MICRO

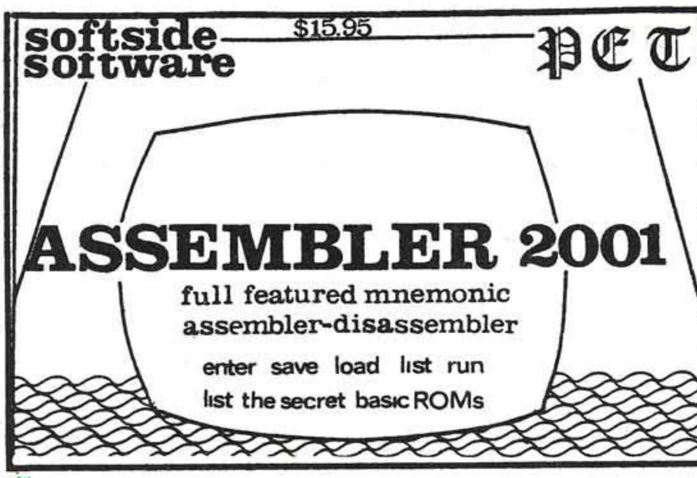
THE 6502 JOURNAL

A Digital Thermometer for the APPLE II

Clocking KIM

A Home Message Center

NO. 22 MARCH 1980 $2.00



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

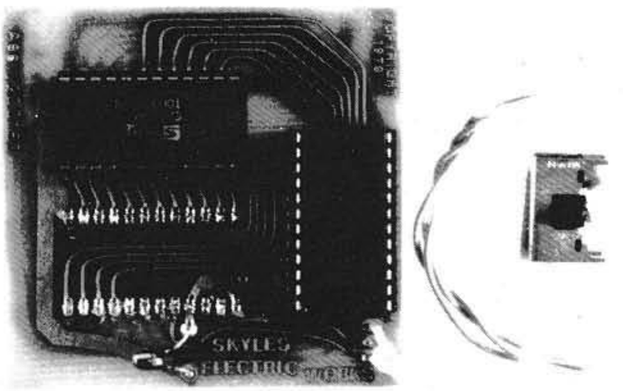
- UNTIL....GOTO....
- SYMBOLIC OPTIONAL
- LINE NUMBERS

305 RIVERSIDE DRIVE, NEW YORK, NEW YORK 10025

Said the Toolkit to the Word Processor: "You're in My Space!" Said the Word Processor to the Toolkit: "Let's Share...here's

Socket 2 Me™!"

From the original producer of peripherals exclusively for PET lovers everywhere . . . the device that allows you to select between the BASIC Programmer's Toolkit and the Commodore Word Processor II while they occupy the same address space.



The **Socket 2 ME . . . doubles** your **m**emory **e**xpansion in a single **socket**. It's a 2.5" x 2.75" board that fits neatly into the Toolkit/Word Processor socket on the main logic board of all *new* PETs. Then both the Word Processor and the Toolkit plug into the **Socket 2 ME.**

A miniature slide switch – part of the kit – mounts with double-stick tape (supplied) to the front part of the right side of the PET base, almost hidden by the overhang of the top of the PET cabinet. The slide switch is connected to the **Socket 2 ME** by a special cable (also supplied) . . . and you're up and running.

Up and running; installation took only a minute or so. Flip the switch from Toolkit to Word Processor. And back. No need to open the PET.

Complete with the first-rate installation and operating instructions you've come to expect from all Skyles documentation.

YOU HAVE AN ORIGINAL 2001-8 PET?

No problem. The **Socket 2 ME** interfaces with the BASIC Programmer's Toolkit model TK 160E or TK 160S connector board, the Word Processor II interfaces with the **Socket 2 ME**, the slide switch is placed on the PET base. Then, as long as the PET 2001-8 has at least 8K of memory expansion, the system is up and running.

YOU HAVE A COMPUTHINK DISK AND YOU WANT A BASIC PROGRAMMER'S TOOLKIT?

How would you like to switch between the Computhink and the Toolkit with a single **SYS** *command?*

Just add two small jumpers to the Computhink system and a short program to the DOS diskette. Plug in the BASIC Toolkit **TK 80E**, enter the **SYS** command and your system is up and running.

NO ROOM ON THE *PET 2 COMPUTHINK* DISK BOARD?

All your sockets are booked? Fret not; Skyles comes to the rescue. **Skyles Electric Works** now has a modified EPROM board available with sockets for the Toolkit and Computhink ROM chips. Plug in the ROMs, add a jumper (supplied) to the PET 2 Computhink disk board, plug the new EPROM board into the Computhink disk board. Power up and enter a short switching program into the DOS diskette. Switch between the Computhink disk and the Toolkit with a single **SYS** command.

ORDER NOW – with Skyles' 10-day money-back guarantee:

Socket 2 ME: $$22.50*
Commodore Word Processor II: $100.00*
Commodore Word Processor III: $200.00*

BASIC Programmer's Toolkit

Model TK 80ED	$85.00*
Model TK 160ED	$95.00*†

**Add $2.50 to each for shipping and handling.*

†Note: If Computhink EPROM board is returned, after purchase of TK 160ED, Skyles will refund $20.00.

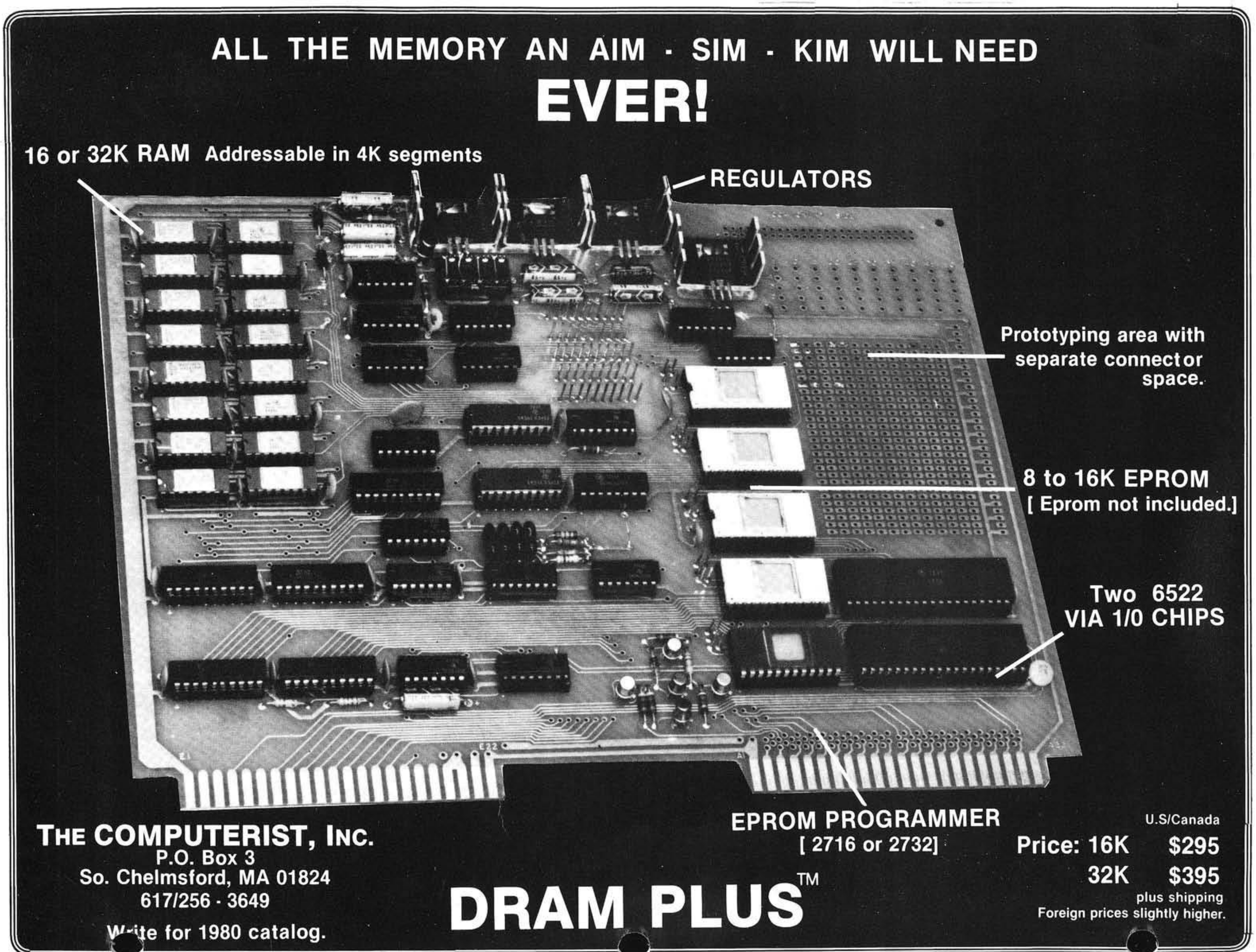
"Socket 2 ME" is the trademark of Skyles Electric Works.

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

VISA, MASTERCHARGE ORDERS CALL (800) 538-3083 (except California residents)
CALIFORNIA ORDERS PLEASE CALL (408) 257-9140

Skyles Electric Works

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


ALL THE MEMORY AN AIM · SIM · KIM WILL NEED
EVER!
16 or 32K RAM Addressable in 4K segments
REGULATORS
Prototyping area with separate connector space.
8 to 16K EPROM
[Eprom not included.]
Two 6522 VIA 1/0 CHIPS
EPROM PROGRAMMER
[2716 or 2732]
THE COMPUTERIST, INC.
P.O. Box 3
So. Chelmsford, MA 01824
617/256 · 3649
Write for 1980 catalog.
DRAM PLUS™
U.S/Canada
Price: 16K $295
32K $395
plus shipping
Foreign prices slightly higher.

MICRO™

March 1980
Issue Number 22

Table of Contents

Staff

Editor/Publisher
Robert M. Tripp

Associate Editor
Mary Ann Curtis

Assistant Editor
Evelyn M. Heinrich

Business Manager
Maggie E. Fisher

Circulation Manager
Carol A. Stark

Art/Advertising Coordinator
Terry Spillane

Comptroller
Donna M. Tripp

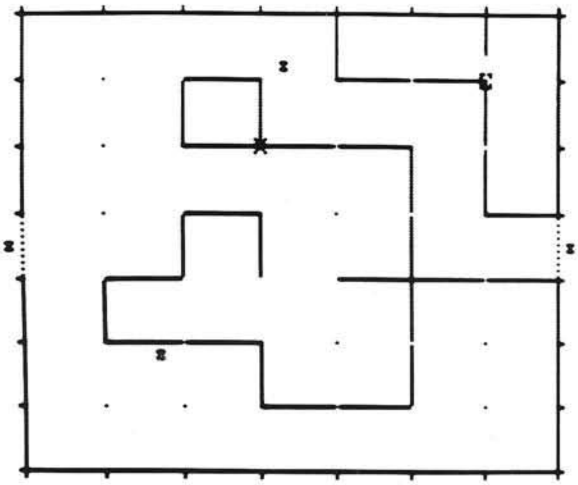
Production Assistant
L. Catherine Bland

MICRO Lab
R. Keith Beal

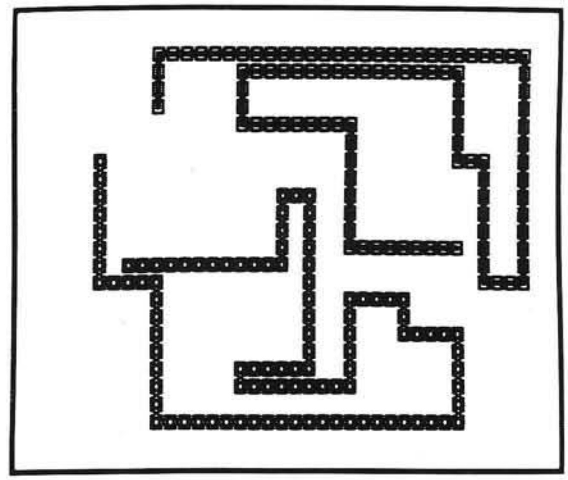
MICRO™ is published monthly by MICRO INK, Inc., Chelmsford, MA 01824. Tel. 617/256-5515.
Second Class postage paid at Chelmsford, Ma 01824.
Publication Number: COTR 395770.
Circulation: Paid subscriptions: U.S.: 3800, Foreign: 350; Dealers: U.S.: 4500, Foreign: 1900.
Subscription rates: U.S.: $15 per 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

Software for the Apple II



SCORE:108 SCORE:105



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

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

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

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

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

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

Available at your local computer store

Call or write for our free SOFTWARE CATALOG

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

DEALER INQUIRIES INVITED

POWERSOFT, INC.

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

Programs Available on Diskette at $5.00 Additional

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

A 16 Bit 6502?

Several microprocessor manufacturers who make 8 bit micros have recently come out with 16 bit versions or variations. Some of these are true 16 bit micros and some might be best termed "pseudo" 16 bit, that is micros that are basically 8 bit and have an 8 bit data bus, but which can perform some 16 bit operations. There has been talk from time to time about a pseudo 16 bit version of the 6502. It has even had a number, 6509 or 6516, but at present it looks as though it is only a designer's dream.

If someone were to come up with a pseudo 16 bit version of the 6502, what should it contain? And, is there really a need for such a device? I have a few ideas on these matters which I will present next month. An article by Randall Hyde will describe in detail the specifications which were generated for a Synertek 6516. Between now and then, why don't you think about these topics and see what you would like to have in a 16 bit version of the 6502? Maybe with enough input from various sources we can help get such a project moving.

A MICROscope Note

If you have applied to be a reviewer for the **MICROscope** product review, thank you. We will be sending out acknowledgements and additional information this month. A number of products have been submitted for review, so we will be getting these out to the reviewers soon, and the first **MICROscope** should appear within a couple of months. If you have a product to be reviewed: hardware, software, book, or whatever, contact us for the proper forms.

Robert M. Tripp

Statement of Ownership, Management, etc., required by the act of Congress of October 23, 1962 of **MICRO** published monthly at Chelmsford, Massachusetts for March 1980.

The name and address of the publisher is **MICRO INK, Inc.**, 34 Chelmsford Street, Chelmsford, Massachusetts. The Editor/Publisher is Robert M. Tripp of Chelmsford, Mass.

The owner is **MICRO INK, Inc.**, Chelmsford, Mass. and the name and address of stockholders owning or holding one percent or more of the total amount of stock is: Robert M. Tripp and Donna M. Tripp of Chelmsford, Mass.

The known bondholders, mortgagees and other security holders owning one percent or more of the total amount of bonds, mortgages or other securities are: none.

The average number of copies of each issue of this publication sold or distributed though the mails or otherwise to paid subscribers during the twelve months preceding the date shown above is: 7063.

I certify that the statements made by me above are correct and complete.

Signed: Robert M. Tripp
Editor/Publisher

APPLE II Floating Point Utility Routines

Here is a guide to the Applesoft BASIC floating point utility routines which will permit them to be used effectively from assembly language programs. Get the best of both worlds: optimize your programs by writing them in assembly, and, use these excellent floating point math routines directly.

Harry L. Pruetz
2213A Lanier Drive
Austin, TX 78758

Although floating point capabilities are available in Applesoft BASIC, it is still useful to do floating point operations from machine language. This is especially true when the floating point operations are needed at some point in a machine language routine and it would be very tedious to pass parameters back up to a FP BASIC program. The alternative of writing special machine language routines for the solution of a few calculations can also delay a programming project unnecessarily. The purpose of this article is to give an idea of how the AFPUR (Apple Floating Point Utility Routines) work and how to use them.

AFPUR uses 248 bytes from $F425 to$F4FB and the FIX and underflow routines from $F63D to $F65D. In case of overflow, a jump to OVLOC ($3F5) is taken, where a jump to your own code should be stored. Floating Point work space is given in the reference manual as $FO to $FF although only $F3 to $FF are used. The floating point work space bytes and their uses are given in Table 1

E is an extra copy of the FPI mantissa saved during all arithmetic operations although it is actually used only in division. EG is an extra byte changed by the align and normalize code when FP1 and FP2 are being shifted. FP = X,H,M,L is the floating point format of a number where X is the exponent and H, M, and L are the high, medium, and low bytes of the mantissa. Note that the order of the bytes according to significance is opposite that in Integer and FP BASIC and the Sweet 16 interpreter.

The floating point format is similar to 32-bit hardware on many larger computers. The exponent is excess 128 so that $80 represents 2^0. The mantissa is two complement normalized until the two leading bits are different so that + 1.0 = $80 40 00 00 and —1.0 = $7F 80 00 00. Here are some more examples of decimal numbers in hex floating point format:

```
0.0 = $00 00 00 00
1.5 = $80 60 00 00
1.75 = $80 70 00 00
10.0 = $83 40 00 00
256.0 = $88 40 00 00
0.1 = $7C 66 66 66
```

Note that the floating point form $7C 66 66 66 is a truncated approximation of 0.1 so that 0.1 multiplied by 10.0 will give $7F 7F FF FE instead of $80 40 00 00. The AFPUR will work on unnormalized numbers although there can be a loss of accuracy because of the way the alignment code works. For example, in adding the numbers $7F 80 00 00 and $69 40 00 00, the result is $7F 80 00 00 + $7F 00 00 01 = $7F 80 00 01. Using the unnormalized $80 C0 00 00 would give $80 C0 00 00 + $80 00 00 00 = $7F 80 00 00. These cases give trivial losses of accuracy, but more extreme cases can make your Apple II seem like it can't add. Since results of arithmetic operations are normalized in all cases, unnormalized numbers can only be input to the AFPUR by the programmer in the form of stored constants.

Table 2 gives a general idea of how the AFPUR works.

Trying to POKE or PEEK from Integer BASIC will not work because critical information is stored in the same locations as the floating point work space. For example, $F6 and $F7 contain the current Integer BASIC line number and $F8 contains the automatic line numbering mode flag. If a machine language routine is called from Integer BASIC and AFPUR routines are used, then location $F8 should be set to 0 before returning to the Integer BASIC program.

In the following examples of calls to AFPUR, FP1 and FP2 are used as the 4-byte FP registers at $F8 and $F4. If you can try the monitor calls on your Apple II, the examples will be more instructive.

FCOMPL is called by FSUB one time, FMUL two or three times, and FDIV two or three times. It may be called directly by the user. The only FP number which can cause an overflow error is $FF 80 00 00. FCOMPL is easy to use from the monitor:

```
*F8:80 60 00 00
*F8.FB F4A4G F8.FB.
```

An example of a call to give A = – A can be coded by the steps:

```
        *0) Declare hex storage as a hex
        string
A             .HS 00000000
        *1) Load FP1 with A
              LDX #3
LDAL          LDA A,X
              STA FP1,X
              DEX
              BPL LDAL
```

```
*2) Floating complement the number
        JSR $F4A4    FCOMPL
*3) Store FP1 in A
        LDX #3
STAL    LDA FP1, x
        STA A,X
        DEX
        BPL STAL
```

FLOAT of a fixed point number assumes a 15-bit signed integer in M1H and M1M and an 8-bit fraction in M1L. If no fraction is intended, then M1L must be set to $00. FLOAT is a special entry point preceeding normalization code and is called no other place in APFUR. FLOAT is solely for the user. Because of the 15-bit limit on the magnitude of the integer, there can be no overflow errors.

An example of using FLOAT from the monitor is:

```
*F9:00 64 00
*F8. FB F451G F8.FB.
```

An example of a call to give FPA = float (IA) can be coded by the steps:

```
*0) Declare hex strings for IA and FPA
IA      .HS 0000
FPA     .HS 00000000
*1) Load IA into M1
        LDA IA+1 Intg high byte
        STA M1H
        LDA IA Intg low byte
        STA M1M
        LDA #0
        STA M1L
*2) Float the integer
        JSR $F451  FLOAT
*3) Store FP1 into FPA
        LDX #3
STAL    LDA FP1, X
        STA FPA, X
        DEX
        BPL STAL
```

The FIX of an FP number returns a 15-bit signed integer in M1H and M1M and an 8-bit fraction in M1L. Depending on the size of the FP number, EH, EM and EL may also contain parts of a fraction which could be useful in some calculation. In the more typical uses of FIX, only M1H and M1M are of practical use. FIX has a flaw in the way it treats negative numbers. The FIX1 ($F63D) entry point must be used for negative FP numbers. Calling FIX for FP numbers with exponents larger than $8E will cause overflow errors. Calling FIX1 for negative FP numbers with $8E 7F FF 00, $8E 80 00 00, or exponents larger than $8E will cause overflow errors. To insure that the overflow routine given later in this article will operate properly, a CLV should preceed all FIX and FIX1 calls. Some examples of fixing FP numbers from the Monitor are:

```
*F8:7F 80 00 00
*F8.FB F63DG F8.FB
*F8:80 7F FF 00
*F8.FB F640G F8.FB
```

An example of using an intermediate routine UFIX to give IA = fix(FPA) is:

```
*0) Declare hex strings for IA and FPA

IA      .HS 0000
FPA     .hs 0000000
*1) General UFIX routine
UFIX    CLV           FOR
        OVERFLOW PROCESS-
        ING
        LDA M1H GET SIGN OF
        FP1
        BPL UFIM
        JSR  $F68D     FIX1
        (NEGATIVE FP1)
        RTS
UFIM    JSR    $F640    FIX
        (POSITIVE FP1)
        RTS
*2) Load FPA into FP1
        LDX #3
LDAL    LDA FPA, X
        STA FP1, X
        DEX
        BPL LDAL
*3) Fix the FP number
        JSR UFIX
*4) Store M1 into IA, reversing
byte order

        LDA M1M
        STA IA
        LDA M1H
        STA IA+1
```

Unfortunately, FP2 is sometimes changed depending on the signs and exponents of FP1 and FP2 in routines FADD, FSUB, FMUL, and FDIV. Thus, if FP2 is a constant being used in a series of calculations, it would be wise to restore FP2 each time.

An overflow error can occur in the FSUB call to FCOMPL. FP1 can be corrected and control returned to FSUB. FADD and FSUB may both have overflow errors after the operation is completed and normalization is being done. FP2 and FP1 are swapped (interchanged) if FP1 has the larger exponent since alignment operates by shifting the mantissa of FP1 right until the exponents X1 and X2 are equal.

An example of using FADD and FSUB from the monitor is:

```
*F4:81 40 00 00 80 40 00 00
*F0.FF F46EG F0.FF
*F4:81 40 00 00 80 40 00 00
*F0.FF F468G F0.FF
```

FMUL and FDIV both call FCOMPL to get the absolute value of the operand in FP1. This is done by swapping FP1 and FP2, taking the absolute value of FP1, swapping FP1 and FP2 again, and taking the absolute value of FP1 again. The sign of the result (product or quotient) is stored at location $F3 in the right-most bit. Before returning to the user's program, the sign is tested and FCOMPL is called if bit 0 of SIGN is set. An overflow error can occur on any of these FCOMPL calls if FP1 is $FF 80 00 00. Both FMUL and FDIV check for overflow and underflow when calculating the exponent of the unnormalized result. Underflow is handled by UNDFL at $F657 and overflow by the user's own routine.

Table 1

ADDR	NAME	USE
$F3	SIGN	Product/Quotient sign
$F4	X2	FP2 exponent
$F5	M2H	FP2 mantissa high byte
$F6	M2M	FP2 mantissa medium byte
$F7	M2L	FP2 mantissa low byte
$F8	X1	FP1 exponent
$F9	M1H	FP1 mantissa high byte
$FA	M1M	FP1 mantissa medium byte
$FB	M1L	FP1 mantissa low byte
$FC	EH	M1H copy
$FD	EM	M1M copy
$FE	EL	M1L copy
$FF	EG	garbage

Table 2

ADDR	ROUTINE	OPERATION	TIME
$F46E	FADD	FP1 = FP2 + FP1	1.5 millisec
$F468	FSUB	FP1 = FP2 − FP1	1.6 millisec
$F48C	FMUL	FP1 = FP2*FP1	3.5 millisec
$F4B2	FDIV	FP1 = FP2/FP1	5.6 millisec
$F451	FLOAT	FP1 = float(M1)	105 microsec
$F640	FIX	M1 = fix(FP1)	125 microsec
$F4A4	FCOMPL	FP1 = − FP1	135 microsec

FP constants may be calculated by using the monitor as in finding 29.43. The steps are:

1) load M1M with 43 = $2B
2) float
3) move FP1 to FP2
4) load M1M with 100 = $64
5) float
6) divide giving FP1 = float (43)/float (100)
7) move FP1 to FP2
8) load M1M with 29 = $1C
9) float
10) add giving FP1 = 29.43

```
*F8:00 00 2B 00 F8.FF
*F451G F4  F8.FBM F0.FF
*F8:00 00 64 00 F8.FF
*F451G F4B2G F4 F8.FBM F0.FF
*F8:00 00 1C 00 F8.FF
*F451G F46EG F0.FF
```

The final result is FP1 = $84 71 B8 51 or 29.43.

As is obvious to the most casual observer, calculating very many constants using the monitor is hazardous to your enthusiasm.

The order in which operands are loaded in FP1 and FP2 for addition and multiplication can be chosen so that the user's code is more efficient. For example, the statement D = A*B + C can be coded by the steps:

```
*0) Declare hex strings for A,B,C,D
A       .HS 80 60 00 00  1.5
B       .HS 82 40 00 00  4.0
C       .HS 7F 80 00 00  -1.0
D       .HS 00 00 00 00
*1) Load FP1 with A
        LDX #3
LDAL    LDA A,X
        STA FP1.X
        DEX
        BPL LDAL
*2) Load FP2 with B
        LDA #3
LDBL    LDA B,X
        STA FP2,X
        DEX
        BPL LDBL
*3) Multiply with product left in FP1
        JSR $F48C
*4) Load FP2 with C
        LDX #3
LDCL    LDA C,X
        STA FP2,X
        DEX
        BPL LDCL
*5) Add with sum left in FP1
        JSR $F46E
*6) Store answer in D
        LDX #3
STDL    LDA FP1,X
        STA D,X
        DEX
        BPL STDL
```

The statement D = A/B - C can be coded by the steps:

```
*7) Load FP2 with A and FP1 with B
        LDX #3
LABL    LDA A,X
        STA FP2,X
        LDA B,X
        STA FP1,X
        DEX
        BPL LABL
*8) Divide leaving the result in FP1
        JSR $F4B2
*9) Move FP1 to FP2 and load FP1 with C
        LDX #3
LDCL    LDA FP1,X
        STA FP2,X
        LDA C,X
        STA FP1,X
        DEX
        BPL LDCL
*10) Subtract leaving the difference in FP1
        JSR $F468
*11) Store the answer in D
        LDX #3
STDL    LDA FP1,X
        STA D,X
        DEX
        BPL STDL
```

All of the overflow errors detected in the AFPUR jump to OVLOC and then to the user's code. When CLV is used before fixing a FP number, the status register bits N,V,Z can be used to determine the routine where the error occured. Table 3 demonstrates this.

Table 3

Routine	N V Z	EOR Test	STK
FIX	0 0 1	$02	0
FIXL	0 0 1	$02	0
FCOMPL	0 1 1	$42	0
FADD	0 1 1	$42	0
FSUB	0 1 1	$42	0
FMUL	1 0 0	$80	2
FDIV	1 0 0	$80	2

The routine in Listing 1 determines which minimum or maximum integer or FP number to store in FP1 before returning to the AFPUR. The contents of the status register and the AFPUR return address are stored in locations $F0, $F1, $F2.

The times for the AFPUR given at the first of this article are for the routines themselves. As can be seen by the previous examples, much more code is required to make practical use of the AFPUR. Two fairly simple programs were used to time the AFPUR. The program used to time FADD consisted of summing the floated values of the integers 1 to 32,768. The program required about 55 seconds to get $9C 7F 64 00, which is close to $5.37*10^8$. Part of the time was spent in the FLOAT routine and the summing program itself. A listing of the program is given in Listing 2.

The relative offsets of labels from SUMI are ZL = $04, INC2 = $0A, FLT = $2D, SL = $3E and IH = $49.

The program used to time FMUL consisted of summing a geometric progression with a factor close to 1.0. The multiplications involved calculating the next term in the sequence. For a factor too close to 1.0, the loss of accuracy in the floating point operations gave an incorrect answer. However, for factors like $80 3F FF F0, the answer was close enough. A listing of the program for finding (1 - R N)/(1 - R) = 1 + R + ... + R N is given in Listing 3.

The code in the AFPUR demonstrates many useful machine language programming tricks. I also think too much speed was sacrificed to get a minimum amount of code. Although the floating point format used is fairly standard, the methods used would work better with a signed magnitude floating point format so that negative operands could be easily complemented before multiplication or division.

Finally, a 3-byte floating point format would be entirely sufficient such that integers and FP numbers had the same byte order for many machine language programming applications.

Listing 1

```
            *       OVFL
         *AFPUR OVERFLOW ROUTINE
*

       .OR $3F5
       JMP OVFL
       .OR $900
SIGN   .EQ $F3
FP1    .EQ $F8
OVFL   PHP        SAVE STATUS
       PLA
       AND #$C2
       STA $F0    STORE STATUS
       PHA
       EOR #$80
       BEQ MLDV   MULTIPLY/DIVIDE
       PLA
       EOR #$42
       BEQ ADSB   ADD/SUBTRACT
*FIX   OVERFLOW--SIGN BIT IN C
       BCS FIXM
       LDX #3     INTEGER MINIMUM
       BPL OVST
FIXM   LDX #7     INTEGER MAXIMUM
       BPL OVST
*MULTIPLY/DIVIDE OVERFLOW--SIGN BIT IN SIGN
MLDV FOR SIGN
       PLA        ADJUST STACK
       PLA
*ADD/SUBTRACT OVERFLOW--SIGN BIT IN C
ADSB BCS ASM
       LDX #11    FP MINIMUM
       BPL OVST
ASM    LDX #15    FP MAXIMUM
*STORE OVERFLOW VALUE IN FP1 FOR ALL CASES
OVST   LDY #3
OVSL   LDA OTBL,Y
       STA FP1,Y
       DEX
       DEY
       BPL OVSL
*SAVE AFPUR RETURN ADDRESS
       CLV        CLEAR V STATUS BIT
       PLA
       STA $F1
       INC 1F1
       PLA
       STA $F2
       JMP ($F1)
OTBL   .HS 8E7FFF00
       .HS 8E800100
       .HS FF7FFFFF
       .HS FF800001
```

Listing 2

```
*      SUMI
*SUM INTEGERS PROGRAM
*
       FP1 .EQ  $F8
       FP2 .EQ  $F4
       M1  .EQ  $F9
SUMI   LDX #6     ZERO IH THROUGH S
       LDA #0
ZL     STA IH,X
       DEX
       BPL ZL
INC2   INC IM     INCREMENT IM AND IH
       BNE FLT
       INC IH     INCREMENT IH EVERY 256 ITERATIONS
       LDA IH
       CMP #$80
       BNE INC2
       RTS
*LOAD INTEGER INTO M1
FLT    LDA IH
       STA M1
       LDA IL
       STA M1+1
       LDA #0
       STA M1+2
       JSR $F451 FLOAT
       LDX #3     ADD INTEGER TO SUM
AL     LDA FPI,X
       STA FP2,X FP2=FP1
       LDA S.X
       STA FP1,X FP1=S
       DEX
       BPL AL
       JSR $F46E FADD
       LDX #3     SAVE SUM
SL     LDA FP1,X
       STA S,X    S=FP1
       DEX
       BPL SL
       JMP INC2
IH     .DA #0     INTEGER HIGH BYTE
IM     .DA #0     INTEGER MIDDLE BYTE
IL     .DA #0     INTEGER LOW BYTE
S      .HS 00000000        SUM
```

Listing 3

```
       GSUM
*GEOMETRIC PROGRESSION SUM
*
FP1    .EQ $F8
FP2    .EQ $F4
       .OR $800
GSUM   LDX #6     ZERO I AND S
       LDA #0
ZL     STA IH,X
       DEX
       BPL ZL
*S E T T = 1.0
       LDA #$40
       STA T
       LDA #$40
       STA T +1
       LDA #0
       STA T+2
       STA T+3
*S T   R = 1.0 - 2 (-18)
       LDA #$80
       STA R
       LDA #$3F
       STA R+1
       LDA #$FF
       STA R+2
       LDA #$F0
       STA R+3
*INCREMENT IM AND IH
INC2 INC IM
       BNE CALC
       INC IH
       LDA IH
       CMP #$80
```

```
        BNE INC2
        RTS
*LOAD S AND T FOR ADD
CALC    LDX #3
AL      LDA S,X
        STA FP1,X   FP1=S
        LDA T,X
        STA FP2,X   FP2=T
        DEX
        BPL AL
        JSR $F46E   FADD S+T
*SAVE S
        LDX #3
SL      LDA FP1,X
        STA S,X
        DEX
        BPL SL
*LOAD T AND R FOR MULT
        LDX#3
ML      LDA T,X
        STA FP1,X   FP1=T
        LDA R,X
        STA FP2,X   FP2=R
        DEX
        BPL ML
        JSR $F48C   FMUL R*T
*SAVE T
        LDX #3
TL      LDA FP1,X
        STA T,X
        DEX
        BPL TL
        JMP INC2
IH      .DA #0      LOOP INDEX HIGH
IM      .DA #0      LOOP INDEX MIDDLE
S       .HS 00000000SUM
T       .HS 00000000TERM
R       .HS 00000000FACTOR
```

A JSR FDIV was substituted in this program to get the FDIV timing.

A Machine Language Screen Print Program-for the Old (or New) PET

A program is presented which gives the user control over printing from the old PET screen. The commented assembly language program provides information on printing and can be used as a starting point for other print utilities.

Kenneth Finn
Little Old Farm
Bedford, NY 10506

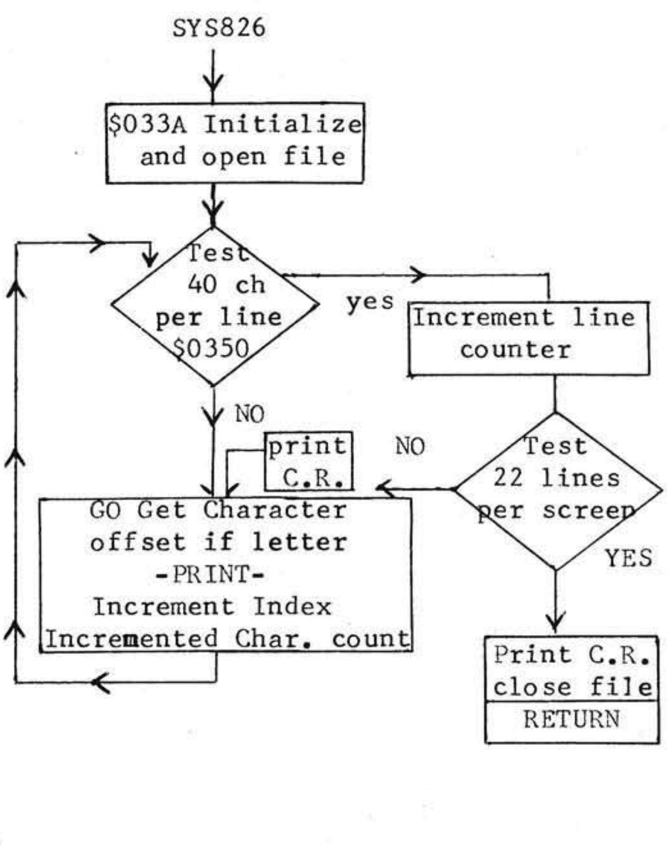
After waiting almost a year, I finally received the dot matrix, friction feed printer for the Commodore PET. The printer plugs right into the IEEE port and will print all the PET graphics as well as upper and lower case letters.

When I received the printer, I also got some very scanty documentation. What I learned from it is that you can print in your programs by using PRINT# statements after OPEN-ing the file. I also learned that you could set up the printer as the primary device by using the following code:

OPEN 4,4,0 : CMD 4

This is fine to have the printer print *everything* that would be on the screen but still not very good if you just want to print *some* things.

What I needed was a short program that would print what was on the screen when I wanted it printed. This dictated a machine language program stored in the second cassette buffer that I could call with a *SYS826* when I wanted anything printed. After some trepidation and a lot of help from other programs, the following is the result. It can reside in the second cassette buffer and will print the *top 22 lines* of the screen at *40 characters per line* when you want it.



```
033A  A9 00        LDA    #$00
033C  8D 56 02     STA    $ 0256       Printer Secondary Address
033F  8D FE 03     STA    $ 03FE       Screen Line Counter
0342  8D FF 03     STA    $ 03FF       Character counter
0345  85 DA        STA    $ DA         Index Lo-Byte
0347  A9 01        LDA    #$01
0349  8D 62 02     STA    $ 0262       GPIB File Length
034C  A9 04        LDA    #$04
034E  8D 42 02     STA    $ 0242       Logical File #
0351  8D 4C 02     STA    $ 024C       Device Number
0354  A2 04        LDX    #$04
0356  20 C9 FF     JSR    $ FFC9       Open File (i.e. Open 4,4,0)
0359  A9 80        LDA    #$80
035B  85 DB        STA    $ DB         Index Hi-Byte
035D  AD FF 03     LDA    $ 03FF
0360  C9 28        CMP    #$28         40 Characters per line, test
0362  D0 14        BNE    $ 0378       (GO)
0364  EE FE 03     INC    $ 03FE       Increment Line Counter
0367  AD FE 03     LDA    $ 03FE
036A  C9 16        CMP    #$16         22 lines/screen test
036C  F0 23        BEQ    $ 0391        (END)
036E  A9 00        LDA    #$00
0370  8D FF 03     STA    $ 03FF
0373  A9 0D        LDA    #$0D         C.R. in ASCII
0375  20 30 F2     JSR    $ F230       Print #4,C.R.
0378  18       (GO)CLC
0379  A2 00        LDX    #$00
037B  A1 DA        LDA    ($DA,X)      Get next character via indirect addr.
037D  C9 1F        CMP    #$1F          Test for Letter
037F  10 02        BPL    $ 0383
0381  69 40        ADC    #$40         Offset for Pet screen to ASCII
0383  20 30 F2     JSR    $ F230        Print #4
0386  E6 DA        INC    $ DA         Increment index Lo-byte
0388  D0 02        BNE    $ 038C       Branch if not zero
038A  E6 DB        INC    $ DB          Increment Index Hi-byte
038C  EE FF 03     INC    $ 03FF       Increment Character counter
038F  10 CC        BPL    $ 035D       Return for next character
0391  A9 0D   (end)LDA    #$0D          Ascii C.R.
0393  20 30 F2     JSR    $ F230       Print #4
0396  A9 04        LDA    #$04
0398  20 CC FF     JSR    $ FFCC       Close 4
039B  60           RTS                 Return
```

```
.   033A,039B
         | 0  1  2  3  4  5  6  7
.:  033A |A9 00 8D 56 02 8D FE 03
.:  0342 |8D FF 03 85 DA A9 01 8D
.:  034A |62 02 A9 04 8D 42 02 8D
.:  0352 |4C 02 A2 04 20 C9 FF A9
.:  035A |80 85 DB AD FF 03 C9 28
.:  0362 |D0 14 EE FE 03 AD FE 03
.:  036A |C9 16 F0 23 A9 00 8D FF
```

```
.   033A,039B
         | 0  1  2  3  4  5  6  7
.:  0372 |03 A9 0D 20 30 F2 18 A2
.:  037A |00 A1 DA C9 1F 10 02 69
.:  0382 |40 20 30 F2 E6 DA D0 02
.:  038A |E6 DB EE FF 03 10 CC A9
.:  0392 |0D 20 30 F2 A9 04 20 CC
.:  039A |FF 60 EA EA EA EA EA EA
.
```

"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

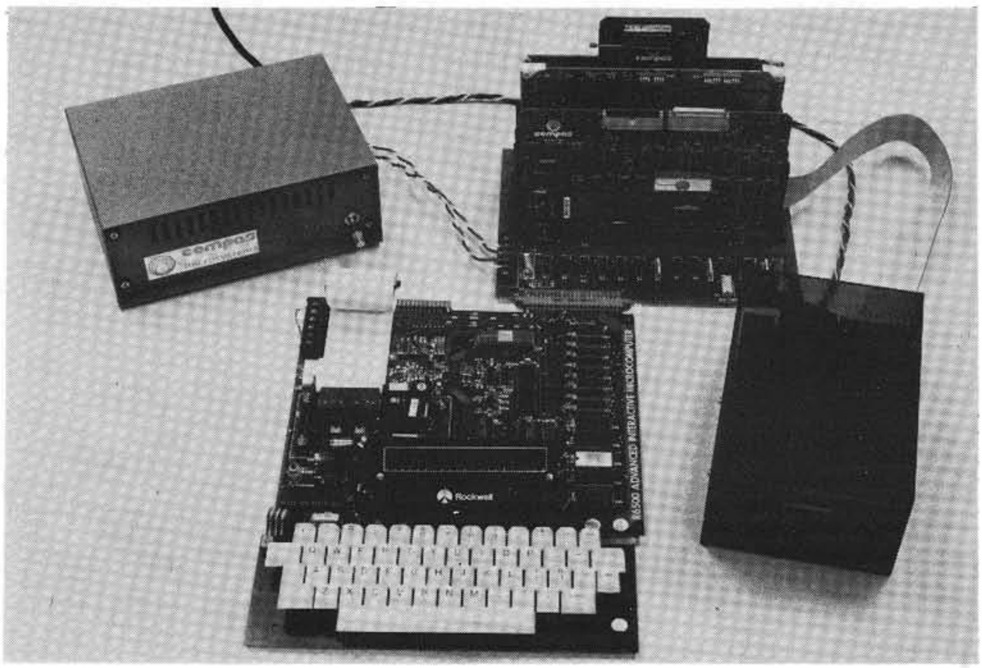
- EasyWriter is a TM of Cap'n Software, Inc.
- Apple II is a TM of Apple Computers, Inc.

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

compas
microsystems

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

DAIM



DAIM is a complete disk operating system for the ROCKWELL INTERNATIONAL AIM 65. The DAIM system includes a controller board (with 3.3K operating system in EPROM) which plugs into the ROCKWELL expansion motherboard, packaged power supply capable of driving two 5 1/4 inch floppy drives and one or two disk drives mounted in a unique, smoked plastic enclosure. DAIM is completely compatible in both disk format and operating system functions with the SYSTEM 65. Commands are provided to load/save source and object files, initialize a disk, list a file, list a disk directory, rename files, delete and recover files and compress a disk to recover unused space. Everything is complete — plug it in and you're ready to go! DAIM provides the ideal way to turn your AIM 65 into a complete 6500 development system. Also pictured are CSB 20 (EPROM/RAM) and CSB 10 (EPROM programmer) which may be used in conjunction with the DAIM to provide enhanced functional capability. Base price of $850 includes controller board with all software in EPROM, power supply and one disk drive. Now you know why we say —

There is nothing like a

DAIM

Phone 515-232-8187

Polling OSI's Keyboard

The "Polled Keyboard" technique used by OSI and others permits the user to define the function of the various keys to his own specifications, and to change them at will. Even though your keyboard may appear to be UPPER case only, it is easy to make it lower case as well.

Edward H. Carlson
3872 Raleigh Drive
Okemos, MI 49964

OSI machines come with a polled keyboard arranged in the standard 53 key format. Each key is a switch whose state (open or closed) can be ascertained under software control. Polled keyboard hardware affords maximum flexibility to the programmer. The most immediate use of the keyboard is for input to BASIC and other standard software which expects letters to be ASCII capitals but numbers and symbols to be unshifted. For this reason, the keyboard ROM in OSI machines has been programmed to yield capital letters and unshifted numbers when the SHIFT LOCK key is depressed. I am writing an editor program and desire the keyboard to act in the conventional "typewriter" manner with respect to shifting. The Program Listing gives a subroutine (lines 80 to 730) for returning the standard ASCII from the keyboard, and a driver program (lines 10 to 70) to demonstrate the subroutine. As a bonus, all the function keys (ESC, etc.) ignored by the OSI ROM are implemented to yield standard ASCII code.

When called, the subroutine loops until a key (other than SHIFT or CTRL) is depressed, then returns with the appropriate ASCII code in bits 0 through 6 of the accumulator. If CTRL was also depressed, bit 7 in the accumulator is set (1), otherwise it is reset (0). Except for the CTRL and SHIFT keys, the routine expects only one key to be depressed at a time. The routine detects the first character key depressed if several are depressed at the same time. The subroutine clobbers the X register.

Several choices have been made which you can easily change. The SHIFT LOCK key is ignored. The program works the same whether it is depressed or not. AND #7 in line 540 will enable the SHIFT LOCK key. LEFT and RIGHT SHIFT keys are made equivalent, just as on a typewriter. Since REPT is not an ASCII signal, I chose the code $00 for it arbitrarily. The BREAK key on OSI machines is hard wired to the reset line of the 65XX chips and so is not detectable by this program. ESC, RUB OUT, LINE FEED, and RETURN have ASCII codes $1B, 7F, 0A, and 0D respectively.

The keys are arranged electrically as an 8x8 matrix. I will not discuss this matrix in detail. It is shown in the OSI Graphics Manual. The first row of the matrix contains only control keys: LEFT SHIFT, RIGHT SHIFT, SHIFT LOCK, REPT, CTRL, and ESC. I call this row CTLROW and read it first. If the REPT or ESC keys are depressed, the program returns immediately with the appropriate code. If not, CTLROW is saved and rows 2 through 8 are polled for character keys. RUB OUT, LINE FEED, and RETURN are included among the character keys. When shifted, they give $20 "space" as their code. You could change line 730 so that some other ASCII function not represented on the keyboard (for example, $07 "bell") would be signaled. The polling for character keys continues in a loop until a key closure is detected. Then its ASCII code is put in the accumulator. If a SHIFT key is down, the shifted code is put in the accumulator. Then the CTRL key closure is tested. Bit 7 of the accumulator is set if appropriate.

All this happens in a millisecond or so. Many uses of the subroutine will require a check to see if the keyboard is clear of the old keystroke so that a new keystroke can be sought. The KYDONE subroutine (lines 740 to 780) accomplishes this. Once entered, KYDONE (ignoring the CTLROW keys) loops until there are no depressed keys on the keyboard, then returns.

A modified KYDONE could be a useful element in a more sophisticated keyboard program. One may wish to implement the repeat-after-a-delay mode that OSI uses in its keyboard routine. Or a two-key-rollover mode can be implemented which allows recovery from errors induced by fast, sloppy typing.

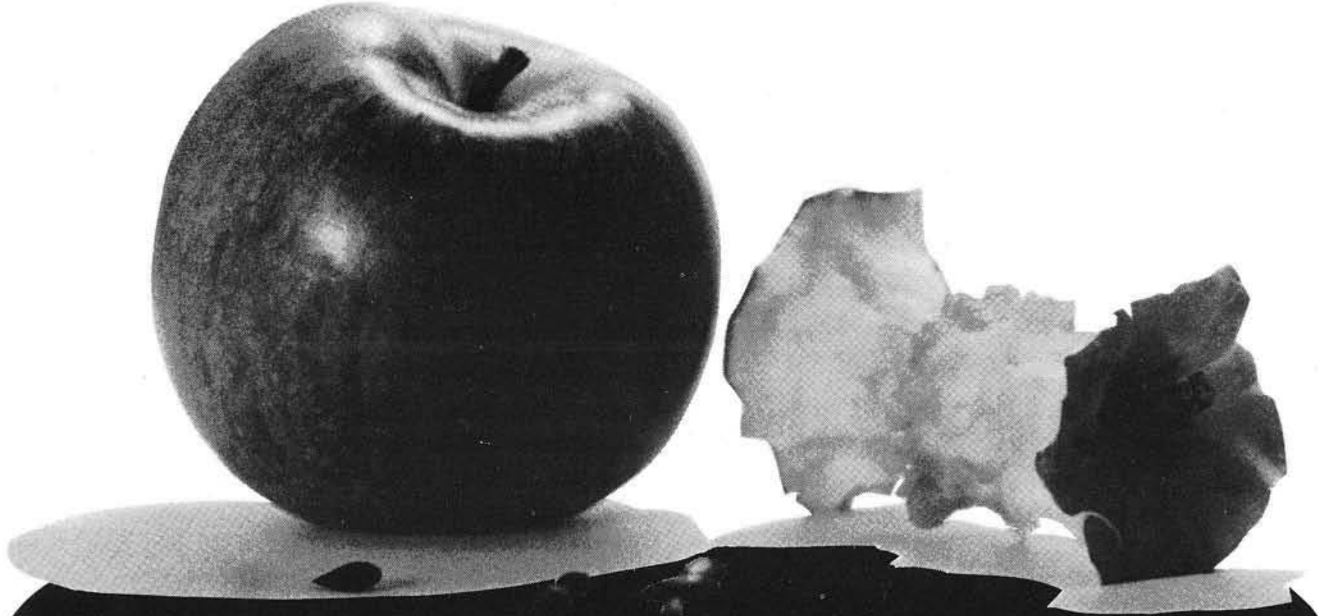
Listing 1

```
0000                    LOC=$D000+256
C200                    *=$C200
C200 2000C0  MAIN       JSR KYBD
C203 8D00D1             STA LOC
C206 20D7C0             JSR KYDONE
C209 D0F5               BNE MAIN
C20B F0F3               BEQ MAIN
C20D                    KYPORT=$DF00
C20E                    CTLROW*=*+1
C20F                    ACC*=*+1
C210                    XREG*=*+1
C000                    *=$C000
C000 A901    KYBD       LDA #1
C002 D00DF             STA KYPORT    LOOK FOR CONTROL KEYS
C005 AD00DF             LDA KYPORT
C008 8D0DC2             STA CTLROW
C00B 18                 CLC
C00C 0A                 ASL A         IS IT 'REPT'?
C00D 9003               BCC KY1       NO, BRANCH
C00F A900               LDA #0        YES
C011 60                 RTS
C012 0A      KY1        ASL A         SKIP 'CTRL' FOR NOW
C013 18                 CLC
C014 0A                 ASL A         IS IT 'ESC'
C015 9003               BCC KY2       NO, BRANCH
C017 A91B               LDA #$1B      YES
C019 60                 RTS
C01A A902    KY2        LDA #2        LOOK FOR CHAR. KEYS
C01C 8D00DF  KYBD1      STA KYPORT
C01F AE00DF             LDX KYPORT
C022 D005               BNE CHR
C024 0A                 ASL A
C025 D0F5               BNE KYBD1
C027 F0D7               BEQ KYBD      NONE FOUND, LOOK AGAIN
C029 8D0EC2  CHR        STA ACC       WHICH ROW
C02C 8E0FC2             STX XREG      WHICH COL.
C02F 18      CHR1       CLC
C030 A900               LDA #0        DECODE MATRIX
C032 4E0EC2             LSR ACC
C035 4E0EC2  SHIFT      LSR ACC
C038 B004               BCS FNDROW
C03A 6907               ADC #7
C03C 90F7               BCC SHIFT     BRANCH ALWAYS
C03E 18      FNDROW     CLC           FOUND ROW, WHICH COL.?
C03F 4E0FC2             LSR XREG
C042 4E0FC2  SHIFT2     LSR XREG
C045 B004               BCS FNDCOL
C047 6901               ADC #1
C049 90F7               BCC SHIFT2
C04B AA      FNDCOL     TAX           FOUND COL., STORE
C04C BD75C0             LDA CHRTBL,X
C04F 8D0EC2             STA ACC
C052 AD0DC2             LDA CTLROW
C055 2906               AND #6        IS IT A L. OR R. SHIFT?
C057 F00B               BEQ KY3       NO, BRANCH
C059 18                 CLC           YES

C05A 8A                 TXA
C05B 6931               ADC #49
C05D AA                 TAX
C05E BD75C0             LDA CHRTBL,X
C061 8D0EC2             STA ACC       CAP. CHAR. IN ACC
C064 AD0DC2  KY3        LDA CTLROW
C067 2940               AND #$40      IS IT 'CTRL'?
C069 F006               BEQ KY4       NO, BRANCH
C06B AD0EC2             LDA ACC       YES
C06E 0980               ORA #$80      BIT 7 OF CHAR. IS SET
C070 60                 RTS           RETURN WITH CHAR. IN A
C071 AD0EC2  KY4        LDA ACC
C074 60                 RTS           RETURN WITH CHAR. IN A

CHRTBL  .BYTE'p;/ zaq,mnbvcxk jhgfdsiuytrew'
        .BYTE'  ',$0D,$0A,'ol. ',$7F,'-:0987654321'
        .BYTE'P+? ZAQ<MNBVCXKJHGFDSIUYTREW'
        .BYTE'    OL>   =*@)(''&%$#"!'

C0D5 A9FE    KYDONE     LDA #$FE      IS KEY STROKE OVER?
C0D7 8D00DF             STA KYPORT
C0DA AD00DF             LDA KYPORT
C0DD D0F6               BNE KYDONE
C0DF 60                 RTS           KEY IS RELEASED
```



A Digital Thermometer for the APPLE II

Thermistor probes can be connected directly to the APPLE II Game I/O Connector and their output signals processed via a linearizing algorithm to produce a digital display in both degrees Celsius and Fahrenheit.

Carl J. Kershner
2123 Timberidge Circle
Dayton, OH 45459

Thermistor probes can be connected directly to the APPLE II Game I/O Connector and their output signals processed via a linearizing algorithm to produce a digital display in both Celcius and Fahrenheit.

A thermistor temperature measuring probe can be directly connected to the APPLE II computer via its built-in Game I/O Connector. This is possible since thermistors are "thermal resistors" which exhibit large resistance changes in response to a change in temperature and paddle input ports, PDL(0,1,2,&3), on the APPLE are essentially eight bit A/D converters for such variable resistance sources.

The APPLE and the thermistor are quite suited for one another since the inherent nonlinearity of the thermistor can be easily handled with a simple algorithm in software. In addition, the small current drain during the sampling cycle of the RC network on the APPLE's 553 timer closely approaches the ideal zero-power operating condition for a thermistor. Both the nonlinearity and the induced temperature due to the probing current have been particularly troublesome characteristics which engineers have had to find ways of working around when applying thermistors.

The program written in Applesoft consists of an input section, a data reduction section and a display section. The input section calls for the selection of a paddle input and two thermistor spacifications used by most manufacturers; the room temperature resistance designated as RO and a value representing the ratio of the resistance at 25°C to that at 50°C designated as RA. The selected paddle input is then read and scaled to represent the resistance value at the input port. The corresponding temperature in both degrees Celcius and Fahrenheit are calculated from the resistance via a temperature-resistance relationship:

$$R_1/R_2 = e^{\beta(1/T_1 - 1/T_2)}$$

where R_1 and R_2 are the resistances at the absolute temperature T_1 and T_2 respectively, and β is a constant for the particular thermistor material. The results are rounded to the nearest integer and displayed in a three digit format with the blanking of leading zeros and a negative sign for temperatures belwo zero.

A thermistor probe can be connected to the APPLE II by merely attaching one of its leads to the +5 volt supply, pin 1, and the other to one of the PDL ports, pins 6,7, 10, or 11 on the Game I/O connector J 14. No other components or modifications are required so long as a thermistor is chosen with a room temperature resistance and ratio which suits the temperature range and sensitivity desired for application. A 40,000 ohm thermistor with a ratio of 9 or 10 will provide at least one degree Fahrenheit sensitivity and a working range suitable for an indoor thermometer application. The best way to choose a thermistor for your particular application is to run the program using a game paddle as input, enter values for RO and the RA from a manufacturer's specification sheet, and observe the useful operating range and sensitivity of the selected thermistor. This latted proceedure demonstrates the additional usefulness of the program as an engineering design aid in selecting thermistor for other applications.

Thermistors suitable for this application can be purchased for less than five dollars from most supply houses or directly from a manufacturer. A Fenwal GA44P2 glass probe type thermistor with a room temperature resistance of 40,000 ohms and a ratio of 9.53 is a good choice for an indoor thermometer application, whereas a Fenwal GA42P2 with a room temperature resistance of 15,000 ohms and a ratio of 9.1 is a good compromise for indoor-outdoor use. It is best to house the thermistor probe in a small metal tube to protect it from mechanical damage and to provide thermal inertia to minimize effects of short term temperature transients. It is also advisable to calibrate the thermistor probes against a laboratory type thermometer, if high accuracy is desired, because the manufacturing tolerances on RO and RA values for the inexpensive probes described here are generally no better than ± 10%.

Because thermistors can be used that have relatively high resistances, transmission line and contact temperature effects can be neglected and the probes can be situated far from the computer console. Thus the APPLE II digital thermometer can perform many useful temperature monitoring tasks in and around the house.

The Fenwal products mentioned in this article can be purchased from Fenwal Electronics, 63 Fountain St., PO Box 585, Framingham, MA 01701.

```
LIST
100  REM   DIGITAL THERMOMETER FO
     R THERMISTOR PROBE(DISPLAYS
     BOTH CELCIUS &FAHRENHEIT)
110  PRINT "WHICH INPUT DO YOU WA
     NT(0,1,2,3)": INPUT NUMBER
120  PRINT "WHAT THERMISTOR CONST
     ANTS DO YOU WANT(RO,RATIO)":
      INPUT RO,RA
125 BETA = 1.7636E3 *  LOG (RA)
130  HOME : REM   CLEAR SCREEN
140  REM   PRINT TEMPERATURE SCALE
      CHARACTERS
150  GR : COLOR= 15
160  HLIN 26,27 AT 6: HLIN 26,27 AT
     7: HLIN 26,27 AT 9: HLIN 26,
     27 AT 10: VLIN 7,9 AT 25: VLIN
     7,9 AT 28
170  HLIN 34,38 AT 9: HLIN 34,38 AT
     10: HLIN 34,36 AT 14: HLIN 3
     4,36 AT 15: VLIN 9,20 AT 33
180  HLIN 26,27 AT 23: HLIN 26,27
      AT 24: HLIN 26,27 AT 26: HLIN
     26,27 AT 27: VLIN 24,26 AT 2
     5: VLIN 24,26 AT 28
190  VLIN 28,29 AT 38: VLIN 27,28
      AT 37: VLIN 26,27 AT 36: VLIN
     26,27 AT 35: VLIN 27,28 AT 3
     4
200  VLIN 28,35 AT 33: VLIN 35,36
      AT 34: VLIN 36,37 AT 35: VLIN
     36,37 AT 36: VLIN 35,36 AT 3
     7: VLIN 34,35 AT 38
210 T = 298: REM   SET T(0) AT 29
     8 DEGREES ABSOLUTE
220 RI = 589.94 *  PDL (NUMBER): REM
       READ INPUT & SCALE TO OHMS

230  IF RI = 0 THEN RI = 1: REM
      PREVENT DIVISION BY ZERO
240 TC =  INT (1 / (1 / T -  LOG
     (RO / RI) / BETA) - 272.5): REM
       CALCULATE TEMPERATURE IN D
     EGREES CELCIUS AND ROUND TO
     NEAREST INTEGER
245  IF  ABS (TC) > 999 THEN  GOTO
      220: REM   LIMIT OVERFLOWING
       DISPLAY
250 SIGN = 0
260  IF TC < 0 THEN SIGN = 15
270  COLOR= SIGN
280  HLIN 3,5 AT 29: HLIN 3,5 AT
     30: REM   DISPLAY NEGATIVE S
     IGN
290 TC =  ABS (TC)
300 J =  INT (TC / 100):I = J: REM
       SEPARATE HUNDRED'S DIGIT
310  IF J = 0 THEN J = 10: REM
     BLANK LEADING ZERO
320 X = 1:Y = 26: GOSUB 1000: REM
       DISPLAY CELCIUS HUNDRED'S
330 J =  INT ((TC - J * 100) / 10
     ): REM   SEPARATE TEN'S DIGI
     T
340  IF I = 0 AND J = 0 THEN J =
     10: REM   BLANK BOTH HUNDRED
     'S AND TEN'S LEADING ZEROS I
     F J&I ARE BOTH ZERO
350 X = 9:Y = 26: GOSUB 1000: REM
       DISPLAY CELCIUS TEN'S DIGI
     T
360 J = TC - I * 100 - J * 10: REM
       SEPARATE ONE'S DIGIT
370 X = 17:Y = 26: GOSUB 1000: REM
       DISPLAY CELCIUS ONE'S DIGI
     T
380 TF =  INT (9 * (1 / (1 / T -
      LOG (RO / RI) / BETA) - 273
     ) / 5 + 32.5): REM   CALCULA
     TE FAHRENHEIT & ROUND TO NEA
     REST INTEGER
390 SIGN = 0
400  IF TF < 0 THEN SIGN = 15
410  COLOR= SIGN
420  HLIN 3,5 AT 12: HLIN 3,5 AT
     13: REM   DISPLAY NEGATIVE S
     IGN
430 TF =  ABS (TF)
440 J =  INT (TF / 100):I = J: REM
       SEPARATE HUNDRED'S DIGIT
```

```
450  IF J = 0 THEN J = 10: REM  BLANK LEADING ZERO
460 X = 1:Y = 9: GOSUB 1000: REM  DISPLAY FAHRENHEIT HUNDRED'S DIGIT
470 J =  INT ((TF - I * 100) / 10): REM   SEPARATE TEN'S DIGIT
480  IF I = 0 AND J = 0 THEN J = 10: REM   BLANK BOTH HUNDRED'S AND TEN'S LEADING ZEROS
490 X = 9:Y = 9: GOSUB 1000: REM  DISPLAY FAHRENHEIT TEN'S DIGIT
500 J = TF - I * 100 - J * 10: REM  SEPARATE ONE'S DIGIT
510 X = 17:Y = 9: GOSUB 1000: REM  DISPLAY FAHRENHEIT ONE'S DIGIT
520  GOTO 220
1000  REM   SEVEN SEGMENT ENCODER
1010  ON J GOTO 1110,1120,1130,1140,1150,1160,1170,1180,1190,1200
1100 A = 15:B = 15:C = 15:D = 15:E = 15:F = 15:G = 0: GOTO 2000
1110 A = 0:B = 15:C = 15:D = 0:E = 0:F = 0:G = 0: GOTO 2000
1120 A = 15:B = 15:C = 0:D = 15:E = 15:F = 0:G = 15: GOTO 2000
1130 A = 15:B = 15:C = 15:D = 15:E = 0:F = 0:G = 15: GOTO 2000
1140 A = 0:B = 15:C = 15:D = 0:E = 0:F = 15:G = 15: GOTO 2000
1150 A = 15:B = 0:C = 15:D = 15:E = 0:F = 15:G = 15: GOTO 2000
1160 A = 15:B = 0:C = 15:D = 15:E = 15:F = 15:G = 15: GOTO 2000
1170 A = 15:B = 15:C = 15:D = 0:E = 0:F = 0:G = 0: GOTO 2000
1180 A = 15:B = 15:C = 15:D = 15:E = 15:F = 15:G = 15: GOTO 2000
1190 A = 15:B = 15:C = 15:D = 15:E = 0:F = 15:G = 15: GOTO 2000
1200 A = 0:B = 0:C = 0:D = 0:E = 0:F = 0:G = O:J = 0: GOTO 2000
2000  REM  SEVEN SEGMENT DISPLAY
2010  COLOR= A
2020  HLIN X + 1,X + 4 AT Y
2030  HLIN X + 1,X + 4 AT Y + 1
2040  COLOR= G
2050  HLIN X + 1,X + 4 AT Y + 5
2060  HLIN X + 1,X + 4 AT Y + 6
2070  COLOR= D
2080  HLIN X + 1,X + 4 AT Y + 10
2090  HLIN X + 1,X + 4 AT Y + 11
2100  COLOR= F
2110  VLIN Y + 1,Y + 5 AT X
2120  COLOR= B
2130  VLIN Y + 1,Y + 5 AT X + 5
2140  COLOR= E
2150  VLIN Y + 6,Y + 10 AT X
2160  COLOR= C
2170  VLIN Y + 6,Y + 10 AT X + 5
2180  RETURN
```

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 ______

Challenger II Cassette Techniques

The Challenger II has available a useful feature which allows the storage and retrieval of sequential data files on cassette using SAVE and LOAD commands in a program. This can be used to extend the size of your BASIC programs by permitting DATA to be INPUT from tape as needed.

Richard A. Lary
P.O. Box 234
Kilausa, HI 96754

Well, I knew it would happen sooner or later. I came across a program which I wished to run on my Challenger II but my 8K of memory was not enough to satisfy the program's appetite.

The desired program used several arrays to store variable values with DATA statements being used to supply the required values for the arrays. After dimensioning the arrays and entering all the required DATA statements, I discovered, much to my dismay, that these two steps had consumed nearly the entire 8K. What to do...?

After staring blankly at the CRT for several minutes wondering what I was going to do, I remembered reading something in my system documentation about entering data files from the cassette interface using the INPUT statement. This seemed to be my only hope to get the program running.

The Challenger II has a useful feature available which allows you to conveniently store and retrieve sequential data files on cassette using SAVE and LOAD commands as part of a program. The remainder of this article will describe a simple method to make use of this feature.

The first step is to store the data in a sequential file on cassette tape. Program 1 shows how this can be done. Program line 20 allows for setup and start of the recorder before the data file is recorded. Line 30 is a programmed SAVE instruction which, when executed, turns on the cassette output such that any ASCII characters listed or printed after the SAVE instruction will be output to the cassette tape. Lines 40-70 form a loop which reads data from lines 100 and 110, prints the data on the screen and outputs the data to the cassette, one variable at a time, each variable being followed by the PRINT command's carriage return.

Program 1 shows how to use DATA statements as the data source. Program 1 can be modified, as shown in Program 2 to load the data variables into an array via the keyboard and INPUT statement and then dump the array variables to the cassette. In Program 2, lines 20-60 input and store the variables in the array "D"; lines 80-110 create the sequential data file on tape as in Program 1. Line 70 allows you to set up and start your recorder before the data file is actually created.

It is very important to insure that there is no unwanted data stored on the cassette tape immediately before the start of the data file because this erroneous data will be mistaken for real data by the program which retrieves the data file. This is easy to accomplish if your recorder erases previous data before it records new data. If your recorder operates in this manner, simply allow the recorder to run in it's record mode for approximately 10 seconds before the save portion of Programs 1 or 2 are executed. Doing so will create a leader free of erroneous data before the start of the data file. If your recorder does not erase before it records, you will have to use a method compatible with your recorder which will erase a portion of the tape before the start of the data file.

After you have recorded your data file using Program 1 or 2, the next task is to retrieve the data.

Program 3 demonstrates a method for retrieving the data. Program 3 will allow you to retrieve the sample data file you created using Program 1. Line 20 dimensions the array into which the data is to be stored as it is retrieved from the data file. You must be sure to dimension the array so it will be large enough to store all your data variables. In this case, the array is dimensioned to ten since we'll only have ten variables. Line 30 is a programmed LOAD instruction which allows the INPUT statement in line 50 to accept inputs from the cassette. Lines 40-60 form a loop which reads the data file from the cassette and stores the data variables in array "D".

Line 70 stores a decimal 0 at decimal memory location 515. On the Challenger II this memory location is a flag which controls the system monitor's cassette load routine. A decimal 0 stored at the location exits the routine and a decimal 255 stored at the same location will enter the load routine. It is necessary to exit the load routine in this manner so that the program using the array variables will be executed directly, without the program stopping after the array is filled. If the program was stopped after the array had been filled to exit the load routine in the usual fashion (space bar, carriage return, etc.), it would be necessary to type RUN to restart the program. Each time you type RUN all variables are set to zero; this would include the array we just filled with data from the data file.

Lines 80-130 in Program 3 simply list the variables which were retrieved from the data file so you can see how this technique works.

In the actual use of Program 3, the program which will use the retrieved data would follow immediately after line 70.

```
10 REM WRITE DATA FILE TO CASSETTE FROM DATA STATEMENTS
20 INPUT ''SET UP AND START RECORDER...TYPE'1' TO RECORD DATA''; A
30 SAVE
40 FOR I = 1 TO 10
50 READ D
60 PRINT D
70 NEXT I
80 END
100 DATA 1,2,3,4,5
110 DATA 6,7,8,9,10
```

Listing 1

```
10 REM WRITE DATA FILE TO CASSETTE FROM AN ARRAY
20 INPUT ''HOW MANY FILES IN DATA FILE'';N
30 DIM D(N)
40 FOR I = 1 TO N
50 INPUT ''DATA''; D(I)
60 NEXT I
70 INPUT ''SET UP AND START RECORDER...TYPE '1' TO RECORD DATA'';A
80 SAVE
100 PRINT D(I)
110 NEXT I
120 END
```

Listing 2

```
10 REM RETRIEVE DATA FILE
20 DIM D(10)
30 LOAD
40 FOR I = 1 TO 10
50 INPUT D(I)
60 NEXT I
70 POKE 515,0 : REM EXIT MONITER CASSETTE LOAD ROUTINE
80 REM THE PROGRAM USING DATA ARRAY WOULD START HERE
90 REM PRINT OUT ARRAY FOR TEST OF TECHNIQUE
100 FOR I = 1 TO 10
110 PRINT D(I)
120 NEXT I
130 END
```

Listing 3

To demonstrate the retrieval of a data file, enter Program 3; place the tape with the data file you created with Program 1 into your recorder. Rewind the tape to the erased leader portion you created. Type RUN. The INPUT statement's question mark will appear to signify that the program is waiting for input from the cassette interface.

You can now start your recorder in it's playback mode, and upon the tape reaching the start of the data file the first data variable will appear following the question mark. Another question mark will appear followed by the second data variable and so on until all data has been retrieved.

When the last data variable has been retrieved, Program 3 will list the "D" array so you can see that the array now contains the data retrieved from the data file.

If you should discover that the first data variable is something other than what it should be, chances are that the leader before the data file had not been adequately erased or you may have started the tape playback somewhere other than in the erased leader portion of the tape.

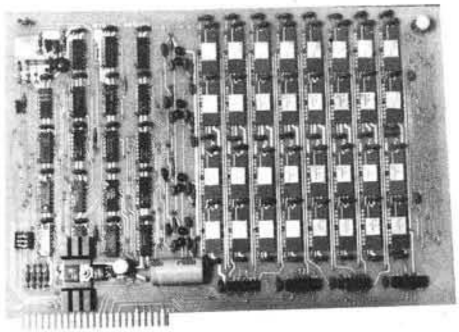
To keep these programs short and simple, I used numerical data and a single one dimensional array. By modifying these programs using nested FOR-NEXT loops in place of the single loops, you can save and retrieve data in two dimensional arrays.

Data can also be saved and retrieved in several different arrays by using one or more FOR-NEXT loops, one for each array, one after another in each of the programs. It is also possible to save and retrieve string data files by using string variables in place of the numerical which were used in these simple programs.

I have obtained reliable results using these programs. Simple modifications such as I have mentioned have allowed me the pleasure of running some programs which I have previously been unable to run.

Hopefully I have provided you with a simple but useful technique to create and retrieve cassette data files with your Challenger II.

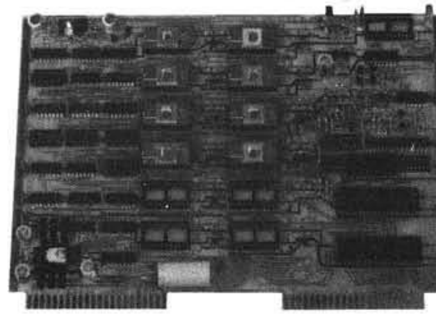
16K MEMORY



K-1016

- ADDRESSED AS CONTIGUOUS 16K STARTING AT ANY 8K BOUNDARY
- LOW POWER — 1.6 WATTS TOTAL
- K-1016A — $340 6 MONTH WARRANTY

SYSTEM EXPANSION

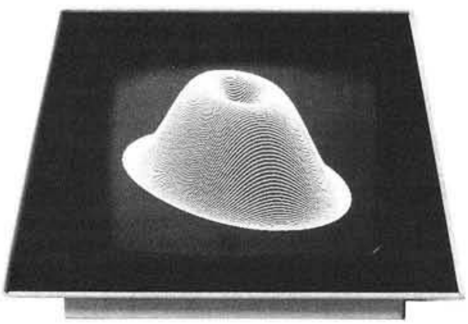


K-1012

- 12 PROM SOCKETS — 2708/TMS 2716, USES THE POWER OF ONLY 1 PROM.
- 32 BIDIRECTIONAL I/O LINES
- FULL RS-232 ASYNC SERIAL COMMUNICATIONS, 75-4800 BAUD
- PROM PROGRAMMER
- K-1012A — $295

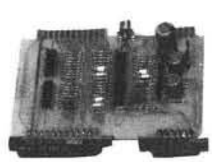
EXPANSION
FOR YOUR 6502 COMPUTER

HIGH RESOLUTION GRAPHICS



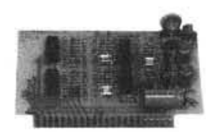
- 320 x 200 BIT MAPPED GRAPHICS
- 8K RAM AVAILABLE FOR USE
- EACH POINT INDIVIDUALLY ADDRESSABLE
- K-1008A — $240, PET — $243 (PLUS PET INTERFACE)

MULTI-HARMONIC 4 VOICE MUSIC



K-1002-2

MODEL FOR ALL PETS



K-1002

MODEL FOR KIM, AIM, SYM

- FORIER SYNTHESIZED WAVEFORMS — UP TO 16 HARMONICS
- 4 VOICES PLAY SIMULTANEOUSLY
- QUALITY D/A CONVERTER, 6 POLE FILTER AND AMPLIFIER
- HARDWARE — $40-50, SOFTWARE — $20

ALL MTU PRODUCTS ARE SUPPLIED WITH FULL DOCUMENTATION CLASSED AS "BEST IN THE INDUSTRY". MANUALS MAY BE PURCHASED SEPARATELY.

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

Beginning Bolean: A Brief Introduction to Boolean Algebra for Computerists

It makes no difference if your computer is maxi or micro, if you program in machine code, BASIC or Pascal, if you do simple games or complex real-time simulations. In the final analysis: "It's All Ones and Zeros". How these ones and zeros are used is the topic of this primer.

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

Boolean algebra, invented by George Boole in the early 1800's, is useful in programming a microcomputer for logic design, designing interface circuits, and understanding the functions of integrated circuits. The last point is illustrated by opening the *TTL Data Book* and looking for logical expressions. For example, the 7453 is described by $Y = \overline{AB + CD + EF + GH + X}$ which has to be somewhat of a mystery without any Boolean background. The name tends to scare people, but it turns out that there is a very simple approach to learning Boolean algebra, namely Boolean arithmetic. Anyone who can accept that 1 + 1 = 1, can also learn Boolean. If you're an electrical engineer or a professional computer scientist, turn to the next article; otherwise give it a try.

Beginning Boolean

Boolean arithmetic is super-simple; there are only two numbers, zero and one. There are only two operations symbolized by + and ·.* Long division is out, and there is not a minus sign in sight. Figure 1 summarizes all you need to know about the + operation which is called "OR" rather than addition.

*The symbols ∨ and ∧ frequently replace + and ·, respectively. The dot (·) is sometimes implied, that is A·B = AB.

The OR facts are read "0 or 0 equals 0," *not* "0 plus 0 equals 0." Mumble these facts to yourself several times in the privacy of your own home. That will help you get a feeling for them. It is important to relate the OR operation with the circuit in Figure 1. A and B stand for switches. If switch A is closed then A = *1; if it is open then A* = 0. The same holds for switch B. Light L is off when L = 0 and it shines when L = 1. Referring to either the OR table or the OR facts in Figure 1, it is seen that if both switches are open we have 0 + 0 = 0 so the light is off. Likewise, the fact that 1 + 0 = 1 means that if switch A is closed, but B is open, then L = 1 so the light is on. This should also be obvious from the circuit. The last two OR facts are equally obvious to anyone who has played with switches and light bulbs. Slipping in a little algebra, unnoticed of course, the circuit is summarized by the simple equation:

$$A + B = L, \qquad (1)$$

which gives the correct value for L for each of the four possible combinations of switch settings. Go back and study the table and this paragraph again if you haven't understood.

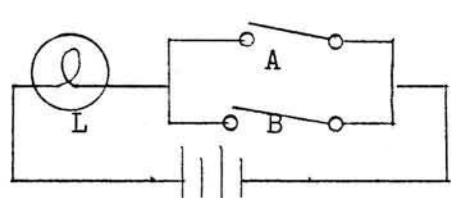
Before proceeding, a more conventional representation of the OR operation should be given, and surprisingly enough it appears in Figure 2. The information in Figure 2 is no different than in Figure 1, only the form has been changed. Actually the truth table has nothing to do with telling the truth or telling lies, but that's another story. Suffice it to say that somebody thought if everyone were

+	0	1
0	0	1
1	1	1

a) OR Table

0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 1

b) OR Facts



c) OR Circuit

Figure 1: **Summary of the properties of the OR operation.**

logical we could sort truth from lies and the world would be a better place to live. Dream on!

If you will drop your conservative image for a moment, we might let A and B stand for a string of eight digits each instead of one digit each. Also, let the OR operation be applied to the digits of A and B in sequence. An example is shown in Figure 3. Hopefully you can reproduce this calculation in your own mind. Don't do anything heavy like "carry" or "borrow," just take two digits at a time, one from A and one from B, and apply the OR rule to them.

Since you are very likely the proud owner of an 8-bit computer, an examination of the instruction set will reveal an OR command which does what has just been described. The reader is left with a few problems. Assuming that you are familiar with representing 8-bit binary numbers with hexadecimal (hex) numbers, do the following OR problems. Answers to the first two are given.

11 + FE = FF	FF + 2B =
7F + 7F = 7F	00 + 3E =
22 + 01 =	FO + 5E =
3C + 00 =	FF + 00 =
12 + 34 =	

Having experienced the intellectual rewards of having mastered the OR operation, you will want to proceed to the AND operation.

AND Away We Go

Figure 4 summarizes the AND operation in the same fashion as Figure 1 treated the OR operation. The AND circuit is a series circuit, requiring that both A *and* B be on (hence the name) for the light L to light. This is in contrast to the OR circuit which lights if either A *or* B is on.

Notice that "ANDING" works the same way as old-fashioned multiplication with no weird results like 1 + 1 = 1 which we obtained with the OR. The AND facts are read "0 and 0 equals 0," or "1 and 1 equals 1." The equation which describes the circuit is:

$$A \cdot B = L, \qquad (2)$$

the truth of which may be verified by substitution and comparison with the simple series circuit. As before, a more conventional representation of the AND operation is given by a truth table and logic symbol shown in Figure 5.

As before, A and B may be taken to represent 8-bit numbers, and an example of such an AND operation is given in Figure 6. Your microprocessor's instruction set will include an AND command which takes two 8-bit words and ANDs them, as illustrated in Figure 6.

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

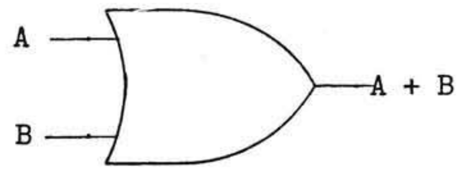


Figure 2: **Truth Table and logic symbol for the OR operation.**

```
  01010011
+ 11000101
= 11010111
```

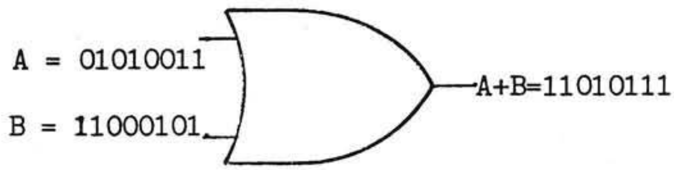


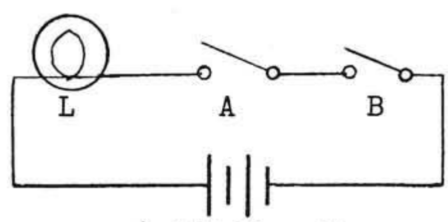
Figure 3: **Example of an 8-bit OR operation.**

•	0	1
0	0	0
1	0	1

a) AND Table

0 • 0 = 0
0 • 1 = 0
1 • 0 = 0
1 • 1 = 1

b) AND Facts



c) AND Circuit

Figure 4: **Summary of the properties of the AND operation.**

A	B	A•B
0	0	0
0	1	0
1	0	0
1	1	1

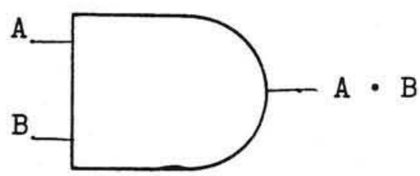


Figure 5: **Truth Table and logic symbol for the AND operation.**

```
  01010011
• 11000101
= 01000001
```

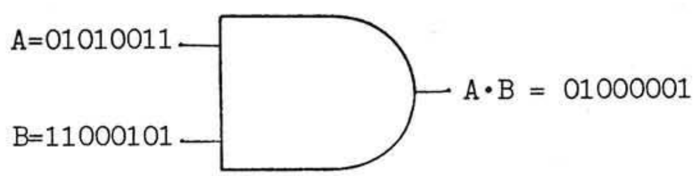


Figure 6: **Example of an 8-bit AND operation.**

Try the following AND problems where the 8-bit binary numbers have been represented in hex.

FF·11 = 11	33·BC =
7F·7F = 7F	00·3E =
OF·37 =	FO·37 =
80·FF =	80·11 =
55·40 =	

Everyone Loves a COMPLEMENT

There is another Boolean process called complementation or negation. It is a simple but very important idea. All complementation facts are summarized in Figure 7, using a truth table and the logic symbol. Clearly, complementation simply changes 0 to 1 and 1 to 0. The bar over the variable indicates complementation, and the inversion circle at the end of the triangle symbolizes complementation. A triangle without such a circle performs no inversion and in computer literature usually refers to a buffer. $\overline{A}$ is read as "not A."

It's Not a Complement to be Called EXCLUSIVE

There is another operation in Boolean mathematics which is the exclusive or, which for our purposes we shall call EOR, and we shall give it the symbol ⊕ . I didn't really lie to you in the second paragraph when I said there were only two operations. It turns out that the EOR operation can be accomplished with ORs and ANDs, but it is somewhat simpler to think of it as a third operation. Figure 8 gives the truth table for EOR, the logic symbol, and an example of an 8-bit computation. Later we shall see how it can be implemented with ORs and ANDs. Try some of the problems given earlier for OR and AND operations only do EOR operations. For example, FF ⊕ 11 = EE and 7F ⊕ 7F = 00. Doing some EOR arithmetic may lead you to some interesting but important generalizations. In any case, in an EOR operation, if the digits are alike, the result is 0; but if the digits are different, the result is 1.

If You're Exclusive You May Get A Complement

In checking over the instructions for my microprocessor, I find that no complementation command exists. Wow! Here is a fundamental Boolean concept which is missing. If you played with some EOR problems you may have already discovered how to produce a complement with the EOR operation. Suppose we deal with 1 digit numbers for the time being. Consider $1 \oplus A$ for a starting problem. Clearly if A is 1, $1 \oplus 1 = 0$ which is the complement of A. On the other hand, if A is 0, then $1 \oplus 0 = 1$, which is the complement of A. So both possibilities give the complement of A. Summarizing,

$$1 \oplus A = \overline{A} \quad (3)$$

If A is an 8-bit binary number represented by a hex number, equation (3) becomes $FF \oplus A = \overline{A}$. In other words, if you want the complement of a number, do an exclusive or with it and a word containing all ones.

Designing Circuits — A Simple Application

You now know how to OR, EOR, AND and COMPLEMENT. You would like to know how to do something with what you have learned, right? Let's start with a simple problem; namely, constructing an electric coin flipping game. Actually, no coins will be flipped, but the principle is the same. Our machine will have two switches A and B. When the switches are the same a light will light, when they are different the light will be off. This, of course, corresponds to the case of both coins being the same (light on) or one coin coming up heads while the other comes up tails.

The first step in the design is to construct a truth table (sometimes called a closure table) for the system. We require that when A and B are both on the light is lit, when they are both off the light is lit, but when they have different settings (one on, the other off) the light is off. The truth table we would like to implement is clearly the one shown in Figure 9. This is constructed by first listing the four possible combinations for two switch settings, that is 00, 01, 10, and 11. For each of these switch settings the desired value of L is listed, completing the truth table. It is seen that when the switches "match" then L = 1, otherwise it is 0.

The next step in the design is to develop the Boolean equation which is equivalent to the truth table. This is accomplished by the following two steps:

1. Identify all rows with a 1 in the last column. AND the elements making up these rows, complementing those with a 0 in the row.

A	$\overline{A}$
0	1
1	0

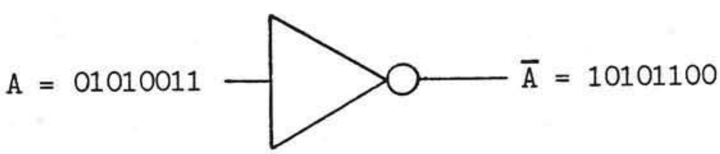


Figure 7: **Truth Table and logic symbol for complementation.**

A	B	A ⊕ B
0	0	0
1	0	1
0	1	1
1	1	0

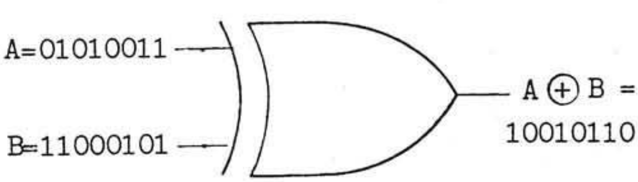


Figure 8: **Truth Table and logic symbol for EOR operation.**

A	B	L
0	0	1
0	1	0
1	0	0
1	1	1

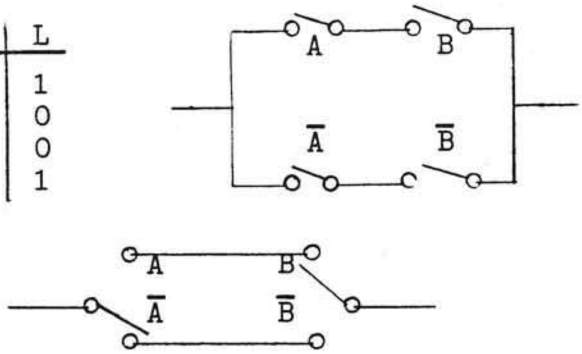


Figure 9: **Truth Table and circuit diagram for the coin matcher.**

2. OR the products obtained in step 1 and set the result equal to the variable in the last column.

For the first step above and the truth table of Figure 9, we obtain the products $\overline{A}\cdot\overline{B}$ and $A\cdot B$ from the first row and the last row, respectively. Step two then gives the equation,

$$\overline{A}\cdot\overline{B} + A\cdot B = L, \quad (4)$$

which is the equation for the circuit. It is important for you to verify, by substituting in the various values of A and B given in Figure 9, that this equation does not in fact give the desired values of L.

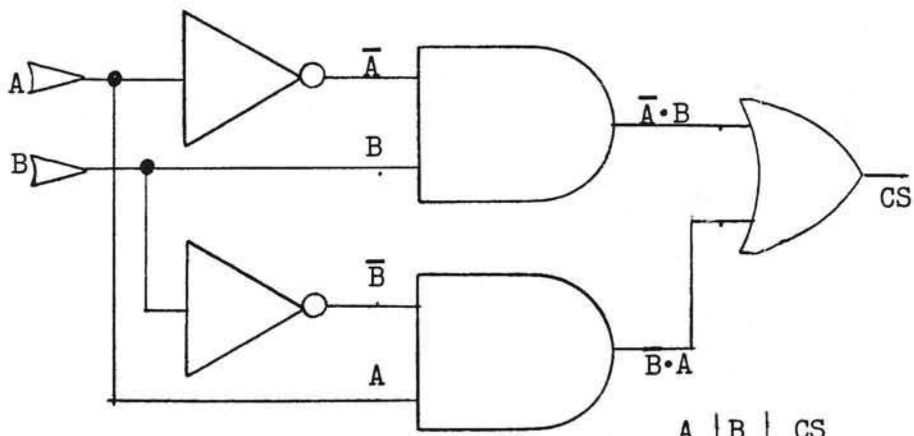
The final step in the design is to construct a circuit which is equivalent to the equation (4). An examination of this equation indicates that we need two parallel branches, one containing $\overline{A}$ in the series with $\overline{B}$, the other containing A series with B. This circuit is shown in Figure 9. The battery and the light have been omitted for simplicity. Clearly the circuit could be constructed with two SPDT switches.

It is important to realize that the steps we took to design this particular circuit are perfectly general, that is, they are the same steps one would go through to design any logic circuit. Of course, with more switches the truth tables and the equations get more complicated. For example, with three switches our truth table would have 8 rows; four switches, 16 rows, and so on. Since this article is not meant to be an exhaustive (although you may feel that way) explanation of Boolean algebra, we will not proceed to more complex situations. For those you might want to pick up a textbook on digital electronics or computer science. But if you made it this far, you shouldn't have much trouble with the textbooks.

One other design study will illustrate several points. Suppose we require a logic 0 signal on a chip select pin when either of two other signals, call them A and B, are either both logic 0 or both logic 1. When A and B have opposite logic levels our chip select must be 1. This is clearly an artificial situation which originated in my mind and not in a computer interface circuit, but it illustrates a point. The truth table which fits the description demanded by the design is shown in Figure 10. Following the steps outlined earlier we find that the Boolean equation which implements the truth table is

$$\overline{A}\cdot B + A\cdot\overline{B} = CS. \quad (5)$$

An examination of the truth table will show that it is identical to the EOR table, and thus we have proved that EOR can be implemented with ORs and ANDs. A second point worth mentioning is that if A and B were single digit binary numbers, the value of CS is the value of the least significant digit in the binary sum of A and B. Thus equation (5) is also part of an adding circuit. If A and B are ANDED the correct value for the "carry" part of the binary addition is also produced. Together these circuits form what is a "half adder." Figure 10 also gives the logic symbol implementation of equation (5).



A	B	CS
0	0	0
0	1	1
1	0	1
0	0	0

***Figure 10:* Truth Table and logic circuit for A·B + A·B = CS**

TO BE OR $\overline{\text{TO B}}$ = 1:
Some Boolean Theorems

Once a truth table, closure table, or function table has been constructed for a particular design problem and the Boolean equation has been derived using the steps outlined in the previous section, then one usually tries to simplify the equation to minimize the number of integrated circuits which will be required for the circuit. Here is where Boolean algebra really becomes useful, for it is the theorems of Boolean algebra which allow complex looking equations and circuits to be simplified.

Because Boolean theorems are quite easy to understand and prove, and because they look different from the equations of real number algebra, a few of the simple theorems are listed in Table 1. An interesting property of Boolean algebra is illustrated by the first two columns. Note that column two can be obtained from column one by replacing all + signs with •, if you replace all 1's with 0's.

It is quite easy and it is good practice to prove these theorems. They are proved by the method of exhaustion, namely all possible values for the variable are tried. For example, the first theorem can be proved by reasoning that A can be 1 or 0. If it is 1, then from the OR table $1+1=1$. If it is 0, then from the OR table $1+0=1$. So, for all possible values of A, $1+A=1$ and the theorem is proved. All the theorems in the first two columns may be proved in this manner. Theorems involving two variable A and B are usually proved using the four possible values for

$1 + A = 1$	$1\cdot A = A$	$A + B = B + A$
$0 + A = A$	$0\cdot A = 0$	$A\cdot B = B\cdot A$
$A + A = A$	$A\cdot A = A$	$\overline{A + B} = \overline{A}\cdot\overline{B}$
$A + \overline{A} = 1$	$A\cdot\overline{A} = 0$	$\overline{A\cdot B} = \overline{A} + \overline{B}$

$$A\cdot B + A\cdot\overline{B} = A \qquad \overline{\overline{A}} = A$$
$$A + A\cdot B = A$$
$$A + (B\cdot C) = (A+B)\cdot(A+C)$$
$$A\cdot(B+C) = (A\cdot B)+(A\cdot C)$$

***Table 1:* Some Basic Theorems from Boolean Algebra**

A and B, namely

A = 0011 = 03Hex

B = 0101 = 05Hex.

Theorems with three variables require 8 possible combinations to exhaust all the possible arrangements:

A = 10101010 = AAh

B = 11001100 = CCh

C = 11110000 = FØh

The purpose of expressing these in hex is so that you can try to prove the theorems on your computer, and at the same time get some experience in performing logical operations. All of the problems given earlier can also be solved on your computer. Theorems three and four in column three are the famous DE Morgan's theorems with which one can connect the ANDs and ORs with the NANDs and NORs of the real world.

To conclude, go back for a minute to that Boolean expression in the first paragraph. Suppose we ask what the value of Y will be if A and B are both 1. Using what you have just learned, the answer should be easy. If A and B are both 1 the AB (the dots are frequently omitted in AND operations) is 1. From the theorem in the first column of the theorem table it is clear that 1 OR anything is 1. Consequently, no matter what the other variables are, the value under the inversion or complementation bar is 1 if A and B are both 1. Inverting the 1 gives 0, so the answer is 0. Also if X = 1, then Y = 0, regardless of the states of the other variables.

I hope that you had some fun with this weird arithmetic. Perhaps your mind got bent out of shape as an added feature. But my main hope is that some of the mystery in those words "Boolean Algebra" has disappeared. I'll leave you with a homework problem. Draw the logic diagram to implement a full-adder, then expand it to handle 8-bit numbers. Finally, implement it with software on your 8-bit machine, and check your answer using the ADD instruction. Have some fun and get some books on digital electronics and/or computer science and dig into this stuff.

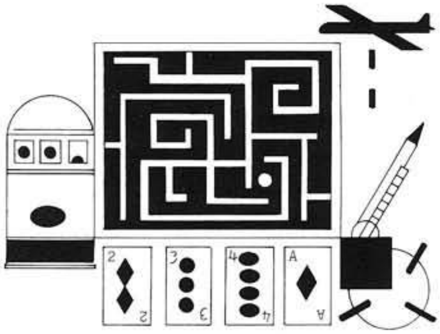
Bibliography

Digital Principals and Applications, Malvino, A.P. and Leach, D.P., McGraw-Hill, 1975, New York.
Introduction to Computer Science Scheid, F., Schaum's Outline Series, McGraw-Hill, 1970, New York.
The Bugbook V, Rony, Peter R., Larsen, David G., and Titus, Jonathan A., E&L Instruments, Inc., Derby, Connecticut, 1977.

Our Latest Arrivals

Education, games, business, electronics—you'll find what you have been looking for in this batch of new programs from Instant Software.

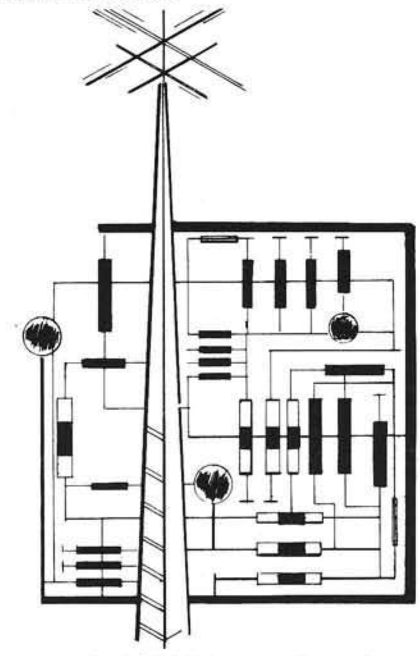
PET*



PET DEMO I You can give yourself, your family, and your friends hours of fun and excitement with this gem of a package.
•**Slot Machine**—You won't be able to resist the enticing messages from this computerized one-armed bandit.
•**Chase**—You must find the black piece as you search through the ever-changing maze.
•**Flying Pheasant**—Try to shoot the flying pheasant on the wing.
•**Sitting Ducks**—Try to get your archer to shoot as many ducks as possible for a high score.
•**Craps**—It's Snake Eyes, Little Joe, or Boxcars as you roll the dice and try to make your point.
•**Gran Prix 2001**—Drivers with experience ranging from novice to professional will enjoy this multi-leveled race game.
•**Fox and Hounds**—It's you against the computer as your four hounds try to capture the computer's fox.
For true excitement, you'll need a PET 8K. **Order No. 0035P $7.95.**

CODE NAME:CIPHER Empires have risen and fallen because of secret codes and code breakers. Now you can enjoy that same feeling of intrigue and discovery with the Code Name: Cipher package. Included in the package are:
•**Memory Game**— Would you like to match your memory against the computer's? You can with the Memory Game.
•**Codemaster**—You and another player can compete to see who will be the "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. The computer will give you hints as to how close your guesses are.
•**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, if you want to pit your wits against the faultless logic of the computer, then Code Name: Cipher is for you. For the 8K PET. **Order No. 0112P. $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.**

DECORATOR'S ASSISTANT This integrated set of five programs will compute the amount of materials needed to redecorate any room, and their cost. All you do is enter the room dimensions, the number of windows and doors, and the base cost of the materials. These programs can handle wallpaper, paint, panelling, and carpeting, letting you compare the cost of different finishing materials. All you'll need is a PET 8K. **Order No. 0104P $7.95.**

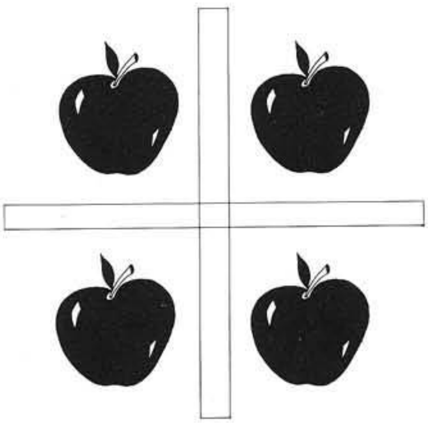
ACCOUNTING ASSISTANT This package will help any businessman solve many of those day-to-day financial problems. Included are:
•**Loan Amortization Schedule**—This program will give you a complete breakdown of any loan or investment. All you do is enter the principal amount, interest rate, term of the loan or investment, and the number of payments per year. You see a month-by-month list of the principal, interest, total amount paid, and the remaining balance.
•**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. Your computer will display a list of the item's lifespan, the annual depreciation, the accumulated depreciation, and the remaining book value. This package requires the PET 8K. **Order No. 0048P $7.95.**

ELECTRONIC ENGINEER'S ASSISTANT
Now you can use your computer to analyze designs for filter circuits and microstrip transmission lines.
•**Network Analysis**—Your computer can help you design and analyze four-terminal AC networks. Just enter the input load impedance, component values, and the frequency range. Your computer will analyze the circuit and display the gain, the real input impedance, and the imaginary input impedance throughout

Instant Software™ Inc. Peterborough, N.H. 03458 603-924-7296

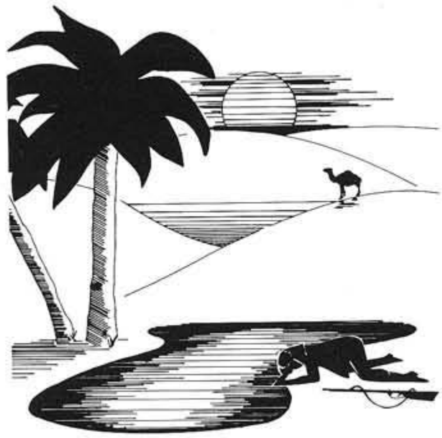
the entire frequency range. There's even an optional plotting routine for graphing frequency response.
•**Microstrip**—This program can help you design microstrip transmission lines for printed circuit boards and other mediums. You can get either the dimensionless width-to-height ratio of the supporting medium or the impedance of the system. For the PET 8K. **Order No. 0085P $9.95.**

APPLE**



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 24K of memory. **Order No. 0025A $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. **Order No. 0094A $7.95.**

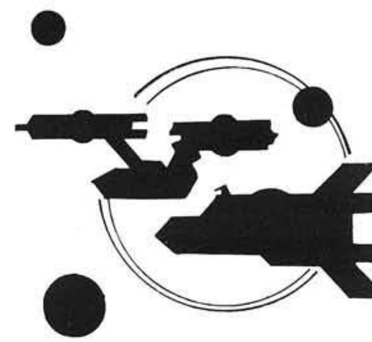


SAHARA WARRIORS Now you can enjoy all the gritty realism of desert warfare with the Sahara Warriors package.
•**Commando**—You must send your commandoes to trap a German general and cut him off from his troops. You'll have the choice of two levels of difficulty.
•**French Foreign Legion**—The battalions 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. This game has four different versions, which even include an occasional sandstorm.

NOTE: Both programs in this package require an Apple 8K and Integer BASIC. The French Foreign Legion program requires the use of the game paddles. **Order No. 0080A $7.95.**

ACCOUNTING ASSISTANT (see the description for the PET version 0048P) This package requires the Apple 16K. **Order No. 0088A $7.95.**

Coming Soon . . .
Disk programs for your Apple



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. **Order No. 0096A $7.95.**

*A trademark of Commodore Business Machines Inc.
** A trademark of Apple Computer, Inc.

For a free catalog listing over 200 programs write: Instant Software Catalog Dept., Peterborough, N.H. 03458.

If the store nearest you does not stock Instant Software, use this order blank to purchase your software directly, or call Toll-Free 1-800-258-5473.

Order Your Instant Software today!

Name ______

Address ______

City ______ State ______ Zip ______

☐ Check ☐ Money Order ☐ VISA ☐ master charge ☐ AMERICAN EXPRESS

Credit card# ______ Exp. date ______

Signature ______ date ______

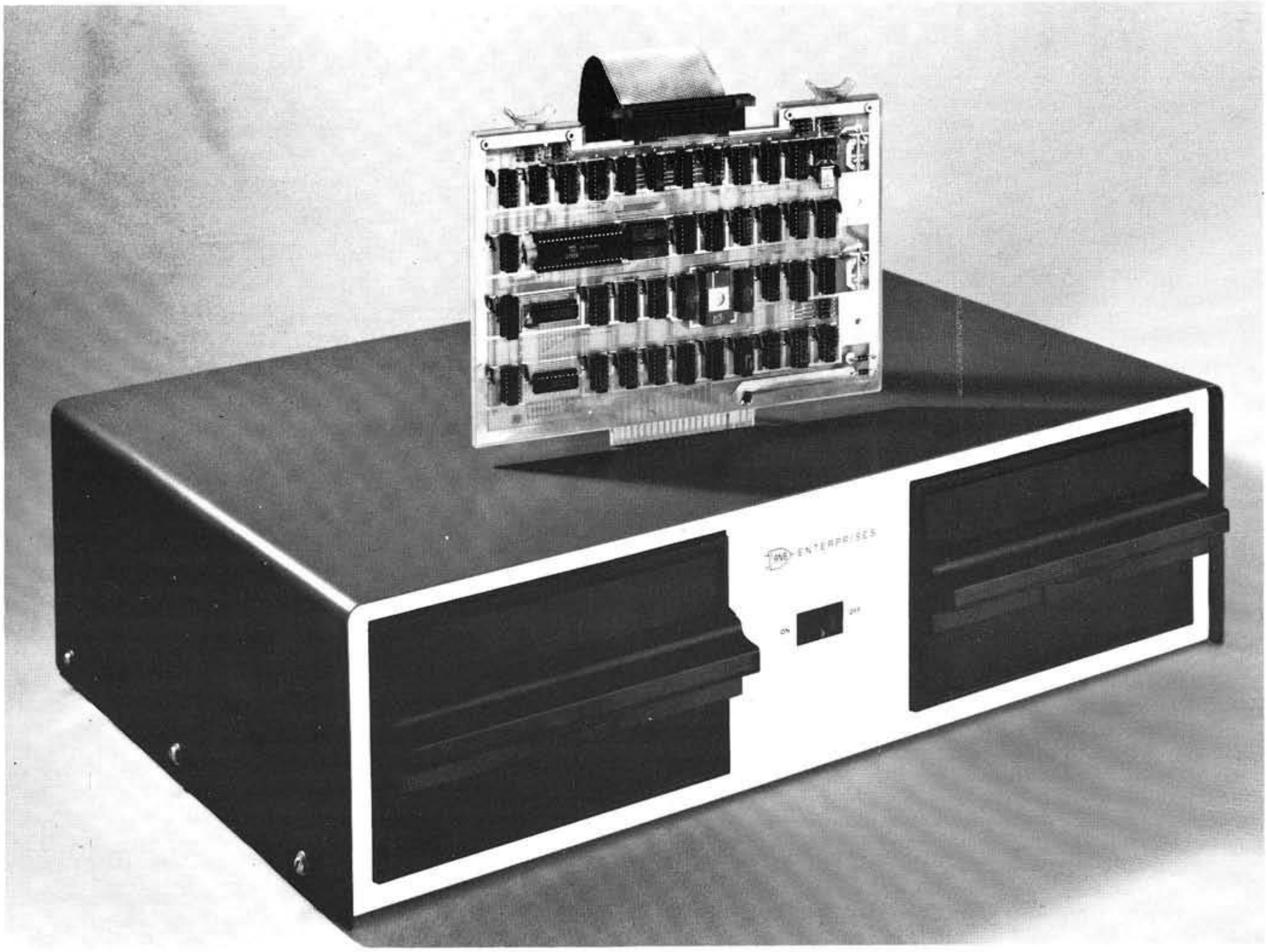
Quantity	Order No.	Program name	Unit Cost	Total cost
		Shipping		$1.00
			Total order	

Mail to:
Instant Software™
Peterborough NH 03458 USA Dept. 60B0

Prices Valid In USA Only

Instant Software™ Inc. Peterborough, N.H. 03458 603-924-7296

VAK-7 FLOPPY



The VAK-7 Disk System was specifically designed for use with AIM-65, SYM-1, and KIM-1 Microcomputer Systems. The VAK-7 will plug directly into the VAK-1 Motherboard or with the addition of regulators into the KIM-4* Motherboard. The VAK-7 is a complete full size (8") FLOPPY DISK SYSTEM. This system will READ, WRITE, and FORMAT both IBM SINGLE and DUAL DENSITY diskettes. Single-Sided is standard and Dual-Sided is optional. Our Single-Sided drives are set up so they can be converted at a later date to Dual-Sided by the factory, if your storage needs increase.

The VAK-7 system occupies a 4K address space. The system has a 1K block of D.M.A. RAM as a transfer buffer. Also, a 1K block of RAM reserved for D.O.S. pointers, drive status, and catalog information. The remainder of the address is occupied by the resident 2K MINI-DOS. This MINI-DOS is a complete set of subroutines to Read, Write, and Format.

DISK SYSTEM

The MINI-DOS is not a high level Disk Operating System, but contains all the elementary subroutines for implementation of a high level DOS. Since all the functions are in subroutines, the implementation of this system into a dedicated system is simplified.

MINI-DOS SUBROUTINES

- Block Move
- Seek Track
- Recalibrate Disk
- Sense Interrupt Status
- Read/Write Data
- Read/Write Deleted Data
- Format Disk/Test For Bad Sectors
- Initialize Disk
- Physical Copy (Disk to Disk)
- Self Test

The VAK-7 is an interrupt driven system, which uses the $\overline{IRQ}$ vector. Since this is an interrupt driven system, your system processor is only used to move data into or out of the 1K of DMA RAM, issue the command, and check status at the end of the disk operation. Your system processor is free to do other functions, during disk operations because the intelligent disk controller will complete the operation without tying up valuable processor time.

The VAK-7 System comes complete with Disk Controller Board, Interconnecting Cable, a Cabinet with Power Supply (for two Disk Drives) and one Disk Drive. The VAK-7 Controller can handle up to Four Drives.

SPECIFICATIONS:

- Completely assembled, tested, and burned in.
- Occupies address $9000-$9FFF for AIM-65, $9000-$9FFF for SYM-1, or $E000-$EFFF for KIM-1.
- IBM Format; Single Density (128 bytes/sector); Dual Density (256, 512, or 1024 bytes/sector).
- All IC's are in sockets.
- Fully buffered address and data bus.
- Standard KIM-4*BUS (both electrical pin-out and card size).
- Designed for use with a regulated power supply, but has provisions for adding regulators for use with an unregulated power supply.
- Dimensions: Board—10" wide x 7" high (including card-edge)
  Cabinet—23.5" wide x 6.5" high x 16" deep.
- Power Requirements: +5V DC @ 2 Amps.
  117V AC 60Hz @ 2 Amps.

**KIM-4 is a product of MOS Technology/C.B.M.*

PRICE:

Single-drive, 1-sided	$1,299.00
Dual-drive, 1-sided	1,898.00
Single-drive, 2-sided	1,499.00
Dual-drive, 2-sided	2,398.00

Plus Shipping	UPS	Mail (APO, FPO)	International
Single Drive	12.00	32.00	Shipped Air Freight. Freight charges collect.
Dual Drive	16.00	44.00	

For Alaska and Hawaii, use mail rates.

We manufacture a complete line of high quality expansion boards. Use reader service card to be added to our mailing list, or U.S. residents send $1.00 (International send $3.00 U.S.) for airmail delivery of our complete catalog.

RNB ENTERPRISES INCORPORATED

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

master charge VISA

Program Checksum Calculator

Whenever you key in a program in machine code, there is some doubt as to whether or not it has been entered correctly. One minor error is all it takes to ruin a program. A technique and program is presented to help overcome this problem on any 6502 computer.

Nicholas Vrtis
5863 Pinetree SE
Kentwood, MI 49508

I decided to write this program for selfish reasons. My hope is that everybody who transfers or writes programs for distribution will use it. The purpose of the program is to compute a sixteen bit checksum by adding up all the bytes in a program. This really isn't a totally new idea. Most methods of transferring programs or data external to a CPU use some sort of checksum routine. Even parity is really a one bit checksum. There is one major method of program transfer which makes almost no use of checksums. That method is listings published in magazines. One of the reasons is probably that noone has published a simple, general program to compute a checksum.

This program was written and tested on a SYM-1, but was designed to be as machine independent as possible. It should run on almost any 6502 system. There are only two monitor routines used, both of which are probably available in most monitors. The routine called OUTBYT outputs the contents of the 'A' register as two HEX digits. Just in case this routine isn't readily available on your system, I have also included a version of one at the end of the program. OUTCHR is a routine that outputs the contents of 'A' as a character; and unless your are using the HEX output routine, it is only used to output a space as a separator. The program does not assume that any registers are saved or restored by either monitor routine. It is completely relocatable (as is HEXOUT), and only uses four bytes of page zero memory. If you want to, you could even put it into an EPROM. The work area for the two byte checksum accumulator is obtained from the stack to avoid any more page zero requirements.

The theory and method of operation is simple. The starting address of the program to be summed is placed at locations $00 and $01 (low order first as usual). The ending address is placed at locations $02 and $03, and the program is started. The program will output an intermediate checksum after the end of each page of the summed program (i.e., each time the high order byte of the current address changes). This intermediate value would be useful in narrowing down the address of where a mistake lies. For a long program there might be a few of these intermediate sums; but then, that is also when they would be most helpful. Remember they still only narrow it down to 256 bytes (or less for the first and last values).

The program starts by zeroing the checksum accumulator by pushing two zero bytes onto the stack. The stack register then points to the next available stack location, which is actually $100 plus the stack register value in absolute address terms. The checksum accumulator is therefore at locations $101 and $102 plus the stack register value, since the stack register starts at $1FF and works toward $100 each time a value is pushed onto the stack. Transferring the stack register to the 'X' register lets us add directly to these two bytes. It would be possible to accomplish the same thing with Pulls and Pushes, but wouldn't have been as interesting. For the output of the last checksum value, the values are Pulled from the stack to keep the stack register the same before as after.

In addition to the aforementioned selfish motives, I have found the program to have other more mundane uses around the computer room. If you suspect a program is modifying itself (possibly by accident), compute the checksum before and after execution. If the checksums are the same, you can be reasonably confident that the program hasn't changed. I say reasonably confident because a simple checksum is not total proof that a program didn't change. If I add to one location and subtract the same amount from another, the checksum will still come out the same. It is orders of magnitude more accurate than guessing or eyeballing a memory dump though.

By the way, the checksum for this program ($04 to $48) is $1AAA! For the HEXOUT subroutine it is $08F8.

```
                        SYM MONITOR ENTRY POINTS USED
0049                    OUTBYT *       $82FA   OUTPUT °A° AS 2 HEX DIGITS
0049                    OUTCHR *       $8A47   OUTPUT °A° AS ASCII
0000                           ORG     $0000
0000 00                 STRTAD =       $00     PROGRAM STARTING ADDRESS LOW
0001 00                        =       $00     PROGRAM STARTING ADDRESS HIGH
0002 00                 ENDAD  =       $00     PROGRAM ENDING ADDRESS LOW
0003 00                        =       $00     PROGRAM ENDING ADDRESS HIGH
0004 A9 00              PGMSUM LDAIM $00       ZERO CHECKSUM ON THE STACK
0006 48                        PHA
0007 48                        PHA
0008 BA                 ADDIN  TSX             MOVE STACK POINTER TO INDEX
0009 A0 00                     LDYIM $00       MAKE SURE Y REG IS ZERO
000B 18                        CLC
000C B1 00                     LDAIY STRTAD GET A PROGRAM BYTE
000E 7D 02 01                  ADCX   $0102    ADD TO CHECKSUM
0011 9D 02 01                  STAX   $0102
0014 90 03                     BCC    NOCARY
0016 FE 01 01                  INCX   $0101
0019 E6 00              NOCARY INC    STRTAD ADVANCE TO NEXT PROGRAM BYTE
001B D0 15                     BNE    CHKEND GO CHECK FOR END OF PROGRAM
001D E6 01                     INC    STRTAD +01 OTHERWISE BUMP TO NEXT PAGE
001F BD 02 01                  LDAX   $0102    OUTPUT INTERMEDIATE CHECKSUM ALSO
0022 48                        PHA             SAVE LOW ORDER ON STACK
0023 BD 01 01                  LDAX   $0101    TO AVOID SAVING X
0026 20 FA 82                  JSR    OUTBYT OUTPUT HIGH ORDER PART
0029 68                        PLA
002A 20 FA 82                  JSR    OUTBYT THEN LOW ORDER
002D A9 20                     LDAIM $20       SPACE
002F 20 47 8A                  JSR    OUTCHR AND A SPACE AS SEPARATOR
0032 A5 01              CHKEND LDA    STRTAD +01 CHECK IF TO END OF PROGRAM
0034 C5 03                     CMP    ENDAD  +01
0036 D0 04                     BNE    CHKNDA
0038 A5 00                     LDA    STRTAD
003A C5 02                     CMP    ENDAD
003C 90 CA              CHKNDA BCC    ADDIN    LESS MEANS MORE TO GO
003E F0 C8                     BEQ    ADDIN
0040 68                        PLA             ELSE GET HIGH ORDER OF CHECKSUM
0041 20 FA 82                  JSR    OUTBYT ENO NEED TO PRESERVE THINGS
0044 68                        PLA
0045 20 FA 82                  JSR    OUTBYT
0048 00                        BRK             CAUSE WE ARE DONE
ID=
HEXOUT    MICRO-WARE ASSEMBLER 65XX-1.0 PAGE 01
```

```
0200                    HEXOUT ORG    $0200    RELOCATABLE
0200 48                        PHA             SAVE EXTRA COPY FOR SECOND HALF
0201 4A                        LSRA            SHIFT HIGH 4 BITS TO LOW ORDER
0202 4A                        LSRA
0203 4A                        LSRA
0204 4A                        LSRA
0205 C9 0A                     CMPIM $0A       CHECK IF >9
0207 90 02                     BCC    HEXOTA WILL SET CARRY IF >=
0209 69 06                     ADCIM $06       PLUS 7 WILL OFFSET TO GET ASCII 'A'
020B 69 30              HEXOTA ADCIM $30
020D 20 47 8A                  JSR    OUTCHR NOW OUTPUT THE FIRST HEX CHARACTER
0210 68                        PLA             GET ORIGINAL BACK
0211 29 0F                     ANDIM $0F       ONLY WANT 4 LOW ORDER BITS NOW
0213 C9 0A                     CMPIM $0A       SAME CONVERT TO ASCII
0215 90 02                     BCC    HEXOTB
0217 69 06                     ADCIM $06
0219 69 30              HEXOTB ADCIM $30
021B 4C 47 8A                  JMP    OUTCHR LET THIS GUY DO THE RETURN
ID=
```

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

EXTRA LONG, HEAVY DUTY cables and connectors

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

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

Ask the Doctor

A technique to solve a problem in the AIM TTY service and a program for easy tape retrieval on the KIM are presented.

Robert M. Tripp
The Computerist, Inc.
P.O. Box 3
S. Chelmsford, MA 01824

The Serial TTY port on the AIM 65 works in a very similar fashion to that of the KIM described by Ben Doutre in MICRO issue number 12, May, l979. The instructions for achieving synchronization at any baud rate (bits per second) are to switch the slide key from KB (keyboard) to TTY (Teletype), next press RESET and type DELETE on the teletype. The routine DETCPS measures the duration of the start bit and since the start bit is at logic level zero, then DELETE is a suitable character to use, because it consists of eight logic 1 signals following the start bit. Actually, about half of the characters on the keyboard would do just as well provided they *start* with a logic level 1 after the start bit. (Note the least significant bit is transmitted first.) The duration of the start bit is stored in CNTH 30 and CNTL 30 as explained previously.

An ordinary teletype works at 110 baud, but I have available a VDU which has a baud rate switch and will work at 110, 300, 600, 1200, 2400, 4800 baud. I tried all the rates with the AIM 65 hooked up and they worked perfectly, except for the 1200 baud. At this rate, no synchronization was achieved and only garbled rubbish appeared on the screen. This was of some concern to me because it was my intention to set up a three-way link between the VDU, the AIM 65 and a PRIME 400 computer and the PRIME was to operate *only* 1200 baud.

On pages 9-29 of the AIM 65 User's Guide the contents of CNTH 30 (at $A417) and CNTL 30 (at $A418) are listed for several baud rates. It is essential that the correct values can be entered by hand if synchronization does not work. The values quoted are only approximate, because the on-board crystal will not be exactly 4 M Hz and the devices linked up may vary slightly in baud rate. A check at the rates that did synchronize, showed some slight differences. Entering the values suggested for 1200 baud, namely, $02 for CNTH 30 (high byte) and $FD for CNTL 30 (low byte), produced perfect functioning of the VDU.

The Monitor Program Listing, page 10, shows that the programming between locations $E11B and $E144 times the start bit, using the timer T2 in the 6522 VIA dedicated to the monitor. The timer is started when the input goes low (start of bit) and when it goes high again (end of bit), the high byte count is read first and slightly later then the low byte count. The counter T2 counts downwards and the counters are initialized with $FF in both. The latch for T2 has $FF put into it permanently during initialization at reset, shown at program addresses $E067 to $E0D0. The value $FF is written into the high byte count at program address $E126, which also writes the latch value into the low byte count.

When the counter is read, the high and low bytes are complimented with $FF to get the duration of a bit in multiples of 1 mS. The counter start is delayed slightly by the time taken by the program instructions $E110 to $E126 and the low byte reading is delayed after the end of the start bit between instructions $E 129 and $E 136. Thus the count is too high and evidently requires reducing by 44 (decimal). This is carried out by the routine PATCH 1 at program address $FE 7C. If the low byte count happens to be less than 44 (it is, only for the 1200 baud rate), then the high byte count will require reducing by one also.

The carry bit is set correctly before the SBC instruction for 44. Unfortunately, the fault is that the carry bit is never examined to see if it is unset and then CNTH 30 reduced by one, either in the subroutine or back in the main program. Programming could be written by a user to overcome this problem, but it is not really worth the effort of loading. The simplest course is to attempt the synchronization, switch back to the AIM keyboard, then change $A417 from $03 to $02 using the M instruction. This gives the optimum count for the particular device connected to the serial interface.

On attempting the automatic synchronization at 1200 baud the values obtained for CNTH 30 and CNTL 30 were respectively $03 and $FO. The high byte value is different enough to cause the timing of each bit to be about 33 percent high, so the bits could not possibly be recognized by the VDU. I decided to investigate the reason for this, in case the VDU was faulty.

Roger Heal
Department of Chemistry
University of Salford
Salford, M5 4WT
England

Fast Tape Retrieval

Although I use the routine primarily for data retrieval for the TVT-6 video interface, it can readily be adapted for many other applications. Basically, it works as follows:

Press STart: Tape recorder starts and tape is read. When seven segment display reads 0000 A9, then

Fast Tape Retrieval

Although I use the routine primarily for data retrieval for the TVT-6 video interface, it can readily be adapted for many other applications. Basically, it works as follows:

Press STart: Tape recorder starts and tape is read. When seven segment display reads 0000 A9, then

Press GO: Tape recorder stops and the information from tape is displayed on a monitor. For next display file, again

Press STart, and so forth.

This provides a two button data retrieval system. The files on tape consist of ASCII data loaded into 0200 to 03FF. This, of course, can vary for other applications and could contain any type of data which would not interfere with the program located at the bottom of page zero. A small gap between files is convenient.

Hardware Interface: Consists of the relay circuit shown in figure 1. PB0 is used as the control port. The tape recorder must be set in the **PLAY** position.

Software Interface: Several items must be inserted memory in order to bring up the system. First, if used with the TVT-6, the SCAN program must be loaded beginning at 1780. Next, 17F9 must contain 00. This will allow continual reading of files without regard to ID numbers. Finally, the NMI vector must be entered:

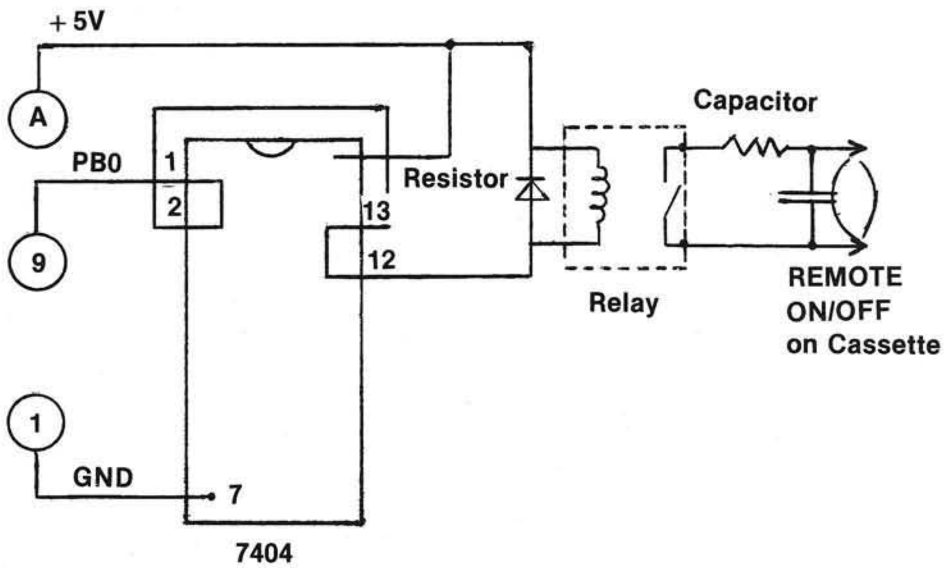
17FA 0B and **17FB 00**

The program utilizes the fact that on a proper tape read, the monitor returns at location 0000 XX. Now, 0000 is a memory location - a location which can be programmed like any other. I use this location to begin the tape/stop/display program. The **ST** button provides an **NMI** which points to the tape/start/read program.

```
0000 A9 01     GO   LDAIM $01    TURN TAPE OFF
0002 4D 03 17       EOR   $1703  BY TOGGLING CONTROL BIT
0005 8D 03 17       STA   $1703
0008 4C AD 17       JMP   SCAN   JUMP TO SOME PROGRAM

000B A9 01     ST   LDAIM $01    ST INTERRUPT
000D 4D 03 17       EOR   $1703  TURN TAPE ON
0010 8D 03 17       STA   $1703  BY TOGGLING CONTROL BIT
0013 4C 73 18       JMP   $1873  JUMP TO TAPE READ

17FA 0B             =     $0B    POINT TO 000B
17FB 00             =     $00    FOR ST KEY
```



Ronald Kushnier
3108 Addison Court
Cornwells Heights, PA 19020

ASK the DOCTOR is presented as an opportunity for you to get information about the AIM, SYM, or KIM out to your fellow computerists. If you have a major article, a good program, a discussion of various features, or whatever, that is long enough for an article, then by all means, submit it as an article and get paid! If, however, you have some short comments, ideas, facts, warnings, etc. that you feel others will be interested in, but which are too short for an entire article, then send them to: ASK the DOCTOR, P.O. Box 6502, Chelmsford, MA 01824. You will not get paid for these "tidbits", but you will get full credit for them.

HDE inc.

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

HUDSON DIGITAL ELECTRONICS INC.

THE HDE DISK SYSTEM.

HERE'S WHAT ONE USER HAS TO SAY . . .

REPRINTED BY PERMISSION FROM THE 6502 USER NOTES - ISSUE NO. 14

PRODUCT REVIEW of the HDE DISC SYSTEM by the editor.

A number of you have asked for details about the HDE full size disc system.

The system is based around the SYKES 8" drive with the 6502 based intelligent controller.

This drive is soft sectored, IBM compatible, and single density which lets you store about a quarter megabyte of data on a disc.

The system software, called FODS (File Oriented Disc System), manages sequential files on the disc much the same way files are written on magnetic tape - one after another. When a file is deleted, from a sequentially managed file system, the space that the file occupied is not immediately reallocated, as in some disc operating systems. As it turns out, this can be an advantage as well as a disadvantage since deleted files on the FODS system can be recovered after the file has been deleted. (This has saved my sanity more than once!) Of course when you want to recover some of the disc space taken up by a number of these deleted files, you can simply re-pack or compress the disc and all the active files will be shifted down until there are no deleted files hanging around using up space.

FODS has this ability to repack a disc.

When saving and loading in FODS you work with named files, not track and sector data or I.D. bytes. This makes life a lot easier. I've seen some disc systems where you have to specify track and sector info and/or I.D. bytes. What a pain that can be!

If you just want to save a source file temporarily, you can do that on what's known as "scratch-pads". There are two of these on a disc, "scratch-pad A" and "scratch-pad B", each of these temporary disc files can hold up to 16K or if "B" is not used, "A" can hold one file up to 32K in length. The only files that can be temporarily saved on scratch pad are files that have been built using the system text editor.

Being a dyed in the wool assembly language programmer, I really appreciate the FODS text editor! This line oriented editor is upwards compatible with the MOS/ARESCO editor but includes about everything you could ask for in a line editor. There is a full and semi-automatic line numbering feature, lines can be edited while they are being entered or recalled and edited later, strings can be located and substituted, the line numbers can be resequenced, the file size can be found, the hex address of a line can be known and comments can be appended to an assembly file after it has been found correct. Oops! I forgot to say lines can also be moved around and deleted. This isn't the complete list of FODS editor commands, just the ones that immediately come to mind.

Another very powerful feature of the system is the ability to actually execute a file containing a string of commands. For example, the newsletter mailing list is now being stored on disc. When I want to make labels, I would normally have to load each letter file and run the labels printing program. But with FODS, I can build up a "JOB" file of commands and execute it.

The job file in turn calls each lettered label file in and runs the label printer automatically. The way computers are supposed to operate right?

Here's a listing of the job file I use to print mailing labels:

```
:LIS PRTLBL
0005 LOD A:RUN %LABEL:LOD B:JMP.E000:
LOD C:JMP.E000:
0010 LOD D:JMP.E000:LOD E:JMP.E000:
LOD F:JMP.E000:
0015 LOD G:JMP.E000:LOD H:JMP.E000:
LOD I:JMP.E000:
0020 LOD J:JMP.E000:LOD K:JMP .E000:
LOD L:JMP.E000:
0025 LOD M:JMP.E000:LOD MC: JMP.E000:
LOD N:JMP.E000:
0030 LOD O:JMP.E000:LOD P:JMP .E000:
LOD R:JMP.E000:
0035 LOD S:JMP.E000:LOD T:JMP .E000:
LOD V:JMP.E000:
0035 LOD S:JMP.E000:LOD T:JMP .E000:
LOD V:JMP.E000:
0040 LOD W:JMP.E000:LOD XYZ: JMP.E000:
0045 LOD EXCH:JMP.E000:LOD COMP:
JMP.E000:
```

Remember the MOS/ARESCO assembler I reviewed several issues ago? Well HDE went and fixed up all the problem areas that I mentioned in the review and then took it several steps further. The HDE assembler is an honest to goodness two-pass assembler which can assemble anywhere in memory using multiple source files from the disc. The assembler is an optional part of the system.

If you're the kind of person (as I am) who enjoys having the ability to customize, modify, and expand everything you own - you'll enjoy the system expansion abilities FODS has to offer. Adding a new command is as simple as writing the program, giving it a unique three letter name and saving it to disc. Whenever you type those three letters the system will first go through its own command table, see that its not there and then go out and read the disc directory to see if it can find it. If it's on the disc it will read it in and execute it. Simple right? I've added several commands to my system and REALLY appreciate having this ability. Some of the things I've added include a disassembler, an expanded version of XIM (the extended machine language monitor from Pyramid Data), Hypertape, and a number of system utilities which make life easier. By the way, to get back to the system, all you need to do is execute a BRK instruction.

HDE also provides a piece of software that lets you interface Microsoft 9 digit BASIC to their disc system. The software allows you to load the BASIC interpreter itself from disc as well as saving and loading BASIC Programs to and from the disc. This particular version of the software doesn't allow for saving BASIC data but HDE mentioned that this ability may be possible with a future version.

The first thing I do with a new piece of software after I get used to using it is try to blow it up. I did manage to find a weak spot or two in the very first version of FODS (a pre-release version) but the later, release version has been very tight.

The standard software that is included with the system consists of the disc driver software, the system text editor and the BASIC software interface. Several command extensions may also be included. All the necessary stuff like a power supply, the KIM-4 interface card, and all cables and connectors are included. It took me about 45 minutes to get things up and running the first time I put the system together.

Admittedly, a dual full size disc system from HDE is probably beyond the means of most hobbyists but if you or your company is looking for a dynamite 6502 development system, I would recommend this one. I've used the Rockwell System 65 while I was at MOS and feel that dollar for dollar, feature for feature, the HDE system comes out on top. The only place the HDE system falls short when stacked up next to the System 65 is in the area of packaging. At this point, there is no cabinet for the disc drives available from HDE.

So far, I've got nothing but good things to say about HDE and their products. Everything I've received from them has been industrial quality. That includes their documentation and product support. I'm very impressed with what I've seen from this company so far and quite enthusiastic over what my KIM has become since acquiring the disc system and its associated software.

ERIC

THANK YOU MR. REHNKE!

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

Clocking KIM

While you probably are not going to use your $180 KIM to replace your $18 digital clock, a lot can be learned about proper use of the KIM and 6502 by building a clock as an exercise. This example includes use of the IRQ interrupt, driving the display, and calculating time. The program is intentionally NOT optimized, providing a challange for the reader.

Ronald A. Guest
12153 Melody Drive, 204
Denver, CO 80234

When I first laid eyes on KIM, something inside my head screamed out at me. It said, "That six digit display was just made to be a clock display." Not being the type of person who likes getting yelled at, I decided to look into the idea.

The idea may seem trivial or worthless, but the design of a clock program is both challenging and educational. When I first started this project I had only owned my KIM-1 slightly more than a month, and I had not made use of some of the KIM's features (such as the interval timer and interrupt capabilities). I had read the MOS Technology manuals carefully, but some things had to be learned the hard way -like the fact that decimal mode does not effect increments and decrements, PB7 of the 6530-03 must be wired to IRQ on the 6502 by the user, and how to use the KIM's displays. I learned through experience.

A clock program is not totally useless, either. Outside of being a good program for keeping the processor busy, it also could be valuable when run with other programs. If, for instance, you had your KIM connected to an A/D converter, the clock program would enable you to take readings at specified times during the day. Or, when you are going to be away from home, your microprocessor could turn lights, radios, etc. on and off as a deterrent to burglary. And, if you had to scrape together your last dollar to buy your KIM, and the old alarm clock just croaked, you can build an alarm to hang on your micro with parts from your tool box (see KIM-1 manual, page 57). You would then be the proud owner of the most intelligent alarm clock on the block.

A project such as this is an excellent way to become familiar with the features of your microprocessor. And, although there are several obstacles to be overcome, none are too difficult to surmount, given a little thought. The following discussion, intended for the KIM-1 could also serve as a guide for the development of a clock program on a similar system.

The first difficulty encountered in designing a clock program is a parallel processing problem: how to scan a display (or execute some other process) while at the same time, count the microseconds as they whiz by. Parallel processing on the KIM-1 can be achieved by the use of the 6502's interrupt capabilities. And since one of our processes is a simple counting mechanism, we can use the interval timer on the 6530-03 as our second "processor". The next two problems revolve around the interval timer.

The KIM's interval timer is only capable of timing intervals of 0.261102 seconds or less (with a 1MHz crystal). Problem number two results because of this. We need to simulate, through software, an interval timer able to time intervals of up to one second. This can be accomplished by writing a value(s) into the interval timer until a certain number of interrupts has occurred, and then updating the time. But, it is not quite that simple.

Which brings us to problem number three. We want to be as efficient as possible, which means we want to interrupt normal processing as little as possible. Thus, as large a value as possible should be written into the timer. The problem, the most difficult of all, is; what value(s) to write where, and how many times. The discussion must now become a little more detailed. Keeping efficiency in mind, we want to delay the maximum value between interrupts as many times as possible, without exceeding one second. This means that we should write a $FF into the ÷1024 location three times. This will give a delay of (3*255*1024)μs, or 0.216640 seconds less than one (1 – 0.783360).

As you can see, this value is less than the maximum interval. The largest value, less than 0.216640 seconds, that we can write to the interval timer is (211*1024)μs, or 0.216064 seconds. About now you are probably thinking, "Oh! Holy Bit Bucket, will this never end?" But don't fuse a power supply, this tedious process is coming to an end. Now, let's see. If we write a 9 into the ÷64 location, that will give us a delay of 576μs, and a grand total of exactly one second. Success! We have accomplished our task. But, wait. We've got some software overhead to consider, right? Right. The software which simulates this interval timer, and the software which updates the time must be taken into consideration. The amount of time allocated to the execution of this software will vary slightly for different programs. But 32μs seems to be a sufficient amount of time. So we want our final delay to be 544μs (instead of 576). We can easily achieve this by writing 68 into the +8 location.

What do we have so far? We have three delays of (255*1024)μs, one delay for (211*1024)μs, and one delay of (68*8)μs; giving us a grand total of 999,968 μs. (Although not the only combination of intervals which yeild one second, this is the most efficient from the standpoint of the interrupted process.) Now, we need to take a look at the software necessary to simulate this timer.

The code needed to simulate a one second interval timer is given in listing 1. The first section of code gives us the delay of (3*255*1024), assuming that "NUMDLY" is initialized to one (1). The second group of instructions gives the delay of (211*1024), and the third group gives the delay of (68*8). To assure that your clock program will be accurate, you should determine how much time your software will actually require. When calculating the amount of software overhead for your particular program, remember that the execution time of instructions executed after the timer has been written into, should not be included. This is because the timer will have already started counting down, resulting in an overlap of the countdown and instruction executions. Taking Listing 1 as an example, and assuming "NUMDLY" is less than three (3), all instructions before "INCDLY" should be included, but the increment and restore instructions should not be, since they are executed after the timing begins.

The code in Listing 1 would be contained in an interrupt program, which consists of instructions to do the following;

1. save all registers on the stack,

2. determine if one second has elapsed,

   if not then (5), else,

3. add one second to the time,

4. reset "NUMDLY" and the timer, and,

5. restore the registers from the stack.

A block diagram of the process is given in Figure 1. The coding for the interrupt program occupies about 96 bytes of memory (most of it is used to update the time).

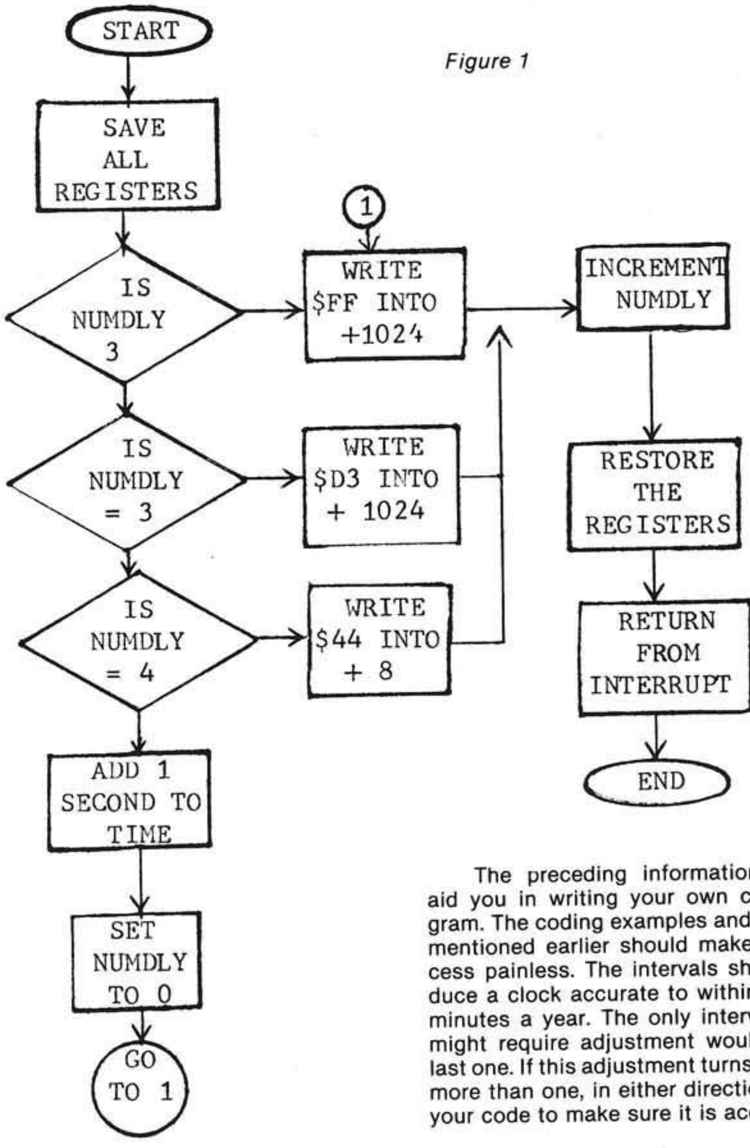


Figure 1

The updating code must:

1. add 1 to the seconds, if not 60 then (4), else zero seconds and

2. add 1 to the minutes, if not 60 then (4), else zero minutes and

3. add 1 to hour, if 13 (or 25 for a 24 hour clock) set hour to 1, else

4. continue.

Since time uses decimal digits, it is necessary to load the byte into the accumulator and perform an add; an increment would not be decimal. The code to update the seconds is given in Listing 2 as an example.

The preceding information should aid you in writing your own clock program. The coding examples and intervals mentioned earlier should make the process painless. The intervals should produce a clock accurate to within about 5 minutes a year. The only interval which might require adjustment would be the last one. If this adjustment turns out to be more than one, in either direction, check your code to make sure it is accurate.

Let me mention a few more things to help you avoid trouble. Don't forget that any program which is to run coincidentally with the clock program you must initialize the interval timer and counter (NUMDLY) at the outset. Also, after you have loaded the clock program into memory, you must also store the starting address of the program in the IRQ interrupt vector locations. And, last but not least, connect the PB7 pin of the 6530-03 to the IRQ pin of the 6502 (A-15 to E-4).

The clock program I have described contains the basic features of any digital clock. It could be expanded to keep track of the month, day, year, and/or just about anything else you could want. If you are willing to spend the time, the clock can be as accurate as the hardware will allow. Theoretically, my clock program should be accurate to within less than 2 minutes a year. In practice, I set the program by WWV and let it run for three days straight. At the end of this time, the seconds were still clicking by, right on the nose. This should be enough accuracy for all but the most exacting and finicky of home-brewers. (I hope no one is going to run their clock program for a solid year.) Hopefully, the information information in this article, and the program itself, will be as useful to others as it has been to me. Good Luck!!

Listing 1

```
                        INTERRUPT SERVICE ROUTINE TO RUN A REAL-TIME
                         CLOCK ON A MOS TECHNOLOGY KIM-1.
                        WRITTEN BY : RONALD A GUEST.
                        COPYRIGHT 1976  BY  RONALN A. GUEST
0020                           ORG    $0020
0020                    NUMDLY *      $000D
0020                    SECNDS *      $000C
0020                    MINUTS *      $000B
0020                    HOURS  *      $000A
0020 48                        PHA           SAVE REGISTERS ON STACK
0021 8A                        TXA               .
0022 48                        PHA               .
0023 98                        TYA               .
0024 48                        PHA               ..
                               DELAY DETERMINED BY VALUE IN NUMDLY.
                               IF DELAY IS < 3 THEN DELAY 255 * 1024
0025 A5 0D                     LDA    NUMDLY
0027 C9 03                     CMPIM  $03
0029 10 0A                     BPL    DELAYF
002B A9 FF              MAXDLY LDAIM  $FF
002D 8D 0F 17                  STA    $170F
0030 E6 0D              INCDLY INC    NUMDLY INCREMENT COUNT AND GO RESTORE
0032 4C 80 00                  JMP    RESTOR     ..
                               IF NUMDLY <> 3 BRANCH TO ONETST
0035 D0 08              DELAYF BNE    ONETST
0037 A9 D3                     LDAIM  $D3    IF = 3 THEN DELAY 211*1024
0039 8D 0F 17                  STA    $170F
003C 4C 30 00                  JMP    INCDLY
                               IF NUMDLY > 4 THEN ONE SECOND HAS ELAPSED
003F C9 04              ONETST CMPIM  $04
0041 D0 08                     BNE    ONESEC
0043 A9 44                     LDAIM  $44    IF = 4 THEN DELAY 68*8
0045 8D 0D 17                  STA    $170D
0048 4C 30 00                  JMP    INCDLY
                               ONE SECOND HAS ELAPSED, SO, INC STORED TIME
004B F8                 ONESEC SED           USE DECIMAL MATH.
004C A5 0C                     LDA    SECNDS SET SECONDS BYTE
004E 18                        CLC
004F 69 01                     ADCIM  $01    ADD DECIMAL 1.
0051 85 0C                     STA    SECNDS
0053 C9 60                     CMPIM  $60    IF < 60 THEN FINISHED
0055 30 22                     BMI    RESET
0057 A9 00                     LDAIM  $00    ZERO SECONDS
0059 85 0C                     STA    SECNDS
                               INCREMENT MINUTES
005B A5 0B                     LDA    MINUTS GET MINUTES BYTE
005D 69 00                     ADCIM  $00    ADD IN CARRY FROM PREVIOUS COMPARE
005F 85 0B                     STA    MINUTS
0061 EA                        NOP           FOR TIMING
```

```
0062 C9 60            CMPIM $60     IF < 60 , FINISHED
0064 30 13            BMI   RESET
0066 A9 00            LDAIM $00     ZERO MINUTES
0068 85 0B            STA   MINUTS

                     INCREMENT HOURS

006A A5 0A            LDA   HOURS
006C 18               CLC           THIS TIME CLEAR CARRY (FOR TIMING)
006D 69 01            ADCIM $01
006F 85 0A            STA   HOURS
0071 C9 25            CMPIM $25     WOULD BE 13 FOR 12 HOUR CLOCK
0073 30 04            BMI   RESET
0075 A9 01            LDAIM $01
0077 85 0A            STA   HOURS

                     RESET TIMER AND NUMDLY

0079 A9 00    RESET   LDAIM $00
007B 85 0D            STA   NUMDLY
007D 4C 2B 00         JMP   MAXDLY

                     RESTORE REGISTERS

0080 68       RESTOR  PLA
0081 A8               TAY
0082 68               PLA
0083 AA               TAX
0084 68               PLA
0085 40               RTI


                     THIS PROGRAM DISPLAYS THE HOURS, MINUTES, AND SECONDS

0200                  ORG   $0200

0200          NUMDLY  *     $000D
0200          CONVD   *     $1F48
0200          TIME    *     $000A

0200 A9 01            LDAIM $01     INITALIZE INTERRUPT COUNTER
0202 85 0D            STA   NUMDLY
0204 A9 FF            LDAIM $FF     INITALIZE DIVIDE BY 1024 COUHTER
0206 8D 0F 17         STA   $170F
0209 58               CLI           ENABLE INTERRUPTS
020A A9 7F            LDAIM $7F     INITALIZE DISPLAY PORT
020C 8D 41 17         STA   $1741
020F A2 09    START   LDXIM $09     X REG POINTS TO DIGIT TO BE DISPLAYED
0211 A0 00            LDYIM $00     Y REG IS INDEX TO TIME BYTE TO BE DISPLAYED

                     THIS IS THE START OF THE LOOP
                     THE PRECEEDING CODE IS INITALIZATION

0213 C0 04    LOOP    CPYIM $04     IF Y REG IS >= 4 THEN JUMP BACK TO START
0215 10 F8            BPL   START

0217 B9 0A 00         LDAY  TIME    GET BYTE OF TIME POINTED TO BY Y REG
021A 4A               LSRA          GET HIGH ORDER 4 BITS
021B 4A               LSRA
021C 4A               LSRA
021D 4A               LSRA
021E 20 48 1F         JSR    CONVD  OUTPUT 4 BITS TO POSITIONPOINTED
                             TO BY X REG
0221 B9 0A 00         LDAY  TIME
0224 29 0F            ANDIM $0F     GET LOW ORDER 4 BITS
0226 20 48 1F         JSR   CONVD   OUTPUT
0229 A9 00            LDAIM $00     TURN OFF DISPLAY
022B 8D 40 17         STA   $1740
022E C8               INY           INCREMENT Y REG
                            X REG IS INCREMENTED BY CONVD
022F 10 E2            BPL    LOOP   BRANCH ALWAYS TO LOOP
```

Missing MICRO Information?

MICRO is devoted exclusively to the 6502. In addition, it is aimed at useful, reference type material, not just "fun and games". Each month MICRO publishes application notes, hardware and software tutorials, a continuing bibliography, software catalog, and so forth. Since MICRO contains lots of reference material and many useful programs, most readers want to get the entire collection of MICRO. MICRO grew very rapidly, and it very quickly became impracticle to reprint back issues for new subscribers. In order to make the older material available, two collections of reprints have been published.

A limited number of back issues are still available for number 7 through current.

The BEST of MICRO Volume 1 contains all of the significant material from the first six issues of MICRO, covering October/November 1977 through August/September 1978. This book form is 176 pages long, plus five removeable reference cards. The material is organized by microcomputer and almost every article is included. Only the ads have been omitted. **Surface . . . $7.00** **Air Mail . . . $10.00**

The BEST of MICRO Volume 2 covers the second six issues, from October/November 1978 through May 1979. Organized by microcomputer, this volume is 224 pages. **Surface . . . $9.00** **Air Mail . . . $13.00**

Use the convenient Order Form on Page 5 to place your order.

MICRO Dealers

This list is published as a service to our readers, and to the dealers that carry MICRO and other 6502 products. The dealers are in zip code order so that you can find the ones in your area quickly. Not all dealers are here; just the ones who sent us the information forms. We will run another list like this in a few months.

The Microcomputer Store
1568 Avenida Central
Caparra Heights, PR 00920
(809) 781-0350
Contact: A. Richner, J. Martinez
6502: PET Computers, peripherals

The Computer Store
Route 9 Deerskin Plaza
Framingham, MA 01701
(617) 879-3720
Contact: L. Virklir
6502: APPLE, Micro Nova, books, software

The Computer Store
63 South Main Street
Windsor Locks, CT 06029
(203) 627-0188
Contact: R. Cirmo
6502: KIM-1, APPLE, Mt. Hardware, Heuristics, Calif. Compu. Systems, software

The Computer Lab, Inc.
130 Jefferson Avenue
New London, CT 06320
(203) 447-1079

Connecticut Microcomputer Inc.
150 Pocano Road
Brookfield, CT 06804
Contact: S. Fletcher

Computerworks
1439 Post Road East
Westport, CT 06880
(203) 255-9096
Contact: S. Merrin
6502: APPLE, PET, Software, peripherals.

Computer Nook
Route 46, Pine Brook Plaza
Pine Brook, NJ 07058
(201) 575-9468
Contact: B. Stroeber
6502: APPLE computers and peripherals, Commodore computers, disk drives and printers.

HDE, Inc.
P.O. Box 120
Allamuchy, NJ 07820
(201) 362-6574
Contact: R. Grabowsky
6502: 8K Static RAM, 15 Slot card cage, prototyping cards, 8" disk systems, 5 1/4" disk systems for KIM, AIM, SYM.

Computerland of Cherry Hill
1442 East Route 70
Cherry Hill, NJ 08034
(609) 795-5900
Contact: D. Phelps
6502: APPLE, PET, ATARI, (servive for all).

Custom Computer Specialists, Inc.
208 Roanoke Avenue
Riverhead, NY 11901
(516) 369-2199

Computer Workshoppe
3848 Wm Penn Highway
Monroeville, PA 15146
(412) 823-6722
Contact: D. Eckert
6502: BASIC classes, APPLE II computers, APPLE software, service center.

Erie Computer
2127 West 8th Street
Erie, PA 16505
(814) 454-7652
6502: APPLE, OSI, software, service.

Computerland, New Castle
Astro Shopping Center
Kirkwood Highway
Newark, DE 19711
(302) 738-9656
Contact: B. Stewart, C. Grady
6502: APPLE, PET, KIM, SYM. Hardware, software, design and repair for most systems.

Computers, Etc.
13A Allegheny Avenue
Towson, MD 21204
(301) 296-0520

Computer Crossroads, Inc.
9143G Red Branch Road
Columbia, MD 21045
(301) 730-5513
Contact: R. Simpson
6502: Full line APPLE dealer.

Computers Unlimited, Inc
907 York Road
Towson, MD 21204
(301) 321-1553

Computerland/Tyson's Corner
8411 Old Courthouse Road
Vienna, VA 22180
(703) 893-0424
6502: APPLE, ATARI, Commodore, authorized factory service.

Computers Plus, Inc.
6120 Franconia Road
Alexandria, VA 22310
(703) 971-1996
Contact: J. Wells
6502: APPLE systems, software, books, maintenance

Home Computer Center
2927 Virginia Beach Blvd.
Virginia Beach, VA 23452
(804) 340-1977
6502: APPLE computers, peripherals, parts, service, distributes own football program.

Byte Shop, Greensboro
218 North Elm Street
Greensboro, NC 27401
(919) 275-2983
Contact: B. Terrell
6502: KIM, PET, APPLE, sales, services, peripherals, books, magazines, software.

Datamart of South Carolina
1901 Laurens Road
Greenville, SC 29607
(803) 233-5753
Contact: B. Thompson
6502: APPLE II

H.I.S. Computermaion, Inc.
1295 Cypress Avenue
Melbourne, FL 32935
Contact: J. Flores, R. Wilson
6502: KIM, APPLE.

Computer Age, Inc.
1308 North Federal Highway
Pompano Beach, FL 33062
(305) 942-1814
Contact: L. Fitzpatrick

The Computer Store
21 Clearwater Mall
Clearwater, FL 33516
(813) 796-1195
6502: APPLE II, level 1 service center, authorized business dealer.

AMF Electronics
11146 North 30th Street
Tampa, FL 33612
(813) 971-4072
Contact: A. Andre
6502: APPLE dealer,authorized service center,centronics dealer.

Computerland
3020 University Drive
Huntsville, AL 35805
(205) 539-1200

Computerlab
627 South Mendenhall
Memphis, TN 38117
(901) 761-4743
Contact: Rick or Jim
6502: AIM, KIM, APPLE II enclosure group

Micro Mini Computer World, Inc.
74 Robinwood Avenue
Columbus, OH 43213
(614) 235-5813, -6058
Contact: Mike, Will, Tom
6502: KIM, SYM, Commodore, peripherals, software, ful service, Memory plus.

ComputerLand of Cleveland
East: 1288 SOM Center Road
Mayfield Heights, OH 44124
(216) 461-1200
West: 4579 Great N. Blvd.
North Olmsted, OH 44070
(216) 777-1433
Contact: any employee
6502: full hardware, software, APPLE, ATARI, Synertek.

The Basic Computer Shop
Fairlawn Plaza
2671 West Market Street
Akron, OH 44313
(216) 867-0808
Contact: P. Reich

Micro Computer Center
7900 Paragon Road
Dayton, OH 45459
(513) 435-9355
6502: APPLE II, sales and service.

Computerland—Merrillville
19 West 80th Place
Merrillville, IN 46410
(219) 769-8020
6502: KIM, SYM, APPLE, PET, MICRO magazine, books.

Computer Center of South Bend
51591 US 31N
South Bend, IN 46637
Contact: B. Milliken
6502: Commodore, OSI, APPLE, hardware, software.

Computerland of Grand Rapids
2927 28th Street, SE
Kentwood, MI 49508
(616) 942-2931
6502: APPLE, Commodore, ATARI, SYM.

Computerland of Madison
690 South Whitney Way
Madison, WI 53711
(608) 273-2020
Contact: J. Sullivan
6502: PET, APPLE.

Zim Computers, Inc.
5717 Xerxes Avenue North
Brooklyn Center, MN 55429
(612) 560-0336
6502: APPLE, Commodore

Computerland of Oak Lawn
10935 South Cicero
Oak Lawn, IL 60453
(312) 422-8080
6502: APPLEs, Level 1 service

Computer Shoppe
3828 Veterans Blvd.
Metairie, LA 70002
(504) 454-6600
Contact: D. Hughes
6502: APPLE, PET, ATARI, software, books and accessories

Computer Shoppe
12-A Westbank Expressway
Gretna, LA 70053
(504) 366-0687

High Technology of Tulsa
2601-D So. Memorial
Tulsa, OK 74129
(918) 664-9754
Contact: D. Deardorff
6502: APPLE computer and related products

Comtel, Inc.
Computers Plus
5970 Broadway Blvd.
Garland, Texas 75043
(214) 840-1383
Contact: B. Jordan

Farnsworth Computer Center
1891 N. Farnsworth Avenue
Aurora, IL 60505
(312) 851-3888
Contact: L. Snyder
6502: APPLE

Computerland
136 Ogden Avenue
Downes Grove, IL 60515

The Computer Store of Rockford
2320 North Central Ave.
Rockford, IL 61103
(815) 962-7580
Contact: C.. Person
6502: APPLE, OSI

Byte Shop
1602 South Neil
Champaign, IL 61820
(217) 352-2323
Contact: J. Harter
6502: APPLE, PET, ATARI, software, full repair service, custom programming.

Computer Station
12 Crossroads Plaza
Granite City, IL 62040
(618) 452-1860
Contact: G. Baltzel
6502: APPLE computers and software

Illinois Computer Mart, Inc.
1114 W. Main Street
Carbondale, IL 62901
(618) 529-BYTE
Contact: E. C. Martin

Forsythe Computers, Inc.
7748 Forsyth Blvd.
Clayton, MO 63106
(314) 721-4300
6502: Authorized APPLE dealer and service center. Books, magazines, programming and programming referrals. OSI C3-OEM

Futureworld, Inc.
12304 Manchester
St. Louis, MO 63131

Personal Computer Center
3819 West 95th
Overland Park, Kansas 66206
(913) 649-5942

Computerland
10049 Santa Fe Drive
Overland Park, Kansas 66212
(913) 492-8882
6502: APPLE, PET, ATARI, KIM; also APPLE and PET service.

Computerland—Independence
1214 So. Noland Road
Independence, MO
(816)461-6502

Micro Store
634 S. Central
Richardson, TX 75080

Computerland
17647 El Camino Real
Houston, TX 77058
(713) 488-8153
Contact K. Arnold
6502: APPLE, PET, SYM, ATARI, hardware, software, service

Computer City
12704 North Freeway
Houston, TX 77060
(713) 821-2702
Contact: R. Bluefarb
6502: APPLE II, full line of software, repair.

Computercraft
3211 Fondren
Houston, TX 77063
(713) 977-0664
Contact: D. Airhart
6502: APPLE II, hardware, software, books, magazines, authorized service center, level 1.

Computer Solutions
5135 Fredericksburg Road
San Antonio, TX 78229
(512) 341-8851
6502: APPLE, software.

Computerland—austin
3300 Anderson Lane
Austin, TX 78757
(512) 452-5701
Contact: C. Phillips
APPLE, authorized dealer and service center

Computerland—Denver
2422 S. Colorado Blvd.
Denver,CO 80222
(303) 759-4685
Contact: S. Cohan

Personal Computer Place
1840 West Southern
Mesa, AZ 85202
(602) 833-8949
Contact: R. Smith
6502: PET, software, ROM updates, schematics, parts, etc.

Home Computers
1775 East Tropicana, 2
Las Vegas, NV 89109
(702) 736-6363
Contact: I. Jordan
6502: APPLE, PET, computers and peripherals.

Computerland—South Bay
16720 Hawthorne Blvd.
Lawndale, CA 90260
(213) 371-4624
Contact: Pete
6502: APPLE

Rainbow Computing, Inc.
9719 Reseda Blvd.
Northridge, CA 91324
(213) 349-5560
Contact: J. Anderson
6502: APPLE, ATARI

Computer Components
3808 West Verdugo Avenue
Burbank, CA 91505
(213) 848-5521
Contact: Linda or Joan
6502: APPLE, ATARI, level 1 service, extensive software.

Southwestern Data Systems
P.O. Box 582
Santee, CA 92071
(714) 562-3670
Contact: R. Wagner
6502: APPLE II software, APPLE-DOC, Roger's Easel, Programmers' Utility Pack, Shoppe Master, LIst Master, The Correspondent.

Computer Age, Inc.
4688 Convoy Street, Suite 105
San Diego, CA 92111
(714) 565-4042
Contact: J. Cheng
6502: APPLE II, PET, ATARI 800, authorized service center for APPLE & PET.

The Byte Shop
14300 Beach Blvd.
Westminster, CA 92683
(714) 892-0500
Contact: J. Hunter
6502: APPLE II, complete line of peripherals.

Computer World
6791 Westminster Avenue
Westminster, CA 92683
(714) 891-2584
Contact: D. Smith
6502: APPLE, ATARI, KIM, AIM, PET, software, servicing, books, magazines, custom software peripherals, disks, joysticks, light pens, printers.

Jay-Kern Electronics
1135 Columbus Avenue
Bakersfield, CA 93305
(805) 871-5800
6502: Commodore

Computers Made Easy
819 East Avenue, Q9
Palmdale, CA 93550
(805) 947-8613
6502: APPLE, KIM, ATARI, PET, all accessories, services.

Computer Place, INc.
P.O. Box 22530
28384 Carmel Rancho Lane
Carmel, CA 93923
(408) 624-7111

A.I.D.S.
301 Balboa Street
San Francisco, CA 94118
(415) 221-8500
Contact: M. Wong
6502: APPLE, authorized service center.

Computerland
22634 Foothill Blvd.
Hayward, CA 94541
(415) 538-8080
6502: full service, full line

Skyles Electric Works
10301 Stonydale Drive
Cupertino, CA 95014
(800) 538-3083
Contact: B. Skyles
6502: keyboards, memories, printers, toolkit, software development sys, all PET models.

Computerland
12020 SW Main Street
Tigard, OR 97223
(503) 620-6170
6502: APPLE, PET

Computerland, South King Co.
1500 5336th Street, Suite 12
Federal Way, WA 98003
(206) 927-8585
Contact: S. Lougee
6502: APPLE, PET, full line, authorized APPLE service center.

Scientific Business Instruments
The Diamond Center Mall
800 East Diamond, Suite 140
Anchorage, AK 99502
(907) 344-8352
6502: APPLE, OSI.

FOREIGN

TJB Microsystems Ltd.
10991 124th Street
Edmonton, Alberta
Canada T5M OH9
(403) 452-9476
Contact: J. Nickerson
6502: APPLE, PET.

Vulcan Computers
20571 Fraser Highway
Langley, British Columbia
Canada V3A 4G4
(604) 530-8572
Contact: G. Tremain
6502: PET, APPLE, Beta, books, software, service.

The Computer Circuit, Ltd.
737 Richmond Street
London, Ontario
Canada N6A 3H2
(519) 672-9370
Contact: W. Solotow
6502: PET, APPLE, KIM, sales, software, service.

The Computer Centre Pte. Ltd.
5345 Woh Hup Complex
Beach Road
Singapore 0719

A Home Message Center

If your house is like mine, there is never a pencil around when you need it, and when a message is left taped to the refrig, nobody ever notices. Put your messages on the APPLE and they will not be missed! This "Message Center" can be a starting point for other automatic display systems.

William McLean
1642 Edgewater Lane
Camarillo, CA 93010

This program was born from a dual need: I wanted to find a daytime use for my new computer, and I wanted to get my family involved in some small way with the APPLE so that they would lose that feeling of awe and "separation" that I see among the families of many computerists.

As I've had my APPLE for less than two months, the program would have to be fairly simple, but I really wanted to incorporate something creative. A tall order for a neophyte programmer who only learned the command LOAD two months ago, but I'm happy with the results and I hope that others will find some value in the ideas incorporated.

A few comments and explanations:

1. If you should happen to get an error message or hit control C, just type in GOTO 0 before listing so you'll have a full page to list without hitting reset.

2. Now you can type in GOTO 90 and the program should recommence without losing your message.

3. Temporary register K$ retains the last message even though it has been erased from the registers C$ through H$. So you can get it repeated by requesting it again even though the name no longer shows on the video screen.

4. This program has been set up to hold from 1 to 6 messages in a total of about 10K of RAM but can be modified to hold almost as mony more as you like if you have adaquate memory. You will need to modify number 90 to allow an extra line or two to display names (POKE 35,4 or 35,5); number 100 to move the display down an equal amount (POKE 34,4 or 34,5); line number 145 to check the extra registers for status; lines 150 through 160 to add more z "name strings"; 170 through 180 to add more "message strings"; and lines 190 through 200 tho erase the strings, once used.

The *Marquis* type of scrolling message is also useful for announcements for club meetings, o attention-getting continuous repla) displays at shows or store windows. Yoι can limit the number of times the message is displayed by deleting line 460 and assigning R in a 300 a value equal to the number of repeats that you want.

At home, we leave the video screen turned off and the APPLE II on. Each member of the family, upon seeing the computer "on" light, checks for messages by turning on the video momentarily and checking the names at the top of the screen.

One last hint: if you leave more than one message for the same person, you must change the "name" slightly. (I suggest a simple addition of a number, as in "John", "John2", "John3", etc.) If you should inadvertantly enter two messages for the same name, you will erase them both after only seeing one of them.

Copies of this program on cassette tapes are available from the author.

```
]PR#6
]PR#6

]LIST, PR#1
0  POKE 32,0: POKE 33,40: POKE 34
     ,0: POKE 35,24
20  DIM A$(255),B$(25),C$(25),D$(
     25),E$(25),F$(25),G$(25),H$(
     25)
21  DIM K$(255),CC$(255),DD$(255)
     ,EE$(255),FF$(255),GG$(255),
     HH$(255)
40 A = 0:C = 1
90  POKE 32,0: POKE 33,40: POKE 3
     4,0: POKE 35,3: HOME
95  INVERSE : PRINT "MESSAGES WAI
     TING FOR: ";: NORMAL : PRINT
     C$;D$;E$;F$;G$;H$
100  POKE 32,0: POKE 33,40: POKE
     34,3: POKE 35,24: HOME
120  HOME : PRINT "      =========
     ==========
        HOME MESSAGE CENTER
                   =============
     ======  "
121  PRINT : PRINT : PRINT
130  PRINT "WHICH DO YOU WANT ---
                         1.  ENTE
     R A NEW MESSAGE
         2.  RECEIVE A MESSAGE
                   ": INPUT "WHI
     CH?  ";X
131  IF X = 2 GOTO 185
140  INPUT "MESSAGE IS FOR (NAME)
       ";B$
141  IF  LEN (B$) > 1 THEN B$ = B
     $ + " - "
145  FLASH : IF  LEN (H$) > 1 AND
      LEN (G$) > 1 AND  LEN (F$) >
     1 AND  LEN (E$) > 1 AND  LEN
     (D$) > 1 AND  LEN (C$) > 1 THEN
      PRINT "SORRY, BUT ALL MY ME
     SSAGE REGISTERS ARE FULL!!":
      FOR Y = 1 TO 500: NEXT Y: GOTO
     90
146  NORMAL
150  IF  LEN (H$) < 1 AND  LEN (G
     $) > 1 AND  LEN (F$) > 1 AND
      LEN (E$) > 1 AND  LEN (D$) >
     1 AND  LEN (C$) > 1 THEN H$ =
     B$
152  IF  LEN (G$) < 1 AND  LEN (F
     $) > 1 AND  LEN (E$) > 1 AND
      LEN (D$) > 1 AND  LEN (C$) >
     1 THEN G$ = B$
154  IF  LEN (F$) < 1 AND  LEN (E
     $) > 1 AND  LEN (D$) > 1 AND
      LEN (C$) > 1 THEN F$ = B$
156  IF  LEN (E$) < 1 AND  LEN (D
     $) > 1 AND  LEN (C$) > 1 THEN
     E$ = B$
158  IF  LEN (D$) < 1 AND  LEN (C
     $) > 1 THEN D$ = B$
160  IF  LEN (C$) < 1 THEN C$ = B
     $
165  INPUT "THE MESSAGE IS?? (5 L
     INES OR LESS, IN   QUOTATION
      MARKS, PLEASE)
        ";A$
170  IF  LEN (HH$) < 1 AND  LEN (
     GG$) > 1 AND  LEN (FF$) > 1 AND
      LEN (EE$) > 1 AND  LEN (DD$
     ) > 1 AND  LEN (CC$) > 1 THEN
     HH$ = A$
172  IF  LEN (GG$) < 1 AND  LEN (
     FF$) > 1 AND  LEN (EE$) > 1 AND
      LEN (DD$) > 1 AND  LEN (CC$
     ) > 1 THEN GG$ = A$
174  IF  LEN (FF$) < 1 AND  LEN (
     EE$) > 1 AND  LEN (DD$) > 1 AND
      LEN (CC$) > 1 THEN FF$ = A$

176  IF  LEN (EE$) < 1 AND  LEN (
     DD$) > 1 AND  LEN (CC$) > 1 THEN
```

```
     EE$ = A$
178  IF  LEN (DD$) < 1 AND  LEN (
     CC$) > 1 THEN DD$ = A$
180  IF  LEN (CC$) < 1 THEN CC$ =
     A$
182  GOTO 90
185  INPUT "WHICH MESSAGE WOULD Y
     OU LIKE TO HAVE    DISPLAYED
     ?  (TYPE IN THE NAME EXACTLY
     )   ";J$:J$ = J$ + " - "
190  IF C$ = J$ THEN K$ = C$ + CC
     $: IF C$ = J$ THEN CC$ = "":
      IF C$ = J$ THEN C$ = ""
192  IF D$ = J$ THEN K$ = D$ + DD
     $: IF D$ = J$ THEN DD$ = "":
      IF D$ = J$ THEN D$ = ""
194  IF E$ = J$ THEN K$ = E$ + EE
     $: IF E$ = J$ THEN EE$ = "":
      IF E$ = J$ THEN E$ = ""
196  IF F$ = J$ THEN K$ = F$ + FF
     $: IF F$ = J$ THEN FF$ = "":
      IF F$ = J$ THEN F$ = ""
198  IF G$ = J$ THEN K$ = G$ + GG
     $: IF G$ = J$ THEN GG$ = "":
      IF G$ = J$ THEN G$ = ""
200  IF H$ = J$ THEN K$ = H$ + HH
     $: IF H$ = J$ THEN HH$ = "":
      IF H$ = J$ THEN H$ = ""
220  HOME : PRINT "      =========
     ==========
        HOME MESSAGE CENTER
                  =============
     ======  ": FOR X = 1 TO 9: PRINT
     : NEXT X
240  PRINT "TO ERASE CURRENT MESS
     AGE AND/OR TO       ENTER NEW
      MESSAGES, PUSH SPACE BAR"
260  FOR W = 1704 TO 1743: POKE W
     ,158: POKE W - 512,159: NEXT
     W
280  POKE 32,0: POKE 33,40: POKE
     34,11: POKE 35,13
300  FOR R = 1 TO 2: FOR M = 1 TO
     ( LEN (K$) + 40):A = M: IF A
      > 38 THEN A = 38
320 C = 1: IF M > 38 THEN C = M -
     37
340  REM   C=BEGINNING CHARACTER
     AND A=NUMBER OF EXTRA CHARAC
     TERS TO BE PRINTED.
360  HOME : HTAB 40 - A
380  PRINT  MID$ (K$,C,A): FOR X =
     1 TO 120: NEXT X
400  IF  PEEK ( - 16384) > 127 THEN
      GOTO 90
420  POKE  - 16368,0
440  NEXT M: NEXT R
460  GOTO 300
10000  REM  "HOME MESSAGE CENTER"
      WAS PROGRAMMED BY  BILL MCL
     EAN, 1642 EDGEWATER LANE, CA
     MARILLO, CALIF.  93010
10001  REM  ANYONE IS FREE TO USE
      AND COPY IT.  MY PHONE IS 8
     05-482-2048.  PLEASE CALL OR
      WRITE IF YOU COME UP WITH M
     EANINGFUL CHANGES.
?SYNTAX ERROR
```

CLUB FORUM Update

Dental Computer Newsletter
Offers: monthly newsletter, software exchange, advice, experience and access to members in your area. Membership: $12 per year
Dental Computer Newsletter
E.J. Neiburger, Editor
1000 North Avenue
Waukegan, IL 60085

Stop That PET!

That fantastic new program you are developing for the PET keeps bombing, causing the keyboard to lockup, and forcing you to RESET - thereby losing information about your program and its problems. Is there a way to STOP the run-away program? Would I ask the question if the answer wasn't "Yes"?

Gary J. Bullard
4808 S. Elwood 675
Tulsa, OK 74107

Commodore, for some reason, decided that the PET did not need a reset button. Since they did not provide a ROM monitor, perhaps they figured a reset button was unnecessary. This decision may have affected their sales somewhat.

They did provide a "STOP" button which serves as the equivalent of a CONTROL C, but this was useless if your cursor disappeared (a commom early PET problem) or your new machine-language program decided to loop forever instead of returning control to BASIC.

Machine-language programming for the PET therefore became an exercise in frustration:

1. Write your machine-language program.
2. Enter your program into the PET.
3. SAVE it. (Important!)
4. Execute your program via SYS or USR functions.
5. Stare impotently at dead PET. Frantically press STOP key as if you really think it will help.
6. Curse! Be creative!
7. Apply Commodore's RESET procedure. (i.e., turn PET off and back on)
8. Go back to step 2.

Somewhere between steps 2 and 3 you should perhaps examine your program for errors of coding or keypunching that could cause the PET to lock up. The difficulty with this procedure (for me anyway) is that if I thought the coding was wrong, I wouldn't have written it that way in the first place. So I'm left to examine five or six pages of coding that look perfectly good to me. What is needed is a way of knowing exactly where the program hung up so as to narrow the search.

If only there was a way to force an interrupt on the PET, then I could recover control and try again. At least it would eliminate steps 5, 6, 7, and 8. I could then insert breakpoints and narrow the problem down to a manageable level.

Commodore was ahead of me again.The non-maskable interrupt is permanently tied down so it vectors to the cold-start routines at power-on time. And since the cold-start routines destroy the contents of RAM, using this line as a reset would have the same effect as turning the PET off and on. I would have to make hardware changes and burn new PROMS to gain a reset capability with the non-maskable interrupt. Commodore wins this round.

But wait! Can Commodore be beaten at their own game? Maybe. . . .

There is one feature of the PET for which Commodore truly deserves a pat on the head . The PET has a built in real-time clock. This clock is updated by software once every sixtieth of a second. A circuit within the PET monitoring the AC line provides sixty interrupts a second. So the PET is not really "dead" at step 5 above. It is merrily executing your infinite loop and keeping good time, but it refuses to answer the keyboard; it is just playing dead.

Let's follow this idea a little way. Every sixtieth of a second, an interrupt is generated that causes the PET to jump indirectly through an address at $FFFE. This routine saves the accumulator, X register, and Y register on the stack and then jumps indirectly through $0219 to the clock routine. This vector is changed,when necessary, by the operating system to keep the clock routines from goofing up the timing for tape reads and writes. Normally, however, this vector remains constant. The exceptions to this rule will become important later.

Get the idea? Let's change that vector so the PET is forced to execute a short routine that checks the keyboard sixty times a second. Then, if the "STOP" key is pressed, we can provide a means of returning control to the operator. If the "STOP" key is not pressed, then the routine can proceed to its normal destination.

Now, since we've gone to this much trouble, why not see if we can derive some useful information from this exercise? It will be necessary to clean the stack so the BASIC warm-start address can replace the "real" interrupted return address. Why not save this information so we can discover just where the interrupt occurred?

Those of you who have had the tenacity to read this far, but who are not the least interested in a desription of a machine-language program may wish to skip this section. How to use the program will be told in the last section.

Listing 1 is the "assembly" listing of my program. A few words would now be appropriate to describe my peculiar "assembler".This assembler is a one-line assembler, approximately equivalent to the assembler provided in the Apple II. However, it is written in BASIC, and it has a few features I find convenient. For instance, I think in decimal, so I like to know the decimal address of an instruction. This is handy, considering that the functions SYS, PEEK, and POKE all use decimal arguments. Therefore, I designed the format of the assembler to be: Decimal address, Hex address, Hex op code, Hex operands/addresses, Label (added later — this *is* a one line assembler), Mnemonic (extended mnemonic set, see below), Operand (decimal, ofcourse, or label added later), then Comments (also added later). Since my output device is an IBM selectric that is in no way connected to my PET, I take the liberty of dressing up my listings at my leisure.

Now, to the program.

Since our intention is to divert the interrupt vector to our own purposes, the routine at 8000 ($1F40) does just that. A SYS8000 done in immediate mode will cause the vector to be changed from its normal value to a value that will cause the PET to execute a routine located at 8016 ($1F50) every time an interrupt occurs. This same routine will reverse that vector. This way, you only need to remember one number to set or reset the vector. This routine works like this: the instruction SEI causes the interrupt disable flag to be set, preventing the cpu from recognizing any interrupts. This is done to prevent an interrupt from occuring while we are half through altering the vector. There is no telling where the thing would go if we don't take this precaution! The LDY VECTOR + 1 instruction gets a byte of the current vector so we can determine whether it is the normal or the altered vector. Then we JSR to the SET subroutine which sets the normal vector. If the vector was already set to normal, then we have wasted a few microseconds, but I can wait. Then we compare the Y register to see if the vector was normal. If Y equals 230 ($E6) then we JSR to the RESET subroutine and alter it. Else the BNE NOTSET causes execution to resume at 8014 ($1F4E), the CLI clears the interrupt flag, enabling interrupts once again, and exits via the RTS back to BASIC.

Now that we have altered the vector, the PET will execute the routine located at 8016 ($1F50) every time an interrupt occurs, which happens sixty times a second. Let's examine this routine.

First we jump to a subroutine located at 62250 ($F32A) which checks the keyboard and returns a zero in the accumulator if the STOP key is pressed. Upon return from that subroutine, we check the accumulator for zero, BEQ STOPPD. If the accumulator is zero, we jump to STOPPD, which is our recovery routine. If the accumulator is not zero we jump to CONINT, which is where the interrupt would normally have gone.

The instruction at 8024 ($1F58) loads 8 into the X register, then uses this as a counter and displacement to pull 8 bytes off the stack and store them in memory beginning at location 8165 ($1FE5). We pull 8 bytes off the stack instead of 6 because we assume at least one level of subroutine call beyond the interrupt. After all, we had to SYS or USR into our machine-language program, didn't we? We don't want the stack to get too cluttered. This isn't going to be a complete answer, but it will help. More will be said about this later.

Once we have stripped and saved the stack contents, we need to restore it. The routine beginning at 8033 ($1F61) does this. First, we load and stack the high byte and low bytes of the BASIC warm-start routine. Then we recall the original value of the status register and stack it. This is important only because of the interrupt flag. Then we stuff the stack with three bytes of anything just to fill it out. After all, the interrupt handling routine thinks it has stored the accumulator and the X and Y registers on the stack; it will expect them to be there. Since we are not returning to the place where the interrupt occurred, we don't care what those bytes are, so let's just push the status byte onto the stack three more times.

Now that we have cleaned the stack and provided for a fake return from the interrupt, how about displaying the information thus recovered? Sounds easy, yes? Not so.

If we are to display our hard-won information, we can either write our own display routines or use the routines so conveniently provided for in ROM. Being naturally lazy, I much prefer to use the routines already written. But this is a problem. When I first wrote this program I got strange results that I eventually traced to the fact that the interrupt flag was being cleared somewhere, permitting the routine to be interrupted if I don't remove my finger from the STOP button within one sixtieth of a second. I discovered that at location 58816 ($E5CO) in ROM there was a CLI instruction. Since this is the print routine, I needed to find a way to use it without allowing the interrupt to screw things up. It was necessary to exit from our routine, return to BASIC, and then display our information. How to do it? I chose to use a TRICK!! Follow closely...

The PET has a keyboard buffer that collects keystrokes until the operating system can service them. This gives the effect of the PET "remembering" keystrokes even if you enter them while your program is doing something other than looking for INPUTs or GETs. If you load this buffer while in a machine language program and then exit to BASIC, the PET will suddenly "see" a command in its keyboard buffer and proceed to execute it.

After we recreate the stack and set the normal interrupt vector (JSR SET) then we load the keyboard buffer with the command "SYS8066cr". The "cr" is a carriage return. The instruction at 8049 ($1F71) loads 9 into the X register. This number, which is the count for the buffer, is then stored in location 525 ($020D). Then the command is retrieved byte by byte from location 8128 ($1FCO) and placed in the keyboard buffer. When that is finished, the PET jumps to CONINT, the normal interrupt routine.

What follows is this: the PET updates the clock register, then jumps to the interrupt routine which restores the registers and "returns" to the BASIC warm-start routine. The prompt "READY." is printed, and control returns to BASIC. BASIC checks the keyboard buffer, sees the command "SYS8066", prints it and executes it. Control is now given to our display routine. Roundabout way of doing it, no?

The display routine begins at location 8066 ($1F82). First we load the accumulator and Y register with the address of our header line. Then we JSR to the print routine in ROM at 51751 ($CA27). After that we load the accumulator and X register with the bytes we stored that represent the interrupted return address, and JSR to the number display routine in ROM at 56479 ($DC9F). This routine displays numbers in decimal, so if you wish them displayed in hex you will have to convert them yourself. Next, a loop is set up that retrieves each byte and displays them — first the status byte, then the accumulator, and the X and Y registers — all in decimal. At the end of all this wonderful activity, the routine exits through the START which alters the interrupt vector so we can do the whole thing over again.

Now, aren't you sorry you decided to read this?

How to Load and Use This Program

Listing 1 is an assembly listing of this program. It may be of interest to people who enjoy machine-language programming, or those who like convoluted logic. It is difficult to enter machine-language programs into the PET, however. So I have provided listing 2, which is a BASIC version of this program. I have converted the machine-language program into DATA statements and built a small loader routine around them to make it easier to key into the PET and SAVE onto tape.

For those of you with the fortitude to attempt to enter this program into your PET, attend: First, this program will not work if your PET is one of the newer models. If your PET comes awake with ### COMMODORE BASIC ### instead of *** COMMODORE BASIC ***, then forget this program. Rumor has it that you don't need it anyway. Second, if you still qualify, notice that each DATA statement has exactly eight numbers. If you key this program in exactly as shown it will be easier to find errors later if you miss-key a number.

OK. Are you still with me? Then key the program into your PET exactly as shown but do not RUN!! When finished, SAVE the program on tape. Then replace line 36 with:

```
36 FOR X=8000 TO 8159: READ A:
P=P+X*A:NEXT X:PRINT P
```

Now RUN the program. If the number printed is 146725222, then the odds are very good that you made no mistakes. You may now LOAD and RUN the original program. If, on the other hand, your number did not match the above number, then recheck your keypunching, correct your error, and try again.

NOTE: I would like to take this opportunity to encourage all programmers who write programs involving many DATA statements to include a verify routine. It need not be a permanent part of the program, but should be designed to give the hobbyist some solid evidence that his efforts were accurate.

Now you have LOADed and RUN the program. Notice that line 15 protects the last page of an 8K PET from interference by BASIC. This routine will reside safely here while you write your new machine-language program, debug it, and test it. I put this program in high memory because it is a machine-language debugging aid and most machine-language routines for the PET are located in the second cassette tape buffer area 826-1023 ($033A-$03FF) for convenience. If you wish to relocate it, be very sure you understand how it works.

Notice we have not yet activated this routine. First LOAD your machine-language program if you have it on tape, or your assembler if you are just starting. Then execute a SYS8000. This activates the Reset routine. From now on, whenever you hit the STOP key, the machine will halt and display the address where it was interrupted plus the register contents. There are a couple of exceptions which will be noted below.

OK. The Reset routine is activated, you have entered your machine-language program into the second cassette buffer, and executed a SYS826. Somehow the expected results do not materialize and the new program does not return control to BASIC. Now what? Hit the STOP key! Suddenly the PET comes back to life and prints:

```
ADDR ST AC XR YR

835 34 129 66 202
```

Eureka! Somewhere around location 835 there was an error in your program. The status register contained 34, the accumulator had 129 in it, the X register had 66 and the Y register was holding a 202. Were they supposed to have these values? Perhaps a register was initialized wrong or a relative branch was figured wrong. At least you know where to begin looking. Isn't it wonderful?

If the STOP key is pressed during normal command mode, you will likely get:

```
ADDR ST AC XR YR

58013 34 0 1 3

READY.
```

Notice that the numbers do not align perfectly under the header. They are printed without regard to the size of the number but are printed in the order depicted by the header, separated by blanks. Also, while it happens too fast to see, what actually printed was:

```
READY.
SYS8066
  ADDR ST AC XR YR
  58013 34 0 1 3
READY.
```

The SYS8066 and the header are both prefaced with an up-cursor character, so that they are printed on top of each other. This makes less scrolling, in case the screen contents were important.

Now for the exceptions to the rules. Since it is impossible to know how deeply nested in subroutines the PET was when the interrupt occurred we cannot properly clean the stack. After a while (approximately 23 times) pressing the STOP key will cause a ?OUT OF MEMORY ERROR remark to appear. The Reset routine will be deactivated and it will be necessary to SYS8000 again. Nothing to worry about; the PET corrected the stack before it printed the error message. Another message that will deactivate the Reset routine is ?ILLEGAL QUANTITY ERROR. Same response: SYS8000.

The command SYS8000 will activate the Reset routine if it was not already activated. If the Reset routine is active, SYS8000 will turn it off. Be sure it is on when you are executing your new machine-language program. And be sure it is off whenever you want to use the tape recorder to LOAD or SAVE a program or read or write data files. The cassette routines will not work if the interrupt vector has been altered. If in doubt as to the current status of the Reset routine, simply hit the STOP key. If nothing happens, the routine is off. If you get a register display, the routine is on.

One final exception to the rule. This routine is excellent if you get caught in an infinite loop or you just want to exit from a machine-language program early. It won't protect you from all invalid op codes. Some invalid op codes act like NOPs or do some mysterious, undefined function. These are OK. There are some op codes, however, that will cause the PET to lock up in spite of our marvelous Reset routine. Hex 04 will do this for example. I suspect that these obstinate op codes do something to affect the interrupt flag, thus disabling our routine. Until Commodore see fit to provide a non-maskable interrupt, I guess we will have to live with this incovenience. But even so, it gives us a place to start looking, doesn't it? If your PET locks up in spite of the Reset routine, look for invalid op codes.

There you are. I hope this encourages more machine-language programming for the PET. Let's get full power out of our computers!

For your free information regarding the extended mnemonic set that was referenced above, send a self-addressed and stamped envelope to:

MICRO
P.O. Box 6502
Chelmsford, MA 01824

```
DEC  HEX #1 #2 #3 LABEL  MNEM   OPER        COMMENTS

8000 1F40 78       START  SEI                :Disable interrupts
8001 1F41 AC 1A 02        LDY    VECTOR+1    :Get interrupt vector Hi byte
8004 1F44 20 AE 1F        JSR    SET         :Set normal vector
8007 1F47 C0 E6           CPYIM  230         :Was normal vector already set?
8009 1F49 D0 03           BNE    NOTSET      :
8011 1F4B 20 B9 1F        JSR    RESET       :No - alter vector to CKSTOP routine
8014 1F4E 58       NOTSET CLI                :Enable interrupts
8015 1F4F 60              RTS                :Return to BASIC

           Check if STOP key pressed

8016 1F50 20 2A F3 CKSTOP JSR    STOP        :Check if STOP key pressed
8019 1F53 F0 03           BEQ    STOPPD      :Skip next instruction if STOPped
8021 1F55 4C 85 E6        JMP    CONINT      : Else continue normal interrupt
8024 1F58 A2 08    STOPPD LDXIM  8           :Set strip counter
8026 1F5A 68       SAVEM  PLA                :Strip stack
8027 1F5B 9D E4 1F        STAX   STK-1       :Save bytes in memory
8030 1F5E CA              DEX                :Reduce counter
8031 1F5F D0 F9           BNE    SAVEM       :Continue until 8 bytes removed

           Recreate the stack

8033 1F61 A9 C3    RESTK  LDAIM  195         :BASIC warm start Hi byte
8035 1F63 48              PHA                :Stack it
8036 1F64 A9 8B           LDAIM  139         :BASIC warm start Lo byte
8038 1F66 48              PHA                :Stack it
8039 1F67 AD E9 1F        LDA    STATUS      :Get real status byte
8042 1F6A 48              PHA                :Stack it
8043 1F6B 48              PHA                :Pad stack
8044 1F6C 48              PHA                :   with dummy
8045 1F6D 48              PHA                :      bytes
8046 1F6E 20 AE 1F        JSR    SET         :Set normal interrupt vector

           Create re-entry from BASIC warm start

8049 1F71 A2 09           LDXIM  9           :Get character count
8051 1F73 8E 0D 02        STX    KNT         :Store in keyboard buffer counter
8054 1F76 BD BF 1F SETSYS LDAX   SYS66-1     :Retrieve instruction
8057 1F79 9D 0E 02        STAX   KBUF-1      :Store in keyboard buffer
8060 1F7C CA              DEX                :Reduce counter
8061 1F7D D0 F7           BNE    SETSYS      :Continue until done
8063 1F7F 4C 85 E6        JMP    CONINT      :Exit to normal interrupt routine

8066 1F82 A0 1F    DISPLY LDYIM  31          :Header Hi byte
8068 1F84 A9 CA           LDAIM  202         :Header Lo byte
8070 1F86 20 27 CA        JSR    ASPR        :Print header
8073 1F89 AD E7 1F        LDA    LOC         :Interrupted location Hi byte
8076 1F8C AE E8 1F        LDX    LOC+1       :Interrupted location Lo byte
8079 1F8F 20 9F DC        JSR    DCPR        :Print interrupted location
8082 1F92 A0 00           LDYIM  0           :Set and
8084 1F94 8C ED 1F        STY    CNTR        :  store counter
8087 1F97 AC ED 1F ALLREG LDY    CNTR        :Get count
8090 1F9A C0 04           CPYIM  4           :Have all been displayed?
8092 1F9C F0 A2           BEQ    START       :If so, reset vector, exit
8094 1F9E BE E9 1F        LDXY   STATUS      :Get next byte
8097 1FA1 A9 00           LDAIM  0           :Hi byte = zero
8099 1FA3 20 9F DC        JSR    DCPR        :Display byte
8102 1FA6 EE ED 1F        INC    CNTR        :Increment counter
8105 1FA9 4C 97 1F        JMP    ALLREG      :Continue until done

8108 1FAC EA              NOP                :
8109 1FAD EA              NOP                :

           SET and RESET subroutines

8110 1FAE A9 E6    SET    LDAIM  230         :Normal vector Hi byte
8112 1FB0 A2 85           LDXIM  133         :Normal vector Lo byte
8114 1FB2 8D 1A 02 SAVIT  STA    VECTOR+1    :Store Hi
8117 1FB5 8E 19 02        STX    VECTOR      :Store Lo
8120 1FB8 60              RTS                :Return
8121 1FB9 A9 1F    RESET  LDAIM  31          :New vector Hi byte
8123 1FBB A2 50           LDXIM  80          :New vector Lo byte
8125 1FBD 4C B2 1F        JMP    SAVIT       :Store & exit

           SYS8066 constants

8128 1FC0 91       SYS66  DC     145         :Up cursor
8129 1FC1 53              DC     83          :   S
8130 1FC2 59              DC     89          :   Y
8131 1FC3 53              DC     83          :   S
8132 1FC4 38              DC     56          :   8
8133 1FC5 30              DC     48          :   0
8134 1FC6 36              DC     54          :   6
8135 1FC7 36              DC     54          :   6
8136 1FC8 0D              DC     13          :Carriage return

8137 1FC9 EA              NOP                :


DEC  HEX #1 #2 #3 LABEL  MNEM   OPER        COMMENTS

           Header constants

8138 1FCA 91       HEADER DC     145         :Up cursor
8139 1FCB 20              DC     32          :Space
8140 1FCC 41              DC     65          :   A
8141 1FCD 44              DC     68          :   D
8142 1FCE 44              DC     68          :   D
8143 1FCF 52              DC     82          :   R
8144 1FD0 20              DC     32          :Space
8145 1FD1 53              DC     83          :   S
8146 1FD2 54              DC     84          :   T
8147 1FD3 20              DC     32          :Space
8148 1FD4 41              DC     65          :   A
8149 1FD5 43              DC     67          :   C
8150 1FD6 20              DC     32          :Space
8151 1FD7 58              DC     88          :   X
8152 1FD8 52              DC     82          :   R
8153 1FD9 20              DC     32          :Space
8154 1FDA 59              DC     89          :   Y
8155 1FDB 52              DC     82          :   R
8156 1FDC 0D              DC     13          :Carriage return
8157 1FDD 00              DC     0           :End of message character
```

Working storage -- need not be initialized

```
8165 1FE5 00      STK     DC    0        :Extra Hi byte
8166 1FE6 00              DC    0        :Extra Lo byte
8167 1FE7 00      LOC     DC    0        :Interrupted Hi byte
8168 1FE8 00              DC    0        :Interrupted Lo byte
8169 1FE9 00      STATUS  DC    0        :Status byte
8170 1FEA 00              DC    0        :Accumulator
8171 1FEB 00              DC    0        :X register
8172 1FEC 00              DC    0        :Y register

8173 1FED 00      CNTR    DC    0        :Loop counter
```

Memory locations and ROM routines used

LABEL	DECIMAL	HEX	COMMENT
VECTOR	537	0219	Interrupt vector
KNT	525	020D	Keystroke counter
KBUF	527-536	020F-0218	Keyboard buffer
BASIC	50059	C38B	BASIC warm start
ASPR	51751	CA27	Print ASCII string terminated with a zero. Enter with ADH in Y, and ADL in A.
DCPR	56479	DC9F	Print the decimal integer whose binary value is in A (weight=256) and X (weight=1).
CONINT	59013	E685	Normal interrupt routine.
STOP	62250	F32A	Check to see if STOP key pressed. Returns a zero in accumulator if key is pressed.

```
1 REM"████████                *** RESET ***
2 REM"████████
3 REM"████████                 WRITTEN BY:
4 REM"████████
5 REM"████████               GARY J. BULLARD
6 REM"████████               4808 S. ELWOOD
7 REM"████████               TULSA, OKLAHOMA
8 REM"████████                         74107
9 REM"████████
10 REM"██████████
11 REM"██████████
15 POKE135,31:POKE134,64
16 DATA 120,172,26,2,32,174,31,192
17 DATA 230,208,3,32,185,31,88,96
18 DATA 32,42,243,240,3,76,133,230
19 DATA 162,8,104,157,228,31,202,208
20 DATA 249,169,195,72,169,139,72,173
21 DATA 233,31,72,72,72,72,32,174
22 DATA 31,162,9,142,13,2,189,191
23 DATA 31,157,14,2,202,208,247,76
24 DATA 133,230,160,31,169,202,32,39
25 DATA 202,173,231,31,174,232,31,32
26 DATA 159,220,160,0,140,237,31,172
27 DATA 237,31,192,4,240,162,190,233
28 DATA 31,169,0,32,159,220,238,237
29 DATA 31,76,151,31,234,234,169,230
30 DATA 162,133,141,26,2,142,25,2
31 DATA 96,169,31,162,80,76,178,31
32 DATA 145,83,89,83,56,48,54,54
33 DATA 13,234,145,32,65,68,68,82
34 DATA 32,83,84,32,65,67,32,88
35 DATA 82,32,89,82,13,0,234,234
36 FORX=8000TO8159:READA:POKEX,A:NEXT
37 :
38 :
39 PRINT"███SYS8000 TO TURN ON OR OFF."
40 PRINT"███ NOTE: THIS ROUTINE MUST BE TURNED OFF █"
41 PRINT"█IF CASSETTE I/O IS USED."
 READY.
```

Listing 2

The MICRO Software Catalogue: XVIII

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

Name: **APPLESOFT-PLUS (APLUS) Structured Basic**
Memory: **16K (min.)**
Language: **6502 Assembly Language**
Hardware: **Standard (Disk, Autostart, and APPLESOFT ROM optional)**

Description: You don't have to buy APPLE PASCAL to begin enjoying the benefits of structured programming. APPLESOFT PLUS (APLUS), a 4K assembly-language program, is now available to add the following structured-programming commands to APPLESOFT Basic: When..Else..Fin, Until, While, (multi-line) If..Fin, Unless, Case, Select (variable), (Otherwise), 'DO (named procedure)', and 'TO (define a named procedure). The APLUS '&CONVERT' command converts Applesoft programs that contain the above commands into standard Applesoft by replacing the structured-programming commands with appropriate 'GOTO' statements. The APLUS '&LIST' command automatically indents listings to clarify the logic flow inside a program. APLUS automatically checks for logic errors each time the Applesoft program is listed and allows start/stop/continue control of APLUS listings. APLUS also provides a separate list of all 'named procedures' used in a program. APLUS allows new Applesoft programs to be created without entering any 'GOTO's or 'GOSUB's. These Applesfot-Plus programs are then significantly easier to develop, mâintain, and understand that 'regular' Applesoft programs. APLUS can also be used to maintain existing Applesoft programs without special conversion.

Price: **$25.00**
includes: Cassette or disk with instructions
Available: Sensible Software
P.O. Box 2395
Dearborn, MI 48123

Name: **Integer BASIC Floating Point Interface (2.5K)**
System: **APPLE II**
Memory: **16K or more**
Language: **Machine Language**
Hardware: **Basic Apple II**

Description: The Integer BASIC Floating Point Interface, or IBFPI, is an integrated set of machine language routines which allows the user to indirectly use the Apple II Floating Point Utility Routines. The first eight variables defined in an Integer BASIC program allow floating point variables and arrays to be referenced from within an Integer BASIC program. IBFPI also includes utility routines for floating point exponention, input, and output. IBFPI FP output allows Fortran-like field widths. Arithmetic errors are processed automatically, and when an error is detected, an error message is printed and the current Integer BASIC line is displayed. The eight-page documentation includes features, background, program cassette operation, programming, page zero usage, error messages, structure, and notes. A demonstration program is included and is explained in the operation part of the documentation.

Copies: **7 delivered**
Price: **$8.95** postpaid
Includes: Cassette and documentation
Author: **Harry L. Pruetz**
Available: Microspan Software
P.O. Box 9692
Austin, TX 78766

Name: **Restaurant Food/Beverage Analysis and Menu Pricing**
System: **APPLE II**
Memory: **16K**
Language: **APPLESOFT II (ROM or RAM)**

Description: A self-prompting program that establishes menu prices of virtually any food/liquor item. Predetermined, and industry proved, percentages in the program define what the menu price should be in order to make an acceptable profit. The program may be utilized both for initiating a new or revising an existing menu. Re-evaluates existing menu prices and indicates what present food/liquor costs are. Takes the guess work out of menu development. No intelligent restauranteur should be without this aid.

Price: **$14.95 Diskette, $9.95 Cassette, $5.95 Listing,** plus $1.00 postage and handling.
Author: **M. Goldstein**
Available: Mind Machine, Inc.
31 Woodhollow Lane
Huntington, N.Y. 11743

Name: **Disk Utilities**
System: **APPLE II**
Memory: **48K with APPLESOFT ROM**
Language: **APPLESOFT/MACHINE**

Description: Disk Utilities is a set of 3 programs: One Drive Copy, Disk Statistics, and Patch. One Drive Copy allows disks to be copied on a system with only one disk drive. Disk Statistics displays the unused sectors of a disk as a number and a percentage of the total. Patch is a powerful tool that allows the reading, displaying (in ASCII and HEX), modification and writing of any sector on a disk.

Price: **$19.95** on diskette with user manual.
Author: **Hal Clark**
Available: ON-GOING IDEAS
P.O. Box 132
Rosemount, MN 55068

MN residents please buy at your local APPLE dealers.

Name: **AIM Microchess**
System: **AIM 65**
Memory: **2K**
Language: **Assembly**

Description: An AIM version of Microchess, Peter Jennings' original chess program for the 6502. This version features several keyboard selectable speeds, a chess clock, optional printout of the current chessboard, display of moves in standard chess notation, and more. A great way to "show off" your AIM, as well as a means to learn more about using your AIM since source listings are provided.

Copies: **AIM version, just released. KIM version, thousands!**
Price: **$15.00**
Includes: Cassette tape object, operating instructions, commented source listing.
Authors: Mel Evans - AIM Version
Peter Jennings - Original
Available: MICRO Software
P.O. Box 6502
Chelmsford, MA 01824

Name: **Micro Memo**
System: **APPLE II**
Memory: **48K**
Languagez; **RAM or ROM Applesoft**
Hardware: **Optional**—Mountain Hardware Clock (any slot) and/or any APPLE compatible printer

Description: Micro Memo is a powerful "desk calendar" program. It can handle one-time, weekly, monthly, semi-annual, and annual reminders. Monthly reminders may be for fixed dates (e.g., the 15th of the month) or "floating " dates (e.g., the 1st Saturday of every month). Each reminder allows you the choice of 1 week, 2 week or 1 month advance notice, so the system can remind you ahead of time to prepare for meetings, purchase birthday presents, make reservations, etc. The program will print out or display any day's or week's reminders, including most major holidays. This is a "perpetual" calendar which automatically creates new months with all appropriate reminders (birthdays, anniversaries, monthly meetings, etc.) as past months are dropped. The system holds a full year's reminders (beginning with any month) on one disk.

Price: **$39.95**
Includes: Disk with program and 6 pages of documentation, including information on custom modifications.
Author: **Barney Stone**
Available: STONEWARE
Microcomputer Software
P.O. Box 7218
Berkeley, CA 94707

Name: **IBM PRINT**
System: **APPLE II or APPLE II PLUS, Disk II and IDS 440 Printer**
Memory: **16K with ROM—32K without**
Language: **APPLESOFT II**

Description: IBM Print is a simple utility program for printing "TEXT" files with ANSI standard carriage control characters in column one of the records. This capability allows the user to create files for printing with overprint and paging capabilities using any text editor or program. The program is most useful for local printing of large files created on main-frames and moved to the APPLE using the terminal communications program: MOVE 370. This technique allows the user to create large print files using IBM's very powerful word processing systems, move them to the APPLE, and then print them locally. The program is easy to use and easy to modify for other brands of printers.

Price: **$20.00**
Includes: One diskette and program
Author: **Gary M. Grandon, Ph.D.**
Available: Rosen Grandon Associates
296 Peter Green Road
Tolland, CT 06084

Name: **Wilderness Campaign**
System: **APPLE II**
Memory: **48K**
Language: **Integer Basic**

Description: Wilderness Campaign is a game of high adventure in which you undertake a crusade to free the kingdom of Draconia from the Evil Necromancer that is tyrannizing it. As you direct your party across the high resolution graphics map of Draconia, you must overcome obstacles, defeat hostile inhabitants, survive various natural hazards (avalanches, quicksand, etc.) and explore numerous tombs, temples, castles, and ruins in search of gold and magical devices. When treasure is found, you will go to nearby villages to hire men and purchase weapons, armor, and assorted useful supplies. The supplies and any magical devices that you find will aid you in your ultimate quest: to find the ancient weapons of power required to defeat the Necromancer. Once you have found the required magical weapon and have gathered and equipped a suitable army, you are ready to attack the fortress of the Necromancer itself. The future of Draconia rests on your shoulders.

Copies: **Many**
Price: **$15.00 cassestte, $17.50 disk** (WA residents add 5.3% sales tax)
Author: **Robert C. Clardy**
Available: Synergistic Software
5221—120th Ave. S.E.
Bellevue, WA 98006
(206)641-1917

Name: **MAE Development Software**
System: **32K ROM PET and 2040 Disk Drive**
Memory: **10K**
Language: **Machine Language**

Description: A new Assembler/Editor and associated development software to be used for developing programs in assembly language on the new ROM 32K PET and 2040 Disk Drive. No tape is supported. User has option of connecting an extgernal CRT or TTY to obtain 80 column display operation. Features include MACROS, Conditional Assembly, and Interactive Assembly. The MAE ASSM/TED occupies 10K memory starting at $5000 and handles labels up to 31 characters. Includes relocating loader plus a copy of the loader in relocatable form so it can be relocated practically anywhere in RAM memory.

Copies: **Just Released**
Price: **$169.95** postpaid in U.S.—requires completion of software license agreement.
Includes: Diskette and Manual
Author: **C.W. Moser**
Available: Eastern House Software
3239 Linda Drive
Winston-Salem, N.C. 27106

Name: **One-Arm Bandit**
System: APPLE II
Memory: **32K**
Language: **Interger Basic**
Hardware: **APPLE II (32K), Integer Basic, Video Monitor or TV**

Description: One-Arm Bandit is a slot machine program for one to four players. It uses a combination of text graphics, low-res color graphics and sound effects to display a realistic one arm bandit. Pay off odds are based on those of a real slot machine. The program also displays many personalized messages and keeps track of each player's winnings or losses. See and hear the wheels turn when you pull the bandit's arm. Three "ORANGES"—YOU WIN!.

Price: **$9.95** for cassette, **$14.95** for disk (Listing included)
Authors: **Ken and Dawn Ellis**
Available: Progressive Computer Software
405 Corbin Road
York, PA 17403

Name: **ROSTER**
System: **APPLE II**
Memory: **48K**
Language: **APPLESOFT II**
Hardware: **APPLESOFT ROM Card, Disk II, Printer**

Description: A general purpose disk-based record-keeping program for teachers at all levels. Allows instructors to create and change class rosters, label, enter and change test or assignment scores, sort roster based on student number, student name, or rank in class, assign character or numeric grades based on any of five criteria (raw score, percent, rank, percedntile rank, or z-score), and list scores, totals (or averages), and/or grades according to any of these options. The program will (at the instructor's option) automatically weight each score's contribution to the student's cumulative total (or mean), and/or drop the lowest (weighted or unweighted) score from among a selected group of scores prior to grade assignment. Two separate list formats are available. A class roster list prints the name of the class, section, term, etc., student numbers and any or all of the following: student names, individual scores (as raw scores, percents, ranks, percentile ranks, or z-scores), totals (or means) of individual scores (using the weighting or drop options, if requested), and assigned grades (according to user-entered grading scale). Means and standard deviations for each selected item are printed automatically. The individualized student list prints any or all of the above statistics for each test or assignment separately (particularly useful for student or parent consultation). Program is password protected (you enter your own password).

Copies: **Just released**
Price: **$39.95**
Includes: Software on disk, listing, documentation, and user's manual
Available: **Dr. Douglas Eamon**
105 West Oak Street
Albion, MI 49224

Name: **Higher Graphics**
System: **APPLE II**
Memory: **32K**
Language: **Integer Basic**
Hardware: **Disk Drive**

Description: A collection of programs and shape tables that lets any programmer create detailed and beautiful high resolution displays and animation effects. Make your programs come alive by utilizing the full graphical capabilities of the Apple II. Package contains: Shape Maker— create, correct, or delete shapes, start new shape tables or add to existing ones, display any/all shapes with any scale or rotation at any time. Table Combiner—pull shapes from existing general purpose tables and add the ones you want. Screen Creator—place your shapes on the high-res screen, add areas of color and text to make detailed displays etc. Shapes—four shape tables with over 100 shapes are provided. High Res Text—how to use high resolution graphics in your program. Animation effects and display techniques.

Copies: **Many**
Price: **$25.00** (WA residents add 5.3% sales tax)
Author: **Robert C. Clardy**
Available: Synergistic Software
5221 —120th Avenue,S.E.
Bellevue, WA 98006

Name: **PET RABBIT**
System: **16K or 32K New ROM PET**
Memory: **2K**
Language: **Machine Language**

Description: Provides 12 commands which can be executed in Basics direct mode plus provision of automatic repeat of any key (including cursor control keys) held down for 0.5 seconds. Rabbit provides fast load and save commands (38 seconds versus 2 minutes 44 seconds for PET to record 8K memory), an exhaustive memory test, convert hex to decimal and vice versa, plus more. Specify memory location as: $3000, $3800, $7000, or $7800.

Copies: **Just Released**
Price: **$29.95 postpaid**
Includes: Manual and Cassette
Authors: **J.R. Hall and C.W. Moser**
Available: Eastern House Software
3239 Linda Drive
Winston-Salem, N.C. 27106

Name: **Matrix Manipulator**
System: **APPLE II or APPLE II Plus**
Memory: **32K Minimum**
Language: **ROM Applesoft**
Hardware **Disk II (opt.) Printer (opt.)**

Description: Matrix Manipulator is an interactive program having the following modes; add, subtract, multiply, invert, transpose, scalar multiply, square root, orthonormalization, row reduce, column augment, input, edit, display/print, to/from disk, and 11 other modes. Matrix names and sizes are user definable, typified by 15 (32K) or 40 (48K) 10x10 matrices in use at once with DOS active. User's guide explains modes and applications to linear system solutions, etc. Program is supplied in two versions, one with in-line instructions.

Price: **$24.95 disk**
$22.95 tape plus 4.5% Ohio tax in Ohio
Includes: Both program versions and User's Guide
Author: **Robert Rennard**
Available: SmartWare
2281 Cobble Stone Ct.
Dayton, OH 45431

Name: **MOVE 370**
System: **APPLE II or APPLE II PLUS, Disk II and Communications**
Memory: **16K with ROM—32K without**
Language: **APPLESOFT II**

Description: MOVE 370 is a telecommunications interface program for moving complete "TEXT" files to and from IBM 370 systems. The program communicates with IBM's interactive CMS EDITOR (or other IBM editors with minor modifications) to pass files to or from the IBM system from the remote APPLE II. This capability enables the user to treat the Micro as a genuine distributed processor for data-entry and other off-loaded applications. It's easy to use and easy to modify.

Price: **$20.00**
Includes: One diskette plus program
Author: **Gary M. Grandon, Ph.D.**
Available: Rosen Grandon Associates
296 Peter Green Road
Tolland, CT 06084

Software Catalogue Note

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

6502 Bibliography: Part XVIII

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

554. Call Apple 2, No. 6 (July/August 1979)(cont'd)

Neulen, Bob and Golding, Val. "Change Catalog to C," pg. 30.
A short utility...but watch out playing WUMPUS with that C.

Golding, Val J. "Monitor String Writer," pg. 32.
Two programs for converting machine language to HEX Strings and writes them into program memory automatically, one each for Integer Basic and for Applesoft Basic.

Dunmire, Darrell. "The Missing Link," pg. 34.
LINK was incompletely listed earlier; now it is complete.

Anon. "HIRES Dump," pg. 36.
Mod for centering with Parallel Card, for the Apple.

Wagner, Roger. "Exceeding the Speed Limit with your Apple II," pg. 36.
How to coax more speed out of the Apple.

555. Creative Computing 5, No 9 (September 1979)

Anon. "Apple Cart 3-D Graphics," pg. 17.
Program missing in July Apple-Cart article.

Scarpelli, Anthony T. "A 6502 Disassembler in Microsoft BASIC," pg. 124-129.
A program that will disassemble the 6502 op-codes, printing out the op-code, address, data, all in hex and the mnemonics and addressing mode.

Piele, Donald T. "Micros 'GOTO' School," pg. 132-134.
Discussion of the use of the Apple II in school computer programs.

Hunter, Jim. "A Grade Maintenance Program for the Apple II with Disk Drive," pg. 142-149.
Maintaining accurate grades in school computer systems.

Tubb, Philip. "Q and A - The 'Tiny' Interpreter Exercise," pg. 161-176.
Discussion of Applesoft Basic.

Yob, Gregory, "Personal Electronic Transactions," pgs. 178-182.
Discussion of the "Basic Programmer's Toolkit, the Structure of Basic, encrypted messages on the PET.

556. PET User Notes 2 No. 1 (Jan/Feb 1979)

Beals, Gene, "New Commodore Products," pg. 2.
Discussion of new model PETs, etc.

Covitz, F., "Visible Memory," pg. 4.
Control individual pixels on the PET screen with the Visible Memory board, and PET interface.

Butterfield, Jim, "Watching a Cassette Load," pg. 7.
Procedure for visible loading of a program from cassette tape.

Wuchter, Earl, "Controlled Restore," pg. 7.
Routine to reset the DATA pointer to the beginning of any DATA line.

Zimmermann, Mark, "ROM Subroutines," pg. 11.
Some of the ROM subs of the PET are revealed.

Swan, Warren, "Machine Language Routines for Fast Graphics," pgs. 13-19.
Seven routines including a demo.

Russo, Jim, "Single-Step Routine," pg. 16.
MT6671 is a machine-language monitor program incorporating a single-step trace feature.

Velders, Jerry A. "PLOT," pg. 18-19.
Machine-language plot routine which is faster than an equivalent Basic routine.

O'Brien, Roy, "Ramblin'," pgs. 20-21
How the CB2line of the PET's VIA chip (6522) can be used for generating musical tones.

Velders, Jerry A., "Memory Test," pg. 22.
MTEST is a Non-destructive Memory test.

557. Dr. Dobb's Journal 4, Iss.8 (Sept. 1979)

Zant, R.F., "Modular Programming with the Apple II,"pgs. 37-38.
The modular approach to programming encourages the use of common routines in different programs.

558. Washington Apple Pi 1, No. 7 (August 1979)

Dial, William R., "6502 Information Resources," pg. 2.
Reprinted from MICRO and updated by Mark Crosby.

Cirillo, Nicholas B., "Changing 'Beneath Apple Manor',"
pgs. 3-4.
A way to interrupt the commercial program, Apple Manor, and save data so that it can be resumed later.

Moon, John L., "APPLE Programming With Style," pgs. 4-5.
How to structure a program, add a menu, etc.

559. Rainbow 1, No. 7, (August, 1979)

Simpson, Rick, "Introduction to Machine Language Programming," pgs. 1-5.
The internal structure of the 6502 itself is discussed.

Landereau, Terry, "What's the Difference," pg. 7.
Discusses the differences in the terms assembler, interpreter, compiler, source code, object code, etc. for the Apple.

Wright, Walter, "Modifications to Gary Dawkin's Shape Drawing Program." pg. 8.
A version modified for the Programmer's Aid ROM No. 1 of the Apple.

Blue, Bill and Massimo, Alex, "Applesoft Program for Lower Case Letters with Dan Paymar's Lower Case Adapter Install- e d ,
pg. 13.
Short listing—the title tells it all.

Blue, Bill and Massimo, Alex, "Stop the Cursor Flashing," pg. 15.
Simple hardware mod. for the Apple.

Blue, Bill and Massimo, Alex, "Keyboard Mod for Faster Cursor Control," pg. 16.
Simple hardware mod to move the cursor faster on the Apple.

560. 73 Magazine, No. 228 (September, 1979)

DeJong, Marvin L., "Build the KIM Keyer," pgs. 80-84.
With a simple interface and a short application program the KIM-1 can send any of three messages entered into memory.

561. Byte 4, No. 9 (September 1979)

Teeters, Jeff, "Interface a Chessboard to your KIM-1," pgs. 34-54.
By using a specially electrified chessboard, the KIM-1 can be used, avoiding keyboard entry of moves.

O'Haver, T. C., "A Similarity Comparator for Strings," pgs. 58-60.
A program comparing strings or doing searches reports 'best' match if no exact match is found. In OSI Microsoft Basic.

Hallgren, Richard C., "A Low-Speed Analog-to-Digital Converter for the Apple II," pgs. 70-78.
Hardware and two software routines.

Powers, William T., "The Nature of Robots: Part 4: Looking for Controlled Variables," pgs. 96-112.
An Apple II graphics program is given to simplify a control-variable simulation.

562. The Paper 2, Iss. 3 (August 1979)

Sparks, Paul W., "An Inexpensive Printer Option," pg. 3.
Interfacing the SWPTC PR-40 printer with the PET.

Busdiecker, Roy, "6502 Relocating Macro Assembler/Test Editor 1.0," pgs. 7-8.
A review of a PET program.

Busdiecker, Roy, "Odds and Ends," pg. 9.
Discusses the PET ROM Monitor, the care of cassettes, renumber program, etc.

Anon, "Forth and Backwards," pgs. 12-13.
Discussion of PETFORTH and other versions of Forth.

Wachtel, A., and Szepesi, Z., "The Development of a Basic Program," pgs. 14-15.
Illustration of the many ways a program can be improved.

Busdiecker, Roy, "Computers Do It In Binary," pgs. 22-24.
The second in a series of articles on computer arithmetic.

563. Abacus 1, Iss. 8 (August 1979)

Anon, "Routine to Center Titles," pg. 2.
Short utility for the Apple II.

Anon, "Integer Basic Internals," pgs. 3-5.
A list of Variables used by Basic and Integer Basic Subroutines for the Apple.

Hertsfeld, Andy, "Text File Mover," pg. 4.
How to get a text file from one disk to another on the Apple.

Staff, "Super HI-RES," pg. 4.
A short listing for the Apple Hires Graphics.

Avelar, Ed, "Survey Analysis," pgs. 7-8.
Analyzes data for number of responses in each category, etc.

Paymar, Dan, "Disc Access Utility," pgs. 9-10.
Allows you to dump the contents of any specified sector and track. Useful for recovering crashed disks, on the Apple.

Hyde, Randy, "The Apple Monitor," pgs. 11-12.
Description of routines and registers affected by calling the routine.

Crosby, Mark, "Changing 'Catalog' to 'C'," pg. 12.
Two short listings for Apple DOS 3.1 and 3.2.

Staff, "Convert Decimal Input to Hex Output," pg. 12.
Short Applesoft routine for the Apple II.

Staff, "Use of Control Y with Parameter on the Apple," pg. 14.
Demo of Control Y and the Apple Paddles.

Staff, "Statistics Programs," pg. 15.
A routine for the Hello program for the Apple that will bring up a menu and allow a given program to be selected. Example is with the Osborn list of statistics programs.

564. The Seed 1, No. 3 (August 1979).

Staff, "Documentation Library," pgs. 3-9.
In addition to maintaining a large program library, The Apple Pi user Group of Denver, CO has published an extensive list of User Group publications and Lists of documentation of Apple programs available for copying.

Borgerding, Jim, "Star Location," pg. 12.
Locate stars with the Apple.

Egan, Linda, "Hires Shape Generator," pg. 17.
An integer program to go with the Apple Programmer's Aid No. 1.

McClure, Allen and Taylor, T.N., "Disc Speed Adjustment with the DSPEED Programs," pg. 18.
Adjust the speed of your Apple II disk drive.

Anon, "B/BSTAT Change," pg. 21.
A few simple changes of variables will insure that B/BSTAT will work on Apple DOS 3.2.

565. 6502 User Notes, No. 16 (September 1979)

Martin, Richard, "Four Part Harmony," pgs. 1-4.
This KIM-1 music program plays on an unexpanded KIM-1.

Adams, Jim, "Circular List Processing Subroutines,;; pgs. 5-6.

Hawkins, George W., "Pong Sound Effects," pg. 8.
Add BEEP, BOOPand ZONK sounds to your KIM p-rograms.

Leedom, Robert, "Baseball for the KIM," pgs. 10-11.
A video style action game for the KIM-1.

McKenna, Sean, "Language Lab," pgs. 12-19.
An Editor for Microsoft Basic.

Clements, William C., "Adding a Cassette Interface to Focal," pgs. 14-16.
Add a cassette interface and User function to Focal 65-E.

Mulder, Bernhard, "Focal LED Output," pg. 17.
This routine makes KIMs, LEDs another output device.

Allen, Michael, "TVT-6/TINY BASIC Interface," pgs. 18-19.
How to get Tom Pitman's Tiny Basic to work with the KIM-1/TVT-6 combination.

McKenzie, Bruce, "SYM On-Board Speaker Toggle Routine," pg. 21.
A routine to invert the state of the SYM speaker, producing an audible tone.

Davidson, Al, "Some Useful AIM Notes," pgs. 21-23.
How to make the AIM 65 run at MHz; AIM and KIMSI hints; etc.

Breson, Steve, "AIM Tape Problems," pg. 22.
Hints on using the AIM Tape interface.

Butterfield, Jim, "Personal Views on How Aim Compares...," pg. 23.
Point by point comparisons with PET and KIM.

Boisvert, Conrad, "A Couple of 6522 Applications Notes," pg. 24.
SY6522 Generating Long Time Intervals and SY6522 use in Generating a 1Hz Square-Wave Signal.

Solomon, Robert, "Read KIM Tapes on Your OSI System," pgs. 26-27.
How to take advantage of the large supply of KIM software using your OSI system.

566. MICRO 16, (September 1979).

Sherburne, John, "Plotting a Revolution," pgs. 5-10.
A PET Basic program for plotting.

DeJong, Dr. Marvin L., "An AIM-65 Notepad," pgs. 11-13.
Several assembly language programs that may be used to desplay input/output information.

Childress, J.D., "Applesoft Renumbering," pgs. 15-16.
A fast and reliable utility for Apple programmers, adaptable to PET and other Microsoft BASIC systems.

Herman, Prof. Harvey B., "MOVE IT: Relocating PET Source Programs and Object Code," pgs. 17-19.
Let the PET move it—this article tells how.

Auricchio, Richard R., "Life in the Fast Lane," pgs. 21-24.
A high speed version of the game of LIFE. For the Apple.

Faris, Stephen J., "SYM-1 Event Timer," pgs. 26-27.
A 100 KHz event timer using the on-board 6532 can handle up to 50 elapsed time intervals or successive timed events.

DeJong, Dr. Marvin L., "AIM-65 in the Ham Shack," pgs. 29-31.
A message transmitter and keyer which will accept and save messages to be transmitted automatically on request.

Husbands, Charles R., "Speech Processor for the PET," pgs. 35-39.
Digitized speech can be stored, cataloged, procesed as discrete data, and output through a D/A converter.

Vrtis, Nicholas, "Tiny PILOT: An Educational Language for the 6502," pgs. 41-48.
PILOT, a higher level language used for computer aided instruction, for the 6502, including an editor and an interpreter.

Rowe, Mike (staff), "The Micro Software Catalog: XII," pgs. 51-52.
Eight programs are reviewed.

McCreary, Dann, "8080 Simulation with a 6502," pgs. 53-56.
How your micro can assist program development for other machines.

Spilman, Shawn, "Writing for MICRO," pgs. 59-62.
6502 enthusiasts should try their hand at contributing articles for publication.

567. Stems from the Apple 2, Iss. 8 (August 1979).

Hoggatt, Ken, "Ken's Korner," pg. 2.
How to check for duplicates in a file. For the Apple.

Smith, Eric, "Scientific Programming 1," pg. 3.
Circle and spiral plotting in Hires on the Apple.

Anon, "Dick Sedgewick's New Computing Language, IMA," pg. 4.
IMA is an anarcism for Integer-Machine-Applesoft, and allows the use of all three languages in a single program, on the Apple.

Rivers, Jerry, "Graphically Speaking," pgs. 5-10.
A tutorial on HIRES shapes with several demo listings.

568. The Apple Shoppe, Vol 1, Iss. 3 (August 1979).

Clancy, Lee, "File Cabinet Fix," pg. 6.
File Cabinet (Apple User Library No. 3) search function can be extended to make it more useful.

Anon, "Graphics Workshop," pg. 7.
A tutorial on the famous Shapes program of Apple Hires.

Clark, Kim, "Water Cooled PETs," pgs. 9-11.
A drastic and probably fatal solution to overheated PET ROMs.

Wayne, Phil, "Language Lab," pgs. 11-17.
A tutorial on Pascal. In addition to the official "Apple Pascal" there is Programma's "Clarity Pascal (Tiny Pascal) and the soon to be released Programma "APscaLLE."

Williams, Ed., "Apple Speed," pgs. 19-20.
A utility to find the ratio of two different speed settings for Apple programs.

Welman, Chuck, "Binary Program Saver," pgs. 20-21.
This program determines the starting address and length of a binary program and then saves it.

Welman, Chuck, "THE DOS Dumper," pgs. 21-22.
This program lists the Apple II DOS Commands and their Memory locations.

Chavez, Franklin, "INDEX FILE by Programma," pgs. 22-23.
The reviewer gives this program high scores for quality and ease of use, but there are restrictions in format.

Crouch, Bill, "Making WHATSIT? Print," pgs. 24-27.
Modification of this popular Apple program for hard copy.

Crouch, Bill, "WHATSIT?"—48K Apple Version," pgs. 28-29.
A very good report on this expensive but still worthwhile program.

569. Apple Peelings 1 No. 2 (September 1979)

Silverman, Ken, "September DOM," pg. 5.
A discussion of the programs on the Septgember DISK OF THE MONTH of the Apple Core.

Baldwin, G.R., "Don't Delay Until You've Read This," pg. 6.
The use of an Apple Monitor program called WAIT will give small delays of 1 to 162 milliseconds, Loops can be used to multiply this to longer delays.

Nareff, Max J., "Screenpauses—A Compilation," pg. 7.
A series of subroutines for conditional and unconditional delays for Integer or Applesoft Basic.

Burr, Richard, "Pascal Tips—Single Disk Users," pg. 7.
Special precautions if only one disk is used.

Nareff, Max J., "Avoiding Line Overruns and Blanks," pg. 8.
How to set up templets or formats for print statements.

Bernheim, Phil, "POKE 51,0—AUseful DOS COmmand," pg. 9.
An Applesoft/DOS fix.

570. Kilobaud Microcomputing No. 34 (October 1979).

Lindsay, Len, "PET-pourri," pgs. 16-20.
Discussion of the Commodore Disk and the Computhink Disk, Disk software, Printer Update, Stop Key Disable, Merge Programs, etc.

Pepper, Clement S., "KIMCTR Measures Capacitance," pg. 64-66.
Measure capacitance 1pf to 999.999mfd.

Rogers, Kendal T., "Beefing Up PET," pg. 122.
You can have machine language and BASIC too in your PET. The machine language is up to 500 times faster.

Miles, Kenneth, "Apple's Documentation Strikes Again," pgs. 132-133.
The Apple Programmer's AID ROM comes with an excellent 96 page manual.

Klosson, Ken, "Pig Latin," pgs. 162-163.
Pig Latin for the OSI Computers using OSI 6502 Microsoft 8K Basic.

571. Interface Age 4, No. 10 (October 1979

Inman, Don, "Apples, Computers and Teachers," pgs. 68-72.
A workshop for teachers using strings to introduce BASIC statements and programming techniques.

572. Southeastern Software Newsletter, Iss. 12 (September 1979).

Carpenter, Chuck. "Indexing Fundamentals: Part I," pg. 2.
A tutorial on absolute addressing, to be followed by another article on indirect addressing, and symbolic addressing.

Burnett, Carol. "LO-RES Graphics Program," pg. 4-5.
Graphics with sound.

Anon. "Formatting $ and Cents," pg. 5.
A useful utility.

Anon. "Gas and Mileage Chart," pg. 6-7.
Uses variables of miles per gallon, price per gallon, mileage and cost.

573. Recreational Computing 8, No 2 (Sept/Oct 1979)

Wells, Arthur, J. "An Apple PILOT Interpreter," pg. 24-27.
How to write and use programs written in PILOT using a language interpreter program on your Apple II.

Saal, Harry. "SPOT," pg. 54-55.
New games for the PET, etc.

Day, Jim. "Graphic Triples for Apple II," pg. 56.
Program generates and displays Pythagorean triples.

574. Stems from the Apple 2, Iss. 9. (Sept. 1979)

Hoggatt, Ken. "Ken's Korner," pg. 2.
Quick test to see which DOS (3.1 or 3.2) you have on disk. Misc. other hints and kinks.

Anon. "Faster than a Speeding Byte," pg. 4.
All about the speed of microprocessors. The OSI Challenger (6502) is the only known production micro on the market to run faster than an Apple II.

Keyes, Patricia. "A String$ Expression Compiler for Applesoft," pg. 8.

575. Dr. Dobbs Journal 4, Iss. 9, No 39 (Oct. 1979)

Diaz, Robert D. "Page List for the Apple," pg. 19-21.
Program to permit listing one page at a time.

Ratliff, Gary. "Just Poking Around with my PET," pg. 22-23.
Method of dumping memory to the screen and how to view the PET Basic Interpreter.

Wheeler, Steve. "Quick and Dirty Routines for the Sweet-16," pg. 24-25.
A modification to the cassette version of the S-C Assembler II which gives it the capability to recognize and correctly assemble SWEET-16 mnemonics.

Morganstein, David. "David and Goliath," pg. 30-31.
How to set up the OSI 430 Board RS232 interface to hook up a modem and phone lines.

Tan, B.T.G. "Common Instructions of the 6800 and 6502," pg 38-39.
A tutorial comparing the instruction sets of two microprocessors.

576. ABACUS II Newsletter, Iss. 9/10 (Sept/Oct 1979)

Luebbert, Prof. William F. "What's Where in the Apple," pg. 1-8.
A very comprehensive memory map (reprinted from MICRO).

Anon. "Real Handy Data," pg. 9-11.
A series of Routines for the Apple, Tables of Apple Data, ASCII chart, Monitor Cross reference tables, Applesoft and Integer Token charts, Applesoft Interpreter Set, Apple DOS symbol table listing, Zero page usage, Lists of Computer Clubs, Apple User Groups, etc.

577. Byte 4, No 10 (Oct. 1979)

Garber, Joseph P. "Computer Generated Maps," pg. 18.
A Hires Apple program for a map of the U.S.

578. The Seed 1, Iss. 1 (June 1979)

Willmore, Richard. "Recursive Primer," pg. 5-9.
A tutorial on block-structured languages and their most powerful feature, recursion, which allows a program to create new copies of itself as the problem demands.

Knaster, Scott. "Controlling Control-C," pg. 9.
An Applesoft program for use in turn-key systems which disables the Control C function of the Apple.

Anon. "Illegal Line Numbers," pg. 10.
How to use illegal line numbers in a program, for the Apple. Works only on Cassettes. For DOS use 9600 instead of BFFF.

Anon. "Nifty Self Run," pg. 11.
How to make any Applesoft program self-starting as it is loading.

Anon. "Apple Pi Library," pg. 12-15.
A list of 472 programs for the Apple.

Nelsen, Rodney. "String Sorter," pg. 15.
This program alphabetizes your strings on the Apple.

579. The Seed 1, No 2 (July 1979)

Taylor, T.N. "Apple Groups Publications," pg. 5.
A list of Apple User Group Publications.

Willmore, Richard. "Reclusive Primer," pg. 5-7.
A tutorial on block-structured languages.

Willmore, Richard. "Tools for Exploring a Frontier," pg. 8-12.
All about compilers, XPL, PASCAL, Assemblers, File systems, Apple DOS, Floppy File Systems, etc.

Taylor, Terry, "Apple Pi Library," pgs. 13-20.
863 Programs are listed, for the Apple.

Anon, "Apple Pascal," pgs. 3-6.
A well written description of what to do when you get your Pascal package.

Taylor, Terry, "Apple Pi Library," pgs. 7-9.
Some 400 new programs are listed bringing the total to over 1400.

Brown, Donald, "The $7.00 Two-Apple Hookup," pgs. 9-10.
All about the SART Routines that permit extremely cheap communications between two Apples.

Anon, "Information Processing: Surviving a Microcomputer Shift," pgs. 12-13.
Reprinted from Business Week. A discussion with Computer marketing experts. How the Apple and TRS-80 have changed the small computer picture.

580. Compute, Iss. 7 (Fall 1979)

Hulon, Belinda and Hulon, Rick, "Sorting Sorts: A Programming Notebook," pg. 7.
Several Sort methods are evaluated including Selection Sort, Bubble Sort, and Shell Sort. In a later article Machine Language sorts will be discussed.

Lindsay, Len, "Three Word Processors," pgs. 13-20.
An overview of three word processors for the PET.

Byrd, J.S., "Microcomputers for Nuclear Instrumentation," pgs. 24-26.
Describes the use of PET microcomputers in handling several applications at the Savannah River Laboratory.

Herman, Harvey, B., "Tokens Aren't Just for Subways," pgs. 29-30.
A convenient method to list Microsoft Basic Tokens for the PET.

JOIN RAYGAMCO NOW!

Become a member of RAYGAMCO Computer Discount Club.

SAVE 20% AND MORE!

BIG SAVINGS ON EVERY ITEM!

By being a RAYGAMCO Member you receive substantial discounts on every item you purchase, including all hardware, software, accessories, even books and paper! You will also receive a monthly newsletter with all the latest available for your particular computer system, and much, much more — exclusive to RAYGAMCO Members only!

All the famous brand names, including:

APPLE	Alpha Micro	Soroc	Lear Siegler
ATARI	Alpha Pro	Hazeltine	Shugart
EXIDY/Sorcerer	Cromemco	Sektor	Texas Instruments
Kim/Commodore	Xerox	PET	

SAVE 20% AND MORE!

Here's how to join.

Fill out the information, and mail. That's all there is to it. Nothing to buy.
I want to be a RAYGAMCO Computer Discount Club Member. Please send my RAYGAMCO Membership card to:

Name ____________________

Address ____________________

City ____________ State ____________ Zip ________

Computer (Brand Name) ____________________

I would like information on (please specify system, part, accessory, book, program, etc.)

____

____

WE HONOR VISA, MASTERCHARGE, BANKAMERICARD.

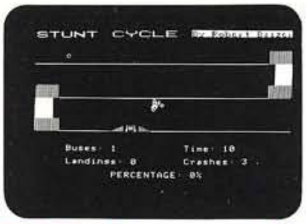
Store Hours: Sat 10-6, Sun 12-4, Tu-Fri 11-8

TOLL FREE, EXCEPT CA
800-854-6455

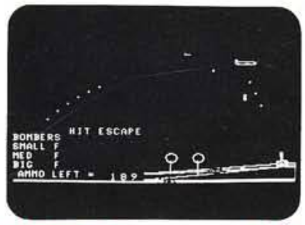
RAYGAM, INC.

6791 WESTMINSTER AVENUE WESTMINSTER, CA 92683
TELEX 182274 (714) 891-2587

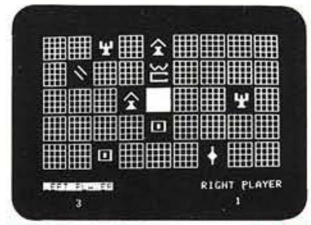
NEW APPLE II SOFTWARE



STUNT CYCLE $15.95

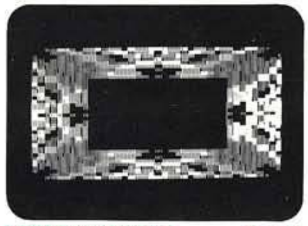


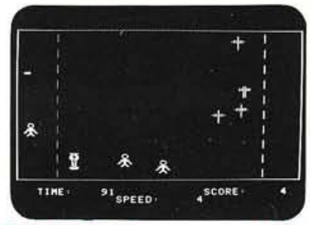
BLITZKRIEG $15.95



FRUSTRATION $ 9.95

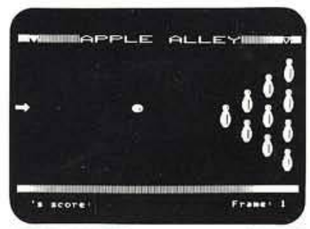
PERPETUAL CALENDAR $ 9.95


LORES HYPERPAK $ 6.95

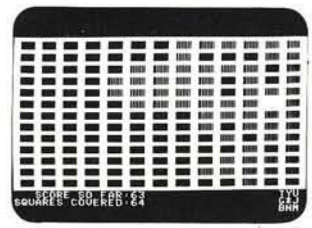


DEATH RACE $15.95

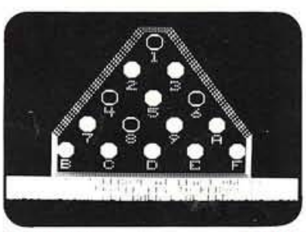
CLOWNS & BALLOONS $15.95

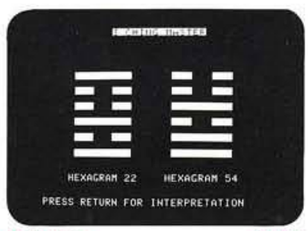


APPLE ALLEY $ 6.95

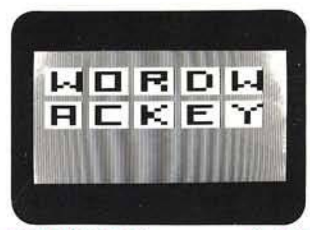


MOUSE HOLE $ 6.95

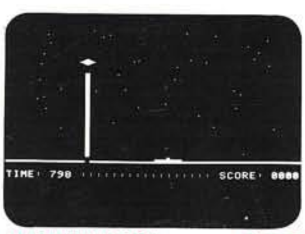

PEG JUMP $ 9.95



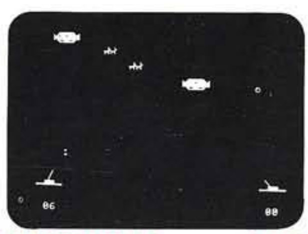
I-CHING $15.95

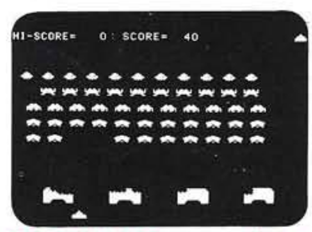


WORDWACKEY $ 6.95

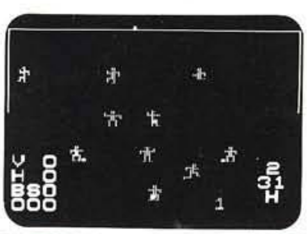


ALIEN INVASION $ 9.95


GUIDED MISSILE $15.95

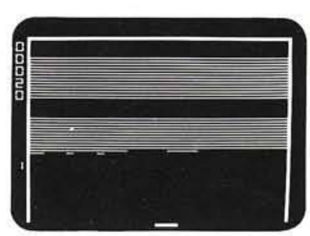


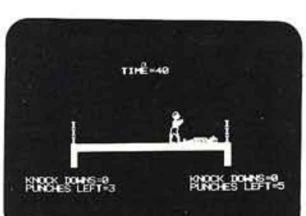
APPLE INVADER $15.95


BASEBALL $15.95

All orders must include 3% postage and handling with a minimum of $1.00. California residents include 6% sales tax.

VISA MASTERCHARGE


BREAKTHRU $ 9.95

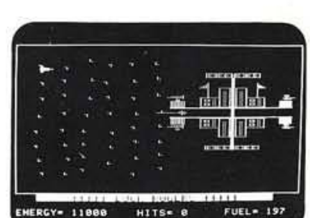


BOXING $ 9.95

Apple II is a trademark of Apple Computers, Inc.

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

Dealer Inquiries Invited



STAR VOYAGER $15.95

PROGRAMMA Software Program Products