SAVE
02C9 D8             CLD           EACK TO BINARY
02CA 60             RTS

             ***************************************
             *                                     *
             *  DISNUM : DISPLAY NUMBER IN BUFFER  *
             *                                     *
             ***************************************
02CB A9 C0   DISNUM LDAIM $C0     MINUS SIGN
02CD A6 94          LDX   MODE    SUBTRACT MODE?
02CF D0 02          BNE   DISMIN YES
02D1 A9 00          LDAIM ZERO    NO, BLANK FIRST DIGIT
02D3 85 99   DISMIN STA   SEGBUF
02D5 A2 02          LDXIM $02     START AT 3RD DIGIT
02D7 86 98          STX   INDEX
02D9 A0 01          LDYIM $01     TWICE THRU
02DB B9 9F 00 NUMLUP LDAAY NUMBUF CET A BYTE
02DE 20 E7 02       JSR   CVTSEG CONVERT TO SEGS IN BUFFER
02E1 88             DEY
02E2 F0 F7          BEQ   NUMLUP QUIT AFTER 2
02E4 4C 29 03       JMP   DISSEG CO DISPLAY


             ***************************************
             *                                     *
             *  CVTSEG : CONVERT 1 BYTE INTO TWO   *
             *           HEX DIGITS IN SEGBUF      *
             *                                     *
             ***************************************
02E7 85 97   CVTSEG STA   TEMP    SAVE BYTE
02E9 4A             LSRA
02EA 4A             LSRA          MOVE HI NYBBLE TO
02EB 4A             LSRA          LOW NYBBLE
02EC 4A             LSRA
02ED AA             TAX           USE AS INDEX AND
02EE BD E7 1F       LDAAX TABLE   LOAD SEGMENT CODE
02F1 A6 98          LDX   INDEX   CET BUFFER INDEX
02F3 95 99          STAZX SEGBUF  AND STORE SEGMENT CODE
02F5 E6 98          INC   INDEX   NEXT BUFFER POSITION
02F7 A5 97          LDA   TEMP    RESTORE BYTE
02F9 29 0F          ANDIM $0F     CLEAR HI NYBBLE
02FB AA             TAX           USE LO NYBBLE AS INDEX
02FC BD E7 1F       LDAAX TABLE   LOAD SEGMENT CODE
02FF A6 98          LDX   INDEX   CET BUFFER INDEX
0301 95 99          STAZX SEGBUF  SAVE IN BUFFER
0303 E6 98          INC   INDEX   NEXT BUFFER POSITION
0305 60             RTS


             ***************************************
             *                                     *
             *  DISGET : DISPLAY CURRENT PLAYER AND *
             *           READ KEYBOARD             *
             *                                     *
             ***************************************
0306 A5 95   DISGET LDA   CURPLA  GET PLAYEQR INDEX
0308 4A             LSRA          DIVIDE BY 2 TO GET PLAYER #
0309 AA             TAX           USE AS INDEX
030A BD E7 1F       LDAAX TABLE   LOAD SEMENT CODE
030D 85 99          STA   SEGBUF  SAVE IN DISPLAY BUFFER
030F A9 02          LDAIM $02     THIRD DIGIT
0311 85 98          STA   INDEX   INIT BUFFER INDEX
0313 A4 95          LDY   CURPLA  PLAYER INDEX
0315 C8             INY           HI BYTE
0316 B9 80 00       LDAAY PLAYER  CET BYTE OF SCORE
0319 20 E7 02       JSR   CVTSEG  CONVERT TO SEGMENT CODES
031C 88             DEY
031D B9 80 00       LDAAY PLAYER  LO BYTE
0320 20 E7 02       JSR   CVTSEC  CONVERT TO SEGMENTS
0323 20 29 03       JSR   DISSEG  DISPLAY BUFFER
0326 4C 6A 1F       JMP   GETKEY  GO READ KEYBOARD


             ***************************************
             *                                     *
             *  DISSEG : LITE KIM DISPLAY WITH     *
             *           CODES FROM SEGBUF         *
             *                                     *
             ***************************************
```

```
0329 A9 7F      DISSEG LDAIM $7F     SET 6530 TO
032B 8D 41 17          STA   PADD    OUTPUT
032E A0 09             LDYIM $09     SELECT DIGIT 1 FIRST
0330 A9 00             LDAIM ZERO
0332 85 98             STA   INDEX   CLEAR BUFFER INDEX
0334 A6 98      DISLUP LDX   INDEX   GET BUFFER INDEX
0336 B5 99             LDAZX SEGBUF  GET A DIGIT FROM BUFFER
0338 A2 00             LDXIM ZERO
033A 8E 40 17          STX   SAD     CLEAR DISPLAY
033D 8C 42 17          STY   SBD     SELECT DIGIT
0340 8D 40 17          STA   SAD     LITE DIGIT
0343 A2 7F             LDXIM $7F
0345 CA         WAIT   DEX           LEAVE IT ON FOR A WHILE
0346 D0 FD             BNE   WAIT
0348 E6 98             INC   INDEX   NEXT BUFFER POSION
034A C8                INY
034B C8                INY           SELECT NEXT DIGIT
034C C0 15             CPYIM $15     DUN YET?
034E 90 E4             BCC   DISLUP  NOPE
0350 A9 00             LDAIM ZERO
0352 8D 42 17          STA   SBD     TURN OFF SEGS
0355 8D 41 17          STA   PADD    TURN OFF 6530
0358 60                RTS


                ***************************************
                *                                     *
                *  SHFKEY : SHIFT KEY INTO NUMBUF     *
                *                                     *
                ***************************************
0359 0A         SHFKEY ASLA
035A 0A                ASLA          MOVE KEY TO
035B 0A                ASLA          HI NYBBLE
035C 0A                ASLA
035D A2 04             LDXIM $04     SHIFT 4 BITS
035F 2A         SHFLUP ROLA          FROM ACCUM INTO
0360 26 9F             ROL   NUMBUF  NUMBER BUFFER
0362 26 A0             ROL   NUMBUF +01
0364 CA                DEX
0365 D0 F8             BNE   SHFLUP
0367 60                RTS
```

Symbol Table

ADDUM	02A6	CKD	0299	CKNUM	02A1	CLRCUR	024F
CLRLUP	0204	CURKEY	0096	CURPLA	0095	CVTSEG	02E7
DISGET	0306	DISLUP	0334	DISMIN	02D3	DISNUM	02CB
DISSEG	0329	GETKEY	1F6A	GETLUP	0216	INDEX	0098
MODE	0094	NOCLR	023B	NOMNUS	0231	NOPC	0245
NOPLUS	0229	NUMBUF	009F	NUMLUP	02DB	PADD	1741
PLAYER	0080	SAD	1740	SBD	1742	SCORER	0200
SEGBUF	0099	SHFKEY	0359	SHFLUP	035F	SUBTRK	02BD
TABLE	1FE7	TEMP	0097	UDLOOP	027B	UPDATE	026E
UPLUP	0278	UPPLAY	0259	WAIT	0345	ZERO	0000

April Fools On Us

We fear that a second class mail bag full of issue 21 may have been lost by the US Postal Service. If you live in the Arkansas, Louisiana, or Georgia area and did not receive your copy of MICRO, 21, please let us know.

NOW YOUR APPLE II CAN PERFORM JUST LIKE THE BIG BOYS

If you're a businessman who demands ultimate performance from your Apple II, then take a look at this outstanding General Ledger Package from Small Business Computer Systems (SBCS).

It features

- 6 digit account numbers
- 31 character account name.
- Ten levels of subtotals — giving you a more detailed income statement and balance sheet.
- Departmentalizing . . . up to nine departments.
- Flexibility — adaptable to any printer and either cash or accural accounting methods.
- Cash Journal allows a 33 character transaction description and automatically generates the appropriate offsetting entry.
- You can print the balance sheet and income statement for the current month, current quarter, or any of the previous three quarters. This year's or last year's totals are also included on the income statement. Or a special report that lists the current account balance for selected accounts.
- Higher number of entries from an external source — as many as 1,000 per session.
- No limit on entries — giving you the opportunity to make your entries as many times or as often as you want.
- With high speed printer routines and other special features of our conversion, processing performance does not decrease dramatically at the system limits.
- Look at these examples of times required to update the chart and print the audit trail.
  With 133 item chart of accounts, 700 postings into 70 regular accounts: less than 20 min.
  With 133 item chart of accounts, 1000 postings into 70 regular accounts: less than 30 min.
  With 210 item chart of accounts, 1000 postings into 125 regular accounts: less than 40 min.
- Coming early this year — capability to archive up to 2,500 postings. The chart of accounts will also be archived to maintain the opening balance for the archive period.

In the final analysis, your financial statements are what this General Ledger is all about. And with this General Ledger Package you can format your own balance sheet and income statement. As well, department financial statements may be formated differently. You have complete freedom to place titles and headings where you want them, skip lines or pages between accounts and generate subtotals and totals throughout the reports — up to ten levels if you need them.

And coming early in 1980, SBCS will present the Accounts Payable/Accounts Receivable Package you have been waiting for.

Just compare these numbers against any package on the market today:*

	5 inch disc	8 inch disc
Vendors or customers	700	1,800
Payable Transactions	350	750
Payable Invoices	380	840
Receivable Transactions	600	1,300
Receivable Invoices	600	1,300

* These are maximum numbers that you can put on a disc if you're using the disc only for these respective data files.

We are an authorized converter for Osborne/McGraw-Hill, providing you with business packages that will do everything the Osborne General Ledger will do in addition to many features we have added.

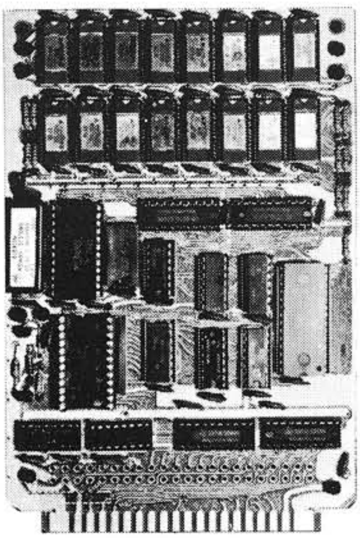
Call or write:

Small Business Computer Systems
4140 Greenwood
Lincoln, Nebraska 68504
(402) 467-1878

KIM/SYM/AIM-65—32K EXPANDABLE RAM

DYNAMIC RAM WITH ON BOARD TRANSPARANT REFRESH THAT IS COMPATIBLE WITH KIM/SYM/AIM-65 AND OTHER 6502 BASED MICROCOMPUTERS.

- PLUG COMPATIBLE WITH KIM/SYM/AIM-65. MAY BE CONNECTED TO PET USING ADAPTOR CABLE. SS44-E BUS EDGE CONNECTOR.
- USES +5V ONLY (SUPPLIED FROM HOST COMPUTER BUS). 4 WATTS MAXIMUM.
- BOARD ADDRESSABLE IN 4K BYTE BLOCKS WHICH CAN BE INDEPENDENTLY PLACED ON 4K BYTE BOUNDARIES ANYWHERE IN A 64K BYTE ADDRESS SPACE.
- ASSEMBLED AND TESTED BOARDS ARE GUARANTEED FOR ONE YEAR, AND PURCHASE PRICE IS FULLY REFUNDABLE IF BOARD IS RETURNED UNDAMAGED WITHIN 14 DAYS.
- BUS BUFFERED WITH 1 LS TTL LOAD.
- 200NSEC 4116 RAMS.
- FULL DOCUMENTATION



ASSEMBLED/TESTED:
- WITH 32K RAM $419.00
- WITH 16K RAM $349.00
- WITHOUT RAM CHIPS $279.00

HARD TO GET PARTS ONLY (NO RAM CHIPS) .. $109.00
BARE BOARD AND MANUAL $49.00

PET INTERFACE KIT $49.00

CONNECTS THE ABOVE 32K EXPANDABLE RAM TO A 4K OR 8K PET. CONTAINS EXPANSION INTERFACE CABLE, BOARD STANDOFFS, POWER SUPPLY MODIFICATION KIT AND COMPLETE INSTRUCTIONS.

6502, 64K BYTE RAM AND CONTROLLER SET

MAKE 64K BYTE MEMORY FOR YOUR 6800 OR 6502. THIS CHIP SET INCLUDES:
- 32 MSK 4116-3 16KX1, 200 NSEC RAMS.
- 1 MC3480 MEMORY CONTROLLER.
- 1 MC3242A MEMORY ADDRESS MULTIPLEXER AND COUNTER.
- DATA AND APPLICATION SHEETS. PARTS TESTED AND GUARANTEED.

$295.00 PER SET

16K X 1 DYNAMIC RAM

THE MK4116-3 IS A 16,384 BIT HIGH SPEED NMOS, DYNAMIC RAM. THEY ARE EQUIVALENT TO THE MOSTEK, TEXAS INSTRUMENTS, OR MOTOROLA 4116-3.
- 200 NSEC ACCESS TIME, 375 NSEC CYCLE TIME.
- 16 PIN TTL COMPATIBLE.
- BURNED IN AND FULLY TESTED.
- PARTS REPLACEMENT GUARANTEED FOR ONE YEAR.

$8.50 EACH IN QUANTITIES OF 8

BETA COMPUTER DEVICES
1230 W. COLLINS AVE.
ORANGE, CA 92668
(714) 633-7280

CALIF RESIDENTS PLEASE ADD 6% SALES TAX. MASTERCHARGE & VISA ACCEPTED. PLEASE ALLOW 14 DAYS FOR CHECKS TO CLEAR BANK. PHONE ORDERS WELCOME.

ALL ASSEMBLED BOARDS AND MEMORY CHIPS CARRY A FULL ONE YEAR REPLACEMENT WARANTY.

ACTION, STRATEGY, AND FANTASY
for the SERIOUS games player and his APPLE II

Brain Games - 1 demands ingenuity.

Two players bombard radioactive material with protons and electrons until it reaches critical mass and sets up a **Nuclear Reaction. Dodgem** *requires you to outmaneuver another player to get your pieces across the board first.* **Dueling Digits** *and* **Parrot** *challenges your ability to replicate number and letter sequences.* **Tones** *lets you make music with your Apple (16K) CS-4004 $7.95.* **Strategy Games** *and* **Brain Games** *are on one disk (16K) CS-4503 $14.95.*

THINK

Strategy Games - 1 keeps games players in suspense.

You and your opponent trail around the screen at a quickening pace attempting to trap each other in your **Blockade.** *A 7 category quiz game will certify you as a* **Genius** *(or an errant knave!). Beginners will meet their master in* **Checkers. Skunk** *and* **UFO** *complete this classic collection (16K) CS-4003 $7.95*

Know Yourself through these valid self-tests.

Find out how your life style effects your **Life Expectancy** *or explore the effects of* **Alcohol** *on your behavior.* **Sex Role** *helps you to examine your behavior and attitudes in light of society's concept of sex roles.* **Psychotherapy** *compares your feelings, actions, and phobias to the population's norms and* **Computer Literacy** *tests your microcomputer savvy. A fun and instructional package (16K) CS-4301 $7.95.* **Know Yourself** *and* **CAI Programs** *are on one disk (16K) CS-4503 for $14.95*

IMAGINE

You're in command in Space Games - 1.

Maneuver the TIE fighters into your blaster sights and zap them with your lasers to save the rebel base camp from annihilation in **Star Wars. Rocket Pilot** *is an advanced real time take off and landing game. High resolution graphics, exploding saucers and sound effects add to the suspense as you repel the* **Saucer Invasion.** *Finally, a bonus graphics demonstration,* **Dynamic Bouncer** *(16K) CS-4001 $7.95.* **Space Games** *and* **Sports Games** *are on one disk (16K) CS-4501 for $14.95*

ACTION

Sports Games - 1 puts you in the Apple World Series

Take the field in the **Great American Computer Game.** *Mix up your pitches to keep the batter off balance. Move your fielders to snag the ball before he gets to first. Balls and strikes, double plays, force outs, and errors let you play with a realistic strategy. Also in the line up—* ***Slalom,*** *a championship downhill ski race,* **Torpedo Alley,** *and* **Darts** *(16K) CS-4002 $7.95.* **Space Games** *and* **Sports Games** *are on one disk (16K) CS-4501 for $14.95*

It's easy to order SENSATIONAL SOFTWARE for your Apple II.

Send payment plus $1.00 shipping and handling in the U.S. ($2.00 foreign) to Creative Computing Software, P.O. Box 789-M, Morristown, N.J. 07960. N.J. residents add $1.00 sales tax. Visa, Master Charge and American Express orders may be called in toll free to **800-631-8112** (in N.J. 201-540-0445).

OSI BASIC in ROM

While the various Microsoft BASICs are easy to use, they are difficult to understand due to an intentional lack of documentation. To help understand your OSI BASIC, a table of the locations of the subroutines to service the main commands is presented. The program which generated the table is provided as a starting point for you to explore your BASIC.

E.D. Morris, Jr.
3200 Washington
Midland, MI 48640

A previous article in Micro 18:9 by S.R. Murphy gave a peek into OSI BASIC in ROM by listing a number of scratch pad locations in page zero. In the present article, I wish to delve further into the inner workings of BASIC by explaining the dispatch table.

At the bottom of the BASIC ROMs, between $A000 and $A083, is a list of addresses known as the dispatch table. These are the starting addresses of all the machine subroutines needed to carry out the BASIC keywords such as END, FOR, NEXT etc. The addresses are in hexidecimal in the normal machine format of low byte first followed by the high order byte. For example, starting at $A000 you find the data:

$A000	39
$A001	A6
$A002	55
$A003	A5

Thus the first two entries in the dispatch table are $A639 and $A555. These point to subroutines in the BASIC ROMs.

Now we need to know what each subroutine does. Conviently there is another table starting at $A084 containing a list of all the BASIC keywords. The first entries in this table are:

$A084	45
$A085	4E
$A086	C4
$A087	46
$A088	4F
$A089	D2

Except for the C4 and D2, the data looks like ASCII code. If the high order bit is removed from C4 and D2, then it is ASCII code for ENDFOR. You can demonstrate the list of keywords for yourself by running the program:

```
10  FOR X = 41092 TO 41315
20  Y = PEEK (X)
30  PRINT CHR$ (Y);
40  NEXT
```

If you have the OSI graphics character generator, the last letter of each word will be a graphics character instead of a letter. The high bit being set is used to separate the entries in the word list. To convert these to letters and leave a space between key words, add the following line to the above program:

```
25  IF  Y > 127  THEN  PRINT
CHR$(Y—128);:Y = 32
```

Now we have two lists, one of addresses and one of functions. These can be combined to give an address for each function.

END	$A639
FOR	$A555

However things are not quite that simple. Unfortunately the two tables are not strictly in the same order. Also some of the address entries refer to the subroutine location and others to the location, less one. The address table is further complicated in the case of the arithmetic operators by a third entry which is the precedence value.

Following is a BASIC program that sorts out these quirks and outputs a list of BASIC KEYWORDS together with the hex address of the machine code associated with that keyword. Notice that the program does not contain data statements, rather PEEK's directly at your BASIC ROM's. The program steps through the dispatch table printing out each address. The value of Q is added to each address and is either 1 or 0. The correct keyword is found by PEEKing at D until a character is found with the high bit set.

The subroutine at line 500 converts a binary word into ASCII digits for printing.

For those of you who have trouble with this program or for those who have a sore index finger from typing in that 24K game program, I am providing an output listing. However I urge you to run it yourself to prove all this stuff is really "in there." The BASIC program also contains information about the location and structure of the two tables.

Looking at the sample run, the addresses for END and FOR found earlier, are incorrect by one byte. Users of the USR function know that the subroutine address must be placed at $000B and $000C. The dispatch table associates location $000A with the USR function. Location $000A contains 4C or JMP which completes the three byte instruction.

It is interesting to note that the BASIC keyword table is identical to a numerical listing of the BASIC tokens(MICRO 15:20). The keywords TAB, TO, THEN, and STEP are missing from the dispatch table. However these commands are never used alone but always occur with another BASIC keyword (PRINT, FOR, IF and FOR-NEXT). The purists will note the absence of AND, OR, GREATER, LESS and EQUALS. I must confess, these did not fit neatly into my BASIC program.

If you have ever tried to make sense of "that 8K block of data up there at $A000," it looked like a hopeless task. With the dispatch table at hand, you can break it down and attack one function at a time. These subroutines are available to use if you are into machine code programing. Mr. Murphy is wrong: OSI users are not disinclined to explore their machines. The problem, until now, has been that too little information was available. So let's dig into OSI's BASIC and publish a complete memory map similar to those already out for the PET and APPLE.

Sample Run
(Program output listing)

A63A	END
A556	FOR
AA40	NEXT
A70C	DATA
A923	INPUT
AD01	DIM
A94F	READ
A7B9	LET
A6B9	GOTO
A691	RUN
A73C	IF
A61A	RESTORE
A69C	GOSUB
A6E6	RETURN
A74F	REM
A638	STOP
A75F	ON
A67B	NULL
B432	WAIT
FFF4	LOAD
FFF7	SAVE
AFDE	DEF
B429	POKE
A82F	PRINT
A661	CONT
A4B5	LIST
A68C	CLEAR
A461	NEW
B7D8	SGN
B862	INT
B7F5	ABS
000A	USR
AFAD	FRE
AFCE	POS
BAAC	SQR
BBC0	RND
B5BD	LOG
BB1B	EXP
BBFC	COS
BC03	SIN
BC4C	TAN
BC99	ATN
B41E	PEEK
B38C	LEN
B08C	STR$
B3BD	VAL
B39B	ASC
B2FC	CHR$
B310	LEFT$
B33C	RIGHT$
B347	MID$
B46F	+
B458	−
B5FE	*
B6CD	/
BAB6	↑

BASIC Program

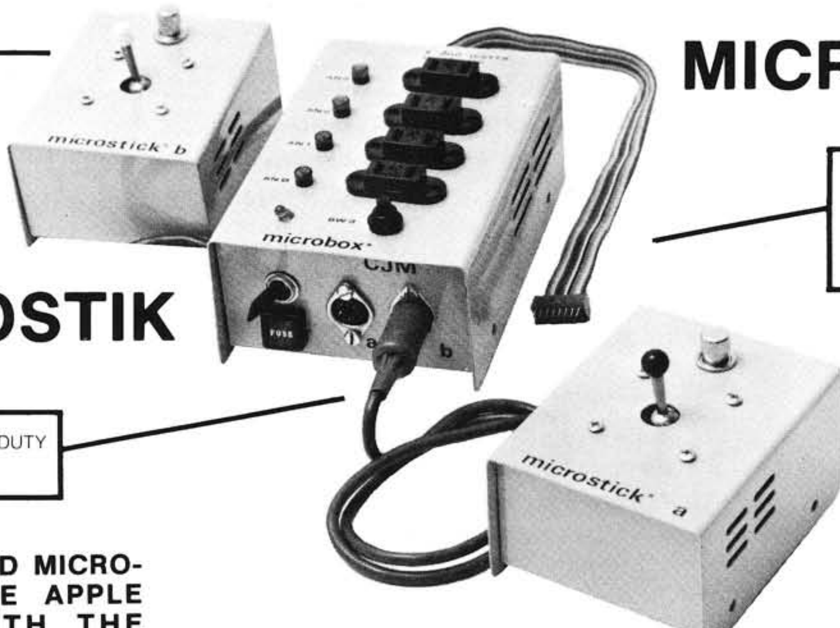
```
10 Q=1:D=41092
20 FOR C=40960 TO 41060 STEP 2
25 IF C=41016 THEN Q=0:D=41237
30 X=PEEK(C+1):GOSUB 500
40 X=Q+PEEK(C):GOSUB 500
50 PRINT"  ";
60 X=PEEK(D)
70 D=D+1
80 IF X<128 THEN PRINT CHR$(X);:GOTO60
90 X=X-128
100 PRINTCHR$(X)
110 NEXT C
115 D=41224
120 FOR C=41062 TO 41074 STEP 3
130 X=PEEK(C+2):GOSUB 500
140 X=1+PEEK(C+1):GOSUB 500
150 PRINT"  ";
160 X=PEEK(D)
170 D=D+1
180 IF X<128 THEN PRINT CHR$(X);:GOTO 160
190 X=X-128
200 PRINT CHR$(X)
210 NEXT C
220 END

500 REM PRINT SUB
510 H=INT(X/16)
520 L=X-16*H
530 IF H<10 THEN H=H+48:GOTO 550
540 H=H+55
550 IF L<10 THEN L=L+48:GOTO 570
560 L=L+55
570 PRINT CHR$(H);CHR$(L);
580 RETURN
```

Now You Can Have
INPUT/OUTPUT
For Your Apple Computer

DUAL AXIS JOYSTICK AND PUSHBUTTON MODULE for Games, Graphics, and Experimental Program Input

MICROBOX

CONNECTS DIRECTLY to the Apple Game Socket. ACCEPTS 2 Dual Axis MICROSTIKS or 4 Paddles



MICROSTIK

ISOLATED SWITCHING of 4 AC loads or relays from a basic program. 4 LED status indicators. Toggle Switch input (sw3)

EXTRA LONG, HEAVY DUTY cables and connectors

MICROBOX AND MICROSTIK PROVIDE APPLE OWNERS WITH THE HARDWARE TO EXPLORE THE INPUT/OUTPUT CAPABILITIES OF THEIR COMPUTERS.

A SIMPLE COMMAND from the Apple keyboard or a Basic Program can switch an external device. Connect AC loads, such as lamps, motors, relays or solenoids directly through the MICROBOX's 4 AC OUTLETS. Loads can range from 0 to 220VAC and draw up to 200 Watts each. Solid State Switching ISOLATES the load from your Apple for complete safety. Four LEDs provide a visual on/off status of each load.

A Complete Instruction/Tutorial Manual is included with the MICROBOX.

CJM

INDUSTRIES

REAL-TIME INPUT

The MICROSTIK is a sturdy, two axis joystick. Metal Cable Connectors assure trouble free usage over time, and enable extension cables to be added easily. Use the MICROSTIK to add real-time input to your game, graphic or experimental programs. Each MICROSTIK contains a PUSHBUTTON for added input possibilities.

MICROBOX and MICROSTIK sit comfortably on, or aside the Apple Computer. They have been designed to match the Apple in color and design.

ORDER TODAY AND CONNECT YOUR APPLE TO THE OUTSIDE WORLD.

The MICROBOX and MICROSTIK can be purchased at most computer stores or can be ordered directly by mail or through our convenient 24 hour telephone service.

TELEPHONE:
(703) 471-4291

Order the MICROSET and receive the MICROBOX, 2 MICROSTIKS, the Manual and Cassette, and SAVE $25.

MICROBOX	**$109.95**
MICROSTIK (each)	**34.95**
Demo Cassette	**9.95**
MICROSET	**164.95**
12 ft. Ext. Cables	**6.95**
Relay Modules	**WRITE**
Solenoid Modules	**WRITE**

Va. residents add 4% sales tax

MASTER CHARGE, VISA accepted
NO C.O.D.s

CJM Industries, Dept. MB
316B Victory Drive
Herndon Industr Park
Herndon, Va. 22070

Dear MICRO magazine,

My Dad and I have had an APPLE II for about 9 months. During this time I have learned of the special joys and sorrows that only computer people can appreciate or experience. This poem was born out of long hours at the keyboard. I hope you like it and feel that it is worth publishing.

Ode to My Disk

I always see verses praising the Apple
But who sees the time saved by Disk II, it's ample?

While Apple sits waiting to digest the data
That trusty old "breadbox" spins round without-
breakdown.

It hasn't been long since I've bought my Disk II
But I know it's worth it and so do you.

I tried Panasonics and Hitachis too
Resetting and loading my Apple I'd do.

Frustrating it was and my hair I did pull
So soon I did tire of ERR MEM FULL.

So now my Hitachi sits dusty and wan
And softly clicks Disk II, no ERR coming on.

Donna Marie Andert
Connelly High School
Anaheim, CA 92801

Dear Editor,

Most articles that are submitted to MICRO are claimed by their authors to execute correctly. The following program has been extensively de-bugged and is guaranteed to run ***neither*** on a PET ***nor*** on an OSI microcomputer.

```
  10 FOR X=1 TO 10
  20 IF X=5 THEN 40
  30 NEXT X
  40 REM
  100 FOR Y=1 TO 10
  110 FOR X⁵1 TO 10
  120 NEXT X
  130 NEXT Y
READY.
```

Can you figure out what is wrong here? If not, the answer is given in the next column

E.D. Morris, Jr
Midland, MI 48640

This program was originally part of a 200 line game program with a "small bug." Through a bit of detective work, I narrowed the bug down to these eight lines. In the original game, these lines occurred in widely different sections of the program and appeared not to be related. When the program is executed, the computer will halt indicating "NEXT WITHOUT ERROR IN LINE 130".

This message is most confusing since line 100 clearly contains a "FOR Y". The program will run if lines 100 and 130 are deleted. Something appears to be wrong with the "Y" loop. If "X" is made the outer loop and "Y" is the nested loop, the program will run without error.

This is all a wild goose chase! Nothing is wrong with the "Y" loop. The first real hint of the cause is that replacing the variable "X" in lines 110 and 120 with a different variable, say "Z", solves the problem. The real culprit is line 20 where the program jumps out of a loop before finishing it. It is simple to see here in an eight line program, but not so obvious in a large program. The problem occurs when a variable from an unclosed loop is used again in a nested loop.

The moral of the story is to close loops whenever possible. For example, line 20 could have been:

```
20 IF X=5 THEN Z=X : X=10 : GOTO 30
```

If you can't close the loop, at least avoid using that variable in another loop.

And here is another poem from a reader, sent to us in May, 1979. We hope that he remembered to renew his subscription.

End of Subscription

There once was a town, Albuquerque,
Wherein lived a genuine turkey
Who, on learning his MICRO had died,
Lost what little was left of his pride.

Hadn't realized how close was the end.
Still, he took out his pencil and penn'd
"Mr. Tripp, won't you give me a chance?
My check will disprove miscreance."

Nelson E. Ingersoll
Albuquerque, NM 87110

We at MICRO would like to thank Donna, Earl, Nelson and all of our readers for their contributions. While all of the letters that we get are not as entertaining or as fun as these, they all certainly give us some things to think about. We welcome reader input and we encourage you to write to us with your comments, and suggestions at any time. We hope to run the Letterbox column in every issue, but it all depends on what we get from you.

The MICRO Staff

Apple-Doc

By Roger Wagner

An Aid to the Development and Documentation of Applesoft Programs

This 3 program set is a must to anyone writing or using programs in Applesoft! It not only provides valuable info. on each of your programs, but allows you to change any element throughout the listing almost as easily as you would change a single line!!

With Apple-Doc you can produce a list of every variable in your program and the lines each is used on, each line called by a GOTO, GOSUB, etc., in fact, every occurance of almost anything!

You can rename variables, change constants and referenced line #'s, or do local or global replacement editing on your listing.

In fact, we guarantee that after purchase, if you don't feel APPLE-DOC is one of the most valuable programs in your library we will even refund your money! (Upon return of product.)

Unheard of? Yes! But that's how good APPLE-DOC really is!

That's not all!! Send for free info. or visit your nearest Apple dealer.

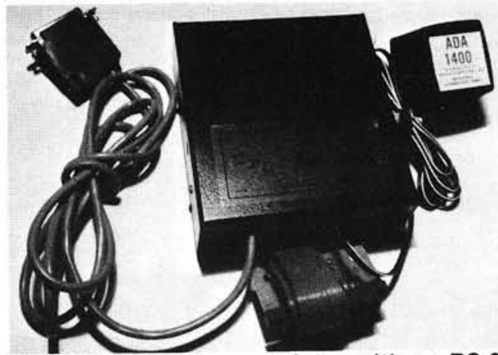
Only $24.95 **Please specify diskette or tape.**
(Calif. residents add 6% Sales Tax)

Available from your local computer store or:

Southwestern Data Systems
P.O. Box 582-M
Santee, CA 92071
(714) 562-3670

(Dealer inquiries invited)

PET Printer Adapter



CmC's ADA 1400 drives a printer with an RS-232 interface from the Commodore PET IEEE-488 bus. The ADA 1400 is addressable, works with the Commodore disk and prints upper and lower case ASCII.

A PET IEEE type port is provided for daisy-chaining other devices.

A cassette tape is included with programs for plot routines, data formatting and screen dumps. The ADA 1400 sells for $179.00 and includes a PET IEEE cable, RS-232 cable, power supply, case, instructions and software.

Order direct or contact your local computer store.

VISA AND M/C ACCEPTED-SEND ACCOUNT NUMBER, EXPIRATION DATE AND SIGN ORDER.
ADD $3 PER ORDER FOR SHIPPING & HANDLING - FOREIGN ORDERS ADD 10% FOR AIR POSTAGE.

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659 TWX: 710-456-0052

STONEWARE for APPLE II ⁂

For the Serious Business or Home User:
MICRO MEMO

MICRO MEMO is the first sophisticated "Desk Calendar" program to make good use of your computer's power.

- ★ Micro Memo includes one time, weekly, monthly, semi-annual and annual reminders.
- ★ Monthly reminders may be for fixed or "floating" dates (ex. 1st Saturday of every month).
- ★ Each reminder allows choice of one week, 2 week or 1 month advance notice—reminds you ahead of time to prepare for meetings, purchase tickets, make reservations, etc.
- ★ Micro Memo includes "shorthand" for fast memo entry, greater capacity.
- ★ Micro Memo will display or print any day's or week's reminders.
- ★ Micro Memo is a "perpetual" calendar—automatically creates new months with all appropriate memos (birthdays, anniversaries, monthly meetings, etc.) as past months are dropped—system holds full year's reminders on one disk.
- ★ Micro Memo "knows" most major holidays.
- ★ Supports Mountain Hardware clock (optional).
- ★ "Bomb Proof" menu driven command and data entry.
- ★ Requires 48K, disk, RAM or ROM Applesoft.

$39.95

STONEWARE
Microcomputer Software
P.O. Box 7218, Berkeley, CA 94707
(415) 548-3763

And Just for Fun:
TRANQUILITY BASE

$24.95

TRANQUILITY BASE is a fast high resolution Lunar Lander game by Bill Budge, creator of Apple's "Penny Arcade." TRANQUILITY BASE is just like the popular arcade game, including multiple moonscapes, craft rotation, and zoom in for a close-up view as you approach the Lunar surface.

TRANQUILITY BASE requires 32K and disk.

Available at your favorite computer store or direct from STONEWARE (add $2 shipping & handling; Calif. residents add sales tax. Visa & MasterCharge accepted, no C.O.D.'s).

DEALER INQUIRIES INVITED

⁂ Apple II is a Trademark of Apple Computer, Inc.

The MICRO Software Catalogue: XIX

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

Name:	**Dakin 5 Programming Aids II**
System:	**Apple II**
Memory:	**48K**
Language:	**Assembler/Applesoft II**
Hardware:	**Apple II, 2 Disk II's, and Printer**

Description: Set of seven programs: 1) *Copier*- copies absolutely any kind of file or program from one diskette to another. 2) *Variable Cross Reference*- creates a cross-reference for all variable neames used in an Applesoft BASIC program, showing all line numbers where a given variable name is used. 3) *Line Cross Reference*- creates a cross-reference for all referenced lines in an Applesoft BASIC program, showing where a given line is referenced by GOTO, GOSUB, THEN, or LIST statements. 4) *Patcher*- allows the user to display any sector of a given file or program, and then to update any data within that sector. A second option enables the user to specify the particular sector he wishes to update. 5) *Screen Printer* - permits contents of the screen to be sent to the printer at any time the keyboard is active. The progrm remains in effect until you press RESET or reboot the system. 6) *Array Editor*- a simple word processor that allows you to create, modify, print and save your own text files. 7) *Calculator II*- a multiplication/division subroutine that handles numeric string data. Written in Assembler code, and using twenty place accuracy, it runs much faster thatn an equivalent BASIC subroutine. It is also compatible with the addition/subtraction subroutine, the Calculator, included in the first Dakin5 Programming Aids package reviewed in the December 1979 issue of The MICRO Software Catalogue XV.

Copies:	**Just released**
Price:	**$49.95**
Includes:	Professionally bound documentation and program diskette.
Author:	**Dakin5 Corporation** (developer of The Controller for Apple Computer, Inc.)
Available:	Local Apple Dealers

Name:	**Page Format TTY IN/OUT**
System:	**Apple II**
Memory:	**300.3FF (256 Bytes)**
Language:	**Machine**
Hardware:	**Game Conn to TTY**

Description: Program to output to and input from ASK 33 or 35 Teletype. Gives multiple kine feeds at end of each page and waits for you to tear off roll paper or insert new sheet for neat listings. Uses game connector.

Copies:	**Just released**
Price:	**$2.00**
Includes:	Listing and Instructions
Author:	**Ken Ellis**
Available:	Ken Ellis R.D.8 Box 344 York, PA 17403

Name:	**HI-RES GRAPHIC CHARTS GENERATOR**
System:	**APPLE II, APPLE II PLUS**
Memory:	**32K without ROM card, 16k with card**
Language:	**APPLESOFT II BASIC**
Hardware	**APPLE II, Disk II (allows optional features)**

Description: This program will allow you to generate HI-RES graphic charts, either through keyboard or text file input (if using disk). 'Y' axes will be automatically scaled with values. 'X' axes will be marked for plotting points. Best of all, once graph is automatically created, you can add your own titles, comments, or symbols anywhere on the graph. Both upper and lower case characters are provided. Over 30 special symbols are included. Provisions are also included for multiple graph overlays. Disk II users can automatically have graphs made from existing data already stored.

Copies:	**Just releases, 42 copies already sold.**
Price:	**$19.95 + $1.25 for postage.**
Includes:	Cassette containing program, instructions on unique uses. Please specify when your order, if you have ROM card or not.
Author:	**Les Stubbs**
Available:	Les Stubbs 23725 Oakheath Pl. Harbor City, Ca. 90710

Name:	**General Ledger Version 2.0**
System:	**Apple II**
Memory:	**48K**
Language:	**Applesoft**
Hardware:	**Dual Drives, Any Printer**

Description: General Ledger Version 2.0 — This program is a complete double-entry accounting system. User defined flexibility allowing up to 9 individualized departments in all Financial Reports. 10 levels of subtotals throughout each report gives more detailed Financial Statements. Using 5" drives, storing the entire Chart of Accounts and/or all posting approaches minicomputer times when verifying account numbers or sorting records. High-speed printer routines will process 1,000 postings into 70 accounts in less than 30 minutes. Using 8" drives, high-speed sorting routines requiring no additonal disk work space and fast binary searching techniques allow data files to be limited only by your available disk space. Compatible with any printer and printer interface.

Copies:	**Version 1.0, 200; Version 2.0, Just released.**
Price:	**$180.00**
Author:	**David A. McFarling**
Available:	Small Business Computer Systems 4140 Greenwood Lincoln, NE 68504

Name:	**VOCAB 1.1**
System:	**APPLE II or APPLE II PLUS**
Language:	**Applesoft**
Memory:	**32K**
Hardware:	**APPLE II and DISK II**

Description: A vocabulary builder with over 1200 multiple choice questions allows the user to select either synonyms or antonyms. Intended as study aid for college board type exams (e.g., SAT, ACT, GRE, LSAT, etc.). Editor is included for expanding or modifying data lists. Several test formats with grading are options. Ideal for students with little computer experience.

Price:	**$15.00**
Includes:	User documentation and diskette
Author:	**Steven M. Sliwa**
Available:	Sliwa Enterprises 257 C Clemwood Parkway Hampton, VA 23669

Name:	**SORT**
System:	**PET, APPLE**
Memory:	**32K/16K PET; any Apple**
Language:	**6502 Machine Language**
Hardware	**16K/32K PET, any APPLE**

Description: SORT is a 6502 machine language intelligent sort for commercial applications. Requires almost no user set-up when default values are used. Sorts integer, string and floating point arrays of more than one dimension with up to 20 sub-sorts-on-match (if needed).

Copies:	**Just released**
Price:	**depends on end use**
Author:	**David B. Black**
Available:	MATRIX SOFTWARE INC. 1041 N. Main St. Ann Arbor, MI 48104

Name:	**Investment Comparison**
System:	**Apple II or Apple II Plus**
Memory:	**32K with ROM Applesoft, 48K with RAM Applesoft**
Language:	**ROM Applesoft.**Can be used with RAM Applesoft by relocating above HGR2 display area or not using graphics display feature. 1024 bytes of Machine Code is loaded before Main program.
Hardware:	**Cassette tape. Program supports Printer but driver subroutine not included. Apple II.**

Description: We are often faced with decisions such as 'which of two investments is best?' This program provides a means of comparing them by the use of "Cash Discounting." Cash Discounting is a technique that is used to take into consideration the effects of inflation. Often we are faced with a decision of 'buying now' vs waiting a few years or paying cash vs time payments. The effects of inflation are not easy to quantify without some form of computer analysis. For each of two alternatives, entry include:
1. Inflation Rate for both
2. Initial Investment $
3. Number of years to salvage point and value at that time.
4. Monthly expenses (or income)
5. Adjustments on an annual basis for the monthly expenses. This provides a means whereby you can make expenses track at a different rate than inflation.
Display is in a form of a 'cash Flow' by year, and a graphical presentation is also provided. The graphics have labels.

Copies:	**Just released**
Price:	**$16.95**
Includes:	Cassette, loading instructions, description, and example.
Author:	**Neil A. Robin**
Available:	TECH-DIGIT 21 Canter Lane Sherwood, OR 97140

Name:	**The Life Dynamic Transformation Experience**
System:	**Apple II**
Memory:	**48K**
Language:	**Applesoft and Machine Language**
Hardware:	**Apple II Plus, Disk II**

Description: Unique! This program is designed for all those people who desire to experience self-transformation, life-awareness, making relationships work, and "getting your act together," but do NOT desire to pay est or Lifespring or any of the other "trips" of the Human Potential Movement, $300 or so. Includes game playing as a means to a fun way of increasing awareness.

Copies:	**Many**
Price:	**$15.95**
Includes:	(disk) w/instructions
Author:	**Avant-Garde Creations**
Available:	Avant-Garde Creations P.O.Box 30161 Dept. MC Eugene, OR 97403

Name:	**I CHING**
System:	**Apple II or Apple II Plus**
Memory:	**16K**
Language:	**Integer Basic or Applesoft (please specify)**
Hardware:	**Cassette or disk**

Description: Have your own oracle in your home. Consult the I Ching as others have through the ages. Includes a tutorial and a bibliography, as well as an interpretation of the results.

Copies:	**Just released**
Price:	**$9.95 on cassette; $14.95 on disk**
Author:	**C. Brandon Gresham, Jr.**
Available:	Ad Hoc Enterprises 23 Van Buren Street Dayton, OH 45402

Name:	**MUSIC**
System:	**Any 6502 based system**
Memory:	**1.5K**
Language:	**Assembly**
Hardware:	**Terminal or TVT and a speaker connected to one output port**

Description: Music is an interactive programming language for the creation of patterns of sound; "music". It is a compositional tool, not merely a music table compiler or piano roll type of program. Music's language structure is similar to "ROBOT" (see MICRO no. 10, page 15). Complex hierarchies of user defined functions - strings of musical events - which can be called like subroutines, allow the user to program highly intricate and surprising compositions.

Copies:	**Just released**
Price:	**$10.00 (KIM-1 Hypertape cassette: $3.00 extra)**
Includes:	User manual with programming examples and a completely commented source and object code listing
Author:	**Michael Allen**
Available:	Michael Allen 6025 Kimbark Chicago, IL 60637

Get a load of this . . .

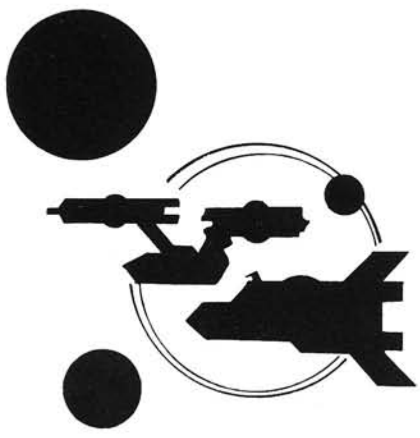
Instant Software™

Ask for Instant Software at a computer store near you.

PET UTILITY I You're working under a serious handicap if you can't write programs in machine language. The PET Utility I package gives you the tools you need:

•**Monitr**—The Monitr program lets you write, edit, save, and verify any machine language and/or BASIC program. Just load and run the Monitr program and then load the program you want to edit.

•**Programmer's Calculator**—This program will convert numbers into the binary, octal, decimal, and hexadecimal systems and function as a floating point calculator. It will also display all four numbering systems simultaneously and allow you to handle large numbers limited only by the size of your screen. For the 8K PET. **Order No. 0105P. $7.95.**



SPACE WARS You must protect your planet against a wide variety of alien attacks. Included are:

•**Space War**—One or two players can pilot their saucers and duel with laser beams or shoot stars.

•**Star Ship Attack**—Your mission is to protect the food station satellites from destruction by the enemy star ship. You must capture, destroy, or drive off the attacking ship.

•**Battlefield**—Guess the location of the four enemy divisions and destroy them before your forces are wiped out.

Engaging in battle requires the Apple 8K and Integer BASIC. **Order No. 0096A $7.95.**

ACCOUNTING ASSISTANT This package will help any businessman solve many of those day-to-day financial problems. Included are:

•**Loan Amortization Schedule**—This program will give you a complete breakdown of any loan or investment.

•**Depreciation Schedule**—You can get a depreciation schedule using any one of the following methods: straight line, sum of years-digits, declining balance, units of production, or machine hours.

This package is available for both the PET and Apple. It requires the Apple 16K and Applesoft II BASIC. **Order No. 0088A $7.95** or the PET 8K. **Order no. 0048P $7.95.**

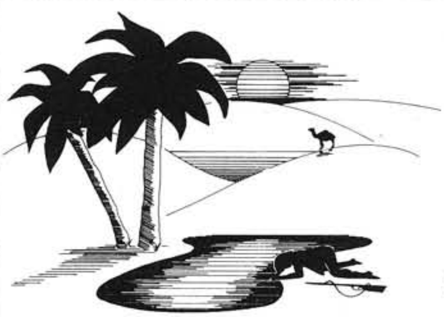
MIMIC Test your memory and reflexes with the five different versions of this game. You must match the sequence and location of signals displayed by your Apple. You'll need an Apple with 24K and Integer BASIC. **Order No. 0025A $7.95.**

HAM PACKAGE I This versatile package lets you solve many of the problems commonly encountered in electronics design. With your 8K PET, you have a choice of:

•**Basic Electronics with Voltage Divider**—Solve problems involving Ohm's Law, voltage dividers, and RC time constants.

•**Dipole and Yagi Antennas**—Design antennas easily, without tedious calculations.

This is the perfect package for any ham or technician. **Order No. 0054P $7.95.**



SAHARA WARRIORS Enjoy all the gritty realism of desert warfare:

•**Commando**—You must send your commandoes to trap a German general and cut him off from his troops.

•**French Foreign Legion**—The battalions of the French Foreign Legion are in a race with the Arabs. Which side will get its battalions into the oasis in the shortest time? You and a friend can find out.

Both programs in this package require an Apple 8K and Integer BASIC. **Order No. 0080A $7.95.**

MORTGAGE WITH PREPAYMENT OPTION/ FINANCIER These two programs will more than pay for themselves if you mortgage a home or make investments:

•**Mortgage with Prepayment Option**—Calculate mortgage payment schedules and save money with prepayments.

•**Financier**—Calculate which investment will pay you the most, figure annual depreciation, and compute the cost of borrowing, *easily and quickly*.

All you need to become a financial wizard with a 16K Apple and Integer BASIC. **Order No. 0094A $7.95.**

CHIMERA If you think the legendary Chimera was hard to handle, wait until you try this package. Included are:

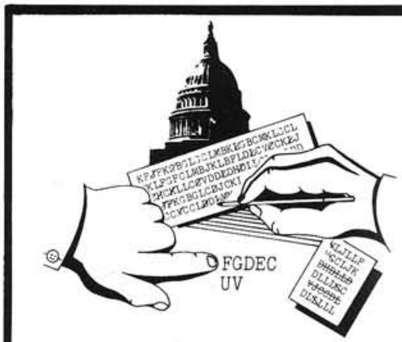
•**Dropoffs**—You must make your opponent's men "dropoff" the board by moving and firing your own men. For one or two players.

•**Dots**—Place your lines carefully as you try to build and capture the squares. For one player.

•**Batter-up**—You and another player take turns at bat as your PET becomes both the pitcher and the umpire. For two players.

•**Reflex**—Round and round the little white ball rolls. Only fast reflexes can guide it into the center of the maze.

You'll almost be able to feel the Chimera's fiery breath as you play the games on your 8K PET. **Order No. 0110P. $7.95.**



CODE NAME:CIPHER Enjoy that same feeling of intrigue and discovery with the Code Name: Cipher package. Included are:

•**Memory Game**—Would you like to match your memory against the computer's? You can with the Memory Game.

•**Codemaster**—One player types in a word, phrase, or sentence, and the PET translates that message into a cryptogram. The other player must break the code and solve the cryptogram in the shortest time possible.

•**Deceitful Mindmaster**—This isn't your ordinary Mastermind-type game. You must guess the five letters in the hidden code word.

•**Code Breaker**—Cracking this code won't be as easy as cracking walnuts. You'll need to flex your mental muscles to win this game.

If you want a mental challenge, then Code Name: Cipher is for you. For the 8K PET. **Order No. 0112P. $7.95.**

Ask your local dealer for the latest Instant Software catalog or write: Instant Software Catalog Dept., Peterborough, N.H. 03458

Instant Software™ Inc. Peterborough, New Hampshire 03458 603-924-7296

HDE inc.

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

HUDSON DIGITAL ELECTRONICS INC.

ANNOUNCING THE HDE OMNIDISK 65/8

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

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

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

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

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

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

6502 Bibliography: Part XIX

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

581. Compute, Iss. 1 (Fall 1979)

Moser, Carl W, "Universal 6502 Memory Test," pgs. 32-33.
A memory test program with ways to adapt it to PET, APPLE II, SYM, KIM, TIM, OSI 65D, Western Data Systems, ATARI, AIM, Super Kim, etc...

Thornburg, Katie A. and David D., "Flying with PET PILOT," pgs.40-45.
Kids and microcomputers at Peninsula School.

Tulloch, Michael, "CORVUS 11A Disc Drive," pg. 61.
How about a 9.6 megabyte memory on line? Compatible with the Apple DOS. Plugs into one slot of the Apple II.

Victor, John, "Atari Computers: The Ultimate Teaching Machines?," pgs. 62-64.
Discussion of the advantages of the Atari in educational use.

Lindsay, Len, "The Evolution of a Magazine," pgs. 65-66.
Discusses the history of the PET Gazette and its successor.

Butterfield, Jim, "PET in Transition," pgs. 68-70.
Discusses new modifications of the PET and how programs must be modified to accomodate the new systems.

Isaacs, Larry, "Retrofitting ROMS," pgs. 76-77.
How to replace the old PET Roms with the new units.

Malmberg, David, "Screen Print Routine," pgs. 78-79.
General utility to print the screen using the new PET printers.

Anon., "Cassette Format Revisited," pgs. 80-81.
Discussion of the PET Cassette format.

Butler, Brett, "Trace for the PET," pgs. 84-85.
TRACE allows you to see Basic executing, displaying each actual line as it is executed.

Hunkins, Arthur, "8-Bit Digital to Analog Converter," pgs. 90-91.
A review of the MicroTechnology Unlimited DAC board for Hal Chamberlin's 4-part music program.

Stuart, Chuck, "Using Direct Access Files with the Dual Drive Disk," pgs. 93-96.
A tutorial on the Commodore 0240 Dual Drive Disk.

582. Creative Computing 5, No 10 (October 1979)

North, Steve, "Mountain Hardware SUPERTALKER," pgs. 42-44.
A review of this new accessory for the Apple II.

Yob, Gregory, "Personal Electronic Transactions," pgs 180-183.
Changes in the New PET; more on the structure of Basic variables; Simple variables, PET Basic Pointers, Discussion of cryptograms.

583. The Target (Sept/Oct 1979)

Roland, Don, "AIM 65 MOVIT," pgs. 2-3.
A move program for the AIM 65.

Butterfield, Jim. "Mortgage," pg. 4.
Mortgage program for AIM Basic.

Clem, Don, "Memory Display," pg. 5.
A program to show what the 6502 is seeing at the output port of the AIM micro.

Riley, Ron, "AIM BASIC," pgs. 6-8.
All about AIM Basic.

Anon., "Statistical Analysis," pg. 10.
Regression programs for the AIM.

584. The Paper 2, Iss. 7 (September 1979)

Busdiecker, Roy, "Think Negative!," pgs. 3-7.
Negative numbers and subtraction are covered in this continuing tutorial on binary numbers, with examples for the PET and 6502.

Busdiecker, Roy, "Decimal to Binary," pg. 12.
PET Basic program for converting Decimal to Binary.

Busdiecker, Roy, "Warning: Prevent Tape Decay," pgs. 21-22.
Use your PET tapes or at least rewind them occasionally to avoid print through on long undisturbed storage.

Oakes, Peter L. A., "PLOT 2-MP," pgs. 23-25.
A plotting program for the PET.

585. Call Apple 2, No. 7 (September 1979)

Wagner, Roger, "Speeding in Applesoft," pg.2.
How to speed up Applesoft Subroutines.

Winston, Alan B., "The Multi-Lingual Apple," pgs. 4-5.
All about Pascal for the Apple.

Hertzfeld, Andy, "Assembler Mini Reviews," pgs. 7-10.
A Consumer's guide to Apple II Assemblers: Apple Mini-Assembler, TED/ASM, Micro Products (Moser/Bishop) Assembler, Randy's Weekend Assembler, Aresco

Assembler, LISA, Microproducts 6-Character Label Assembler, E.A.T. (Apple II Text Processing System), 6502 Macro Assembler (Moser), UCSD Pascal Assembler, S-C Assembler, etc. etc....

Golding, Val J., "Peeking at Numbers," pg. 10.
Demonstration of a couple of ways to print numbers in Integer Basic larger than 32767 on the Apple.

Verlaque, Richard, "Decimal Division/Integer Basic," pg. 16.
Arithmetic program for Integer Basic on the Apple.

Thing, Mike, "Applemash," pgs. 18-19.
All about Modem Operation and the newly established "Apple-Crate," the Call-Apple message system ABBS.

Hoyte, Jim, "Subroutine to Allow Prohibited Character in String Inputs," pg. 24.
Routine lets you use commas, etc. in string inputs on the Apple.

Williams, Rick, "Flash Cards," pg. 28.
A Tutorial program for the Apple.

Capella, Mark, "Hide," pg. 29.
A program to hide program names from appearing in the catalog, but still permitting them to be recovered.

Golding, Val J., "Poke Chr$ to screen," pg. 29.
Routine to put characters on screen at specified locations.

Dunmire, Darrell and Golding, Val J., "So Who Needs Applesoft?" pg. 31.
How to use string functions such as VAL, STR$, and CHR$ in Integer Basic on the Apple.

Neulen, Bob and Golding, Val J., "Change Catalog to CC," pg. 33.
A routine to change catalog to CC for the 3.2 DOS.

Corsetti, Vincent S., "File Restore," pg. 35.
How to restore a program on a disk after you realize you just wiped out something you didn't mean to, on the Apple.

586. Personal Computing 3, No. 11 (November, 1979)

Anon., "Jim and Kay Weir Re-invent the Wheel," pg. 14.
Discusses an application of an Apple II in the tire retreading business.

587. Computer Cassettes Review, No 7 (Fall 1979)

Purser, Robert Elliott, "Software for Apple, PET and TRS-80."
Software Directory, cassette reviews, game reviews, and a list of "The Classics—the best of the past."

588. Rainbow 1, No. 8 (September 1979)

Ladbrooke, Brian D., "Running PIMS on the APPLE II," pg. 1.
How to adapt this TRS-80 or PET program on Personal Information Management System to the Apple II.

Rost, Randi J., "HiRes Color," pgs. 4-7.
A tutorial.

Heller, H. Lewis and Gothrie, Nelson, "Dollar Conversion," pgs. 7-11.
Three programs for the Apple to convert Applesoft outputs to rounded off dollar and cents values.

Buchen, Steven E., "Slow Down," pgs. 16-21.
Two programs to slow down the list function of the Apple

MacDougal, John, "Parallel Interface for the Apple," pgs. 22-24.
Hardware article on a parallel interface providing 2 output ports and 2 input ports.

589. Interface Age 4, No. 12 (December 1979)

Gilbert, Betsy, "The Computerized Artist," pgs 72-73.
A description and evaluation of the Apple Graphics Tablet.

590. Kilobaud Microcomputing No. 35 (November 1979)

Lancaster, Don, "Lower case for your Apple II, Part 1," pgs. 30-36.
Expand the usefulness of your Apple with this inexpensive addition.

Schmeltz, Leslie R., "The Apple Goes to Market," pgs. 70-76.
Gather and analyze data from the stock market.

Derfler, Frank J., "Boy, Did I Make a Killing!" pgs. 112-114.
Real Estate profit guide program for the PET.

Bajcz, Bill, "A 'Pentronics' System," pgs. 158-160.
How to interface a PET to a Centronics 101 Printer.

591. Southeastern Software Newsletter No. 13 (Oct. 1979)

McClelland, George, "Pascal," pgs. 1-3.
Discussion of Pascal for the Apple.

Carpenter, Chuck, "Indexing Fundamentals, Part II," pgs. 4-6.
A tutorial on indirect addressing including a routine to read and print a memory range.

Anon., "Four HI-RES Programs," pgs 7-8.
Several hires routines to frame pages of a program.

592. Ohio Scientific's Small Systems Journal (1979)

Anon., "OSI Programs."
A collection of about 60 programs for OSI computers.

593. The Cider Press 2, No. 5 (September 1979)

Anon., "Disk of the Month," pg. 3.
Seventeen more programs.

Anon., "Serial Handshake Modification," pg. 7.
The high speed serial interface card of Apple Computer Company can be made to run faster than 300 baud by simple modification.

Hockenhull, James L., "Better Sounding Apples," pg. 8.
How to improve the sound of the apple by toggling the cassette output jack.

Nareff, Max J., "Crossfooting," pg. 8.
How to use crossfooting in tabulation of numbers.

594. MICRO 17 (October 1979)

Connolly, Rick, "Nicer Writer," pgs. 5-6.
Eliminate wraparound in your Apple.

Reynolds, William, "Disassembling the DOS 3.2," pgs. 7-10.
Use the Apple DOS 3.2 more effectively with this information on its organization.

Scanlon, C. H., "Hooking PET to Ma Bell," pgs. 11-13.
Use your system as a terminal with an inexpensive modem.

Mimlitch, Thomas R., "Spelunker," pgs. 15-24.
An adventure type game for the Apple.

DeJong, Marvin L., "6522 Timing and Counting Techniques," pgs. 27-39.
Application of the 6522 versatile interface adapter.

Chan Hark, "Card Shuffling Program for KIM-1," pgs. 41-42.
Teach the Kim to shuffle the cards.

Hoyt, Bruce, "How Do You Connect Peripherals to Your Superboard II?" pgs. 43-46.
Some concise information on the configuration and use of the I/O Ports of the OSI Superboard II.

Rowe, Mike (staff), "The MICRO Software Catalog: XIII," pgs. 49-51.
Nine programs are reviewed, mostly for Apples.

Morris, E. D., "Hypocycloids," pgs. 52-53.
A fast graphics program for the OSI.

NOW PRESENTING...

Software for Apple® II

for your Entertainment · Business · Education

Star Attractions:

+WRITE-ON! Professional Word Processing lets you edit, move, delete, find, change and repeat any body of text, merge and save on disk. Does right-justified margins, centering, page numbering. You can enter name & address onto form letters when printing. Edit and merge any text disk file—even files not created by WRITE-ON—and spool text to disk for letter printing or editing. Chain up to 100 files in a single printer run. Needs Applesoft and 32K.
On Disk with operating manual **$99.50**

FILEMASTER I Here's a powerful data file manager giving you two programs—FORMAT & RETRIEVAL. It handles everything from phone lists to legal abstracts. You can design your own data structure with up to 500 characters per record and up to 15 searchable fields in any combination. Needs 32K. ***On Disk*** **$49.95**

FILEMASTER II Has all the same features of FILEMASTER I plus allows for totaling, advanced math routines, more powerful print formatting, larger data fields, and disk-to-disk transfers **$99.50**

+Space (Edu-Ware) Six programs form a unique epic game series. Multi-faceted simulation of life in interstellar society. You and opponents must make life & death decisions. Keeps track of your progress from one game to next. Needs 48K and Applesoft ROM. **Disk** **$29.95**

Pot O'Gold I **or our** All New Pot O' Gold II. A collection of 49 programs for 16K Apple. Everything from Logic to action games. Only a buck a game. Specify I or II.
Price each: **Tape** .... **$49** **Disk** **$54**

+Adventure This original, full-function game is the same as the one developed for large mainframes. Fight off pirates and vicious dwarfs. 700 travel options, 140 locations, 64 objects. Needs Applesoft ROM & 48K. **Disk** **$29.95**

32K Disk Inventory: Use stock numbers, description, vendor, record of purchase and sales date, amount on hand, cost & sell price, total value. Holds up to 300 items.
Disk **$40**
+With Bill of Materials: **Disk** **$50**

32K Data Base Cross file for phone lists, bibliographies, recipes. Run up to 9 lines of 40 columns each. Search by item anywhere.
Disk **$20**

24K Hi-Res Life Simulation Conway's equations on 296x180 screen. A mathematical simulation to demo population growth with birth, death and survival as factors. **Tape** **$10**

16K Rainbow's Casino 9 gambling games: Roulette, Blackjack, Craps, Horserace, Yahtzee, Keno, Slot Machine, Poker, and Acey-Ducey. Needs 16K. **Tape** .... **$29.95** **Disk** **$34.95**

16K Space War: You in your space capsule battle against the computer's saucer . . . in hi-res graphics. **Tape** **$12**

16K Memory Verify Diagnostic routine to check range of memory. Indicates faulty addresses, data in memory cell, and faulty data. **Tape** **$5**

16K Appleodion Music synthesis composes original Irish jigs. Enter your own music and save on tape or disk. Includes 3 Bach fugues.
Tape **$10**

+48K Edu-Pack (Edu-Ware) This package combines COMPU-READ—five speed reading programs; three PERCEPTION games where random shapes and sizes must be matched; and STATISTICS for computing Mean, Variance, Standard Deviation, and much more! Needs Applesoft ROM. **Disk** **$39.95**

+32K Sargon II (Hayden) Here's the program that came in third against the big machines (mainframes and maxis) at the 9th North American Computer Chess Championship and placed first in the European Microcomputer Chess Championships! Has seven levels of play with Levels 0 - 3 playing in tournament time. Need a challenge? This is it!!
Tape .... **$29.95** **Disk** **$34.95**

16K Hi-Res Baseball (Programma International) This animated simulation of a major league baseball game is for *two players*. The scoreboard is in the lower left of screen with the "throw pointer" for directing a throw in the lower right corner. Written entirely in machine language, the action is quick and smooth, making it the finest simulation of its kind.
Tape **$15.95**

OTHER SPECIAL ITEMS FOR YOUR APPLE II

VERSAWRITER This digitizer drawing board, complete with a powerful software package on disk, lets you create any picture in color with high-resolution graphics. It's ideal for mass graphics. You can trace, edit, save and recall what you draw. It's a simple-to-use system for students, artists, engineers and graphic programmers. Has an 8½"x11" working area. New applications added include: •Text Writer adds text to your pictures. You control size, color and direction of text; •Electronic Drawing lets you create schematics and includes commonly used symbols for transistors, OPAMPS, and FETS; •Distance/Area lets you compute distances on maps or area of any frame. Applesoft ROM and 32K required.. **$249.95**
Add $5 (U.S.) or $10 (Foreign) for shipping.

Apple Monitor Peeled Everything you wanted to know about the Apple Monitor but couldn't figure out. All the PEEKS, POKES, and CALLS explained in an easy-to-understand form, written in plain English.
Only **$9.95**

Don't see what you've been looking for, here? Then write for our FREE SOFTWARE CATALOG. We're saving one just for you!

To order, add $2 (U.S.): $5 (foreign) for shipping. California residents add 6% sales tax. Sorry, we cannot ship to P.O. Boxes. VISA/MASTERCHARGE and BANKAMERICARD Welcomed!

RAINBOW COMPUTING INC.

Golden Plaza Shopping Center, Dept. 1A
9719 Reseda Blvd., Northridge, CA 91324
Telephone: (213) 349-5560

+ =Apple Plus compatible

VISA

master charge

What's NEW from SOFTAPE

JOURNEY

You are about to embark on a very hazardous but profitable JOURNEY. The Apple is your eyes, ears, arms and legs. You can "GET" an object that is laying on the ground and you can travel North, East, South, West, Up and Down. You need to acquire tools as you JOURNEY forth and score precious points to become a GRANDMASTER JOURNEYER. JOURNEY REQUIRES 48K and loads on any Apple.

JAB-879 $19.95

BASEBALL FEVER

Catch the fever with this ball game that never gets rained out. You are the pitcher, batter and manager for your team. As manager you control line-ups, pitching staff and base stealers. The optional sound effects enhance the colorful animation. When the game is over you can save the game statistics to tape or disk and you are on your way to a winning season. Requires 32K or 48K with disk. INTEGER BASIC

BFM-879 $12.95

OTHELLO

Play a true competitor of this ancient game of territorial strategy. By flanking a line of the opponents men you "flip" them over to your own color. Be cautious though, for OTHELLO will never say die until the last move. OTHELLO is available for both INTEGER BASIC and APPLESOFT, and loads in 16K.

OHS-279 $14.95

CONEY ISLAND

Enjoy the excitement of an amusement park at home. CONEY ISLAND has 22 varieties of paddle games that are fast. Written in FORTH II for speed, and using the beautiful color graphics of the Apple, one or two can play the most exciting paddle games yet written. CONEY ISLAND can be loaded on any Apple. 16K

CIW-879 $12.95

FORTH II

FORTH II is an extremely well documented version of the Forth language that has been in use since the late 1960's.

It is many times faster than basic and is easy to use. Many of its features are as follows:

NEW FEATURES

*RUNS ON ANY APPLE II COMPUTER
*(24K minimum)
*SUPPORTS DOS 3.2
*CONTROL C BREAK AND CONTINUE
*COMPATIBLE WITH AUTOSTART ROM
*"SAVE IT" FILE FOR CUSTOMIZING SYSTEM

STANDARD FEATURES

*INHERENTLY STRUCTURED LANGUAGE
*DISK BASED EDITOR AND COMPILER
*COMPLETE INSTRUCTIONAL REFERENCE MANUAL
*EXCELLENT EXECUTION SPEED AND MEMORY EFFICIENCY
*SUPPLIED ON MASTER DISKETTE
*VERBS FOR GRAPHICS, GAME I/O, SOUND, DISK AND TAPE I/O

FOG-279 $49.95

PADDLE PLUS

If you have the same problem as Arnold Zieback with constantly changing paddles and PENS, then you too need PADDLE PLUS. This extender plugs into your game I/O port and is conveniently secured for easy access.

PPA-180 $14.95

WHERE TO GET IT: Look for the SOFTAPE Software display in your local computer store. Apple dealers throughout the United States, Canada, South America, Europe and Australia carry the SOFTAPE Software line of quality products.

If your local dealer is sold out of SOFTAPE Software you can order it direct from us by check or Visa/Master Charge. If you have any questions please call us at:

VISA **1-213-985-5763** master charge

Or mail your order to the address below. We'll add your name to our mailing list for free literature and announcements of new products.

SOFTAPE™

10432 Burbank Blvd. • North Hollywood, CA 91601

BRIGHT PEN

What is the difference between a light pen and the BRIGHT PEN. Intelligent software and extensive documentation. The software will help you to calibrate your system for optimum operation. The documentation details the BRIGHT PEN DRIVERS and how they are appended to your INTEGER BASIC programs. BRIGHT PEN includes documentation booklet, two cassettes and, of course, the BRIGHT PEN.

BPE-279 $34.95

DUMP-RESTORE

With DUMP-RESTORE you will be able to back up your disk files to cassette and restore them to disk. This allows you to relocate disk space for maximum efficiency and speed. The programs are saved and restored individually or the entire disk can be saved and restored. DUMP-RESTORE loads with INTEGER BASIC and requires 32K.

DRG-879 $14.95

RESET GUARD

Tired of hitting reset by mistake? If so RESET GUARD will solve the problem. RESET GUARD is a hardware package that plugs directly into your Apple. It protects your programs because it will only Reset if hit twice in one second. Guard your Apple and your sanity with RESET GUARD.

RGA-180 $34.95