Information about your APPLE computer

$5.00

the San Francisco APPLE CORE presents

THE BEST OF

CIDER PRESS

1978-1979

JL

DB Master

THE DATA BASE YOUR APPLE HAS BEEN WAITING FOR!

Available September 1980
at the computer store near you.

48K/ROM APPLESOFT/DISK

"HUMAN ENGINEERED" FOR EASE OF USE BY NON PROGRAMMERS

- No complex "menu trees"
- Easy-to-follow prompting
- Formats displayed on screen as you create them
- Easy . . . Convenient editing for all inputs
- 10 data types, including auto-formatting for dates, phone numbers, social security numbers
- Calculator style right to left input for $ fields

LARGE CAPACITY

- Up to 100 fields (items) per record
- Record size to 1020 bytes
- 1 - 4 fields in primary keys
- Any number of secondary keys
- 1 to 9 user formatted screen pages or "masks" per record, and 1 - 24 fields per "mask"
- Special "update masks" for rapid editing

DATA PACKING FOR INCREASED DISK CAPACITY

- Integers stored in one or 2 bytes, real numbers in 5 bytes
- Recurring characters and un-used portions of alphanumeric fields packed into 2 bytes
- Special code fields store short codes, but print full descriptions on reports

POWERFUL REPORT GENERATOR

- Up to 100 columns on 1 to 9 lines
- Up to 24 computed fields—compute with record values, constants or other computed fields
- Up to 5 horizontal subtotals (assign any field to any subtotal) and grand total
- Up to 6 sort and control break fields—column sub totals at control breaks and page breaks
- Control break or compute on printing and non-printing fields
- Comment lines and fields, auto page numbering and report dating, number formatting, and much more!

FAST ISAM FILING SYSTEM

- Retrieve ANY RECORD from a FULL DISK, IN LESS THAN 3 SECONDS
- Search for records by any combination of range, wild cards, partial strings, and relationals, etc.
- Records are sorted by primary and secondary keys AS THEY ARE ENTERED
- Three levels of password protection

All specifications are preliminary and subject to change.

Also Available:

Aristotle's Apple *by Scot Kamins*
A computerized tutor for ANY subject, at ANY level.
$34.95 48K/Disk/Applesoft

ELECTRONIC PRICE-SHEET *by Scot Kamins*
A sales tool for retailers. **$100 48K/Disk/Applesoft**

MICRO MEMO *by Barney Stone*
$39.95 48K/DISK
A powerful, easy to use appointment calendar.

Micro League Baseball *by Arthur Wells*
Finally . . . The Hi-res Baseball that's as good as the Apple!
$24.95 32K/Disk Applesoft or Integer

TRANQUILITY BASE
by Bill Budge, creator of Trilogy and Penny Arcade
$29.95 32K/Disk/Applesoft or Integer

Bloody murder! *by Arthur Wells*
$19.95 48K/Disk Integer
A 2-player knife fight—rated "R" for violence!

1930 Fourth Street, San Rafael, CA 94901
(415) 454-6500

Calif. Res. Add 6% Sales Tax. No COD's. Add $2.00 for Shipping & Handling. Use Check, Money Order, VISA or MASTERCARD (add Exp.).

APPLE II is a TM of Apple Computer, Inc.

DEALER INQUIRIES INVITED.

INTRODUCTION

The SAN FRANCISCO APPLE CORE started in April of 1978 because I couldn't figure out how to use my Apple II. While the people in the store where I bought my machine weren't exactly disdainful of my approaches for help, it became clear fairly quickly that they knew very little more than I did about the beast.

My background is in the humanities and decidedly not in the hard sciences. But I did seem to have a talent for organizing born out of my work in the various freedom and peace movements of the 60's. "let's see", I foolishly thought. "If I write to a few magazines and contact a store or two, maybe I can get a few guys together once a month to rap..."

Sure, kid.

The Apple Core's membership is now hovering around 800. We have over 400 working programs in the library. There are around six active committees and about a dozen highly active volunteers running things.

There are at least four (count them) meetings per month, including one general, one political/organizational and one beginners's and one newsletter.

We had two publications - APPLE PEELING for input and the CIDER PRESS for ouput.

Personally, I think things have gotten somewhat out of hand.

Be that as it may, the APPLE CORE had its roots in the concept of mutual aid. The idea was for people who did't know much to get together and share what knowledge they had, and I supposed, that the little pieces of knowing would mate and reproduce).

It is to our credit (pat-pat-pat) that the spirit continues. The APPLE CORE is still a place where beginners can come and get help. But we now have a fair contingent of heavies who share arcane knowledge in the dark of the night, too. A bunch of us run computer stores or write contract programs or work at the source itself (Gaw-lee!).

Of course, aside from the above, we have our strange sorts, too.

There's John Scribblemonger, our somewhat competent, crude-but-effective first publications editor. At last report, he had sold his mansion high above Milpitas and has moved to Three Mile Island.

J. Alfred Glitch needs no introduction - so we'll skip him. There's the nefarious Dr. Qwan W. Min and his counterpart, Professor Schmuch the good , both of whom carry out a classic good-and-evil struggle via modems.

And, of course, there's the Commodore's Pet DOS, Tige.

I could go on and on, but HIMEM's being approached.

This edition of the Best of the Cider Press represents our various attempts to pull our computers and our sanity together.

We hope you enjoy reading the articles and applying the suggestions as much as we enjoyed devising and writing them. Keep on coding.

SCOT KAMINS - ONE OF THE FOUNDERS OF APPLE CORE

TABLE OF CONTENTS

SF APPLE CORE

BEST OF

"THE CIDER PRESS"

SECOND PRINTING 8/80

STAFF

EDITOR/PUBLISHER	KEN SILVERMAN
ASSISTANT TO EDITOR	KATHY STARK LEROY LARSEN
ASSISTANT EDITORS	PHIL BERNHEIM RANDY FIELDS GLEN HOAG GENE WILSON
ADVERTISING	MARTY COHEN PETER WEIGLIN
CALLIGRAPHY	JIM LINHART
GRAPHICS	JIM LINHART PEGGY HUGHES EMILIE HANCE

1979 OFFICERS

PRESIDENT	FRED WILKINSON
VICE PRESIDENT	BRUCE TOGNAZZINI
TREASURER	GOEFF BALDWIN
SECRETARY	BILL NEFF

1980 OFFICERS

PRESIDENT	RANDY FIELDS
VICE PRESIDENT	KAREN WEISS
TREASURER	PHIL BERNHEIM
SECRETARY	BILL NEFF

APPLE CORE

The APPLE CORE OF SAN FRANCISCO is a non-profit organization comprised of, and supported by, Apple II Computer owners. The Apple Core is run entirely by volunteer officers and committees. The club endeavors to aid other APPLE owners. All members are individuals (and their families), and NO shops, stores, or corporations are directly registered. (However, any shop may register an employee c/o that shop).

MEETINGS

MONTHLY MEETINGS are held at Fort Mason and is located in San Francisco at Laguna and Marina Blvd. General meetings start at 10 AM. The building is C and the room number is 300. Meetings are held on the first Saturday of each month. Anyone interested in the Apple Computer is welcome to attend.

CIDER PRESS POLICY

All manuscripts, photography, and other materials are submitted free and released for publication. They become the property of the Apple Core and the Cider Press. Authors should clearly mark all material submitted for publication so that credit may be given.

The publishers/editors do not necessarily agree with, nor stand responsible for, opinons expressed or implied by others than themselves in this publication.

ADVERTISER'S INDEX

GENERAL FEATURES

TERMINOLOGY

BY KEN SILVERMAN

Just as in any other speciality, the people involved in data processing equipment have developed their own terminology. Some of these terms are defined in the following list. The definitions are not rigorous and the list is far from complete.

ALPHANUMERIC-Having characters only. A CRT that is an alphanumeric CRT displays only characters and cannot draw lines, etc. for graphic pictures. The characters displayed are usually those specified by ASCII.

ASCII-Acronym for American Standard Code for Information Interchange.

ASSEMBLER-Program that translates assembly language to machine language.

ASSEMBLY LANGUAGE-Symbolic codes representing CPU instructions.

BASIC-Beginners All-purpose Symbolic Intruction Code. The most common high level language for personal computers.

BAUD-A unit of signaling speed derived from the duration of the shortest signaling event. If each signaling event is exactly one bit, then the baud rate is the same as bits per second.

BIT-One binary digit.

BOOTSTRAP-An instruction set usually in ROM or RAM that is executed automatically when the system is turned on, to initialize the system.

BUS-The very high speed communication path between the components of the computer in the mainframe.

BYTE-A group of 8 bits. Usually one byte is used to represent a character in ASCII code.

COMPILER-A program to translate from higher level language such as Pascal to machine language.

CPU-Central Processing Unit, such as the 6502 in the Apple.

DOS-Disk Operating System.

FIRMWARE-Program built into the system, in ROM or RAM.

HARDWARE-The physical components of the system.

HEXIDECIMAL-A numbering system based on powers of 16.

I/O-Input/Output of information to or from the computer.

OCTAL-A numbering system based on powers of 8.

RAM-Random Access Memory that can be both written on and read from.

ROM-Read Only Memory. Memory with permanent instructions that cannot be erased.

SOFTWARE-All programs including systems or applications programs.

There are a great deal more terms used by computer persons today. If you wish to know the definition of some term, just write to the Cider Press, c/o Ken Silverman, and I will try to answer them for you.

APPLE BEEPS TRANSLATED

BY ANDY HERTZFELD

You may have wondered why the APPLE beeps twice when loading a BASIC program from tape (once at the beginning, once at the end), but only once (at the end) when loading a machine language program. The answer is very simple: a BASIC "load" actually consists of TWO monitor loads, each of which produces its own beep.

It is necessary to do two loads for each BASIC program because the monitor must know how long a program is before it can read a program in. Thus before the body of a BASIC program is saved its length is calculated and saved first; thus the monitor can determine how much data is out there when reading it back in. If you listen carefully to a tape of a BASIC program, you'll notice that the leader tone makes a very short dip in pitch about a second or two before the main part of the program begins. This dip in pitch is actually 2 bytes worth of data telling monitor the length of the following BASIC program.

Thus we have a very easy way to find out the length of a BASIC program without even reading the whole thing in! Get into the monitor (one way is to hit reset) and type 300.301R (you can actually use any two adjacent addresses) and start the tape recorder. This will load the length of the program into the specified memory locations, low-order byte first. Then all you need do is read those addresses from the monitor and convert them from hex to decimal - and there you have it!!

SORTING

BY ART MACK

One of the things that every programmer has to do eventually is to sort some data. Described here is one way to write a sort routine in BASIC as follows.

Starting with the first element of an array (the sort shown here is for an array of real numbers but applies to Integer and String arrays as well) the array is searched to find the smallest element (assuming we want to sort into ascending order) and that element is placed in position 1 of the array. Next, we start with element number 2 and find the smallest element left in the array and place it in position number 2. We continue this way with positions 3,4,5, etc. until we reach the last element in the array. Thus when we are finished, the smallest element in the array is in position 1, the second smallest in position 2, and so on.

The following example shows how to sort an array named "D" of dimension "N" (assume that all the elements of "D" have already been put into the array).

```
800 FOR I = 1 TO N-1 : REM LOOP FOR N-1 TIMES, THE LAST ELEMENT WILL AUTOMATICALLY BE IN THE RIGHT PLACE.
801 FOR J = I+1 TO N : REM LOOP AND CHECK THE I'TH ELEMENT WITH ALL REMAINING ELEMENTS IN THE ARRAY.
802 IF D(I)<D(J) THEN 806 : REM IF THE I'TH ELEMENT IS THE SMALLEST TRY THE NEXT REMAINING ELEMENT.
803 A = D(I) : REM OTHERWISE SWITCH THE ELEMENTS (LINES 803 - 805).
804 D(I) = D(J)
805 D(J) = A
806 NEXT J : REM CONTINUE FOR THE NEXT J
807 NEXT I : REM CONTINUE FOR THE NEXT I
```

Note that the "inner" loop (lines 801 - 806) is executed from N-1 to 1 times depending on the value of I, and that the "outer" loop (lines 800 - 807) is executed N-1 times. Also note that the routine can easily be changed to sort the array into descending order (highest to lowest values) simply by changing line 802 to:

```
802 IF D(I)>D(J) THEN 806
```

RAM?

BY TONY HUGHES

You may have the impression now that we have an official animal, or pound on the APPLE keyboard with more than usual force, but in reality, RAM is an acronym that means Random Access Memory. The APPLE can hold several thousand letters, numbers, and special characters in its memory and can look at them in any order (i.e. the first one, then the tenth, then the 1024th, etc.), unlike a tape recorder that must read from the beginning of a tape to the end.

There is another meaning to RAM. It can be erased and written over like a magnetic tape. Unlike a tape, RAM is impermanent; when you turn the APPLE off, your program goes away. You turn it on again and you have to reload your favorite game.

RAM (CONT.)

There is another kind of memory, called ROM, or Read Only Memory. Like RAM, the APPLE can begin reading its characters from any point in memory. Unlike RAM, ROM is permanent, like a phonograph record; its characters are engraved at the factory and can never be changed.

Why does the APPLE have both RAM and ROM? Well, any of the hundreds or thousands of programs we might want to run will fit into the same area of RAM. Just like erasing and reusing a blackboard. But a computer like the APPLE also needs a special program that occasionally takes over to do some mundane housekeeping task. When you press Control and G simultaneously (on the keyboard) you will hear a beep. Now the APPLE isn't electronically wired to produce that beep, like a doorbell, but must manufacture it by performing dozens of steps within itself. Then you hear it. The program that produces beeps, as well as others that perform similar functions are all down in ROM somewhere.

RAM for the APPLE is in the form of an electronic component called a chip. The chip has the size and appearance of a black, rectangular caterpillar. A 4k chip will have 4096 pigeonholes, each and any of which can hold one bit of information; that bit will either be yes or no, off or on. If the APPLE looks at pigeonhole 1234, for example, it may find the bit there to be set on, but we're still a long way from finding a number or letter. If we use, for example, 8 bits in tandem, we can represent up to 256 possible characters, and in fact this is done. The APPLE will look at pigeonhole 1234 in 8 chips; the first chip is off at that location, the second on, and so forth. So just like Morse code, we've spelled out a letter or number (actually, a unique character) at that address.

So a row of 8 4k chips give the APPLE a memory of 4096 characters. The APPLE will hold up to 3 rows of 4k or 16k chips, so its memory could be as high as 49152 characters. There can be 12288 characters in ROM. The rest is taken up by some electronic functions.

MEMORY CELLS

BY KEN SILVERMAN

The 16K dynamic RAM, used in your Apple, is getting cheaper all the time. In fact some mail order houses are advertising 16K bytes (8 units) as low as $65.

Installing your own RAM is not a problem; it is all explained on pages 133 - 135 of your red manual. However there can be a potential problem to the purchaser of these chips, i.e. speed. After talking to the Engineers at Apple Computer I was told that 200ns chips exceed Apple's specifications, 250ns are the standard speed and that 300ns or slower chips should not be used. It seems when 300ns or slower are used a zero will turn into a one once in a while. This has to do with the correct delay time between the two address strobe signals (during the memory access cycle). In addition the Apple timing doesn't allow long enough precharge or row address hold time for the slow 300ns parts.

Find out what the manufacturer's specifications are for the RAM you plan to purchase before you buy. Write first or deal with suppliers who will specify the speed.

SCREENPAUSE

BY SCOT KAMINS

It is always a good idea to include complete documentation as to how a program works within the body of a program. Quite often, however, that information takes up more than a full screen (i.e., more than 24 lines) to display. Thus the programmer needs some device to allow him/her to stop the screen from scrolling before the user is done reading the information being shown on the screen. One way to accomplish this goal is the delay loop. All you need to do is insert "FOR STALL=1 TO 2000:NEXT STALL" in some appropriate place in the printout subroutine.

While that method will halt scrolling for a few seconds, it is not the most desirable way to do things. Slow readers will still have trouble keeping up, and speed readers will get bored.

A less primitive method is here-in offered, making use of "PEEK (37)" (absolute vertical position of the cursor) and "PEEK (-16384)" (read keyboard strobe). Applesoft users will note that the "GET" function can be substituted for the "keyboard-strobe and-clear" peeks and pokes in lines 1030 thru 1050.

```
 0   CALL -936
10   FOR EXAMPLE = 1 TO 1000
20   PRINT EXAMPLE
30   IF PEEK (37) = 20 THEN GOSUB
     1000: REM THIS CHECKS IF
     CURSOR IS AT TEXT LINE 20
40   NEXT EXAMPLE
50   END
.........
.........
1000 REM SCREENPAUSE BEGINS HERE
1010 VTAB 23: TAB 15: REM FORMATS
     THE SCREEN FOR THE
     PROMPTER PRINTOUT
1020 PRINT "PRESS ANY KEY TO CONTINUE"
1030 Z=PEEK (-16384): REM CHECKS TO
     SEE IF ANY KEY HAS BEEN PRESSED
1040 POKE -16368,0: REM RESETS THE
     KEYBOARD "CHECKER"
1050 IF Z>127 THEN 1070: REM IF A KEY
     HAS BEEN PRESSED THEN GOTO 1070
1060 GOTO 1030: REM KEEP CHECKING
1070 CALL -936: REM CLEAR THE SCREEN
1080 RETURN: REM GO PUT MORE
     INFORMATION ON THE SCREEN
```

R&R DEC DUMPS

BY MAX J. NAREFF

Calculations in Applesoft usually result in long multidecimal numbers. While the accuracy of the numbers is commendable, long mantissas are often not necessary; frequently they are disruptive of the three - columns - via - commas screen format the Apple provides.

Following is a simple function for reducing post-decimal numbers and for rounding off the residuals (R&R).

Of the many functions preprogrammed in Applesoft, Integer is used to drop the fractional part of a number. The excision is sharp and clear.

```
Y=INT(3.4729): PRINT Y
```

Will result in "3". It doesn't round off; it truncates. Thus:

```
Z=INT(6.87563): PRINT Z
```

Yields "6", with everything to the right of the decimal ignored. In order to retain numbers to the right of the decimal and to "round them off", we must define a function ourselves. We create this special action by means of the DEF(ined) F(unctio)N command.

The function reads DEF FNA(X)=INT(X) where FN indicates function and the following letter can be any alphabetic character merely serving to define that particular function at the time of its use. The (X) is the value to be acted upon.

The following simple program tells the story:

```
20 DATA 3.4678,19.2062,11.562,1
   41.45917,1000
30 Read x
40 IF X=1000 THEN 70
50 PRINT X, INT(X)
60 GOTO 30
70 END
```

NOTE THAT THE STANDARD INTEGER FUNCTION, INT(X), DROPS THE DECIMALS. NOW ADD -

```
5 DEF FNA(X)=INT(X*100 +.5)/100
```

AND REWRITE LINE 50 TO READ

```
50 PRINT X, INT(X), FNA (X)
```

Note how the DEF FNA(X) limits the post-decimal numbers to two and "rounds them off". The latter is due to the addition of .5, which increases by 1 those decimal numbers .5 or more; anything less is not propelled over the carry-over cliff. When dealing with several variables (X,Y,Z), just plug them between the parentheses.

ALGORHYTHMS

BY BRUCE TOGNAZZINI

A few of you will recall the exciting algorhythm presented several issues ago (August of 1978, I believe) for filling in a block of memory with a desired number. This would be useful for doing things like clearing the hires screen, setting basic memory space to all zeroes and then loading a program for purposes of exploration, or perhaps for "seeding" a Life game. The method presented at that time for filling in the Hexadecimal locations $00.$30 with all EA's (the no-op(eration) command in 6502 code) was as follows: Type 0:EA EA EA EA.....EA. Now this worked very well back in 1973 on the Apple -1 when our memory only went from $00.$30, but now that we've gone from 48 bytes to 48K bytes, it can take several days.

So we offer a solution so daring, so imaginative, that we didn't even think of it. No, it took one of those unsung resident geniuses at APPLE to come up with this one. To fill (for example) $400 to $7FF with EA's, type the following:

```
*400:EA 401<400.7FEM ("RETURN")
```

Now don't panic, that was the screen buffer you just filled. Try preceeding the "400" with "C050" to turn on graphics and change the "EA" to some other "seeds". And when you're ready for the bit time, try this:

```
*C050 C053 C057 2000:10 73 EA 0 14 AF
2006<2000.3FF9M ("RETURN")
```

The general formula follows:

```
(startaddress):((digit(digit))
((space) (startaddress+number of
(digit(digit))s) < (startaddress) .
(endaddress-number of (digit(digit))s)
M(ove) (return)
```

There is another interesting thing that we can do by forcing the computer to repeat its steps. To do this, start your command sequence with the "N" command and terminate it with "(space) 34:0 (space) (space)." For example, try this one:

```
*N C050 C053 C057 2000:01 34 64 82 96
73 2006<2000.3FF9M 2000:FF A0 77 B3 D9
EE 2006<2000.3FF9M 34:0 (space)(space)
("RETURN")
```

For a technical explanation of why this works, we turn now to S. Wozniak, that delightful fellow who whispers *** SYNTAX ERR at us every night.

The APPLE-II MONITOR command input buffer begins at location $200 and is scanned from beginning to end after the user finishes the line by typing a carriage return. An index to the next executable character of the buffer resides in location #34 while any function is being executed. By adding the command "34:0" to the end of a MONITOR command sequence the user causes scanning to resume at the beginning. Because the "34:0" command leaves the MONITOR in "store" mode, an "N" (normal text) command should begin the line. (---printed by special permission of Steven Wozniak)

There, now that wasn't difficult to understand, now was it (Hey, Scot, what's a MONITOR? What about the Civil War? Oh.) Try different "seed" numbers with lores and hires and let us know what you come up with. Have fun.....

DISK? STRINGS ON TAPE

BY PHIL BERNHEIM

The answer is no. All that disk can do for you, as far as I can see, is save time and trouble - and since trouble is merely the trouble of writing slightly more complicated programs, that, too, reduces to time.

Now time may be costly in a commercial environment, but it certainly isn't when it comes to personal computing or fun and games. Of course, if you WANT to go ahead and spend $600.00 to save time, don't let me stop you; but if, like me, you think eleven hundred bucks is enough already, here are some tips to help you do with your little cassette what the big boys do with their disks.

Take that matter of string storage, for instance. On disk, easy - just file it! But on cassette you can't STORE literals - they have to be transcribed to ASCII code first. Only where is the routine for doing that?

Not wanting to re-invent the tea-kettle, I asked around. Nobody seemed to know. What's more, nobody seemed to care. The programming wizards all have disks. So I invented the tea-kettle, and here is a result in a sample demo program. You can use the sub-routines and a few essential lines in the main program in your own programs.

```
10 L$="NOW AND THEN": REM
   SAMPLE STRING FOR TEST PUR-
   POSES; USE YOUR OWN INSTEAD
20 S$="HERE AND HEREAFTER":REM
   SAME AS ABOVE
25 DIM A(25),A$(25),A1$(25):REM
   SET ALL 3 TO LENGTH OF YOUR
   LONGEST STRING
28 L=0:C=0:REM INITIALIZES
   COUNTERS--IF YOU HAVE OVER
   10 STRINGS IN YOUR PROGRAM
   YOU WILL HAVE TO DIMENSION
   C TO THE NUMBER OF STRINGS
30 A$=L$:REM MAKES A$ (THE
   SCRATCH VARIABLE) EQUAL TO
   YOUR FIRST STRING
35 C=C+1:REM ESSENTIAL TO
   INCREMENT COUNTER BEFORE
   EACH CALL TO SUBROUTINE
40 GOSUB 100
60 A$=S$:REM MAKES A$ EQUAL
   TO YOUR SECOND STRING. THIS
   AND THE FOLLOWING STATEMENT
   ARE NECESSARY BEFORE EACH
   CALL TO THE SUBROUTINE
65 C=C+1
70 GOSUB 100
90 GOTO 200:REM INITIATES
   RECALLING ROUTINES--YOU
   WILL WANT TO PUT THIS
   ELSEWHERE IN YOUR PROGRAM
99 REM FOLLOWING IS SUBROUTINE
   FOR STORING EACH STRING
   (LINES 100-180
100 D=LEN (A$):L=L+1:C(L)=D
120 FOR K=1 TO D
130 A(K)=ASC (MID$ (A$,K,1):
    REM TRANSLATES STRING TO
    ASCII CODE LETTER-BY-
    LETTER
140 NEXT
150 PRINT "START TAPE IN RECORD
    MODE; PRESS RETURN": GET W$
160 STORE A
180 RETURN
200 L=0:REM RE-INITIALIZES
    COUNTER
205 GOSUB 300
210 L$=A$:REM SETS A$ (THE
    SCRATCH VARIABLE)BACK TO
    YOUR FIRST STRING VARIABLE
220 GOSUB 300
230 S$=A$:REM SETS A$ BACK TO
    YOUR SECOND STRING VARIABLE
240 PRINT L$: PRINT S$: END :REM
    THIS LINE NOT NECESSARY IN
    YOUR OPERATION
300 L=L+1:A$="": REM IN-
    CREMENTS COUNTER; NULLS A$
    FOR NEXT USE IN SUBROUTINE
310 PRINT "START TAPE IN PLAY
    MODE":GET W$
320 RECALL A
330 FOR K=1 TO C(L): REM
    LIMITS LOOP TO LENGTH OF PAR-
    TICULAR STRING BEING WORKED ON
340 A1$(K)=CHR$ (A(K)): REM
    TRANSLATES ASCII CODE NUMBER
    BACK TO LITERAL
350 A$=A$+A1$(K): REM BUILDS
    UP A$ LETTER BY LETTER
360 NEXT
370 RETURN
```

Problem number 2: I didn't have a printer, so how to stop program results from flashing across the screen so fast I couldn't even read them, much less copy them down? Thanx to Scot Kamins, here's the solution: In any print statement - usually in a loop - use this:

```
PRINT: IF PEEK (37>20THEN GOSUB 500
500 VTAB 23:PRINT "PRESS ANY LETTER
    TO CONTINUE":GET W$: HOME:
RETURN
```

So, so far, no disk for me, but I do have those 600 bucks and a lot of fun!

ARTICLES

BY SCOT KAMINS

A number of members have of late hinted that they would like to submit articles to the CIDER PRESS, but felt inhibited because "I write lousy".

Aside from the obvious comment that a lack of literary ability has so far been no deterrent to the newsletter staff, we do want to impress upon the membership that we are after information, and not deathless prose.

If you have discovered a nifty algorithm, a neat function, some hidden monitor hook or some handy hardware hint that would help a computer user on any level please send it in. Remember: the newsletter is written entirely by the membership for the membership. If we don't write it, it don't get writ.

BELLY-UP PET

CORE BIGGIE ARRESTED

DATALINE...SCANDAL

BY BRUCE TOGNAZZINI

Carnage swept this tiny seaside community this morning during what appears to have been a reiterative altercation centering upon our own beloved John Scribblemonger, just shortly after he had ventured FORTH from his hospital bed.* According to eyewitness accounts, the incident began shortly after nine A.M. as John S. brought his yacht, the "BUSTER BROWN" into the harbor of Scandal. After tying the passenger vessel (of the giant "Floating Bus" RS-232 class) up to Port 1, one of eight which parallel the Deyemmay Memoriable Daisy Field, Scribblemonger disembarked the ship, accompanied only by his trusty dos Tige (he lives there too!) who had been disembarked on a much earlier occasion.

As the Commodore and Tige walked beside the cabled walls of the great hulking slots, the dos suddenly jumped through one of the small openings, lined with dinner plates, which punctuate the port's ponderously planked platform. As Scribblemonger squeezed his massive main-frame through the hole, screaming for his little chum, the frightened dos popped up through another plated-through hole, finding himself beneath the very underpinnings of City Hall, at 6502 H St., corner of 7th. Those of you familiar with Scandal as John Scribblemonger will well understand the plight of the plucky dos as he dodged the screaming traffic. He ran east of H St., past the four-way stop lights on 13th, around the ping-pong courts on 14th near J, and then south on 14th St., past the paint store on 14th and F. He was next spotted cutting through the industrial section, over to 10th St., racing down along side the goat farm.

FOR THE SAKE OF THOSE READERS WHO CANNOT FOR THE LIFE OF THEM FIGURE OUT WHAT IS GOING ON, LET US OFFER A BRIEF EXPLANATION: THIS ARTICLE IS IN FACT A CLEVERLY DISGUISED TOUR-DE-FARE OF THE INSIDE OF YOUR APPLE. IF YOU WILL OPEN YOUR RED BOOK TO PAGE 123 AND REFER TO PAGES 143 TO 151 YOU WILL BE ABLE TO FOLLOW THE PATH OF THE ERRANT PAIR. THE ADDRESS OF THE CITY HALL, FOR EXAMPLE, TELLS YOU THAT IS LOCATED IN ROW H, COLUMN 7, AND IS A 6502 CHIP.

Eyewitness report from farmer Vidgen (rhymes with 'pigeon'): "Ayup, I recon it was long about eleven. The wife come out to the pastcha to tell me that the goats 'uz' thusty. 'Never mind', says I, 'you put on the kettle, and we'll give them rams something nice and refreshing.' It was just then the wife seen the dos go yappin'by, with that big fella lopin' just behind."

And yapping and loping they went, all the way to A St., Broadcast Row, and the local offices of The Apple Core Journal at 2513 A St., corner of 5th. Here the badly scared dos tried to find refuge but was refused entrance by an unidentified guard described only as having a "rather fiendish grin".

With a winded and wilting Scribblemonger chuffing behind him, the tired beast headed North, straight into the path of 16 bi-directional bus lines. He scurried across the rails, pausing momentarily on one of the green safety islands while a burst of bits went screaming by.

We talked with Chip Pfuller, a foreigner, who is, judging from his accent, either from Southern Whales or Northern Milpitas. (Southern Whales are found underwater and are given to a slight drawl; this fellow was both.) "Yes, sir, it shore looked like the little guy was going to make it too. But then he starts kinda verrin' North-west, toward the little Timer-life building (at 555 3rd). Well, sir, all of a sudden, this old cursor, lookin' like Columbo 'cept with a right strange smile, up and flashed him and that ole dos liked to drop right into a wait-state."

Yes, and it was while friendly Tige, the dos was locked helplessly in that wait-state, that the rogue address bus came careening around the corner, and, bearing down on him, scrambled him into apparent pseudo-randomness!

Arriving only moments later, Scribblemonger grabbed his little friend and booted him, repeatedly, in a vain effort to get him back on the right track. The bus lines became crowded with any chatter (cross-talk?) about the violent goings-on. Chip Pfuller: "He booted him pretty good. Even bounced the little sucker off the walls a couple times, but it warn't no use; everybody could see that that ole boy had a dead dos on his hands."

Scribblemonger's rising rage branched toward the offending address bus when he noticed to his horror that the bus driver was none-other than J. Alfred Glitch, that Evil Genius behind so many of our beloved editor's recent tribulations. And as the Commodore's pet lay belly-up, J. Alfred sped away from the scene, with a cursing Scribblemonger, having forced an interrupt on a data bus, taking off after him. (John had accidentally left his Lone Ranger costume at the cleaners, thereby making this his very first (sigh) non-maskable interrupt.)

Hip Flash, ace photographer for the Journal, reports that the police were shortly in hot pursuit of the pair, even going so far as to notify Oregon and Nevada of the chase, to enable (of course) the setting up of a tri-state buffer. (Nobody's forcing you to read this, you know.)

Scribblemonger proved to be a valiant bus-driver, but he was no match for the nefarious Glitch, who, being able to leap from bus line to bus line, finally managed to completely delude (and thus elude) the baffled John S. by trapping him in a loop. Yes, poor John, coming upon what he assumed to be Glitch's bus, climbed aboard and, not realizing that he had in fact looped back upon himself, repeatedly rammed the driver until he had beaten himself into something that appeared to be (according to photog Flash) an interrupt driven, real-time clock.

As we go to press, John S. has been taken into custody by the local constabulary and is being held for observation: "We want to find out what makes this guy tick." We are currently raising money to pay for John's bail and a ni-cad stand-by power supply. But John informs us that, barring any unforseen events, he will definitely be at this upcoming meeting of your Apple Core. And John is one man you can count on.

And as for Mr. Glitch, Hip Flash has just located his hiding place in the area surrounding Scandal, and the Apple Core Journal takes great, nay fiendish, pleasure in revealing it to everyone. Yes, J. Alfred, we know you are located at SOcwo3himem:30KSLS/::: FORMULAE TOO COMPLEX
ERR39LCLCATALOGS039GKD:::SYNTAX ERREIFLSRESET.....

PROGRAM LAW

COPYRIGHT (C) 1978

BY ARTHUR WELLS

This article will discuss some of the laws pertaining to the protection of computer programs and data bases. As the law in this area is complex and confused. and as I am not an expert in the field, and as this is just a general article, I am sure you will be careful about relying on it before you do something you think is important.

There are theoretically several ways you can try to legally protect computer programs from unauthorized use. Basically, these are to try to patent them, copyright them, or treat them as trade secrets or "know how". Patent protection, even if available for computer programs, is an expensive process from the outset. As contrasted with other possible protections, the granting of a patent requires that a search be conducted to demonstrate that the item for which the patent protection is being sought is novel, that is, no one else has "invented" it. Patent protection for programs, if available at all, might be had where firmware is integrally involved with hardware.

The most common attempt at protection is via a copyright claim. In order to claim copyright protection, one need only put a notice of copyright on the work, which consists of the word "copyright" or the letter C in a circle with the name of the claimant and the year of publication. This claim must be followed by registration, which involves filling out a simple form (available from the Copyright office in Washington, D.C.), payment of a small sum, and submission of a "copy" of the work for which copyright is claimed.

PROGRAM LAW (CONT.)

Having the notice and the registration do not mean that you actually have the copyright protection you claim. You get that protection only after a lawsuit in which the court says you have protection. Such lawsuits are expensive and time-consuming and it is extremely doubtful that anyone but a large company would bring such a suit over a program and then only if the program were very valuable. Many specific problems and questions are presently unresolved as far as copyright protection for computer programs and data bases. In fact, a new copyright law effective in 1978 specifically left questions about how to treat computer material for future discussion and several professional committees seem to be still grappling with the problems.

Some of the questions and problems are:

1. To claim a copyright, the work involved must be "original". When is the assembling, selecting, arranging, editing, and literary expression (if any) that goes into the set of operating instructions (program) or compilation of reference material (data base) sufficient to constitute original authorship?
2. What protection attaches to works created by a computer, based on a program?
3. In which form ought the program be protected; input, output, or both?
4. When does unauthorized use of only a part of a program, such as one elegant subroutine, constitute an infringement?
5a. How do we distinguish between the situation where the program "communicates information" and is thus copyrightable, and where it is an integral part of the mechanical elements of some device, and therefore protection of its "utility" can be obtained by patent only.
5b. Does this mean we draw distinctions between systems firmware or software (compilers, assemblers, etc.) and other programs? Is a program ever non-"functional"? Or, is it always "communicating" information, even if it is only to a machine?
6. Copyright protection gives the exclusive right to reproduce copies, and to prepare derivative works. Where is the line to be drawn regarding the use of programming algorithms?
7. Copyright protects original works of authorship fixed in a tangible medium that does not extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery regardless of the form in which it is described, explained, illustrated, or embodied in the work. How is this rule to be applied to computer programs and data bases?
8. Copyright is supposed to encourage people to produce original works by giving them monopoly-like rights in their works for a specified period of time. But use of copyrighted material without permission is sometimes allowed to further societal goals such as teaching, criticism, scholarship and research. Whether such "fair use" is permitted depends on the purpose and character of the use, the nature of the material, the amount of material used, and the effect of the use on the market for or value of the copyrighted material. How is the fair use doctrine to be applied to computer material?
There are a host of other, and/or subsidiary questions, but the above will do for a start.

Given the problems with patents and copyrights, it is not surprising to find that many companies dealing with intellectual property do not rely completely on those legal tools for protection. Surveys of firms providing consulting, feasibility studies, systems analysis and designs, contract programming, proprietary software packages, time-sharing, telecommunicatios, data center management, and computer research indicates use of various protective devices. Preferred methods seem to be contracts requiring that material be treated as confidential both by employees and users. In addition to treating the material as a trade secret, and to patent and copyright, companies also use limited access to technology, cryptographic coding, software locks and similiar devices. Protection of software seems to be viewed as most important for general business and financial applications and for systems software, and least important for complex production and management and for engineering and scientific applications. Probably this is so because marketing considerations and design and development costs play a great role in providing meaningful protection for the latter.

As applied to the hobby user, it is very doubtful that anyone would or could ever successfully prosecute an infringement or misuse suit against someone who pirated a program. Conversely, hobby users who develop their own material should either keep it secret to themselves or give it to all in the hopes of making a good reputation. The hobby user will hardly be in a postion to enforce his right, if he has any.

The fundamental concepts for this field have been well stated by others:
1. Creators and developers should receive compensation for the use of their product.
2. Society should have maximum access to the creative products of its members.

The resolution of these two somewhat conflicting principles, especially as applied to a new complex technology, will need years to be worked out; that is why, in summary, the field of computer program law will contrive to be obtuse and obscure.

In light of the above, you serious programmers and other computer professional have my sympathy and my condolences. Also my apologies for such faults of this article as may contribute to those attributes.

ANY QUESTIONS ABOUT THE ARTICLE CAN BE SENT TO ARTHUR, CARE OF US HERE AT:

SF APPLE CORE
1515 Sloat Blvd, Suite 2
San Francisco, CA 94132

TEACHING PGM

BY ARTHUR WELLS

Young children apparently have little trouble learning to use a computer. My own child, now age 8, as well as a number of other children I've run across and heard about, have no trouble loading programs and running the machine. This includes both tape and disc load, restarting tapes after glitches, rebooting, and other minor troubleshooting. Strong motivation for this obviously exists in that the reward is immediate and satisfying: the child gets to play interactive games on or with the computer.

Teaching your child to program also achieves a variety of goals and satisfactions, not the least of which are:

1. Your child begins to learn a skill;
2. You justify having your computer and spending money on it;
3. It is a productive way to interact with your kid;
4. The time the child spends with you frees up your spouse's time;
5. Your spouse will get off your back about spending so much time with the computer; and
6. Your child learns that computers do what people tell them and not vice versa.

To learn programming your child needs to know elementary counting and how to read. By the end of the second grade, maybe earlier if you're lucky or work hard with your kid on basic skills, your child can learn simple programming.

The easiest thing to teach the child is that which produces immediate or dramatic results. If the child is 4-6 and is just learning to spell, use the Giant Typewriter program. The child gets immediate satisfaction from using the keyboard.

The child can also be shown the PRINT command for simple adding and subtracting. This is fine if you work with the child, otherwise he will not know if the answers are right or wrong.

The first program kids seem to learn easily is the infinite loop with a PRINT statement, such as:

```
10 PRINT "HI DADDY!"
20 GOTO 10
30 END
```

This program is useful to teach the child the concept of a simple statement, a simple program, the RUN command and that rules must be followed to get results. Many different results can be had using the semi-colon command and dummy PRINT to vary the output. Putting blanks within the quotes, and using asterisks, pluses, minuses, parentheses, etc., lets the child draw designs. The design will go by on the screen too fast for the child to see the design, so you can teach him CONTROL C to stop the program (or CTRL S with Auto-boot ROM).

By the end of the second grade most children have been exposed to, and can grasp the concept of, using coordinates to define a location. The APPLE low resolution graphics is thus another way to get your child interested in programming the computer. While the manual doesn't say so, all the low resolution graphic statements will operate as commands, that is, immediately and without line numbers. This means the child does not have to write a program and then RUN it to see what he did, and then try to figure out what in the program produced which result on the screen.

Without a line number type GR. The 40 x 40 graphics will be called up. Then set COLOR. Then PLOT, HLIN, and VLIN can be used to draw on the screen. An outline of marks to help the child count the coordinate locations can be made with the following simple program:

```
 10 GR:  COLOR = 10
 20 FOR A=0 TO 39 STEP 2
 30 HLIN 0,1 AT A; VLIN 0,1 AT A;
    HLIN 38, 39 AT A; VLIN 38,39 AT A
 40 NEXT A
 50 END
```

Obviously, the child will need your help. But he will eventually learn the command format. Show him the difference between PLOT 3,20 and PLOT 20,3 etc. To help him write programs, have hime draw a picture on a 40 x 40 grid. Then help him list the commands he needs to draw the picture.

LET FINGERS DO THE SINGING

BY TONY HUGHES

One of my happy memories of the TV show Star Trek pictures a scene in the transporter room with Scotty or some technician at the controls. As his (or her) fingers go over the controls, we can hear each beep and buzz. This detail of human engineering is also present in our modern day cash registers, calculators, etc. Audio feedback as it is called, is an asset because it coordinates two senses, makes us that much more sure.

To try it out on your APPLE enter the code below while in MONITOR. Then enter 300G (return) and start zinging away. The effect will still be present if you go into BASIC. As you will discover, each key has a distinguishing tone, so you can tell them apart.

```
300:AD 47 03 85 38 AD 48 03
308:85 39 60 E6 4E D0 02 E6
310:4F 2C 00 C0 10 F5 91 28
318:8A 48 98 48 08 AD 00 C0
320:0A 0A 85 06 A9 18 85 07
328:AD 30 C0 88 D0 04 C6 07
330:F0 09 CA D0 F6 A6 06 F0
338:EF D0 ED 28 68 A8 68 AA
340:AD 00 C0 2C 10 C0 60 0B
348:03
```

The routine works by replacing the address of the MONITOR get-a-character routine with its own address (the address goes into hex location $38-$39). Floppy disk users should beware, though, since the DOS will put its own address into this location. It will also overlay that part of memory where our keytone routine resides. If you want to relocate it, you can put it anywhere you want as long as the last two bytes point to your load address + 11 bytes. It will even work with DOS if you load it before you load DOS.

HOW TO SELECT THE PRINTER YOU NEED.

CHOOSING A PRINTER

Picking the printer best suited to your application is easier once you isolate the basics. This report will tell you what type of printers are available, how they differ from one another, and how to evaluate their value in terms of cost to you.

Since they represent the most popular and fastest growing segment of the printer market, our discussion will be limited to low- to medium-speed printers. They encompass those applications using minicomputers and microcomputers, process control, small business systems, data acquisition, data entry and data communications. Printing speeds can range from 15 characters per second (8 lines per minute) up to 600 lines per minute.

WHAT SPEED DO YOU NEED?

The proper printer speed can be influenced by factors such as the average amount of data to be printed, available system time or the minimization of communications line charges.

For continuous output, a rule of thumb technique for determining speed needs is multiplying the number of forms required each day by the average number of lines per form. The total is the number of lines to be printed per day. Hence, the number of printed lines required per day divided by printer speed equals the number of minutes per day of actual printing.

The above is greatly simplified because it doesn't account for line length, carriage return time factors, slew speed in skipping over blank space, etc. Finer calculations need to be equated for each given situation.

PRICE vs. PERFORMANCE

While it is typically true that the lower the speed the less expensive the machine, this simplistic approach can lead to downstream headaches. To save money, a user with a heavy workload often buys a printer designed for light duty. While the printer may produce the desired output, it does so at the expense of being overworked. The cost of replacing prematurely worn out parts soon offsets any initial savings. Plus, the lost time of downtime can be even more costly.

So, in addition to rated speed, the astute printer purchaser looks at duty cycle limitations, MTBF specifications, maintenance requirements, and, most importantly, the mechanical structure of the machine. Always remember, with mechanical devices, simplicity is a forerunner of reliability. *Inherent design simplicity and a minimum number of stress and wear points means a superior mechanical system.*

There is more to printer cost than initial purchase price. The total cost of all expenses over the life of the printer must be factored. Often, the printer with the lowest purchase price ends up being the most expensive buy.

In the price performance category, the purchaser should also look to see what basic features are standard and what features are "options," and hence, add-on costs.

THE PITFALLS OF SPEED RATINGS

While manufacturers quote serial printer speeds in characters per second and line printer speeds in lines per minute, the figures can often be misleading. Some specs don't account for the length of a printed line or the number of control characters in a stream of text. Hence, speed ratings are given for full lines, or only partial lines, or for 64-character sets or 48-character sets. To properly assess the true speed of a printer, ask the manufacturer or vendor for a throughput curve as shown here.

IMPACT vs. NONIMPACT

Simply stated, an impact printer transfers ink to the paper forcefully by a hammer assembly such as the common office typewriter. Non-impact printers use sensitized papers that respond to thermal or electrostatic stimuli to form an image. Hence, the basic advantage of impact printers is multiple copies using low-cost, standard paper and inked ribbons. The advantage of non-impact printers is quiet operation, and, often, speed. However, they produce only single-page output, and the special paper can be expensive (2 to 3 times the cost of standard paper) and not readily available. Also, use of pre-printed forms is impractical.

Since most data processing applications require multiple copies, and cheap copies, the remainder of the discussion will deal only with impact printers.

LINE vs. SERIAL

The choice between selecting a line printer or a serial printer generally pivots around speed needs and purchase price. Serial printers, which produce one character at a time, are typically less expensive than line printers, which produce an entire line of text at a time. Some of the better-constructed serial printers can comfortably print up to 160 characters per second (about 100 lines per minute) while line printer speeds can attain whatever range one is willing to pay for.

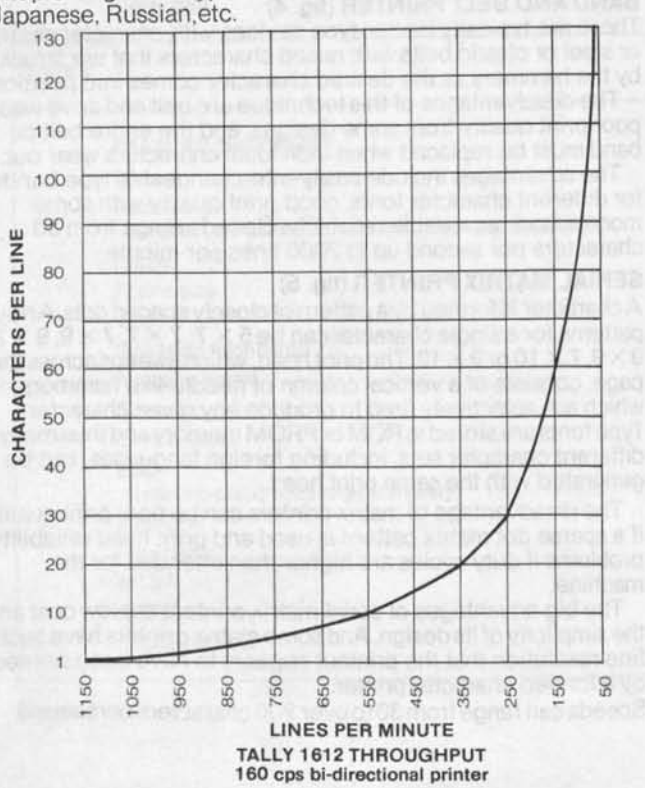
However, with the recent advent of the microprocessor, serial printers are achieving increased throughput efficiencies. An example is "optimized bi-directional" printing, whereby the print head moves left to right or right to left, thus eliminating the wasted time of carriage returns. This technique also "searches" for the shortest path to pick up the closest character on the next print line. A printer equipped with optimized bi-directional printing can typically double or triple throughput. It's an important bonus to look for.

MATRIX vs. FORMED CHARACTER

Another major distinction between printer types is shaped characters vs. matrix image formation. The shaped character (such as found on the office typewriter) is contained on print mechanism devices such as drums, chains, belts, bands, daisy wheels or type balls. However, there is the drawback of higher purchase price and mechanical complexity. The latter has a bearing on maintenance costs and the probability of downtime.

Matrix printers form a character from an array of closely spaced dots. The foremost advantage of matrix printers is mechanical simplicity which translates into a low-cost device capable of highly reliable operation. Good matrix printers produce characters that achieve a formed character look. Also print quality is more consistent from matrix printers because alignment seldom wavers and character density is uniform. Formed characters have a tendency to get out of alignment and character density can vary. (The user should pay particular heed to the quality of printed output when comparing matrix printers.)

Another advantage of matrix printers is the ability to obtain different type styles—condensed, expanded, double width, upper/lower case, negative print, OCR, bar code—from a single mechanical system. Because the matrix array is contained in a Read-Only-Memory (ROM), a simple electronics swap changes the type face or even the language, e.g., Japanese, Russian, etc.



TALLY 1612 THROUGHPUT
160 cps bi-directional printer

PRINT MECHANISM TYPES

The heart of a printer is the print mechanism which imparts the character image to the paper. A thoughtful examination of impact technique is important from the standpoint of long-term reliability and consistent print quality.

DRUM PRINTER (fig. 2)

A complete set of formed characters is embossed around the circumference of a cylindrical drum. The drum rotates at a constant speed and a hammer, located at each print position, strikes the desired character at each print position each time the drum rotates on its axis.

As is readily apparent, timing is critical to maintain print quality. As such, there is the need for periodic hammer flight time adjustments. Other disadvantages include a limited choice of character fonts and the constant possibility of vertical misregistration of characters.

On the plus side, operation is generally reliable, print speeds from 300 to 2000 lines per minute can be achieved, and as such, drum printers offer a good cost/performance ratio for higher duty cycle applications.

CHAIN OR TRAIN PRINTER (fig. 3)

Character slugs move horizontally past hammers located at each print position. The characters pass each hammer in sequence. In chain printers, the slugs are not connected and they push themselves around the track. With both methods, several complete character sets revolve past the hammer positions at a constant speed. The hammer is fired at the precise instant that the character to be printed moves into position.

Again, hammer and electronic adjustments are critical and require periodic maintenance. Reliability is a problem because wear in the tracks is common. However, for higher speeds, up to 2000 lines per minute, these printers do the job.

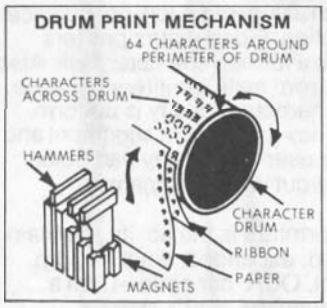


Figure 2

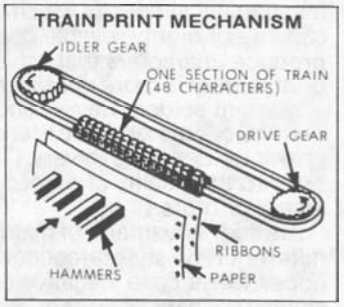


Figure 3

BAND AND BELT PRINTER (fig. 4)

These are typically tractor-type devices with character slugs, or steel or plastic belts with raised characters that are struck by the hammers as the desired character comes into position.

The disadvantages of this technique are belt and drive wear, poor print quality from some designs, and the entire belt or band must be replaced when individual characters wear out.

The advantages include easily-interchangeable type bands for different character fonts, good print quality with some models, and reasonable reliability. Speeds range from 30 characters per second up to 2000 lines per minute.

SERIAL MATRIX PRINTER (fig. 5)

A character is formed by a pattern of closely spaced dots. Array patterns for a single character can be 5×7, 7×7, 7×9, 9×7, 9×9, 7×10 or 9×12. The print head, which sweeps across the page, consists of a vertical column of needle-like hammers which are selectively fired to produce any given character. Type fonts are stored in ROM or PROM memory and thus many different character sets, including foreign languages, can be generated with the same print head.

The disadvantage of matrix printers can be poor print quality if a sparse dot matrix pattern is used and print head reliability problems if duty cycles are higher than intended for the machine.

The big advantages of serial matrix printers are low cost and the simplicity of its design. And some matrix printers have such fine resolution that the printout appears to have been printed by a formed character printer.

Speeds can range from 30 to over 200 characters per second.

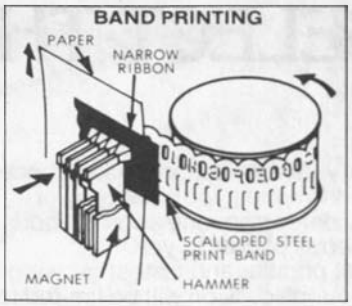


Figure 4

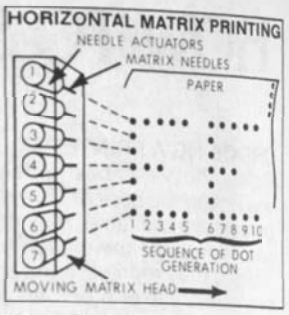


Figure 5

COMB MATRIX PRINTER (fig. 6)

A unique variation of matrix printing, the print comb technique, prints a line of data at a time rather than a character at a time as is common with most matrix printers. A single piece print comb has 132 fingers, each with a steel ball impact face. Each finger corresponds to a character position, thus there is a hammer for each print position.

Each finger is pulled back by its own electromagnet and then released forward to create a dot. A slight horizontal movement of the comb locates the adjacent dot position and the finger is released again. This is repeated until one dot row is completed. The paper advances vertically to the next dot row and the process is repeated until the complete line has been printed.

This technique can achieve reliable printing speeds up to 300 lines per minute. Print quality is very good, approximating formed character output. Because the print mechanism never works very hard, and mechanical movement is slight, machine reliability is unmatched. Preventive maintenance is unnecessary. There are no lubrication or adjustment requirements nor duty cycle limitations. The comb matrix hammer has proven to be the most reliable mechanism available.

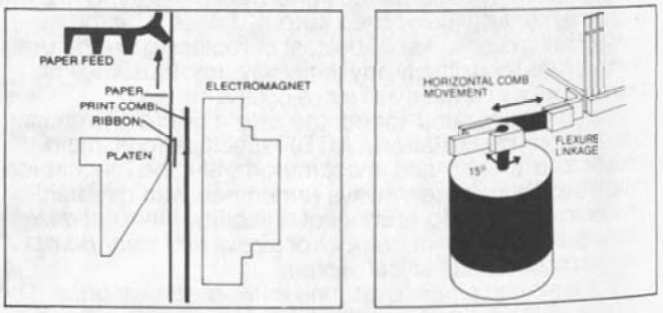


Figure 6

HELIX MATRIX PRINTER (fig. 7)

Another matrix technique employed in a line printer, the helix printing method, has a helical print mechanism with a knife edge spiral on its periphery. As the helix rotates by each character print position, hammers fire at appropriate positions of the dots, and the required character is completed during one rotation of the helix.

The advantage of this technique is high speed printing, up to 500 lines per minute, but at prices appreciably less than drum or chain printers. Print quality is good and machine reliability is good because of a minimum of moving parts.

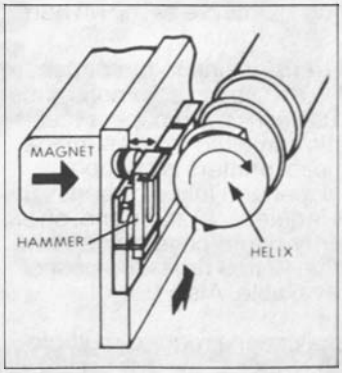


Figure 7

IS RELIABILITY IMPORTANT?

Even after deciding on the type of printing method best for the application, the user will be faced with competing manufacturers offering, on the surface, like machines. But careful analysis can soon determine the best value. Rather than being swayed by purchase price, the important consideration is total cost of ownership.

THE REAL COST OF OWNERSHIP

User cost goes beyond the initial purchase order. *Total* cost must include cost of supplies, maintenance, service calls and spare parts over the expected life of the machine. *And the immeasurable cost of downtime!* When these factors are added up, often the machine with the higher initial price proves to be the lowest cost in the long run.

Three basic factors can be evaluated toward determining the real cost of ownership—inherent machine reliability, periodic maintenance requirements, and the consumption of expendables such as paper and ribbon.

BASIC RELIABILITY

Complexity means cost. Even with today's advanced electronics, some printers still have elaborate mechanical works. Remember, any part eliminated through good design is one less part to fail. Therefore, look for mechanical simplicity that requires a minimum of, or no routine adjustments, a minimum of, or no lubrication procedures, and no special tools for normal service.

Mechanical assemblies should be modular in design for easy and quick replacement. Look out for extensive use of clutches and brakes, as these are high failure-rate components.

While evaluating the reliability of printer electronics can be difficult, there are some things to look for that indicate sound design. Microprogramming eliminates many potential circuit failures by greatly reducing the number of electronic components. Modularity indicates good design. Few boards that are easily replaceable normally indicate quick solutions to problems. Electronic adjustments by the operator should never be required.

In summary, the critical reliability components of any printer are the hammer assembly, the paper feed system, the ribbon system, and the printed circuit boards or electronics. Each should be examined with simplicity and soundness of design in mind.

PERIODIC MAINTENANCE

The requirement for and extent of periodic maintenance is tied directly to the electro-mechanical design of the printer. The simpler the design, the higher the expectation of reliability. In an inherently simple electro-mechanical system, a minimum amount of maintenance will be required. Or no maintenance whatsoever!

At today's prices, a field service call can typically cost about $100 plus parts. With these prices escalating yearly, maintenance can become the greatest portion of ownership cost if reliability is poor.

EXPENDABLE CONSUMPTION

The major expendable supplies a printer goes through are paper and ribbons. Look for a printer that can easily handle standard forms of varying widths, and for multiple copy requirements, make sure the last carbon is clearly legible. If preprinted forms are utilized, make sure the printer has the needed forms controls so printing conforms to the allocated space. Additionally, printers are available that offer special forms handling capability to cut the printed form to the desired length or size and minimize paper wastage.

When you realize that, with typical usage, more money is spent on paper than was originally expended on the printer, it's easy to see that paper cost is a big item.

Standard line spacing for most printers is 6 lines to the inch. Some printers, for paper economy applications, have an 8 line per inch spacing option. For high volume runs, savings can be significant. In addition, some of the better matrix printers can alternate their character pitch to achieve 10, 12, 14 or 16.5 character per inch spacing. As an example of paper savings, 132 columns of data can be printed on 80 column width paper using 16.5 cpi spacing.

Ribbon replacement can become expensive if an impractical ribbon system is employed. For economy, a reel-to-reel system with a reusable fabric ribbon offers longer life. Many printers offer cassette-type cartridges that feature easy-loading characteristics. With both types, look for a good design that minimizes moving parts and maximizes ribbon life. Because ribbon replacement costs can become a big expense factor over the life of a printer, it should be considered in with the total cost of ownership.

IMPLEMENTATION COSTS

An often-overlooked factor in printer price evaluation is the cost of implementation; namely, the manufacturer's ability to service the product, provide the needed documentation to integrate the printer into the system, and in general, solve any problems that arise. The chart below is a good checklist to refer to in making an evaluation of a printer vendor. (Fig. 8)

THE VALUE OF VERSATILITY

Whether approached from the standpoint of cost or the standpoint of operator convenience, the standard features offered by a printer are important parameters. Also, be sure to determine if any given feature is included in the standard price of the machine or if it is an additional cost option. A check-list (Fig. 9) comparison of printers can assist the evaluation.

The true measure of machine features is the degree to which they enhance the ease and efficiency of operation. Here are some considerations.

TOTAL COST OF OWNERSHIP

- **Purchase cost**
- **Operating cost**
  - paper
  - ribbon
- **Cost of integration**
  - controller/interface
    - purchase or manufacture
    - checkout
    - programming
- **Support cost**
  - spares
  - manuals
  - training
  - warranty
  - refurbishment
  - preventive maintenance
  - corrective maintenance
- **Other cost**
  - impact on existing inventory
    - spares
    - training
- **Total cost**

Figure 8

FEATURE CHECKLIST

- Speed
- Technology
  - (impact, non-impact, matrix/formed characters)
- Copies
- Forms size
- VFU
- Self-test/diagnostics
- Acoustic noise
- Size and weight
- Environment
- ESD
  - (electro-static discharge immunity)
- Interface
- Flexibility
- MTBF
- MTTR
- Character sets

Figure 9

INTERFACE CAPABILITY

The interface aspect of a printer purchase can be an expense and technical headache if the machine is not equipped for compatibility within the system it will live. While it seems most printers are offered with an RS-232 compatible port, if the system requirement calls for something else, the user should determine up front if the machine can be adapted to his interface needs and at what expense. Many manufacturers offer standard controller packages to allow plug-in compatibility with the host system.

NUMBER OF CHARACTER SETS

Numerous applications can make use of different character sets. With matrix printers, since character generation codes are contained on printed circuit cards, more than a single character set is at the user's disposal. The user can even configure special custom characters. And of course, foreign languages are easily accommodated.

Another advantage of the matrix printer is that the sizes of characters and interline spacing can be varied. Condensed printing at 16.5 characters per inch, as opposed to the standard 10 characters per inch, can save paper. Or, double-width printing, where a character is printed twice as wide as normal, allows certain portions of text to emphatically stand out.

FORMS CONTROL FEATURES

Since most print-out follows a set format, the types of forms control available must be considered. While most printers have the same standard offerings, lower-priced units may take shortcuts by eliminating some necessary features.

Any printer should have provisions for initially aligning the top of form and left-hand margin for pre-printed forms. From this point, there are a number of forms length control offerings. Typically, vertical format (called the VFU) can be controlled via a 66/88 line count program, a forms length selector switch, and 8 or 12 channel punched paper tape loop program, or an electronic VFU loaded through the I/O. The best method depends upon user requirements—but the more sophisticated the VFU, the higher the price.

For consistent print quality, especially if alternating between different thickness layers of multi-part forms, look for a forms thickness control. Also, a paper-out alarm should be standard.

For users that require very specialized forms handling capability, printers are available that can be customized to perform very specific tasks such as printing airline ticket passbooks, printing entries in bank passbooks and simultaneously recording the transaction on a permanent journal, printing and cutting tiny tickets and labels, or even printing two different forms simultaneously. Additionally, automatic front feed attachments to standard printers enable efficient control and printing of individual cut forms such as ledger cards or invoices.

OPERATOR CONVENIENCES

From the operator's standpoint, the acceptance of a printer generally depends on whether the manufacturer originally designed the machine with the operator in mind.

The primary operator function is paper loading. Loading should be easy and fast without awkward fumbling. The more accessible the paper path, the better. Tractor face plates (for pinfeed paper) should be exposed and the tractor teeth clearly visible. Tractors should be easily adjustable to accommodate various width forms.

Ribbon changing should likewise be easy and fast—and clean.

Control indicators should be conveniently, and safely, located for operator accessibility. They should be clearly labelled and logically organized.

Some of the more sophisticated printers offer status check panels that register the fault condition if the printer stops operating. This allows the operator to initiate corrective action in a timely manner. Often these panels differentiate between operator correctable faults and those that require a service technician.

An important consideration to the operator, or anyone in close proximity to a printer, is noise level. While most manufacturers publish a decibel level for their machines, the user should find out under what conditions the tests were made, i.e., where was the sound measured, what kind of printout pattern was being used. Any slight variation in any of these conditions can have a significant impact on the test results.

Another convenience consideration is whether the printer is available as a desk-top unit or requires a stand. Some printers are interchangeable, i.e., the unit can be used in either situation. Also, determine if paper-stackers, both supply and take-up, are available or can be easily accommodated.

PRINT QUALITY

One of the most important factors in choosing a printer is print quality. While no standards exist as to what constitutes good print quality, the eye of the beholder can easily pass judgement. The expectation should be clear, crisp and concise characters. In formed-character printers, the pitfalls to look for are horizontal or vertical misregistration, tipped characters, ghosting or smearing, clipping off of ascenders or descenders, voids or variations in character density.

The critical examination of matrix printing should be concentrated on the dot spacing. Good matrix print quality has the individual dots appearing to overlap to give the impression of a formed character. An inherent benefit of matrix printing is that character density is always uniform because the force required to print a dot is always uniform. Conversely, in a formed character printer, a design compromise exists in order to legibly imprint a large land area character such as a "W," yet not perforate the paper with a character such as a period. Also, a natural benefit of the uniform printing characteristic of matrix technology is better carbon copies from a legibility and consistency standpoint.

In matrix printers, there should be no excuse for any poor character registration. A sound mechanical design literally locks the needles or hammers into an alignment position so the result is straight line registration, vertical and horizontal, that never wavers. If alignment problems occur, it indicates a faulty paper movement mechanism. Before selecting a printer, make sure you carefully review print samples.

TALLY®
PRINTERS
WORLDWIDE

This report is furnished compliments of Tally Corporation, 8301 South 180th St., Kent, WA. 98031. (206) 251-5500. Additional copies can be obtained by writing to the above address.

Tally Corporation is a member of the MANNESMANN Group.

So You Bought Your Computer From Somebody Else . . .

Business and math and education and such. Do we have lots of programs? We'd say very much . . .

. . . these are the programs that go on the disks that slip into the drives that connect to your Apple.

. . . this is a printer to output your programs and list your HEXs. Centronics and Trendcom (and others from Texas).

To err is human. Besides, that was the past, this is the present, and we want to connect with your future. Sooner or later, you're going to want food for your Apple. When that time comes, have we got the goodies for you! We have add-ons like you wouldn't believe: software that just doesn't quit; books, magazines, Apple-users' publications and tutorials. We also have coffee, a comfortable environment and the time-of-day. All of this has been personably selected and tested by our staff of Apple specialists. Want details? Check out the flow chart on this page. Then stop in and we'll forgive you for buying your Apple at That Other Place.

. . . these are the diskettes that go in your drive. We sell these and the storage binders to keep them in . . .

This is the Apple you may have bought elsewhere. We'll get you up on a floating point ROM card, or get you started in Pascal . . .

. . . this is the disc drive that connects to your Apple. We've got these, too, and . . .

ComputerLand®

of the Castro

2272 Market Street
San Francisco, CA 94114
(415) 864-8080

APPLE DOS

WHAT'S DOS

Congratulations on your new disk drive. Playing with programs and data will be quicker and easier, but there are some things you should know before you start. For example, D.O.S.

The DOS (Disk Operating System) is what it sounds like. There are many steps in getting data from and to the diskette's magnetic recording surfaces, and the DOS has them in its innards somewhere in machine code. But it performs another function, that of control.

A tape-oriented system has a limited set of commands--not much you can do with a cassette recorder; a disk drive has many, many more. The commands to control the disk will be read by the DOS, and control passed to the appropriate set of machine codes. When we load a named program, the DOS has to first go to a part of the disk containing program names and corresponding locations to find our program. Then it goes to each sector on which the program is located, picks up each byte on each sector, and puts it into the Apple's memory.

Let's say something about floppy diskettes at this point. They are surfaced on both sides with magnetic material, just like magnetic tape. The disk drive normally uses just one side, since the manufacturer puts his best side on top. The diskette will rotate inside the drive while the recording head is held stationary. It looks like a magnetic phonograph record. Instead of one continuous groove, the recording head will be moved in and out to record data in 35 concentric circles, called tracks. The data does not completely fill a track but is placed in 13 sections, called sectors, with 256 data bytes per diskette. So, 35 tracks * 13 sectors * 256 bytes per sector = 116480 bytes per diskette.

Introducing commands to control the disk drive posed a problem; what is going to read these commands and execute them? Other computers feature special versions of BASIC with more commands and higher memory requirements. The Apple, however, executes disk commands differently. All data that is to be printed will first pass through the DOS for its examination. It will intercept its own commands and execute them, but not pass them on to the Apple. The DOS commands can then work with any other program product, such as Applesoft, Integer BASIC, the moniter, anything that prints.

The DOS is normally kept on disk and loaded whenever you enter PR# in BASIC or Control P in the monitor. What happens is that a program on the ROM Card is executed that takes the DOS off of disk and loads it into memory. Say your disk controller is in Slot 7. Then if you get into Monitor by a 'CALL -151' and then execute the ROM program at C700 by 'C700G', you will boot the DOS. It will be placed into the highest location in your Apple's memory. Himem will be reset.

3.1 OR3.2?

BY ALLEN JOHNSON

Hi there, you say your feeling blue because you just updated your disks to 3.2 and now they don't run right. Not only that but now you can't remember which is 3.1 and 3.2? My friend Bob Burns and I sat down and came up with a nifty little program that tells you which is which.

```
>LIST
10 TEXT : CALL -936
20 X=PEEK (1002)
30 VTAB 5: TAB 12: IF X=32 THEN
   PRINT "DOS VER 3.1"
40 VTAB 5: TAB 12: IF X=76 THEN
   PRINT "DOS VER 3.2": VTAB 20: END
```

VOL MISMATCH MATCHED

BY ANDY HERTZFELD

NOTE: THIS ARTICLE APPLIED TO DOS 3.1 AND PROBLEM DOES NOT OCCUR WITH DOS 3.2 BUT HAS SOME INTERESTING DOS INFORMATION

If you are running your APPLE with a disk, by now you have probably encountered the problem of"VOLUME MISMATCH ERROR" messages. Here's a solution to this problem produced by Andy Hertzfeld's exploration into the DOS.

Every diskette has a volume number from 1 to 254 associated with it. It is assigned when the diskette is initialized and there is currently no easy way to change it. The volume number of the current disk is stored at $B7F6. Before most DOS commands are executed the system checks to see if a "volume mismatch" error is generated. While this "feature" may be nice for large business applications that don't want dumb operators inserting the wrong disks, it is very annoying to most average users. It is most difficult when you want to transfer a number of programs between two disks with different volume numbers.

After much searching I located the place where the volume check is performed and devised a patch to disable it. It's only two bytes long; just enter the monitor and type: "BDFE: A9 00". This will disable all volume checking until the next bootstrap. It works by replacing the comparison instruction which performs the volume check with a "LDA #00" instruction which sets the "equality" or Z flag, effectively forcing the match to succeed.

Note that the addresses given here are only true for a 48k system. If you have a 32k system, read "77F6" for B7F6" and "7DFE" for "BDFE". In general, if your system has memory size X, subtract ($C000 - X) from the given addresses.

FP DISK TRACE

BY SCOT KAMINS

Handy hint for the month comes from our esteemed VP, Bruce Tognazzini, and has to do with the elusive "Tracing an Applesoft Program with Disk Booted" problem.

The solution: D$=CHR$(13) + CHR$(4)

which is the same thing as having D$ equal a carriage return (CTRL M) concatenated with good friend CTRL D of disk command fame.

I haven't the faintest idea how it works, and B.T. ain't talkin'; but it does, indeed, work. One is cautioned, however, to return D$ to equal only CTRL D after debugging is completed, however, lest odd and unpredictable events ensue.

DOS NOTE

BY SCOT KAMINS

The DOS will accept a colon (:) from a Textfile written by an Integer BASIC program without acting upon it. But it treats it as a statement delimiter in Applesoft, ignoring everything else that comes after it in a record.

Thus you can say "THIS IS IT: THE BIG ONE" in a Textfile read from an Integer program. Read from Applesoft, the program says "THIS IS IT" with an error message.
Silly, isn't it?

1. Larry Fish from Los Altos Hills, CA, points out that, rather than typing "VO" for the wild card to avoid disk mismatches, a simple "V" will do. Every stroke counts...

2. Assuming that your "HELLO" program contains a "CATALOG" command, you can add some text in a string statement to appear after the catalog listing. An example follows.

3. Incorporating Arthur Wells' "N" title, the following routine will do nicely for a HELLO program:

```
 0 D$ = " ": REM CTRL D
10 POKE -16298,0: REM  CLEAR HIRES.
20 TEXT : REM  CLEAR GRAPHICS, RESET
   SCROLLING WINDOWS.
30 CALL -936: REM  CLEAR SCREEN.
40 PRINT D$;"CATALOG"
50 PRINT "TO RUN A PROGRAM, COPY THE
   TITLE"
60 PRINT "EXACTLY AS IT APPEARS
   ABOVE": REM  ILLUSTRATES #2,ABOVE.
70 END
```

NAME GAME

BY ANDY HERTZFELD

Some people might wish to alter the names of some of the DOS commands to suit their own personal tastes (it is, after all, a personal computer). For example, I know many folks would like to abbreviate the "CATALOG" command to a simple "C". This is surprisingly easy to do.

Since the DOS lives in RAM the contents of its command table are easily changed. The command table is located from $A7E0 - $A863. Each command name is represented as an ASCII string with the high bits off, except for the last character of the string, which has its high order bit set. The strings are associated with the commands by their position in the command table (the first string corresponds to the INIT command, the second to the LOAD command, etc).

Thus you can dream up your own names for the commands by storing new strings in the command table. For example; to change the name of the INIT command to DNEW you would enter the monitor and type "A7E0: 44 4E 45 D7". However, some caution is required when you change the length of a command name; in general you will probably have to rewrite the entire command table to achieve the desired effect.

It is hard to use the input and output "hooks" in conjunction with the DOS since you cannot simply change the hooks as they are the only way the DOS interacts with the rest of the system. Also, if you only change one of them, the DOS has the nasty habit of changing it back. Fortunately, the DOS has its own internal hooks it uses for keyboard input and video output. Its output hook is at $A996 - $A997 and the input hook immediately follows at $A998 - $A999. If you change the contents of these addresses instead of the usual hooks at $36 - $39, everything should work out just fine. For example, lets say you wanted to divert output to a line printer without disabling the DOS. If the line printer output routine is located at $300, all you would have to do is enter the moniter and type "A996: 00 03"(For 3.1 DOS only).

- - - - -

By the way, in case you missed it, typing "9DB9G" (9DBFG for 3.2 DOS) from the monitor will reinitialize the DOS. This routine should be called after every reset to restore the hooks. It is exactly like typing "3D0G" as APPLE's documentation recommends but is a little bit safer since the 3D0 location is often destroyed by various programs.

This article is merely the tip of the proverbial iceberg; most of the DOS's internals still remain a mystery to me. I hope APPLE eventually distributes complete documentation but until then other curious users can use this article as a starting point for their own explorations. Hopefully they will report back what they find.

RWTS

BY TONY HUGHES

New information has come out about accessing data on the disk directly. The DISK II has its data recorded on 35 tracks (think of a 4 track cassette or an 8 track cartridge) arranged as concentric circles around the center hole. Each of these tracks is divided into 13 sectors, of 256 bytes each. Using the following method, you can read from or write to an individual sector on any track.

First, you must build a table or two in memory to specify the parameters of your call to the DOS. It will contain:

	POSITION	CONTENTS
01	Must be 01.	
02	Slot number, times 16.	
03	Drive number.	
04	Vol. number to be found. Make it 00.	
05	Track number that the sector is on. Must be 00-34	
06	Sector number. Must be 00-12.	
07-08	Low and high order addresses of device characteristics table (see below).	
09-10	Location of data. Where DOS will get it or put it.	
11-12	Amount of data to be processed. Should be 255.	
13	Command codes. They are:	00 - Position the head 01 - Read a sector 02 - Write a sector 04 - Format a disk
14	Error code returned by DOS if operation failed.	10 - the diskette was write protected 20 - the volume number was incorrect 40 - the drive had a problem 80 - the data was bad
15	Volume number actually found is placed here	
16	Previous slot number, times 16, from last access of the disk. found. Low and high addresses.	
17	Previous drive number from last access of the disk.	
18	Previous slot number from last access of the disk.	

The big table above is called the IOB. You must place the low order address of the IOB in the Y register and the high order address in the A register. Then, call the subroutine in the DOS that will perform the operation. The addresses should be as follows:

MEM	SIZE	ADD(hex)	INT	FP
16k		3D00	15616	15616
20k		4D00	19712	19712
24k		5D00	23808	23808
32k		7D00	32000	32000
36k		8D00	-29440	36096
48k		BD00	-17152	48384

NOTE: An easier method would be to CALL $3D9, which contains a jump to the proper address.

DEVICE CHARACTERISTICS TABLE

This table describes the DISK II to the DOS. The information will always be the same if the current disk drive from APPLE is used. The table contains the following: 00 01 EF D8.

Good Luck....

IN 9 EASY STEPS DISK-BASED PGM

BY PAUL WYMAN

Listen my friends, to a little fable about a certain nameless prince, who once upon a time shared a wonderful and magical little program with his friends. Each day they learned a little bit more about how to use the program, finding more and more magic all the time. It seemed that the program was not so little after all, and that it could practically do everything for everybody. Finally it became obvious that the equally nameless program did have a limitation. It was only capable of storing its output on cassette tape instead of disk. The ability to save and load the results of the nameless program to and from disk was a feature devoutly to be wished. Finally a way was discovered, and albeit crude, was in fact an answer to the problem.

The technique works like this: once you have used the program to create the results you want to save drop into the monitor and display locations 74.77(Hex). Before your very eyes will appear two copies of the pointer to the "end" of your results. The pointer to the "beginning" lies in 70/71 and will be "00 20" or $2000 make note of the 'end' pointer, and perform the following steps:

(1.) Reboot DOS
(2.) BSAVE RESULT.74: LLHH, A$2000, L$LENG

In (2) above, RESULT should be a descriptive name for your purposes. Next save the actual low (LL) and high (HH) byte values from 74.75 into the name of the file, so you won't forget the end pointer. You now have a binary image of the results you developed in the form of a disk file. LENG is the length of the file.
To restore the file and use the nameless program again to change or display your previous results simply follow these steps:

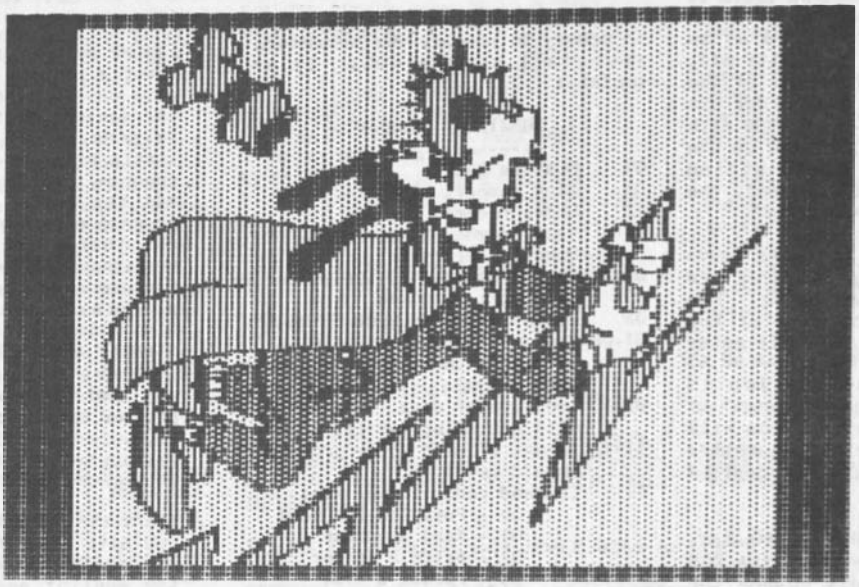
(1.) >BLOAD RESULT.74:LLHH
(2.) >BLOAD NAMELESS.PROGRAM
(3.) >CALL 2048
(4.) : ADD
(5.) Ctrl-D to escape add function
(6.) Hit Reset
(7.) *74: LL HH LL HH
(8.) *803g
(9.) :LIST

Don't forget to re-establish the end pointer two times into both 74.75 and 76.77, because the prince once said "two is a very good number".

Enjoy....

ComputerLand®
of San Francisco

apple® computer
Sales and Service



HARDWARE	SOFTWARE	PRINTERS
T I Color Monitors	Information Unlimited Software	Centronics 737
Micro-Works Digitizers	Personal Software	NEC Spinwriters
Mountain Hardware MUSIC SYSTEM!	Programma International	Silentype
ALF Music Synthesizer	Microsoft	Trendcom
Z-80 Softcard	Synergistic Software	Qume
Videx video-term	Automated Simulations	Diablo
M&R Sup-R-Terminal	California Pacific Computer	Paper Tigers
Mi-Plot Plotter	Apple	
Graphics Tablet	Bits & Pits	

ComputerLand™

117 Fremont Street • San Francisco, CA 94105 • (415) 546-1592

REVIEWS

JL

REVIEWS

REVIEWS

REFERENCE MANUAL

BY KEN SILVERMAN

The RED BOOK has been replaced. The new Reference Manual is following the new size of 6" by 8" and is 196 pages long with a complete circuit drawing of the Apple in the back.

It should be at your local computer store for the price of $19.95 (A2L0001A).

The book was written by Chris Espinosa, an Apple Core member, and is the best manual I have seen come out of Apple. It covers subjects that were in the original reference manual (Red Book) but in more detail with subjects easier to find. There are good examples included with each topic. There is a complete listing of both monitors (old and new autoboot).

The table of contents is 6 pages long with the following chapter headings:

1 - APPROACHING YOUR APPLE
2 - CONVERSATION WITH APPLES
3 - THE SYSTEM MONITOR
4 - MEMORY ORGANIZATION
5 - INPUT/OUTPUT STRUCTURE
6 - HARDWARE CONFIGURATION
7 - APPENDIX A, 6502 INSTRUCTION SET
8 - APPENDIX B, SPECIAL LOCATIONS
9 - APPENDIX C, ROM LISTINGS
10- GLOSSARY
11- BIBLIOGRAPHY

This is one publication you will want in your reference library. Check with your local computer store.

HANDBOOK

BY KEN SILVERMAN

The basic computer language in most home type computers is BASIC. But BASIC has never been standardized; there are more than 100 "dialects". A guide called "The Basic Handbook: an Encyclopedia of the Basic Computer Language", written by David A. Lien, provides good deal of help in coping with these assorted dialects.

The book covers 78 of the most popular versions of BASIC in such a way that its readers can convert nearly any program to run on a computer that uses a different version. These include more than 50 computers from IBM, Cromemco, Apple, Exidy, IMSAI, Heath, OSI, Radio Shack, Commodore, HP, and Wang, among others. The book is 360 pages and is available for $14.95 plus $1.35 postage from:

CompuSoft Publishing
8643 Navajo Rd.
San Diego, CA 92119

ROGER'S EASEL

BY GENE WILSON

I first met Roger Wagner at last May's Computer Faire, where he was selling the Apple II Utility Disc (fantastic set of programs that are still in heavy use on my machine). Roger's attention to detail was immaculate, and the documentation was 'complete' in every way. ('Complete' is when the documentation includes a detailed tutorial-one that has examples as needed, and gives insights into the critical areas. A program is 'complete' when the documentation allows me to fully understand what is taking place, and why.)

Roger's Easel is another 'complete' program. Easel is not another Electric Crayon. It is a Lo-Res Color Sketch program that employs the game paddles for fast pictures, logos and diagrams, etc.. (My joy stick draws truly fast pictures). Integer and Applesoft routines are included to 'link' these finished pictures to user programs. Complete documentation describes the entire process and the procedure is made crystal clear.

Actual use of the Easel is accomplished by following instructions listed in the bottom-of-screen window. Typing 'N' will give current color & X, and Y positions. 'H'elp switches to Page 2, for a complete list of commands. The left and right arrows and REPT keys together allow fast scrolling of this information, and shows a detailed description of each command. (Figure out how the scroll works--It's impressive). Roger states that user created pictures can be successively linked together for animation.

The only flaw that I could find is that at $13.95 (plus tax), the programs are too cheap. In today's market place price does not always denote quality or excellence, and many twenty to forty dollar packages lack any of the nice extras included in the Easel. To order ROGER'S EASEL, or for more information, please contact:

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

As a footnote, it is worth mentioning that for anyone who has lost track of his/her Applesoft programs there is another offering from Santee--APPLE-DOC. This three program set gives you the ability to list every variable used in a program and all the lines each is used on, a list of all the lines called by a GOTO, GOSUB, etc., or to rename any or all occurrences of any variable, change variable types, replace constants. This program set is also only $13.95 (cassette version $9.95).

Support your local programmer. You need programs, programmers need income-reach a happy compromise. Everyone benefits.

PILGRIMAGE

BY GENE WILSON

On a recent trip to the Los Angeles area (I think the city limit signs are posted about twenty miles south of San Jose) I took time out to stop at the all time "shrine" of Apple users-Computer Components of Orange County.

The place is filled with all the goodies that cast spells on the computer user's wallet; software galore, gadgets, super joysticks, magazines, books, lessons, tutorial sessions, hardware on line and working. It is possible to get many special items (cranked out in the back rooms, no doubt) including interface cards, lower case boards, and many more things that I don't fully understand.

Among the many magazines is a newcomer called "The Apple shoppe", which is published more or less monthly for the purpose of promoting practical uses of Apple Computers, by Compu-Tutor Co. Subscriptions are $12.00 a year, and are available from:

The Apple Shoppe
P.O. Box 701
Placentia, CA 92670

Volume 1, No.3 has several short programs, a Graphics Workshop section on Hi-Res Shapes, a discussion on Personal Computing's Family Tree, a short primer on Pascal, and a very useful printing subroutine for the WHATSIT program (something sorely needed). You don't want to miss Kim Clark's method of Water Cooling a PET.

And finally, is anyone able to "really" review some of the assemblers that are available for the Apple II? Just reading the documentation for Programma's ASM/65 EDITOR ASSEMBLER scared me off. I would love to hear from anybody with "hands-on" experience on the ASM/65. In all fairness, the documentation is thorough, and will probably be a future challenge.
LISA, Programma's Interactive Assembler changes a few mnemonics, and syntax is not identical to MOS Technology, which requires some thought before jumping in. Improved version 1.5 was just announced, and (you guessed) mine is the "older" 1.3 (looks like software promoters have learned something from the car makers).
I've just started using the Software Concepts Text Processing System & Assembler--Seems to do what I need done without a lot of frills (that I can't remember anyway).

Anybody with knowledge of new products should send a memo to the Cider Press as there are over 800 members who can benefit from your experience(s). Maybe someone from Programma could tell us how well their different assemblers have been accepted, and how "updates" are handled.

LISA AUTHOR STRIKES BACK

LETTER FROM RANDY HYDE

Dear "Press",

I read "Pilgrimage Plunder" by Gene Wilson in the September Issue. Being "someone from Programma" and, in fact, the author of LISA I was prompted to reply to Gene's comments on ASM/65 and LISA. First, ASM/65 is a "professional" product (whatever that is) and the documentation resembles an IBM manual more than anything else (because it was written by an ex-IBM user) and as such it is not easy reading for the average Apple owner. There are a few "cyber-snobs" out there however who eat the stuff alive, not to mention pay $69.95 for a product which is "professional" even though it may not offer any advantages over similar programs. For the person who likes to impress people, ASM/65 is definitely the choice. ASM/65's main advantage is that it incorporates the Apple PIE text editing system which is an order of magnitude better than anything else. If you plan to do a lot of text manipulation on source files, ASM/65 is your only choice.

With regards to LISA I have several comments to make. First, the "few" mnemonics which were changed were the Sweet-16 mnemonics, none of the 6502 mnemonics were affected. Several "extended" mnemonics were added, possibly this is where the confusion lies. Extended mnemonics are simply duplications of an existing mnemonic with a different name. For instance, the BCC (branch if carry clear) test can be used after a comparison to test for the less than condition. "Carry Clear" does not register in most peoples minds as "less than". So I added the mnemonic "BLT" for "branch if less than". It generates the same code as "BCC", yet is much easier to remember than "BCC". Likewise "BGE" "branch if greater than or equal" has been added as has "BTR" & "BFL" "branch if true and branch if false".

XOR has also been added and may be used in place of EOR. Please note that these "extended mnemonics" are included IN ADDITION to the existing mnemonics, if you don't like them you don't have to use them.

Some of the Sweet-16 mnemonics have been changed so that they conform to the MOS three character mnemonic syntax. All two character mnemonics were converted to three character mnemonics and all four character mnemonics were also converted to three character mnemonics. The reason behind this is simple, it improves the listings enormously if the mnemonics are all the same length.

As for the syntax not being identical to MOS's syntax I must point out that only one assembler for the Apple II uses MOS syntax and that's ASM/65. When I wrote LISA I researched the field carefully and used a syntax which was compatible with most of the existing assemblers at the time.

LISA's syntax, with the exception of decimal addresses and octal numbers (and who uses octal numbers anyway?) is IDENTICAL to MOS's. Again, I added some extensions to make LISA easier to use than would normally be the case. Since none of the assemblers available for the Apple II have a completely compatible syntax, there really is no "standard" syntax. I recognized this problem when writing LISA and as such I tried to include a mixture of each of the available syntaxes. For example, load the accumulator with a hex constant is specified as:

```
LDA 00 ($0 IS THE CONSTANT
TO BE LOADED)
```

when using the original Microproducts assembler, or as:

```
LDA #$0
```

When using a MOS compatible assembler (such as ASM/65 or the SC Assembler II). I wrote LISA so that both methods of specifying an immediate hex constant could be used. The drawback to this scheme? Well in the "standard" MOS syntax world the Microproducts version means "load the accumulator from decimal location 0". Since I didn't want to disallow this function I decided to differ from MOS rather than not allow the Microproducts' syntax. The reason behind this is simple, "LDA 0" is easier to type into the machine than is "LDA #$0" (in the latter version you have to type two extra shifted characters) Since I wanted to make LISA extremely easy to use I decided to use a different syntax to specify decimal numbers. To load the accumulator from location 0 you would type:

```
LDA !0
```

where the "!" tells LISA that the following is a decimal number. The only other variance from the "MOS standard" concerns the specification of a low or high order byte in an immediate expression. To load the accumulator with the low order byte of an address you would use:

```
LDA #address
    -or-
LDA #<address
```

To specify the low order byte you may use:

```
LDA #address
```

only. The "#<"option is not supported (the reason will be clear when version 2.0 is released).

To specify the high order byte of an address you would use:

```
LDA #>address
```

When using a MOS syntax assembler. Again I didn't like the idea of having to type two shifted characters, so once again I "borrowed" LISA's syntax from the SC Assembler II. To specify the high order byte you would use the following:

```
LDA/address
```

And that (unless I've forgotten something) is the extent of the syntax differences. Naturally LISA's pseudo opcodes are incompatible with the MOS syntax, but then so are everyone else's. LISA was designed to be easy to use, not follow blindly the syntax rules designated by some electrical engineer with considerable computer background.

I'm also wondering why Gene knocks LISA and praises "EAT". "EAT" (from Software Concepts) contains as many syntax irregularities as does LISA! I'm not knocking "EAT", it is a fairly good assembler (although overpriced), but Gene seems to imply that LISA is incompatible with MOS syntax whereas "EAT" is not.

Concerning updates, Programma Int'l has one of the most liberal update policies around. Simply send in your original disk (so we know you actually bought the product), plus the difference in cost between the two versions plus a $5.00 update fee and you will receive the update and the new documentation by return mail. Speaking of updates I would like to take a moment to mention the new features in LISA 1.5. LISA 1.5 works with the DOS 3.2 subsystem, as well as the auto-start ROM and the Apple II PLUS (or minus in some people's opinions!). Several new commands have been added to the command processor which allow you to modify (edit) existing lines, determine the length of the file, break to the monitor, and write your textfile to the disk as a text type file so that you may edit your file with the Apple PIE text editing system should your program require extensive modification. Several new pseudo opcodes have been added to the system increasing LISA's flexibility immensely. LISA now supports 23 pseudo opcodes making it easily the most flexible assembler around (and also the assembler best integrated into the Apple II). The pseudo opcodes include:

```
NLS - NO LISTING
LST - LISTING ON
EQU - EQUATE
EPZ - EQUATE PAGE ZERO
ORG - PROGRAM ORIGIN
OBJ - OBJECT CODE ADDRESS
ADR - ADDRESS PSEUDO OPCODE
BYT - BYTE DATA       (* NEW *)
HBY - HIGH BYTE DATA (* NEW *)
HEX - HEX STRING
ASC - ASCII STRING
STR - STRING W/LENGTH BYTE
DCI - DEFINE CHARACTERS IMMEDIATE
      (* NEW *)
INV - INVERTED CHARACTERS (* NEW *)
BLK - BLINKING CHARACTERS (* NEW *)
PAG - SKIP TO TOP OF FORM (* NEW *)
PAU - PAUSE, FORCE ERROR  (* NEW *)
END - END OF TEXTFILE
ICL - INCLUDE, CHAINS TEXTFILES
DCM - DISK COMMANDS, MULTI-PURPOSE
DFS - DEFINE STORAGE
```

VERSION 2.0 PSEUDO OPS

```
LET - REDEFINES A LABEL(FOR LOCAL
STORAGE)
FLT - FLOATING POINT NUMBER
```

In addition, error and listing options have been improved considerably. Version 2.0 (for the language card, to be released in January) has several improved error messages, supports eight character labels, signed and unsigned decimal integers (now you can specify "JSR !-936" just like in BASIC!), improved syntax concerning decimal constants, multiplication and division in address expressions are also supported as are local labels.

LISA STRIKES BACK (CONT)

If you ever have any questions about a software product, or wish to report a bug, or want to obtain an update, please contact the author or distributor, "We're here to help."

Signed,
Randell Hyde
Lazer Systems
12804 Magnolia
Chino, CA 91710

Editorial note: In Cider Press Vol. 2, No. 5 I asked for someone to come forward and "really" review some of the assemblers that are available. I'm glad that Randy has answered the call. It's good to get an 'in depth' survey of LISA. I must, however, question where Randy found a comparison between LISA and any other assembler. I use the Software Concept's Text Processing System/Assembler - I'm comfortable with it, and don't feel at all that it's 'overpriced'. (Using a product is a form of praise that certainly can't be ignored). Perhaps we can find some 'neutral' arbitrator to decide the relative merits of each product. (Sounds like another call for a review-and, in fact, it is. The actual merits of any program will be ultimately decided in the marketplace, and, having purchased both products - my vote is therefore cancelled out.)

The Interactive aspect of LISA certainly makes it one of the more desirable products on the market, and I personally like the approach that Randy has taken to continually improve his work.

I thank Randy for providing the Core with the above information, and hope that he will continue to enrich us with his knowledge.

G.W.

APPLE '21'

BY JIM LINHART

I want to spread the word of a card game that's a gas to play and has the best hi-res graphics of any card game I've ever seen. It's Apple 21 and it's a game worth adding to your library.

Apple 21 is the game of Las Vegas blackjack and it provided me with a good feel for the game. One thing I know, I'm no gambler and it's easier to pay a $10,000 debt by mashing "RESET" than by selling plasma or refinancing my house.

Apple 21 plays with a 52 card deck and when it reaches the end of the deck it goes on to reshuffle. Card counters now have a chance to perfect their skills, and the game includes such aspects as insurance bets and doubling. I sure enjoy playing this game and am surprised sometimes at how much time (and money) goes by.

TRENDCOM 100

BY SCOT KAMINS

"What's that you say? A reliable hobbyist thermal printer for less than $500? With an interface? And paper readily available? Why, you must be mad." While the final statement is undoubtably true, it does not negate the previous ones.

The printer of which I speak is the Trendcom 100 - a little 40 column, bidirectional job that hums along at 40 CPS. It is, in the vernacular, a real sweetheart.

The system consists of the printer, an intelligent interface card and a roll of paper (80 ft.) for $450. Extra paper is $5.00 for two rolls which prints in blue,; black is a little higher.

Regular readers of this journal are familiar with my propensity for panning printers. Alas, I find little to complain about with this gem. I have been using one since mid-October without a glitch. True, its narrow format makes it impractical for business uses. Its 40 columns allow for calendars and biorhythms, and little else; but it sure is terrific for listings. Anyone who has struggled along for a year editing programs 24 lines at a time knows how valuable a cheap printer would be - Trendcom fills the bill admirably.

The company itself (Trendcom, 484 Oakmead Pkwy, Sunnyvale CA 94086) impresses me for a number of reasons. First, it sells only to dealers - which is a good step in helping the industry to mature.

Secondly, it actually delivers its product before it is due, as opposed to the 3-months-late de-facto-standard we have been plagued by up to now. And third, they don't lie about the product. What they claim about their product in their talks to dealers is true.

Without hesitation, I recommend this printer to hobbyists; if I were you, I'd begin leaving Christmasy hints around the house, naming the Trendcom and your favorite computermonger. Happy listings...

DISK II

BY ANDY HERTZFELD

The APPLE II Floppy Disk Subsystem was released on July 7, 1978 only about a month after it was originally promised (although it was tantalizingly displayed at the 2nd Annual Computer Faire in March). For the most part, it was well worth the wait.

The drive itself is based on the standard Shugart Mini-floppy drive but with its electronics completely revamped by APPLE. The specs are rather impressive; it has a transfer rate of 156K bits per second, nearly 100 times faster than the cassette interface. It has an impressive capacity of 116K bytes (143K with Pascal), about 30 more than most mini-floppies get. It can access any data stored on it in well under a second, which sure beats listening to obscene squawks from a tape recorder. An average of 15 to 20 programs will fit on a single diskette (depending, of course, on program size); the diskettes cost around $5.00 apiece - shopping around pays.

For a list price of $595 you get one disk drive with case and connector ribbon, one controller card capable of handling up to two disk drives, and a complete disk operating system (DOS) with 19 commands. All the hardware and software you need to get going are included, even a blank diskette; you just plug the controller card into any peripheral slot except 0, plug the drive into the controller card, and you're ready to go. The bootstrapping routine (which gets the thing running) is stored on 512 bytes of PROM on the controller card, so all you have to do is reference the proper slot and the firmware takes over to load in the DOS and even start executing a program of your choice.

The DOS is a real memory hog. APPLE claims you can get "full disk capability in systems with as little as 16K RAM" - which I suppose is true, if you don't need APPLESOFT, HIRES graphics and you're willing to write programs that use less than 3K or so. Apparently the entire DOS must be resident at all times, and it needs 8 - 12K depending on how many file buffers are used. This is a bad way to set up a microcomputer's operating system, unless you're in the business of selling memory.

Since you usually only use one command at a time, it should be able to overlay itself to conserve its memory requirements; only about a 2K nucleus should have to be resident. For some inexplicable reason, APPLE did not overlay; consequently you really need a 32K system to use the DOS comfortably.

The disk system is an important advancement; 100K + of online, direct-access storage transforms the APPLE from a sophisticated toy into a full-fledged "real" computer. Its price is a bargain in the current market, and it should make a significant addition to anybody's system.

80 COLUMNS

BY KEN SILVERMAN

Recently your roving reporter was in Sunnyvale, CA to see and talk to Marty Spergal of M & R Enterprises. M & R as you might know manufactures the Sup 'R' Mod we use in our Apple.

While getting a tour of the plant I was shown a working prototype of a new I/O card. BELIEVE IT OR NOT I saw 80 columns by 24 lines, upper and lower case, coming out of the Apple into a monitor. The resolution on the inexpensive monitor (8MHz bandwidth) was fantastic. I asked about this and the designer, John Wilbur, told me there is a new circuit (being patented) that adjusts the contrast between the horizontal and vertical lines to make it look like an expensive monitor.

According to M & R, the new card will be in computer stores for your evaluation some time in the end of January (1980). They say the card is compatible with the Apple Pascal System (so don't rush to buy a terminal), works with Apple DOS as a Hello program, and should work with most existing software. This is great for those of you working with word processors or time share programs. In fact John Draper has started to implement a new version of EasyWriter to work with this card with some new features.

M & R Enterprises says ask your local computer store for "SUP 'R' TERMINAL". Take my word for it, I saw it work - as of this time they did not give me a retail price.

APPLE DOC

BY RANDY FIELDS

APPLE-DOC is a 3 program set designed as an aid to the development and documentation of APPLESOFT programs. The 3 programs are 1) Vardoc, 2) Linedoc, and 3) Replace. They were designed, developed and documented by Roger Wagner of Southwestern Data Systems.

Vardoc overcomes one of the fundamental deficiencies of most, if not all, versions of BASIC - the non-centralized location of all variable names. Vardoc, which is short for Variable Documentation, finds all variable names used in your progam, alphabetizes them and lists them on your CRT and/or printer. You also have the option of writing a separate file which contains a description of each variable allowing you to delete the usual REM statements from your program, saving memory and increasing execution speed.

Using Vardoc is the essence of simplicity. For Disk II users, load your program in first, then EXEC Vardoc. Tape users load their program, type in some PEEKs and POKEs, and load Vardoc. Roger's banner appears, hit any key and "Working" appears along with an estimate of the time to complete the search and alphabetizing process, generally less than a minute.

At this point, a complete list of variables is listed. Array variables are indicated by an asterisk (*) for each dimension of the array. After the name of each variable, EVERY line in which the variable is used is listed.

After listing each of the variables, you can at your option create a separate file of descriptors for all or some of the variables. Additionally, as you are supplying the descriptors, typing an 'L' lists the lines that the variable is on. Finally, after you have entered the descriptors, you can save the list of variables and their respective descriptors to a special disk file. For documentation purposes, this list can be output to a printer. Later, after you have modified or extended your program, the list can be read in for additional updating.

Linedoc is similar to Vardoc and performs several functions. First, it constructs a table of every line referenced by GOTOs and GOSUBs. Second, any line numbers which are called by mistake (resulting in an UNDEFINED STATEMENT error) are listed at the end of the table and indicated by an asterisk (*). Descriptors can be added, saved, and retrieved as in Vardoc. The main use for Linedoc is to unscramble the contorted logic which typically appears in even the best of programs.

It seems that I have saved the most interesting program till last: it is called REPLACE and allows you to replace various elements in an Applesoft program with replacement sets of your choice. You can change variable names, change numbers to variables or vice versa, replace a literal, document a literal, list lines, and convert Integer programs to Applesoft with a minimum of typing. A "literal", by the way, is any character or set of characters such as: '+', 'PRINT', 'A1', etc. When replacing either variables or literals, the Replace program asks if you want to replace all or just some. If you respond with 'some', it will list out the entire line and ask you whether or not to do the replacement. Multiple occurrences in the same line are listed separately allowing you complete flexibility in doing the replacements. If you respond with 'all', Replace changes them all without listing. Some useful functions of Replace are: changing real variables to integer variables and vice versa and changing INPUT statements to GET statements.

All three programs - Vardoc, Linedoc, and Replace - come complete with excellent documentation telling you not only how to use the programs, but how the programs work.

APPLE-DOC is available from local computer stores or from SOUTHWESTERN DATA SYSTEMS. P.O. Box 582, Santee CA 92701 for $13.95 for the diskette or $9.95 for the cassette. It is probably the best programming buy for the money, and at these prices, Roger should not be ripped off by bootleg copies.

Ask for APPLE-DOC by name: do not be confused by shoddy imitations!

GRAPHICS TABLET

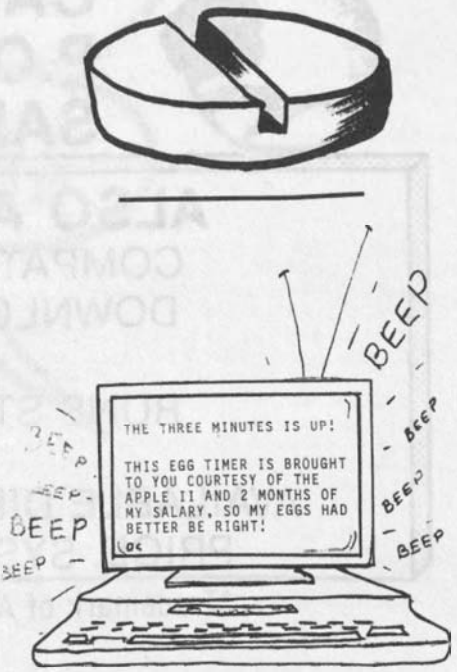
BY KEN SILVERMAN

Apple now has its new Graphics Tablet in the stores. It will allow the user to convert graphic data into digital information that may be processed by traced or drawn freehand on the tablet surface and are instantly displayed on the system monitor. Any image created can be stored on the system disk for later use. Block diagrams, architectural renderings, logic diagrams, etc. are a few of the applications.

The tablet consists of 15 by 15 inch tablet, a mylar overlay, stylus, diskette software and interface. It uses the Apple power supply and results in a low profile tablet, fewer boxes to contend with, easier hook up, and high reliability.

The Software Package is composed of some assembly language fast draw routine and the master control is in Applesoft BASIC. It can process up to 120 coordinate pairs per second. This means that unlike other systems it will keep up with the fastest hand motion.

The tablet is used by pressing the pen on the designated square to select a function. Some items that are selectable are CLEAR, REDUCER, CALIBRATE, PEN COLOR, DRAW, LINES, DOTS, BOX, BACKGROUND COLOR, and some of the commands are selectable from the tablet are CATALOG, SAVE, LOAD, SLIDE, and AREA. These are just a few and a complete listing can be obtained by checking with your local computer store.



FORTH

VER. 1.7 FOR APPLE II* COMPUTERS

100 PAGE, PROFESSIONALLY WRITTEN MANUAL
FORTH INTEREST GROUP COMPATIBLE
DIRECT HOT-LINE TO SYSTEM DEVELOPERS
INCLUDES ITS OWN DOS
CAP'N SOFTWARE HAS DELIVERED 100's OF
WORKING FORTH SYSTEMS
UPDATE OFFER: TRADE IN YOUR VER. 1.6. DISK
FOR FULL CREDIT OF PURCHASE
PRICE TOWARD VER 1.7
RUNS ON APPLE II OR APPLE II+ WITH
1 OR MORE DISKS AND 48K.
ALSO RUNS ON LANGUAGE CARD
AVAILABLE AT COMPUTER STORES OR
DIRECTLY FROM CAP'N SOFTWARE
PRICE, SYSTEM $140, MANUAL ONLY $20

CAP'N SOFTWARE
P.O. BOX 575
SAN FRANCISCO, CA 94101

ALSO AVAILABLE FOR PDP-11†

COMPATIBLE WITH VER. 1.7 FOR APPLE
DOWNLOAD PROGRAM DEVELOPMENT
OR EXECUTION
RUNS STAND-ALONE OR UNDER RT-11,†
RSX-11M†, OR RSTS†
AVAILABLE DIRECTLY FROM CAP'N SOFTWARE
PRICE, SYSTEM $145, MANUAL ONLY $20

*Trademark of Apple Computer Corp. †Trademark of DEC.

LANGUAGES

JL

RAM - FP

BY BARNEY STONE

Apple Computer has quietly decided to drop the RAM versions of Applesoft II Basic. Sometime in the last two or three weeks, and with no notice to their dealers, Apple stopped including the Applesoft II cassette with their computers, and removed the disk version of Applesoft from the master diskettes that come with the disk drives.

The reasons quoted by Apple included the desire to concentrate support on one version of the language - i.e. ROM version (the Pascal/Language system uses the ROM version of Applesoft - that is, although the language will be in RAM, it will use the same memory locations as the current ROM version). Apple in addition stated that the costs and manpower could be better utilized on developing new projects.

As of this time if you call the Hotline about a problem with Applesoft you will get answers but sometime in the near future that support will be stopped.

NOTE: The current version of Applesoft on the card includes the new Auto-Boot ROM.

FP-INT-FP

BY JEFF FRANKEL

In going through the new issue of the Apple user group newsletter "Contact 5" I found a great routine that will aid in converting programs from Applesoft to Integer and vice-versa.

This article assumes that one has a disk drive and Applesoft in ROM. A printer is also helpful especially when going from Applesoft to Integer. When I say that this routine will aid you in program conversion I mean that it will save you the job of tediously re-typing those zillion and a half program lines. The converted program will in most cases not run correctly owing to the differences in the two languages. Making it work is up to you!

Here is what to do.....

1. Please read all of the following instructions before you do anything else. It is easy to goof this up if you don't understand what is happening (I even fouled up my text editor writing this article).

2. Boot the disk drive. Insert a disk containing the program you want to convert. This disk should have plenty of room on it.

You will be writing on this disk so if you are worried about wiping something out .. get another disk... Save the program you want to convert on to it and use that disk.

3. Load the following routine to the source program using any available line numbers. Do this near the beginning of the program if possible:

```
10 REM CONVERSION ROUTINE
20 REM "*"="CONTROL D"
30 PRINT "*OPEN XX"
40 POKE 33,33
50 PRINT "*WRITE XX
60 LIST
70 PRINT "*CLOSE
80 END
```

5. "RUN" the modified source program. The program will open an "exec" file on the disk and list itself into this file. This file will have a "T" prefix and will not respond to a run command. It is accessed through the use of the "EXEC" command.

6. Now use either the "INT" or the "FP" command to jump into whichever language you wish to use for your object program.

7. If your source program is in FP and your object program is to be in Integer turn on your printer. When the EXEC file "types" FP program lines in integer.. all lines of the FP program will be typed in and evaluated by the integer language program itself. Those FP program lines that won't run in integer will be flagged with error messages. If you have your printer on these program lines will be printed along with error messages. However.. integer will not insert these lines into RAM memory.
If you don't have a printer.. go on to step 8. When you list the program back.. incompatible program lines will be conspicuous by their absence. If you are going from integer to FP don't bother with the printer. All program lines will be inserted into RAM and the error messages will appear when you try to run the program.

8. Now execute the file by typing "exec xx". Subject to the limitations as noted above the file will type into RAM memory your object program.

9. Delete the special routine from the object program.

10. Save the object program back onto the disk. (Be sure not to save it under the original name - you will wipe out your source program).

Now comes the fun part of getting the program to run in the new language.

GOOD LUCK.....

FP RENAME

APPLE APPLICATION NOTES

Have you ever written a program that needed to check to see if a data file existed before trying to read it? Most programmers seem to use the method of opening the file, then reading it. If an ONERR statement is in effect, an OUT OF DATA error is trapped. This has the drawback of creating a file in the diskette directory that must then be deleted. Another method is using the VERIFY command. It works but can take a while with a large file.

Here is an example of a program that prompts for a file name and will continue only if the file already exists.

```
]LIST 0,200
100 TEXT: HOME: D$= CHR$ (4): VTAB 11
110 PRINT: INPUT "FILE NAME: ";A$
120 IF A$ = " " THEN END
130 ONERR GOTO 160
140 PRINT D$"RENAME "A$","A$"
150 POKE 216,0: GOTO 180
160 POKE 216,0: PRINT D$;"CATALOG"
170 GOTO 110
180 REM  **MAIN PROGRAM**
```

The work is being done in line 140 by attempting to rename the file with the same name. It's possible to use the UNLOCK command here if the file needs to be unlocked later on in the program.

FP MEM MOVE

APPLE APPLICATION NOTES

Much has been said in articles about how to do a memory block move from Integer Basic. It's almost as simple to do it from Applesoft. You just have to poke a short machine language routine into memory first.

```
]LIST 200-250
200 REM MOVE PG2 TO PG1
210 POKE 60,0:POKE 61,8
211 REM START OF BLOCK
220 POKE 62,255:POKE 63,11
221 REM END OF BLOCK
230 POKE 66,0:POKE 67,4
231 REM START OF OBJECT MEMORY
240 CALL 768:  RETURN
241 REM DO THE MOVE
```

That's all there is to it.

FP PGM MOVER

APPLE APPLICATION NOTES

It is sometimes desirable to have an Applesoft program start at an address other than the default $801 (ROM Applesoft). A "safe" area for machine language, the desire to use page 2 of text or to make room for a large program that uses HIRES graphics are the usual reasons. Here is a good way to do this for Disk II users.

Since a diskette holding this type of program is generally dedicated to a specific purpose, the actual changing of Applesoft's pointers will be done by the "HELLO" program. Here's an example:

```
]LIST
100 TEXT:  HOME:  D$ = CHR$(4)
110 IF PEEK (104) = 12 THEN 150
120 POKE 104,12
130 POKE 103,1
140 POKE 3072,0
150 PRINT D$;"RUN MAIN PROGRAM"
```

This program makes room for using text page 2. This is what happens when the disk boots:

Line 100 simply clears the screen and sets the variable D$ equal to Control D.

Line 110 checks the high order byte of the beginning of program pointer ($68) to see just where the program is located. These pointers are explained on page 140 of the Applesoft manual. If the program is already in the correct location, a branch is made to line 150 where the application program is RUN.

Line 120 is reached only if the test on the line above fails. It POKEs the correct new high-order byte.

Line 130 POKEs the new low-order byte.

Line 140 changes the byte immediately preceeding the new program location to zero. Note that this byte must ALWAYS be zero for Applesoft programs. ($800 is a zero when a normally located program is in memory.)

Line 150 RUNs the main program.

This works "on the fly" because no references are being made to other program lines after the pointers are changed. The program is linear.

Another way of changing these pointers is to do it from within the application program. The drawback is that it is somewhat slower. To do it, use a subroutine to check the pointers like this:

```
100 LET D$ = CHR$(4):  GOSUB 60000

PROGRAM

60000 IF PEEK (104) = 12 THEN RETURN
60010 POKE 104,12
60020 POKE 103,1
60030 POKE 3072,0
60040 PRINT D$;"RUN MAIN PROGRAM"
```

The difference here is that if line 60000 is true the program continues execution. If the expression is false the pointers are changed and the program reruns itself. (The speed difference occurs when relocation must take place and the entire program must be re-loaded from disk.)

"GET" TRAP

BY MAX J. NAREFF

Full many a program bit the dust until the Dec. 1978 issue of "Contact" (from the Apple Computer Co.) revealed the culprit. "GET" was getting in the way. Unless a "GET" statement was followed by a "PRINT", the DOS command would be ignored.

The new 3.2 DOS manual mentions this problem and presents the "cure" on Pg 24; however a clearer picture emerges in the June and October issues of "Call APPLE", edited by Val Golding.

"....first set D$ to equal a carriage return, followed by a CONTROL/D. DOS requires that a carriage return be performed before each DOS command. Now it is possible to have multiple DOS commands in one program line."

"In addition, D$ will now allow (you) to "TRACE" with DOS up, and it WILL permit a DOS command to directly follow a "GET" statement."

Example-A.

```
100 D$=CHR$(13)+CHR$(4):  PRINT "OPEN
COLORS" D$ "WRITE COLORS" (note
absence of delimiting colon and
semi-colons)
```

Example-B.

```
100 D$=CHR$(13)+CHR$(4):  GET A$
110 IF A$=.....
```

SPC(X)-TAB(X)

BY MAX J. NAREFF

Applesoft tab fields are restricted to three as compared to the normal five in Integer. Details appear in the Apple manuals. This limitation sometimes imposes restrictions on the number and format of data outputs and headings. A detour around these roadblocks is illustrated:

```
 .(DATA GENERATING OUTPUT NOT SHOWN)..
150 D$="DATE"
160 R$="RAINFALL"
170 S$="SNOWPACK"
180 F$="RUNOFF"
190 PRINT D$,R$,S$,F$
200 PRINT : PRINT
210 PRINT D$; SPC(3);R$; SPC(3);S$;
    SPC(3);F$
```

Line 210 produces a much nicer format for the data output.

The SPC(X) command can only be used in a print statement and may be located anywhere within that statement. When preceded and followed by a semicolon, SPC(X) introduces X number of spaces between the item preceding and the item following it.

The TAB(X) command is used within a print statement but only immediately following the print command. In contrast to SPC(X), it dictates the absolute position relative to the left margin (defined as position 1) where the item will be printed that follows it. Type in the following three lines to see this work.

```
10 PRINT "P.PECK"; SPC(12);
   "PINK DR."
20 PRINT : PRINT
30 PRINT "P.PECK": PRINT TAB(12);
   "PINK DR."
```

$.XX TRAILING 0'S

BY SCOT KAMINS

The new APPLESOFT language is really terrific - especially if you have been fortunate enough to get it on the ROM card. It does, however, have some pesky quirks - among which is the lack of trailing zeros.

One feels this lack when trying to write business and finance programs that use two digits after the decimal point - i.e., money stuff. Following is a solution to the problem which, while inelegant, does the job.

```
 5 Z2$=".00":Z1$="0":D$="$"
10 CALL -936
15 REM HERE COMES A ROUNDOFF FUNCTION
20 DEF FN A(R)=INT(R*100+.5)/100
25 ?:?:INPUT "TEST ";TEST:?: REM
   GIMME A NUMBER
30 T$=STR$(TEST): REM WE TREAT THE
   THE NUMBER LIKE A STRING
35 REM HERE COMES A "SPECIAL CASE"
   FIX (I DID SAY INELEGANT!)
40 IF LEN(T$=2 AND MID$(TS,1,1)="."
   THEN 1500
45 FOR SEEK=1 TO LEN(T$): REM IS THERE
   A DECIMAL IN THIS NUMBER?
50 IF MID$(T$,SEEK,1)="." THEN 1000:
   REM  IF YES BRANCH TO LOCATION FOR
   FURTHER CHECKING
55 NEXT SEEK: REM KEEP LOCKING UNTIL
   THE CHARACTERS RUN OUT
60 T$=T$+Z2$: REM SINCE THIS NUMBER
   HAS NO DECIMAL, CONCATENATE A
   DECIMAL WITH TWO TRAILING ZEROS
65 GOTO 2000: REM BRANCH TO THE
   PRINTOUT
1000 IF MID$(T$,LEN(T$)-1,2)="." THEN
   2000: REM IF THIS NUMBER HAS A
   DECIMAL W/2 DIGITS AFTER IT GO GET
   PRINTED
1200 IF MID$(T$,LEN(T$)-1,1)='.' THEN
   1500: REM IF 1 DIGIT, GET ANOTHER
1300 T$=STR$(FN A(TEST)): REM ROUND
   OFF THIS NUMBER TO 2 SIGNIFICANT
   DIGITS BEYOND THE DECIMAL
1400 GOTO 30: REM GO BACK TO MAKE
   SURE TWO DIGITS FOLLOW A DECIMAL
1500 T$=T$+Z1$: REM CONCATENATE THE
   NUMBER WITH ONE TRAILING ZERO
2000 ?:?:?  D$+T$:?:?: REM PRINT A
   DOLLAR SIGN FOLLOWED BY THE NUMBER
   WHICH NOW HAS 2 SIGNIFICANT
   DIGITS BEYOND A DECIMAL POINT
2100 GOTO 10: REM GO GET ANOTHER
   NUMBER
```

POKE 51,0

BY PHIL BERNHEIM

Have you ever been de-bugging an APPLESOFT program and after making some changes, had a "NOT DIRECT COMMAND" error message thrown at you when you tried to run the program?

You scratch your head because, plainly, the program is making the command, not you!

There's a very simple answer - POKE 51,0. Now that doesn't make sense, because system monitor location 51 (decimal) is the prompt character. But what you're doing is fooling the computer.

When you interrupt a program that's running in DOS and give any commands from the keyboard, the computer then takes ALL subsequent commands as coming from the keyboard including those in the program itself. So if your program contains any DOS commands that are illegal from the keyboard (Open, Read, Write, etc) you'll get the NOT DIRECT COMMAND error message.

Just type POKE 51,0: GOTO (linenum), and APPLESOFT is fooled into thinking that your keyboard instruction (and all subsequent ones) came from behind the cursor and GOes TO the line.

It's just as useful written into programs. I have one program which uses the STOP command to permit me to make manual (keyboard) changes in a data matrix. The line after the STOP command reads POKE 51,0: PRINT ..., and the program later writes the changed matrix back onto tape. And it simply will not turn on the disk without that all-important POKE line in the program.

OUT OF MEM ERROR IN FP

BY RANDY FIELDS

I have been plagued, albeit occasionally, with a mysterious bug in one of my APPLESOFT programs. The program would be happily executing away, and all of a sudden, a beep and at the bottom of the screen, an 'OUT OF MEMORY IN LINE ###' error message appeared. I did a PRINT FRE(0) and had 8K of memory left!! Impossible, you say, as did I. Well, the problem turned out to be "uncompleted FOR-NEXT loops.

An uncomplicated For-Next loop looks like:

```
10 FOR I = 1 TO 10
20 J = 5 * I
30 IF J = 15 GOTO 50
40 NEXT I
50 continue with the program
```

Typically, line 20 contains a more complicated expression which causes premature termination of the loop. If you have more than 10 of these premature terminations you get the dreaded OUT OF MEMORY error just like the APPLESOFT Manual says for 10 nested For-Next loops.

The fix was supplied by Dick Huston of Apple Computer Inc. and looks like:

```
10 FOR I = 1 TO 10
20 J = 5 * I
30 IF J = 15 THEN I = 10:
   NEXT I: GOTO 50
40 NEXT I
50 continue with the program
```

The problem occurs because the index I and only 9 more indexes can be stored in page 0 locations. The 11th has nowhere to go, and the OUT OF MEMROY error message appears. Prematurely terminated loops leave the page 0 locations waiting to be completed. The fix in line 30 sets the index to its highest value (10 in the example) and the NEXT I before the GOTO de-allocates one of the page 0 locations.

I made the modification after talking to Dick and it works! Thanks, Dick.

P.S.: Leaving the variable names off in NEXT statements significantly improves execution speed.

AVOIDING LINE OVERRUNS

BY MAX J. NAREFF

The beginning programmer, confronted with the 40 character line limit, frequently finds himself with ungrammatical word spillovers and unwanted blank lines. A template is given below for the avoidance of these distortions.

```
10 PRINT".....";
```

This line must be 40 characters or spaces line. Proceed with frequent reference to the template above, keeping the line lengths within the limits outlined by the dots or spaces to prevent line spillover. When line 10 is scrolled away, merely type another appropriately numbered line template. Adjust for changes in the number of digits in the larger lines. The original line 10 may be used as the program opener with a "clear screen" command. The use of the semi-colon as a 'carriage return suppressor' will prevent the occurrence of a subsequent inadvertant blank as is illustrated.

```
10 PRINT".....
```

6503 CPU

BY PAUL KNEVELS

FROM MICHIGAN APPLE

It has been rumored that Apple is working on an apple III computer to be released shortly. At present, the advance information available is that a new microprocessor will be incorporated into the unit - the 6503.

Our research staff has been able to uncover a list of new opcodes that distinguish the 6503 as a breakthrough in computer technology.

The list is presented here for your information (and enjoyment).

```
AD  ADD GARBAGE
BBL BRANCH ON BURNED OUT LIGHT
BAH BRANCH AND HANG
BLI BRANCH AND LOOP INFINITE
BPB BRANCH ON PROGRAM BUT
BPO BRANCH IF POWER OFF
CPB CREATE PROGRAM BUG
CRN CONVERT TO ROMAN NUMERALS
DAO DIVIDE AND OVERFLOW
ERS ERASE READ-ONLY STORAGE
HCF HALT AND CATCH FIRE
IAD ILLOGICAL AND
IOR ILLOGICAL OR
MDB MOVE AND DROP BITS
MW  MULTIPLY WORK
NBC SHOW TERRIBLE PROGRAMS
PAS PRINT AND SMEAR
RBT READ AND BREAK TAPE
RPM READ PROGRAMMER'S MIND
RRT RECORD AND RIP TAPE
RSD READ AND SCRAMBLE DATA
RWD REWIND DISK
SRZ SUBTRACT AND RESET TO ZERO
SSD SEEK AND SCRATCH DISK
TP  TEAR PAPER
WED WRITE AND ERASE DATA
WID WRITE INVALID OP CODE
XIO EXECUTE INVALID OP CODE
XO  EXECUTE OPERATOR
XP  EXECUTE PROGRAMMER
```

CONTROL C

AN APPLE MATRIX

MATRIX PART 1

BY MAX J. NAREFF

Some forms of BASIC provide the matrix commands (MAT READ, MAT PRINT, MAT INPUT, etc,) to facilitate display and manipulation of tables of numbers. Though the MAT commands are unavailable in Apple Basic, there is a substitute.

The term "matrix" is used here to denote a two-dimensional array of numbers consisting of horizontal rows and vertical columns. A matrix is a grid in which each of the elements (numbers) is given a reserved space or cell in the computer's memory.

To accomplish this, the matrix is first dimensioned. Thus DIM A(12,12) allocates 144 cells in memory for the 144 (12*12) cells in the matrix. (Actually since Apple begins dimensioning from zero the number of elements in the variable is 13*13, or 169 - but who's counting?)

Following is a simple program using doubled digits illustrating how a matrix can be constructed or read from data statements in Applesoft. For the sake of brevity the number of rows is here limited to 3. Try more at your computer.

```
 0 DIM A(12,12)
10 DATA 10,20,60,70,90,11,88,42,32
20 DATA 77,41,66,19,20,91,72,72,18
30 DATA 36,24,11,30,93,46,66,33,14
40 DATA 98,16,32
50 FOR I=1 TO 3: REM INDEX LOOP FOR
   ROWS
60 FOR J=1 TO 10: REM INDEX LOOP FOR
   COLUMNS
70 READ A(I,J): REM READS DATA FOR
   MATRIX
80 PRINT A(I,J); SPC( 2);: REM PRINTS
   AND FORMATS MATRIX ELEMENTS
90 NEXT J
100NEXT I

RUN

10 20 60 70 90 11 88 42 32 77
41 66 19 20 91 72 72 18 36 24
11 30 93 46 66 33 14 98 16 32
```

When the matrix is on screen, computations may be made in the Immediate mode.
For Example:
PRINT A(1,5)*A(3,10).
With simple additions to the program all or selected matrix elements may be totaled, averaged, sorted, etc.

Next month: Manipulating the Matrix Elements.

MATRIX PART 2

BY MAX J. NAREFF

This series of articles is designed to simulate various MAT(RIX) functions, statements which are used to simplify the manipulation of large groups of numbers.

Last month MATRIX or table formation was demonstrated with the use of FOR-NEXT loops and double subscripted variables (also called two dimensional arrays). This month we deal with arithmetic manipulation of a MATRIX. Reference is made to last month's program which developed a table consisting of 3 rows and 10 columns of numbers. The following subprograms should be added to it.

1. SUM OF ALL MATRIX ELEMENTS

```
120 T=0:  REM INITIALIZES THE TOTALER
130 FOR I=1 TO 3:  REM LOOP FOR ROWS
140 FOR J=1 TO 10:REM LOOP FOR COLUMNS
150 T=T+A(I,J):REM TOTALS ELEMENTS
160 NEXT J
170 NEXT I
180 PRINT:PRINT TAB(4);"SUM OF MAT
    ELEMENTS=";T
```

2. AVERAGE OF ALL MATRIX ELEMENTS

```
120 T=0:K=0::REM INIT TOTAL & COUNTER
145 K=K+1:REM COUNTS # OF ELEMENTS FOR
    AVERAGING
190 PRINT:PRINT TAB(9);"AVERAGE OF
    ELEMENTS=";T/K
```

3. SUM OF INDIVIDUAL ROWS & COLUMNS

```
200 FOR I=1 TO 3:REM INDEX LOOPS
210 FOR J=1 TO 10
220 REM CALCULATION OF ROW & COLUMN
    TOTALS
230 R(I)=R(I)+A(I,J)
240 C(J)=C(J)+A(I,J)
250 NEXT J
260 NEXT I
270 REM LOOPS FOR PRINTOUTS
280 FOR I=1 TO 3
290 PRINT TAB (14);"SUM OF ROW
    #";I;"=";R(I)
300 NEXT I
310 FOR J=1 TO 10
320 PRINT TAB (14);"SUM OF COL
    #$";J;"=";C(J)
330 NEXT J
```

The terms "MATRIX" and "TABLE" can be used interchangeably. In subprogram 2 line 145 the counter statement can also be expressed as K=I*J another form of iteration as the loop uncoils.

Any component of a MATRIX - whether single elements or rows or columns - can be manipulated as can several matrices. Try "PRINT R(1)+R(2)"; "PRINT R(3)/10" and "PRINT C(2)+C(10)" or any combination required.

More next month-------

MATRIX PART 3 CROSSFOOTING

BY MAX J. NAREFF

As defined by Logsdaon (Programming in Basic, 1977) it is the mathematical process of summing the corresponding numbers (elements) of two arrays in pairs, one number from each array. The sum of each paired number appears in a third array. Originally used in computer card processing, the term has been broadened to include subtraction.

In the previous articles on Matrix simulation, the technique of CROSSFOOTING was demonstrated without being identified as such. Here now is a short application with a paired array.

Three service stations have each sold out their daily quota of two grades of gasoline (A&B). How many gallons has each outlet sold (C), and what is the grand total of company sales (T)?

```
 0 HOME : T=0 : REM CLEARS SCREEN :
   INTITIALIZES TOTALER
10 DIM A(3), B(3), C(3): REM RESERVES
   MEMORY SPACE FOR EACH ARRAY
20 PRINT "REGULAR","ETHYL","R+E SALE":
   PRINT
30 FOR X= 1 TO 3: REM MACHINE LOOP FOR
   READING DATA
40 READ A(X),B(X): REM READS DATA
   (A-REGULAR,B-ETHYL)
50 C(X)=A(X)+B(X): REM CROSSFOOTING
60 PRINT A(X),B(X),C(X)
70 T=T+C(X): REM COMPUTES TOTAL SALES
   3 STATIONS
80 NEXT
90 PRINT : PRINT : TAB 10;"TOTAL FUEL
   SALES= ";T
95 DATA 621.5,433.2,493.8,1217.5,701.8
   ,317.6: REM FUEL SALES CAREFULLY
   ENTERED IN PROPER SEQUENCE
   A, B, A, B, ETC
```

MATRIX PART 4

BY MAX J. NAREFF

MATRIX statements, unavailable in APPLE BASIC, reduce the programming effort required to solve problems involving one or two matrices (tables) by eliminating the need for nested-loops. Large clusters of data can be managed quickly with single MAT statements. As noted previously, several of these statements can be simulated within APPLE BASIC, and while the effort is more tedious, selected mathematical operations can be performed using nested-loops. In previous issues, the MAT READ and MAT PRINT statements were simulated and a single matrix (table) generated. Several arithmetic operations were performed with its contents, both in the direct and indirect modes. Now, two small rectangular matrices, each composed of two rows and four columns of numbers, will be formed and arithmetic interactions between them illustrated. MAT statements involving addition (or subtraction) and multiplication by a constant will be simulated.

Step 1 demonstrates formation of two (2x4) tables labeled MATRIX A & MATRIX B. Here again, the MAT READ & MAT PRINT statements are accommodated by use of nested-loops.

```
 0 CALL -936
10 DIM A(2,4),B(2,4): REM
   DIMENSIONS EACH MATRIX
50 PRINT TAB (15);"MATRIX A"
60 FOR R=1 TO M: REM ROW LOOP:
   M=2
```

```
 70 FOR C=1 TO N: REM COLUMN LOOP:
    N=4
 80 READ A(R,C): REM FROM DATA
    STATEMENTS
 90 PRINT A(R,C);SPC(8);: REM
    PRINTS AND FORMATS MATRIX
100 NEXT C
110 NEXT R
120 PRINT TAB(15);"MATRIX B"
130 FOR R=1 TO M: REM (M=2)
140 FOR C=1 TO N: REM (N=4)
150 READ B(R,C): REM FROM DATA
    STATEMENTS
160 PRINT B(R,C);SPC(8);
170 NEXT C
180 NEXT R
500 DATA 10,20,30,40,50,10,20,30
510 DATA 11,13,15,17,19,21,23,25
```

In step 2, a third MATRIX, C, will be formed by addition of A and B simulating MAT C=A+B

```
 20 DIM C(2,4)
200 PRINT TAB(8);"MATRIX C =
    MATRICES A+B"
210 FOR R=1 TO M
220 FOR C=1 TO N
230 C(R,C)=A(R,C)+B(R,C): REM
    MATRIX ADDITION
240 PRINT C(R,C);SPC(8);
250 NEXT C
260 NEXT R
```

In step 3, scalar multiplication is demonstrated. In this type of operation, each element of matrix may be multiplied by either a constant, a variable or an expression. This step is an example of MAT D=(K)*A where K is a constant.

```
 30 DIM D(2,4)
300 PRINT TAB(8);"MATRIX
    D=(K)*MATRIX A"
310 FOR R=1 TO M
320 FOR C=1 TO N
330 D(R,C)=(K)*A(R,C): REM FOR
    EXAMPLE LET K=5
```

The dimensions of the matrix on the left of the "equals" sign must be equal to or greater than those on the right side of the equation or else some of the elements may not appear; the shape of the new matrix may change, or more likely, the error sign will occur. Where the MAT capability exists, it is most useful and effective, but it is limited by its inability to add or subtract more than two matrices or to perform some other arithmetic.

SCORE ONE FOR THE APPLE: with the procedures outlined, many matrices can be added, subtracted, divided, etc. The interested reader is invited to try these and other options. Note what occurs where MATRIX A is multiplied by MATRIX B. The resultant matrix is produced by the multiplication of EACH element of A by the CORRESPONDING element of B (i.e. C(1,1) = A(1,1) * B(1,1), C(1,2) = A(1,2) * B(1,2) and so forth). In a sense this is a type of "scalar" multiplication where the multiplier varies from element to element. However, this is NOT true Matrix Multiplication in the language of computer science or matrix algebra, where the rows of one matrix are multiplied by the columns of the second and not by the corresponding elements.

FRE(X) FOR INT

Here are a couple of quickie one-lines to implement while waiting to decide how next to complicate your program.

This first one gets you the remaining usable memory. It is equivalent to Applesoft's FRE(X) function.

```
PRINT PEEK (202) + PEEK (203) * 256 -
PEEK (204) - PEEK (205) * 256
```

This one gets you the space taken up by your program:

```
PRINT PEEK (76) + PEEK (204) - PEEK
(202) - PEEK(74) + 256 * (PEEK (77) +
PEEK (205) - PEEK (203) - PEEK (75))
```

The difference you get when you subtract the total of these two functions from the RAM capacity of your machine represents the work space used by the interpreter. It should be 2K (2048) bytes.

Here's yet another program for determining the length of a basic program - except that this one is in machine language and is relocatable.

As it is written, you "CALL 768" to get the length of the commands and statements you've used without the length of the variable table - that is, you get HIMEM down to PP:

```
300:  38 A5 4C E5 CA AA A5 4D E5 CB 20
1B E5 60
```

AND that's it.

WAIT CMD

BY G. R. BALDWIN

Loitering inside the Apple System Monitor is a subroutine called 'WAIT'. It is useful for creating timing delays inside your programs without the bother of writing special code. Delays of up to one-sixth of a second are possible, depending on the value that is in the accumulator when WAIT is called. We can't control this value from a BASIC program, but it turns out we don't have to! Timing delays from one second to several minutes can easily be invoked from either INTEGER or APPLESOFT BASIC. To do this we will resort to a subterfuge involving both trickery and cheating.

The trickery is possible because the accumulator is left at zero when WAIT returns to the program which called it. And a zero in the accumulator will create a delay of 162 milliseconds if WAIT is called again. The cheating is possible because BASIC doesn't mess with the accumulator in the following tight loop:

```
FOR I = 1 TO 7:  CALL -856:  NEXT I
```

We don't know what is in the accumulator the first time the call is made to WAIT, so the delay might be anywhere up to a sixth of a second. But the call leaves the accumulator at zero, and subsequent calls to WAIT take 162 milliseconds each. So this line of BASIC will create a delay of about one second. By changing the limit on the FOR loop from 7 to 31, we get a delay of five seconds, and to 370 gives a delay of one minute. To calculate the proper limit, divide the desired delay in seconds by 0.162 and prune the result to the nearest integer.

For more precision, WAIT must be called using ASSEMBLY language:

```
LDA #nnn,nnn is a value between 0
             and 255 inclusive
JSR $FCA8 ;call WAIT
```

The delay is a shaggy function of the value you put in the accumulator (A):

Delay in milliseconds = (13 + 12.5*a + 2.5*a*a) / 1023.

The divisor is the frequency of the Apple clock in kilohertz, or in other words, 1/1023 is the length of one machine cycle in milliseconds. The formula is valid for accumulator values between 1 and 255. Oddly enough, 255 and zero both give the maximum delay posible. Here are a few representative values:

ACCUMULATOR VALUE	DELAY IN MILLISECONDS
18	1.02
88	20.01
200	100.21
255	162.04

So don't delay a minute - call WAIT now!!

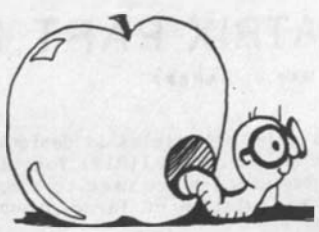
IAC

Fred Wilkinson announces that he's trying to reach out to small user groups nationwide (and worldwide) to get together and communicate regarding the forthcoming formation of the International Apple Core.

If you have questions concerning the International Apple Core's creation or status of current efforts to get going please call Fred Wilkinson at (415)585-2240, or write to:

International Apple Core
P.O.Box 976
Daly City, CA 94017

Now is the time to get involved.



$ARRAYS IN INT

BY JIM DOTY

There is a simple way to get around the lack of string array capability in APPLE's Integer Basic. You can convert the character to a number and then pack two characters in one integer value. Consequently, an integer array dimensioned as A(1000) can hold up to 2000 characters.

The ASC Function will return a number value for a character. This value must be offset since these values are too large to use. I used an offset of -159. This made the character "blank" equal to 1. Consequently, to initialize your integer array you should set each element equal to 101. Why 101 will become obvious shortly (I hope).

Two characters are stored in each integer element. The following example will demonstrate how this is done. Let's use "at" as the two characters to be packed into A(1).

```
ASC("A") = 193.  LESS 159 = 34.
ASC("T") = 212.  LESS 159 = 53.
```

Multiply the left character value by 100 and add the result to the right character value.

```
34 * 100 + 53 = 3453
Therefore A(1) = 3453
```

However, getting the characters back from their integer form is not quite as easy since the inverse of ASC does not exist in integer basic on the APPLE. Here is the needed routine:

First you must break apart the two character values which are combined in the integer array element. Let's use the same value as in the example:

```
A(1) = 3453
LEFT CHAR VALUE = (A(1)/100)+159
                = 193
RIGHT CHAR VALUE= A(1) MOD 100 + 159
                = 212
```

Now all we have to do is convert these values to their ASCII equivalent characters.

You use a string variable which must be the first variable defined in the program. Consequently, you know exactly where in memory this variable resides. To convert the numeric value to a character, just POKE its value into location 2053. For example, if A$ was the first variable defined, the following example would work:

```
POKE 2053, 193
NOW A$ = A
```

This seems to work very well and solves a frustrating problem.

PUT CHARACTER IN PROGRAMS

BY BRUCE TOGNAZZINI

Apple Basic CHR$ Subroutine Function

```
1 CHS = CHR + 128 * (CHR<128)
2 LC1 = PEEK (224): LC2 = PEEK (225) -
  (LC1>243): POKE 79 + LC1 - 256 *
  (Lc2>127) + (LC2 - 255 * (LC2>127))
  * 256,CHS: CHR$ = "A": RETURN
```

Apple Basic CHR$ Matrix Function

```
1 DIM CHR$(128): FOR CHR = 1 TO 128:
  CHR$ = CHR + 128 * (CHR < 128)
2 LC1 = PEEK (224): LC2 = PEEK (225) -
  LC1>243): POKE 84 + LC1 - 256 * (LC2
  >127) + (LC2 - 255 * (LC2>127)) *
  256,CHS: CHR$ (CHR) = "A": NEXT CHR
```

Above are two character string functions for Apple Basic. The CHR$ Subroutine function is ideal where few characters need be called; the CHR$ Matrix function is great for many characters, but takes quite a long time to initialize (approx. 1.5 seconds).

Let's say you wanted to print the following during your program: "How do you say "Apple" in French?". Here is a sample program for the CHR$ Subroutine function:

```
10 CHR =34: GOSUB 1: PRINT "HOW DO YOU
   SAY";CHR$;"APPLE";CHR$;"IN FRENCH?"
```

To do the same thing using the CHR$ Matrix function, you would do the following, having ALREADY executed lines 1 and 2 to initialize the CHR$ matrix:

```
10 PRINT "HOW DO YOU SAY";CHR$(34,34)
;"APPLE";CHR$(34,34);"IN FRENCH?"
```

It is important to note that the Subroutine function is, in fact, a subroutine, whereas the Matrix function is not. Because the matrix is initialized once and the program lines are not called again (because the data exists in the variable table) it is not necessary to separate the lines into a subroutine.

VAL (X$) USE

One use for the VAL(X$) function in Integer Basic:

```
10 V=0: FOR VL=1 TO LEN (V$): V=V+
   (ASC (V$(VL,VL)) - 176)* 10LEN
   (V$)) - VL: NEXT VL: RETURN
```

Going into the subroutine, V$ will be a number, for example, V$ = "1234" - coming out of the subroutine, V will equal 1234. One use for this is to input using a string function and later convert, so that if the user hits return with no input, you can give him or her additional instructions, or just go on with the program, rather than getting the friendly "?"

INTEGERS TIMES A FRACTION

BY PAUL WYMAN

There are numerous business applications where integer BASIC addition and subtraction are quite satisfactory.... until you have to multiply by a fraction or percentage. Preparations of income tax returns is a relevant example. While Applesoft can handle the fractions it chews up vital memory data space at more than twice the rate of integer BASIC. Furthermore, most calculations could be done as integer arithmetic if with a few handy floating point functions in BASIC. The following function is useful only for cases requiring multiplication of a whole number times a decimal fraction. It is not suitable for a chain of successive multiplications due to rapid accumulation of roundable errors. Given these caveats, the reader should be pleased to note that numerous business calculations are relevant such as worksheet spreads by percentages, financial ratio analysis, cost variance analysis, price markups, tax calculations, interpolations in rate tables, etc.

```
10 GOTO 1000
20 REM BASIC SUBROUTINE FOR
   MULTIPLYING TIME WHOLE NUMBERS
25 IN = 0: DE = 0 : THO = 1000
30 FOR Z1 = 1 TO 3: Z2 = 101:
   Z3 = Z2/10
40 Z4 = THO/Z3: Z5 = (PCT MOD Z2)/Z3
50 IN = IN + Z5 * (NUM/Z4)
60 DE = DE + (Z5 * (NUM MOD Z4) * Z3)
70 NEXT Z1: IN = IN + DE/THO
80 DE = DE - (DE/THO) * THO
82 RETURN
85 PCT = 125: REM PCT = 12.5
88 NUM = 5000
90 GOSUB 20
95 PRINT NUM;"*";PCT/101;".";(PCT
   MOD 10);"=";IN;".";DE
98 END
```

IRS AUDIT

MY COMPUTER MAY BE SMALL

BUT IT DOESN'T MAKE

MISTAKES EITHER

PASCAL

PASCAL WHO?

For those of you that are wondering "Pascal who?", Pascal was a French mathematician and also is a very nice modern programming language. A direct descendant of ALGOL 60, Pascal was designed by Niklaus Wirth in the early 70's. It is a clean and elegant language featuring a full set of structured flow of control constructs and superb data structuring capabilities. It is a "strongly typed" language which means that the compiler can catch most of your errors for you. From its origins in academia it is now enjoying increasing popularity in the business and hobbyist sectors as well.

The folks responsible for the APPLE implementation of Pascal (and many others, too) are the Institute for Information Systems at UC San Diego (led by Ken Bowles), a bunch of proverbial "good guys" if there ever was one. They are dedicated to spreading good programming languages throughout the real world at minimum cost. They are very involved in computer-aided instruction, too, so when you obtain your Pascal system you will also get a wonderful program that will teach you the language and how to use it.

COPYING THE BASICS DISK

BY GENE WILSON

Fellow Core member Phil Bernheim relates that his new Pascal/Language System gave him fits when he tried to copy the 'Basics' Disk with a 'Basic' copy program. He called Apple's hot line and got the cryptic answer:
The machine boots up in Pascal. The program on the 'Basics' disk is a Pascal program and must be copied using the'F'(filer) portion of your Pascal System.

Any attempt to copy the 'Basics' disk from Basic will simply be futile. (Did that sound like a challenge?--of course it was. Some Core member has probably already found a sneaky way.)

Our thanks to Phil--his discovery will certainly aid another member in the near future, and doubtless other little tid-bits are just lurking around the corner waiting to be 'flushed' into the open. Send the Cider Press Staff your latest suggestions, tips, traps, etc., and we'll get the whole thing out to the membership.

SINGLE DRIVE

BY GENE WILSON

If you've recently bought Apple's Language System and haven't gotten up and running yet--keep reading. It has probably occurred to you that every place you saw Pascal being demonstrated was done on a 'two drive' system. 'NO SWEAT' you've heard when asked if it works with the one drive you have at home. (If it's so easy, why don't you have it running?)

In mid-October I visited no less than five Apple selling shops, and in all five two drives were in use for each Pascal demo--in no cases were the staff and innocent bystanders able to run the GRAFDEMO program after unplugging one of the drives. It occurred to me that possibly the Reference Manual would clear this up, but among the fantastic information there was nothing that helped.

Cheer up. I know of two solutions; the first being to fork over another five hundred dollars and flaunt a second drive in your neighbor's face. This may place your body out in the street when the wife gets the checkbook back.

For the second method read on:

I will assume that the directions for installation of the card and chips were properly followed. Read the chapters in the Reference Manual covering the Filer and the Editor. Section 6.2 on Starting up an Apple Pascal System that has only one drive must be understood. The most important thing to understand is that with one drive you must switch diskettes 20 times to make a copy. (This tends to 'drive' a person towards solution number 1, above). Go back to your trusting computer store and use their two drives to 'fast copy' an APPLE3, and several APPLE0's (reason later).

Now it's time to get a sample program, so power up per section 6.2.5. Start with APPLE3 in your drive, power up, go to APPLE0, press RESET, and 'WELCOME' message tells all that you can read instructions.
Now we go to the 'F'(filer) for some preliminary play. Go on ahead and type 'F'. Prompt line tells story of success. Make sure that SYSTEM.WRK file is gone elsewhere. (If you wanted that file it was transferred earlier anyway.) Type 'N' (for new) and prompt says 'THROW AWAY CURRENT FILE?'. Type 'Y' and 'WORKFILE CLEARED' tells us that the dirty deed is done.
Let's take a look at the goodies on APPLE3, by placing APPLE3 in the drive, and typing 'L' (for list).
Prompt then says 'DIR LISTING OF?'
Type '#4' <RET>

```
'APPLE3:
SYSTEM.APPLE    32 26-JUL-79
FORMATTER.CODE   4  4-MAY-79
FORMATTER.DATA   6 22-JUN-79
.......'
```

The program we want is 'GRAFDEMO.TEXT.' Type <RET> get all the things off the screen except the command prompts. Typing 'T' produces 'TRANSFER?'. Respond with 'APPLE3:GRAFDEMO.TEXT' <RET>. 'TO WHERE?', answer with 'APPLE0:SYSTEM.WRK.TEXT' <RET>. The computer will now read the requested program, and will prompt 'PUT IN APPLE0: TYPE <SPACE> TO CONTINUE'
Do it!
If you hadn't cleared the old SYSTEM.WRK file the system gives one last chance to save last night's masterpiece by prompting with 'REMOVE OLD APPLE0:SYSTEM.WRK.TEXT?'. If this occurs type 'Y', and get 'APPLE3:GRAFDEMO.TEXT'. If you had done everything correctly your prompt now says '-->APPLE0:SYSTEM.WRK.TEXT'.
Program we want is now where we want it.
Get out of (filer) by typing 'Q'. Get into (editor) with 'E'. Prompt says '>EDIT...
NO WORKFILE IS PRESENT.FILE?(..'
Type 'SYSTEM.WRK' <RET>.
Now it is possible to look over the program at leisure, and in great detail. This example was picked for the author BILL ATKINSON's special knowledge of the Apple II's graphic capabilities. Bill puts the Apple through its paces.

Remember the commands in the Editor? For some special features type '9P' and go to PROCEDURE STUFF, which bitmaps the butterfly demo. See if you can figure it out (If you can write it up we'll put the answer in print).

Oh, we can't go this far and give up.
Type 'Q' (to Quit the Editor). If you've made changes be sure to 'Update' otherwise type 'E' (to Exit) the Editor. Type 'R' (to Run). Apple prompts with 'COMPILING...', 'COMPILE WHAT TEXT?'. Type 'SYSTEM.WRK' <RET>. Apple prompts 'TO WHAT CODEFILE?'. Type 'SYSTEM.WRK'<RET>.
The Compiler will now have a field day, and will eventually run the program GRAFDEMO.

So why did we go such a long way to do so little? It would have been easier to simply place GRAFDEMO upon APPLE0: then run it. You would also have probably put together a program using the Editor which would have crashed due to a lack of available storage on APPLE0. Removing old programs from APPLE0 will certainly become a way of life.
This is why I recommended that several copies of APPLE0 should be made. We can label this diskette as APPLE0:GRAFDEMO. The program can be readily shown by booting up the system and Running by typing 'R'. I don't know of any other way to keep 3 or 4 programs around that can be quickly run. I keep a half dozen diskettes with different programs around. (Surely some one out there knows a better way?). Similar games can be played to get the Assembler up and running, but that's the subject of anybody else's article. APPLE3 also contains program DISKIO which will help with disk use and setting up files in Pascal.

As a practical matter I've thrown the 'PASCAL User Manual and Report' (supplied with System) into a dark corner, and in its place I recommend 'A Practical Introduction to PASCAL' by I.R. Wilson (no relation), available at Computerlands. The language is explained in better detail, and seems to put more emphasis on examples. There is certainly nothing wrong with the supplied 'Problem Solving Using PASCAL', and the 'Apple Pascal Reference Manual' is fantastic stuff, but much more is needed.

In conclusion, Apple Pascal with a single drive is limiting, but still workable. With two drives the system is not only workable, it's a dream, so if you're into 'REAL' systems, Apple Computer will be glad to sell you another drive!

PASCAL LOWER CASE

WITH DAN PAYMAR'S CHIP

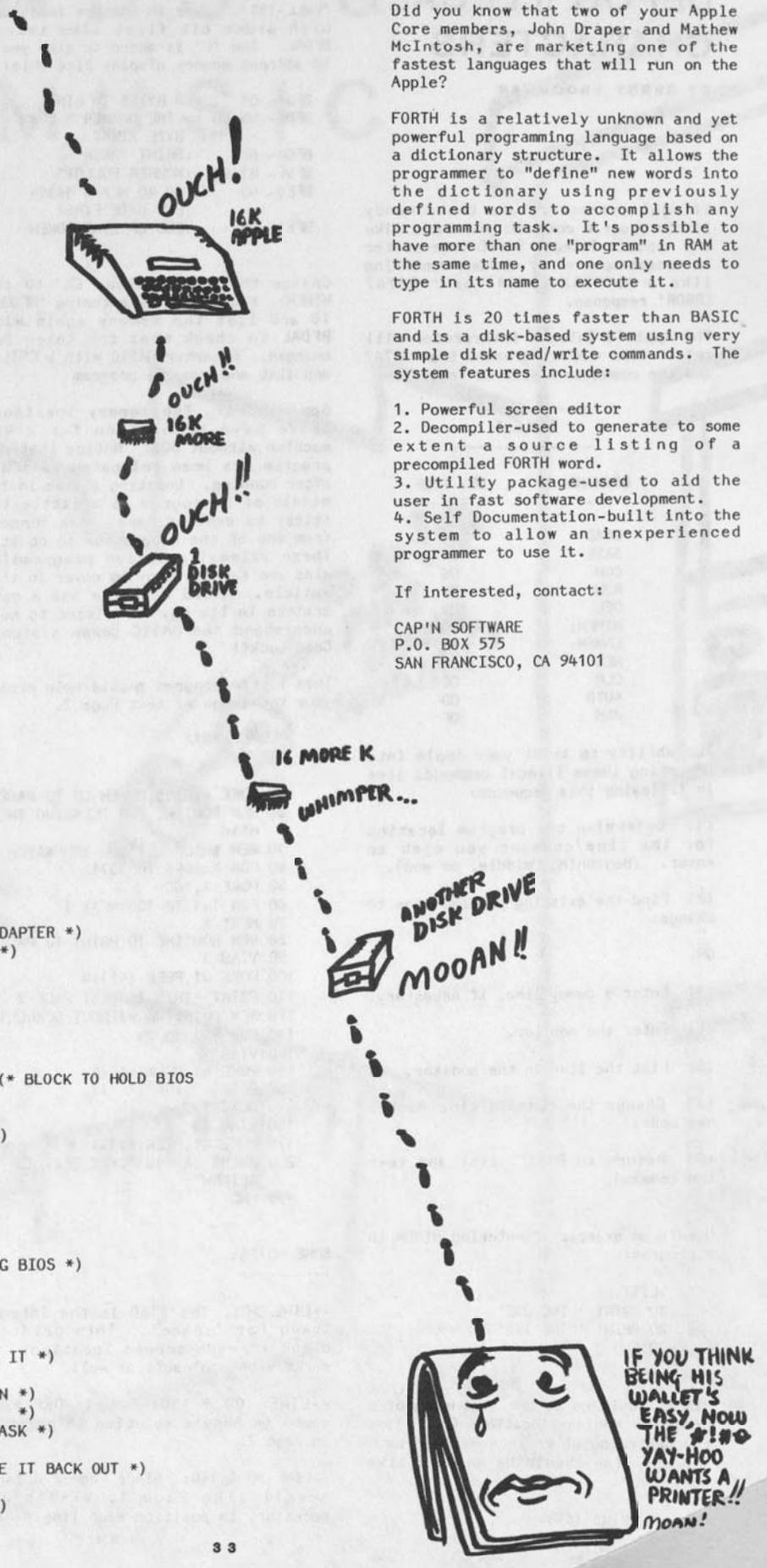
BY CRAIG VAUGHAN

Here is a program to modify BIOS to work with Dan Paymar's lower case adapter. It has not been prepared to download.

ENJOY!

Craig

```
(* PROGRAM TO MODIFY THE BIOS MODULES *)
(* TO WORK WITH DAN PAYMAR'S LOWER CASE ADAPTER *)
(* DEVELOPED AND TESTED BY CRAIG VAUGHAN *)
(* COPYRIGHT WAIVED *)

PROGRAM UPDATE;

VAR

   BLK: PACKED ARRAY(0..511) OF 0..255; (* BLOCK TO HOLD BIOS
   CODE *)
   BLT, (* # OF BLKS TRANSFERRED *)
   BLN: INTEGER; (* BLOCK # TO READ <4> *)
   S :  FILE; (* FILE ID *)
   SRC :  STRING; (* FILENAME *)


BEGIN

   SRC:="SYSTEM.APPLE"; (* FILE CONTAINING BIOS *)

   RESET (S,SRC); (* OPEN IT *)

   BLN := 4; (* READ BLOCK #4 *)
   BLT := BLOCKREAD(S,BLK,1,BLN); (* READ IT *)

   BLK(232) := 234; (* NOP CASE CONVERSION *)
   BLK(233) :  234; (*  "   "       "      *)
   BLK(235) := 127; (* CHANGE CHARACTER MASK *)

   BLT := BLOCKREAD(S,BLK,I,BLN); (* WRITE IT BACK OUT *)

   CLOSE (S,LOCK); (* CLOSE AND LOCK IT *)

END.  (* ALL DONE *)
```



FORTH

Did you know that two of your Apple Core members, John Draper and Mathew McIntosh, are marketing one of the fastest languages that will run on the Apple?

FORTH is a relatively unknown and yet powerful programming language based on a dictionary structure. It allows the programmer to "define" new words into the dictionary using previously defined words to accomplish any programming task. It's possible to have more than one "program" in RAM at the same time, and one only needs to type in its name to execute it.

FORTH is 20 times faster than BASIC and is a disk-based system using very simple disk read/write commands. The system features include:

1. Powerful screen editor
2. Decompiler-used to generate to some extent a source listing of a precompiled FORTH word.
3. Utility package-used to aid the user in fast software development.
4. Self Documentation-built into the system to allow an inexperienced programmer to use it.

If interested, contact:

CAP'N SOFTWARE
P.O. BOX 575
SAN FRANCISCO, CA 94101

DEFOG ILLEGAL CHARACTERS

BY GERRY VROOMMAN

Illegal characters are those handy little program commands you would like the Apple's Integer BASIC interpreter to accept upon entry without something like a 'SYNTAX ERROR' or '>32767 ERROR' response.

The Apple's Integer interpreter will reject line numbers greater than 32767 and the commands listed in Table 1.

TABLE 1

COMMAND	HEX CODE
LOAD	04
SAVE	05
CON	06
RUN	07
DEL	09
HIMEM:	10
LOMEM:	11
NEW	0B
CLR	0C
AUTO	0D
MAN	0F

The ability to trick your Apple into accepting these illegal commands lies in following this sequence:

(1) Determine the program location for the line/command you wish to enter. (Beginning, middle, or end).

(2) Find the existing program line to change.

OR,

(3) Enter a dummy line, if necessary.

(4) Enter the monitor.

(5) List the line in the monitor.

(6) Change the command/line number hex code.

(7) Return to BASIC, list and test the command.

Here's an example of entering HIMEM in a program:

```
>LIST
10 PRINT 'LINE ONE'
20 PRINT 'LINE TWO'
30 END
```

HIMEM should go at the beginning of a program. Monitor locations CA.CB list the beginning of an Integer program. A dummy line should be entered like this

```
>5 PRINT 16384
```

Enter the monitor with the command 'CALL-151'. Type in the hex location, high order bit first like this: BFDAL. The 'L' is added to give you a 40 address memory display like this:

```
BFDA- 08    ::8 BYTES IN LINE
BFDB- 05 00 ::LINE NUMBER 5 LOW
              BYTE FIRST
BFDD- 62    ::PRINT TOKEN
BFDE- B1 00 ::NUMBER FOLLOWS
BFE0- 40    ::00 40 HEX = 16384
              LOW BYTE FIRST
BFE1- 01    ::END OF LINE TOKEN
```

Change the print token '62' to the HIMEM: token '10' by entering 'BFDD: 10 and list the memory again with BFDAL to check that the token has changed. Re-enter BASIC with a CNTL-C and list and run the program

Some notes: The memory locations above have been given for a 48K machine without DOS. Notice that the program has been relocated to 3FD5' after running. Locating a line in the middle of a program is a little too tricky to explain here. Ask someone from one of the stores how to do it. These illegal creative programming aids are far too many to cover in this article. Micro Magazine has a good article in its May 1979 issue to help understand the BASIC token system. Good Luck!!

This little program should help expand your knowledge of text Page 2.

```
>LOMEM:3072
>TEXT
>LIST
 10 POKE -16299,0:REM GO TO PAGE 2
 20 REM ROUTINE FOR CLEARING THE
    HASH
 30 REM WHILE LETTING YOU WATCH
 40 FOR X=2048 TO 3071
 50 POKE X,160
 60 FOR I=1 TO 100:NEXT I
 70 NEXT X
 80 REM ROUTINE TO PRINT TO PAGE 2
 90 VTAB 1
100 POKE 41,PEEK (41)+4
110 PRINT 'THIS IS TEXT PAGE 2
120 REM PRINTING WITHOUT SCROLLING
130 FOR Y=2 TO 23
140 VTAB Y
150 POKE 41,PEEK(41)+4
160 PRINT 'LINE';' '+Y
170 NEXT Y
180 VTAB 23
190 POKE 41,PEEK(41)+4
200 PRINT TAB(10)'TYPE TEXT TO
    RETURN'
999 END
```

SOME NOTES:

--LINE 50: The #160 is the Integer token for 'space'. This prints a blank in each screen location. It works with Applesoft as well.

--LINE 100 & 150: This POKE,PEEK combo is Andy's solution to printing on Page 2.

--LINE 90 & 140: Since Page 2 doesn't scroll like Page 1, VTAB's are necessary to position each line.

--The LOMEM:3072 is necessary to move the Integer variable storage above the Page 2 area so you can write to it.

SETTING REGISTERS

BY JOHN ARKLEY

A CALL instruction in BASIC gets you to a machine language subroutine, but the various registers in the 6502 processor are up to fate - here's how to set them.

Any machine program may be called by POKING the values into the addresses shown below & the program address into PC and PC+1 (low order first) and then enter it via a CALL. This is the same as a GO command in monitor mode.

The following program demonstrates this by using the monitor subroutine PRBYTE that prints the A register as two hex bytes at the next position on the screen.

Your subroutine must be sure that decimal mode is off upon return to BASIC else it will blow up by going into a loop printing on the screen and beeping. This bit of the P register may be cleared before doing the CALL by poking P with a zero or the desired value for the P register insuring that bit 3 is off.

BIT MEANINGS

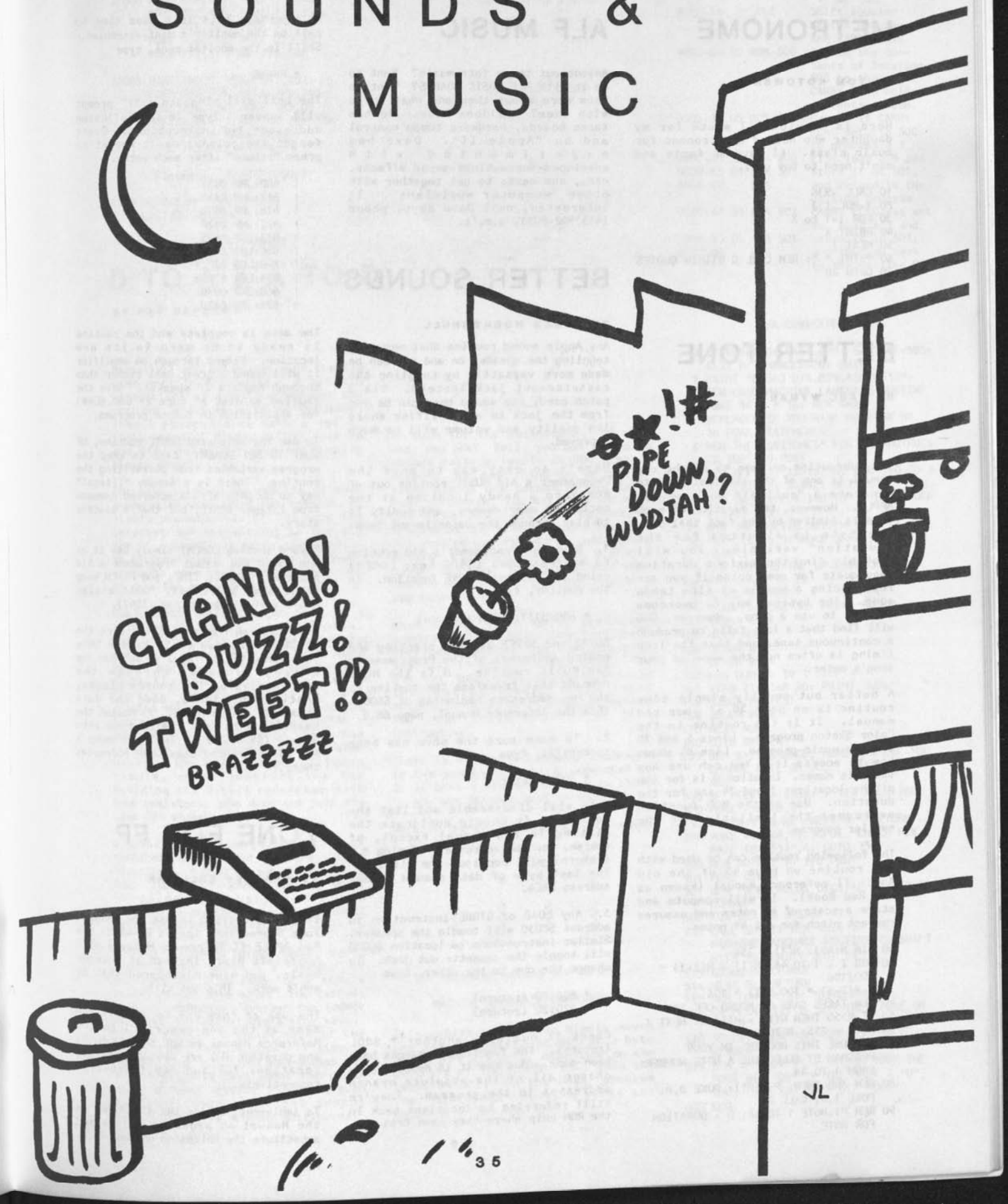
```
 7 6 5 4 3 2 1 0
 N O   B D I Z C
MSB           LSB
```

	BIT	MEANING
MSB	7	Negative result
	6	Overflow Bit
	5	Unused
	4	Decimal mode on
	3	Break inst
	2	Interrupt disable
	1	Zero result bit
LSB	0	Carry bit

PROGRAM

```
100 A=69:  X=70:  Y=71:  P=72
200 PC=58:  GO= -327
300 REM APPLESOFT GO = 65209
350 REM MONITOR ADRS = $FEB9
400 POKE PC,218
450 REM POKE PC, $FDDA
500 POKE PC + 1,253
600 HOME
700 FOR L=0 TO 15
750 FOR I=0 TO 15
800 V=L * 16 + I
820 POKE A,V
830 POKE P,0
840 CALL GO
850 NEXT:  PRINT:  NEXT
900 END
```

SOUNDS & MUSIC



SOUND AND MUSIC

METRONOME

BY TOM KOTOWSK

Here is a quickie I wrote for my daughter who needed a metronome for music class. (I have an Apple and don't need to buy one).

```
10 CALL -936
20 X=PDL (1)
30 FOR I=1 to X
40 PRINT X
50 NEXT I
60 PRINT " ": REM CTRL G WITHIN QUOTES
70 GOTO 20
```

BETTER TONE

BY PAUL WYMAN

The subroutine on page 45 of the red manual is one of the simplest ways to generate a "musical" note on the APPLE. However, the duration of each note is limited by the fact that only one byte is allotted for the "duration" variable. You will probably find the maximum duration inadequate for some notes if you are reproducing a medium or slow tempo song. The typical way to overcome this is to use a loop. However, you will find that a loop fails to produce a continuous tone, and that the loop timing is often not the same as your song's meter.

A better but equally simple tone routine is on page 58 of your red manual. It is the routine in the Color Sketch program. Lines 5 and 10 are the music program. Line 85 shows how to access it. You can use any variable names. Location 0 is for the pitch; locations 1 and 24 are for the duration. Use of the MOD function overcomes the limitation of the earlier program.

The following routine can be used with the routine on page 45 of the old Apple II reference manual (known as the Red Book). It will compute and store a scale of 48 notes and assures correct pitch for all 48 notes.

```
10 DIM N(49): N(1) = 256
20 FOR I = 1 TO 48: N(I) = N(I+1) *
   100/106
30 R=N(I-1) * 100-N(I) * 106
40 REM TAKES CARE OF ROUND OFF ERR
50 IF R>50 THEN N(I) = N(I)+1: NEXT I
60 N(1) = 255: RETURN
70 REM USE THIS ROUTINE IN YOUR
   PROGRAM BY SELECTING A NOTE NUMBER
   FROM 1 TO 48
80 REM AND THEN: P=N(P1): POKE 0,P:
   POKE 1,D; CALL 2
90 REM P1=NOTE 1 TO 48; D = DURATION
   FOR NOTE
```

ALF MUSIC

Anyone out there into music? Want to do it with ALF MUSIC BOARDS? Want to know more about them and what to do with them? So does Dave. He has three boards, hardware tempo control and an "Apple II". Dave has experimented with envelopes-subroutines-sound effects, etc., and wants to get together with other 'computer musicians'. If interested, call Dave days, phone (415)992-9051, a.m.'s.

BETTER SOUNDS

BY JAMES HOCKENHULL

Any Apple sound routine that works by toggling the speaker on and off can be made more versatile by toggling the cassette-out jack instead. Via a patch cord, the sound then can be run from the jack to an amplifier where its quality and volume will be much improved.

Here's an easy way to move the Programmer's Aid MUSIC routine out of ROM into a handy location at the bottom of user memory, and modify it to play through the cassette-out jack:

1. Move the Programmer's Aid routine to address 2048 ($800 hex; LOMEM) using the monitor's MOVE function. In the monitor, type

```
* 800<D717.D7FBM (return)
```

$D717 and $D7FB are the starting and ending addresses of the Programmer's Aid MUSIC routine. M is the MOVE command that transfers the routine to the hex addresses beginning at $800. (See the Reference Manual, page 68.)

2. To make sure the move has been successful, type

```
* 800L (return)
```

This will disassemble and list the routine. It should duplicate the listing in the manual except, of course, for the addresses. Typing * L (return) will continue the listing. The last byte of data should occupy address $8E4.

3. Any LOAD or STORE instruction to address $C030 will toggle the speaker. Similar instructions to location $C020 will toggle the cassette out jack. To change the one to the other, type

```
* 806:20 (return)
* 820:20 (return)
```

Check it out with another * 800L (return). The functional changes have been made. But now it is necessary to change all of the absolute branch addresses in the program. They're still referring to locations back in the ROM chip where they came from.

4. Perhaps this is a good time to call on the monitor's mini-assembler. Still in the monitor mode, type

```
* F666G
```

The bell will ring and a "!" prompt will appear. Type in the following addresses and instructions. Don't forget the colons; don't forget to press "return" after each entry.

```
! 800:JMP 0837
! 812:JMP 0815
! 818:JMP 081B
! 82C:JMP 082F
! 832:JMP 0835
! 83C:LDA 87F,Y
! 84B:LDA 87F,Y
! 854:LDA 87F,Y
! 86D:JMP 0870
! 87A:JMP 087D
```

The move is complete and the routine is ready to be used in its new location. Played through an amplifier it will sound a great deal richer than through Apple's 2" speaker. Save the routine on disk or tape (* 800.8E4W) for application in future programs.

To use the relocated MUSIC routine, BE SURE TO SET LOMEM: 2376 to keep the program variables from overwriting the routine. There is a sneaky "illegal" way to do this with a deferred command from integer BASIC, but that's another story.

Beyond setting LOMEM, simply use it as you would the normal Programmer's Aid program. TIMBRE, TIME, and PITCH keep their same addresses. Just assign MUSIC = 2048 rather than -10473.

(Two tag-end notes: 1) I'm sure the RELOCATE program could handle this just fine, but so far I've not had the energy to weave through the intricacies of its source blocks, destination blocks, code and data segments, and so forth. 2) Resist the temptation to move the routine into page 3 ($300-$3FF). The Programmer's Aid HIRES program uses some addresses there.)

TONE FOR FP

BY CHARLES SULLIVAN

If you have tried to use the "Simple Tone Subroutine" (pages 43-45 in the Red APPLE II Reference Manual) with Applesoft Basic instead of Integer Basic, you have discovered that it won't work. This one will.

The use of this one is exactly the same as the one described in the Reference Manual except the pitch (P) and duration (D) are passed in memory locations 768 and 769 (decimal), respectively.

To implement, follow the directions in the Manual on pages 43 and 44 and substitute the following on page 45.

TONE FOR FP (CONT.)

```
32000 POKE 770,173: POKE 771,48:
      POKE 772,192: POKE 773,136:
      POKE 774,208: POKE 775,5:
      POKE 776,206: POKE 777,1:
      POKE 778,3: POKE 779,240

32005 POKE 780,9: POKE 781,202:
      POKE 782,208: POKE 783,245:
      POKE 784,174: POKE 785,0:
      POKE 786,3: POKE 787,76:
      POKE 788,2: POKE 789,3:
      POKE 790,96: RETURN
```

Figure 2. BASIC "POKES"

```
25 POKE 768,P: POKE 769,0:
   CALL 770: RETURN
```

Figure 3. GOSUB

D TO A & A TO D

BY PAT CAFFREY

Recently I got fascinated with the I/O game port in the Apple. I built a wire extension cable and brought the port out of the Apple to my workbench. Then I proceeded to make a set of light emitting diodes (LEDs) turn on and off, pulse and even to display binary counts.

The very interesting results of that exploration were worth the trouble. I highly recommend it if you have the interest and are willing to buy the few IC sockets and wires necessary. No external power supply was required, although some care must be taken not to draw too much current.

During this period I got hold of a copy of the Apple Organ. Again, fascination. The sounds produced by this program are more realistic than what I had heard before on the Apple. Normally the Apple music has a "brassy" tone because it is all created out of square waves. What the Apple Organ does is to use a DIGITAL - TO - ANALOG converter to shape the waveforms from square waves into sine waves. The Apple Organ includes simple, clear instructions for building the D-to-A converter with four resistors, some wire and just the game I/O socket.

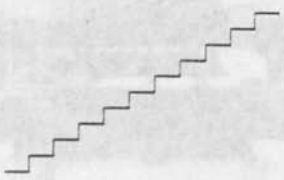
I searched through my pile of resistors and did not find the values specified. I made up some from what I had. The results were adequate, about 30 percent distortion of the waveshapes produced with the actual Apple Organ resistor values.

Drawing A shows a normal square wave, B normal sine waves, C a "ladder" of voltages and D an image of a single reconstructed sine wave. The fact is that the components average the ladder voltages and approximate a sine wave. At the machine language rate of the Apple Organ there is plenty of time to construct up to 20,000 waveshapes per second which is just about the upper range of human hearing. There is even time left over to select tones, timbre, and other musical attributes.

A

B

C



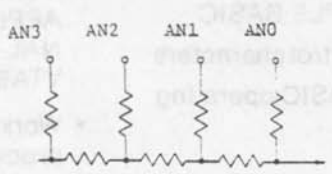
D

So much for the background. Of what use, you ask? Well, you don't need to change voltage levels 20,000 times a second. The same thing may be done at intervals of one second or one day. And, of course, you may stop and hold the rise and fall of the output voltage at any of the ladder levels. You acquire the ability to control the outside world in more than a simple on-off fashion. What you do with that control is a matter of need, ingenuity and time.

What I have done is to provide a program which does the functions necessary to make the voltages rise and fall at predictable rates. You may try such applications as seem interesting (model train speed control), necessary (stepper motors for your solar heater) or just instructive.

There is a machine language component to the program. I have commented on it as best I can and hope it is clear to you how it does what it does.

Here is the resistor schematic I used.



All resistor values are 1K ohm.

MACHINE LANGUAGE COMPONENT

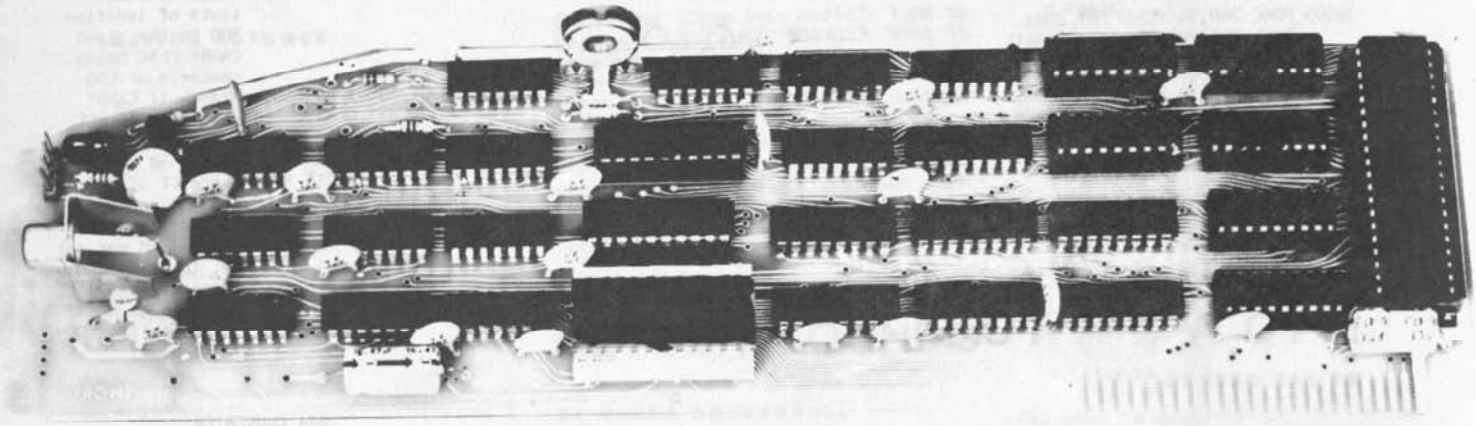
This entire subroutine simply moves the least significant bit of a byte and stores a 1 or 0, depending on the contents of the CARRY FLAG, into the location $01. In the BASIC program the "BITCHANGE" routine PEEKs the contents of $01 back out.

```
0000-                  Holds number to
                       be converted
0001-                  Holds resulting
                       "1" or "0"
0002-18      CLC       Shift routine
                       begins by clear-
                       ing CARRY FLAG
0003-66 00 ROR $00     Rotate the con-
                       tents of location
                       $00 to the right
                       CARRY FLAG holds
                       contents of LSB.
0005-B0 05 BCS $05     Branch if CARRY
                       FLAG set, to $0C
                       to process a 1
0007-A9 00 LDA #00     Else load a 0 and
0009-85 01 STA $01     store it at $01,
000B-60    RTS         and return to the
                       calling program
000C-A9 01 LDA #01     CARRY FLAG was set
                       so, load a 1 and
000E-85 01 STA $01     store it in $01,
0010-60    RTS         and then go back
                       to the calling PGM
```

D/A CONVERTER

```
 0 DRIVER=100: WAIT=299: BITTEST=500:
   A=1: BITCHANGE=520: SHIFT=2
 1 PRINT "BLOAD D/A.BIN.A$0.L$10":
   REM LOAD MACHINE LANGUAGE ROUTINE
 2 POKE 16,6 * 16: REM THIS IS
   REPLACED IF YOU HAVE NO DISK BY
   16 POKE STATEMENTS
 6 REM THE "SWITCHES" FOR THE OUTPUTS
   IN THE I/O PORT
10 DIM OFF(3): X=-16296: FOR I=0 TO 3
   : OFF(I)=X: X=X+2: NEXT I
20 DIM ON(3): FOR I=0 TO 3: ON(I)=OFF
   (I)+1: NEXT I
25 REM INITIALIZE SWITCHES "OFF"
30 FOR I=0 TO 3: POKE OFF(I),0 =
   NEXT I
90 REM THIS IS THE UP-DOWN DRIVER
100 FOR NUM=0 TO 15
110 GOSUB BITTEST
111 VTAB 21: TAB 20: PRINT NUM;"     "
115 GOSUB WAIT
120 NEXT NUM
130 GOSUB WAIT
132 GOSUB WAIT
140 FOR NUM=14 TO 1 STEP -1
141 VTAB 21: TAB 20: PRINT NUM;"     "
150 GOSUB BITTEST
160 GOSUB WAIT
170 NEXT NUM
180 GOTO DRIVER
299 REM WAIT-CHANGE, HERE'S WHERE YOU
    CHANGE RATE OR JUST  STAY AS YOU
    ARE
300 IF PEEK (-16384) > 127 THEN 360
320 POKE -16368,0
350 FOR N=1 TO A:  NEXT N:  RETURN
360 POKE -16368,0: INPUT "ENTER NEW
    RATE (0-255)",A: GOTO 350
500 REM OUTPUT TO I/O PORT
501 POKE 0,NUM
501 REM "NUM" HOLDS DIGITAL NUMBER TO
    CONVERT
505 FOR D=3 TO 0 STEP -1
510 CALL SHIFT: REM THIS IS THE
    MACHINE LANGUAGE ROUTINE TO SHIFT
    BITS TO MEMORY SLOT "1"
515 GOSUB BITCHANGE
516 NEXT D:  RETURN
519 REM CONVERSION "BITCHANGE"
520 C= PEEK (1): IF C=1 THEN POKE ON
    (D),0
530 IF C#1 THEN POKE OFF(D), 0:
    RETURN
531 REM IF SLOT"1" = 1 THEN TURN THE
    PORT BIT "ON" IF THE SLOT = "0"
    THEN TURN THE PORT BIT "OFF"
600 END
```

SUP'R'TERMINAL™



SUP'R'TERMINAL IS AN 80 COLUMN BY 24 LINE PLUG-IN COMPATIBLE BOARD FOR THE APPLE II COMPUTER

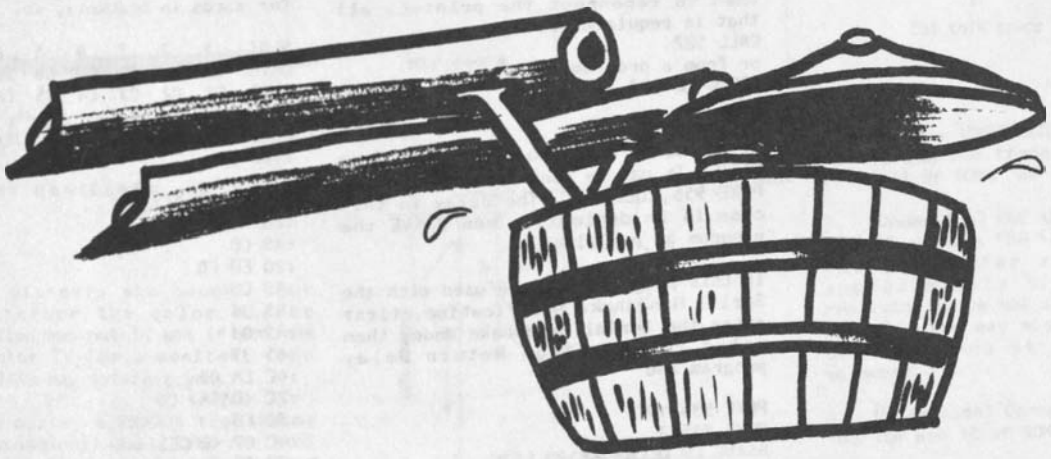
SPECIFICATIONS & FEATURES

- 80 Columns by 24 lines, upper and lower case; all 128 ASCII characters.
- Upper and Lower case data entry using the APPLE II keyboard.
- Includes an Upper and Lower case 5x8 dot matrix ASCII character set, and inverse alpha characters.
- Expands existing keyboard for more ASCII characters
- Character set can be user definable
- Includes VBC™ (video balance circuit) which enables the use of displaying 80 columns on an inexpensive 8 MHz CRT monitor
- Works with LEEDEX monitor (version 2.2) and other inexpensive CRT monitors
- Shift Lock Feature
- KEYPRESS function for PASCAL programs supplied
- Works with APPLE PASCAL and APPLE BASIC
- Incorporates PASCAL and BASIC control characters
- Follows protocols of PASCAL and BASIC operating systems
- ALL monitor-type escapes are valid
- Compatible with ALL APPLE II peripherals.
- Effective baud rate greater than 10,000; fast scrolling and clearing
- Can be used with APPLE II communication interface board to act as self contained terminal for time-sharing or other applications. Terminal program supplied when used with a D.C. Hayes micromodem.
- 3K bytes of bank switched static ram
- 2K bytes of ROM
- The only board with continuous direct memory mapped screened ram.
- The only board that interprets VTABS by firmware (version 2.2)
- The only board with an adjustable scrolling window.
- The only 80 column board that is synchronous with the APPLE II
- Fully programmable cursor
- Conversion program supplied to modify existing APPLESOFT programs to work with SUP'R'TERMINAL (automatically converts HOME,CALL-936 and VTABS) (version 1.0)
- Works with the new Easywriter and APPLE PI word processors.
- Uses less current on the +5V supply than any other 80 column board
- Works with CORVIS hard disc system

PATENT PENDING

M&R ENTERPRISES
P.O. BOX 61011, Sunnyvale, CA 94088

MODIFICATIONS



PRINTER MODIFICATIONS

CR DELAY

FROM APPLE COMPUTER CO.

This program inserts a variable delay after a carriage return to allow time for the printhead to get back to the left margin without missing characters. DOS 3.2 is required to use this modification.

SOFTWARE ENTRY

First you must decide which slot the interface will go in and enter the delay program. The program is customized for this slot number and won't work if used with a different slot. Enter the program using the values from the table for words in brackets, <>.

SLOT	1	2	3	4	5	6	7
CODE	C1	C2	C3	C4	C5	C6	C7

	PARALLEL	COM CARD	SERIAL
TYPE	02	05	07

DELAY. The delay is measured in tenths of a second and entered in hex. Hence 5 is 1/2 second and A is one second. The usual default value is 5.

Enter the monitor with call-151 and type:

```
390:A9 <SLOT>
:20 95 FE
:A9 80
:20 ED FD
:A9 A5
:85 36
:A9 03
:85 37
:4C EA 03
:20 <TYPE> <CODE>
:C9 8D
:D0 0E
:AD BB 03
:48
:A9 C2
:20 A8 FC
:68
:E9 01
:D0 F5
:60
:<DELAY>
```

To check your typing. type (390L 3BB) and compare your listing to the one below for slot 1 and a Parallel Printer Interface.

```
0390-  A9 01      LDS   #$01
0392-  20 95 FE   JSR   $FE95
0395-  A9 80      LDA   #$80
0397-  20 ED FD   JSR   $FDED
039A-  A9 A5      LDA   #$A5
039C-  85 36      STA   $36
039E-  A9 03      LDA   #$03
03A0-  85 37      STA   $37
03A2-  4C EA 03   JMP   $03EA
03A5-  20 02 C1   JSR   $C102
03A8-  C9 8D      CMP   #$8D
03AA-  D0 0E      BME   03BA
03AC-  AD BB 03   LDA   $03BB
03AF-  48         PHA
03B0-  A9 C2      LDA   #$C2
03B2-  20 A8 FC   JSR   $FCA8
03B5-  68         PLA
03B6-  E9 01      SBC   #$01
03B8-  D0 F5      BNE   $03AF
03BA-  60         RTS
03BB-  05
```

Now return to basic with (3D0G)

SAVING THE PROGRAM TO DISK:

The program must be in memory before the printer can be used with the delay. Save the program by typing:

BSAVE CR DELAY,A$390,L$2C

USING THE PRINTER:

The first time you want to use the printer you must load the program and initialize the interface. From command mode type:

BLOAD CR DELAY
CALL 912

This may be done from a program by entering:

100 PRINT D$;"BLOAD CR DELAY": CALL 912
(assuming that D$ is a control D).

If you want to switch back to the video monitor for output then type:
PR#0
or from a program enter
200 PR#0

Then to reconnect the printer, all that is required is:
CALL 922
or from a program enter
300 CALL 922

NOTES:
If the delay needs to be adjusted from BASIC, It can be done with the command POKE 955,<DELAY>. The delay in this case is in decimal. Then BSAVE the program as explained.

If this program is to be used with the Serial Handshake Modification, first enter the Serial Handshake mode; then enter the Carriage Return Delay program and

POKE 934,197
POKE 935,3
BSAVE CR DELAY,A$390,L$2C

Use these instructions to turn the printer on and off.

HANDSHAKE

FROM APPLE COMPUTER CO.

The High Speed Serial Interface card cannot run faster than 300 baud on most printers due to a lack of a printer busy line. This modification uses the existing data input line to sense if the printer is busy and inhibit output if necessary. DOS 3.2 is required to use this modification.

* * *WARNING* * *

Damage to the Serial Interface card may not be convered by your warranty. If you aren't sure of the signal levels and pinout of your printer, find out or get someone who knows to help you.

* * *WIRING CHANGES* * *

First you must determine which wire your printer uses to indicate a printer busy or buffer full condition. Your printer's manual should contain this information or contact the manufacturer. Examples:

IDS 125/225	PIN 4 (may be pin5)
HEATH H-14	PIN 4
TI-810	PIN 11
SPINTERM	PIN 19
COMPRINT	PIN 20

The preferred place to do the wiring change is in the cable, but it can also be done at the card or the printer. Disconnect the wire between pin 2 of the printer and pin 2 on the Serial Card. Then connect the wire with the printer busy signal to the wire for pin 2 on the Serial Card.

* * *SOFTWARE PATCH* * *

Next you must decide which slot the interface will go in and type in the software patch. The patch is customized for this slot number and won't work if used with a different configuration. The patch forces the computer to look to see if the printer is busy and wait if it is. Enter the patch using the values from the table for words in brackets, <>.

SLOT	1	2	3	4	5	6	7
DATA	90	A0	B0	C0	D0	E0	F0
CODE	C1	C2	C3	C4	C5	C6	C7

Enter the monitor with CALL -151 and type:

```
3B0:A9 <SLOT>
:20 95 FE
:A9 00
:20 ED FD
:A9 C5
:85 36
:A9 03
:85 37
:4C EA 03
:2C <DATA> C0
:30 FB
:4C 07 <CODE>
:00 00 00
```

To check your typing, type 3B0L -and compare your listing to the one below for slot 1.

```
03B0-  A9 01      LDA   #$01
03B2-  20 95 FE   JSR   $FE95
03B5-  A9 00      LDA   #$00
03B7-  20 ED FD   JSR   $FDED
03BA-  A9 C5      LDA   #$C5
03BC-  85 36      STA   $36
03BE-  A9 03      LDA   #$03
03C0-  85 37      STA   $37
03C2-  4C EA 03   JMP   $03EA
03C5-  2C 90 C0   BIT   $C090
03C8-  30 FB      BMI   $03C5
03CA-  4C 07 C1   JMP   $C107
03CB-  00         BRK
03CC-  00         BRK
03CD-  00         BRK
```

Now return to BASIC with 3D0G.

* * * SAVING PATCH TO DISK* * *

The patch must be in memory before the printer can be used at the higher speeds. Save the patch by typing BSAVE PATCH, A$3B0, L$20.

* * *USING THE PRINTER* * *

The first time you want to use the printer you must load the patch and initialize the interface. From immediate mode type-

BLOAD PATCH

CALL 944

This may be done in a program by -

100 PRINT D$; "BLOAD PATCH": CALL 944

(D$ = CONTROL D)

To turn off printer use PR#0.

Then to reconnect the printer, all that is required is CALL 954 or from a program 300 CALL 954.

* * *NOTES* * *

If the printer does not print after the CALL 944, it is probably sending the opposite polarity busy signal. The patch can be changed to recognize with

POKE 968, 16

If this doesn't work, have the printer checked.

The modification allows the speed, column width, and other variables to be changed with the POKEs in the manual.

COLOR KILL

BY LARRY DANIELSON

For you pioneers who bought your APPLES before the color killer modification was put in and if you now use a color TV for a monitor, then this article may interest you.

Take two parts, a 2N3904 transistor (or an equivalent) and a 1.6K resistor and solder them to the breadboard section of your APPLE as in Fig. 1. The transistor base and the free end of the resistor should be tied together. Take some hook-up wire and go from the other free end of the resistor to IC #F14-pin 4. With another piece of wire go from the transistor collector to the color trimmer point. Take a third piece of wire and go from the transistor emitter to ground of the capacitor next to the edge on the corner.

Fig. 2 is the schematic of the two parts and how they tie into the APPLE system. To test the final product, just put it back together and turn it on. Go into BASIC, type GR, then type TEXT. The screen should go from color to black and white.

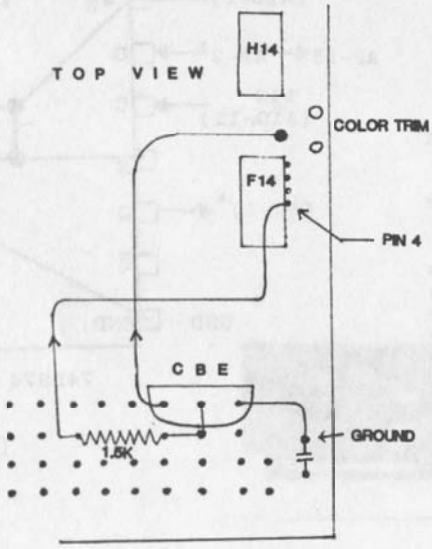


FIG. 1

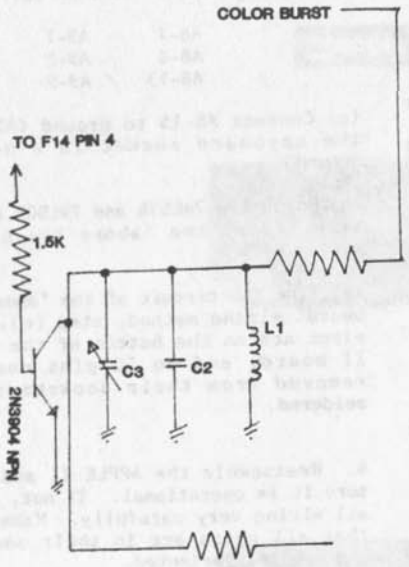


FIG. 2

USER FIRMWARE

APPLE APPLICATION NOTES

NOTE: The modification described in the following instructions will void the warranty on your APPLE II.

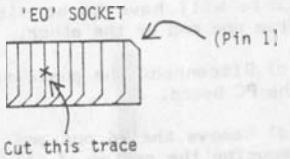
There are times when you may want to create your own firmware for the Apple II. Such firmware can be tailored to your special needs and then installed semi - permanently so that it can be used as conveniently as Apple BASIC or the Monitor.

APPLE MOD'S

This application brief details a simple modification which allows your Apple II to accept industry standard 2716 (2Kx8-bit) Eraseable Read - Only - Memories (EPROM's) in sockets "D0" & "D8". These sockets correspond to memory addresses D000 - DFFF.

EPROM's (2716) are readily available through most semiconductor distributors. Many distributor locations are equipped to program the EPROM's to your specifications, and will do so for a moderate fee.

1. Remove the 'E0' ROM from its socket. On the TOP side of the board, under the 'E0' socket, cut the ROM pin 18 jumper trace. Then reinsert the ROM. This cut will isolate pin 18 or ROM 'D0' & 'D8' from pin 18 of the other ROM. Reinsert the 'E0' ROM when done.



2. On the UNDERSIDE of the Apple board, cut the traces connecting pin 20 to 21 of ROMs 'D0' & 'D8' only.

3. (Underside) Cut the trace going to pin 18 of ROM 'D8' near the chip. Scrape solder resist off of approximately 1/4 inch of the remaining trace not still connected to pin 18. You may wish to tin it with solder since it will later be soldered.

4. (Underside) Connect pin 18 of ROM 'D8' to pin 12 of ROM 'E0' (ground).

5. (Underside) Connect pin 18 of ROM 'E0' to the trace which previously went to pin 18 of ROM 'D8' (and which should be pretinned if step 3 was followed).

6. (Underside) Connect pin 21 of ROM 'D8' to pin 21 of ROM 'D0'. Then connect both of these to pin 24 of either ROM (VCC).

7. Note that the INH control function (pin 32 on the Apple I/O BUS connectors) will not disable the 2716 EPROMs in the 'D0' & 'D8' ROM slots, since pin 21 is a power supply pin and not a chip select input on the EPROMs.

8. EPROM (2716) devices may now be used in sockets 'D0' & 'D8'

NOTE: IF YOU MAKE THIS MODIFICATION, DO NOT INSTALL A ROM CARD OR THE LANGUAGE SYSTEM BEFORE REMOVING THE DEVICES IN SOCKETS 'D0 & D8'.

6 COLOR MOD

APPLE APPLICIATION NOTES

(NULLIFIES WARRANTEE)

Applies only to APPLES with serial numbers below 6000 (REV 0 PC BOARD).

1. REMOVE THE APPLE II PC BOARD FROM ITS ENCLOSURE

(a) Remove the ten (10) screws securing the plastic top piece to the metal bottom plate. Six (6) of these are flat-head screws around the perimeter of the bottom plate and four (4) are round-head screws located at the front lip of the computer. All are removed with a Phillips-head screwdriver. Do not remove the screws securing the power supply or nylon insulating standoffs.

(b) Lift the plastic top piece from the bottom plate while taking care not to damage the ribbon cable connecting the keyboard to the PC board. This cable will have to be disconnected from one end or the other.

(c) Disconnect the power supply from the PC board.

(d) Remove the #6 nut and lockwasher securing the center of the PC board. These will not be found on the earlier APPLE II computers.

(e) Carefully disengage each of the 6 nylon insulating standoffs from the PC board (7 on earlier versions).

(f) Lift the PC board from the bottom plate.

2. ABOVE BOARD WIRING METHOD

(A) lift the following IC pins from their sockets.

A8-1
A8-6
A8-13
A9-1
A9-2
A9-9

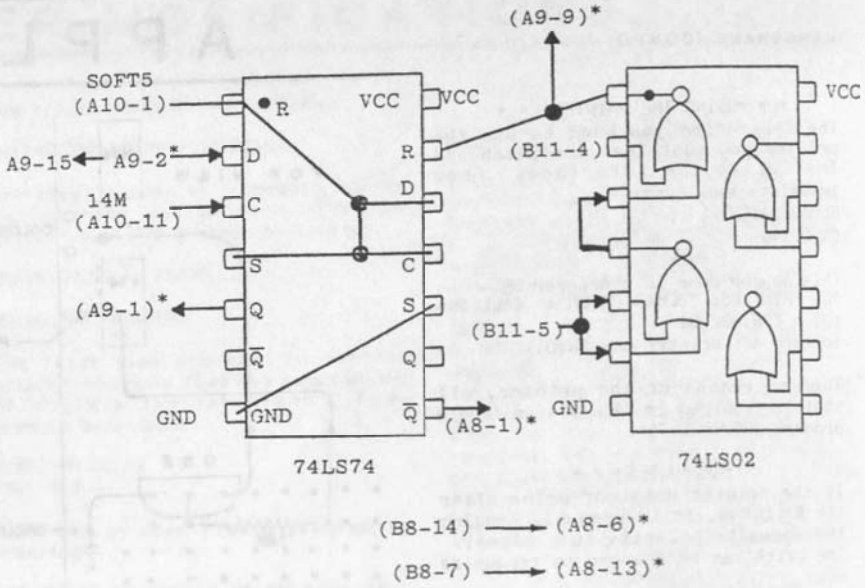
(b) Mount a 74LS74 (dual c-d flip-flop) and a 74LS02 (quad NOR gate) in the APPLE II breadboard area (A11 to A14 region).

(c) Wire the following circuit ('' indicates that wiring is to a pin which is out of its socket).

3. BELOW THE BOARD WIRING METHOD
(for neater appearance)

(a) Desolder all pins of socket A8. Lift the socket and its 74LS257 IC off the PC board taking care not to destroy it. Cut the trace between pins 6 and 13 of A8 on the top side of the board. Also cut the trace between pins 13 and 15 on the top. Reinsert socket A8 and the 74LS257.

BE CAREFUL!!



(b) Cut traces going to the following IC pins on the bottom of the APPLE-II board. Each pin should have a single trace going to it. BE CAREFUL!!

A8-1	A9-1
A8-6	A9-2
A8-13	A9-9

(c) Connect A8-15 to ground (A7-8 on the keyboard socket is a nearby ground).

(d) Mount the 74LS74 and 74LS02 as per step (b) of the 'above the board' wiring method.

(e) Wire the circuit of the 'above the board' wiring method, step (c). All wires are on the bottom of the APPLE II board, and no IC pins need be removed from their sockets or be soldered.

4. Reassemble the APPLE II and make sure it is operational. If not, check all wiring very carefully. Make sure that all chips are in their sockets and properly oriented.

5. The following color values are now applicable to the HI-RES subroutines.

BLACK2	128
ORANGE	170
BLUE	213
WHITE2	255

For example, the program below draws an orange line from (10,20) to (200,140). It is assumed that the HI-RES routines are already in memory locations $800-$bff.

```
 0 XO=YO=COLR
 5 INIT=2048: PLOT=2830: LINE=2836
 7 ORANGE=170: CALL INIT
10 XO=10: YO=20: COLR=ORANGE: CALL
   PLOT
20 XO=200: YO=140: CALL LINE
30 END
```

DUMPLINGS

BY ED AVELAR

This article describes an easy construction project which will allow you to save programs from the APPLE to six (6) or more cassette recorders simultaneously.

Actually the circuit can accommodate as many as (18) recorders, thru the use of (Y) connectors. It is also possible to install (18) MIC jacks initially and jumper them in groups of three (3).

The circuit consist of one (1) 74LS67 (Tri-state hex buffer) IC, pins 1 and 15 are tied low in order to keep the device in the (non) tri-state mode. The IC should be mounted in a small aluminum box along with the required MIC jacks and an LED (optional).

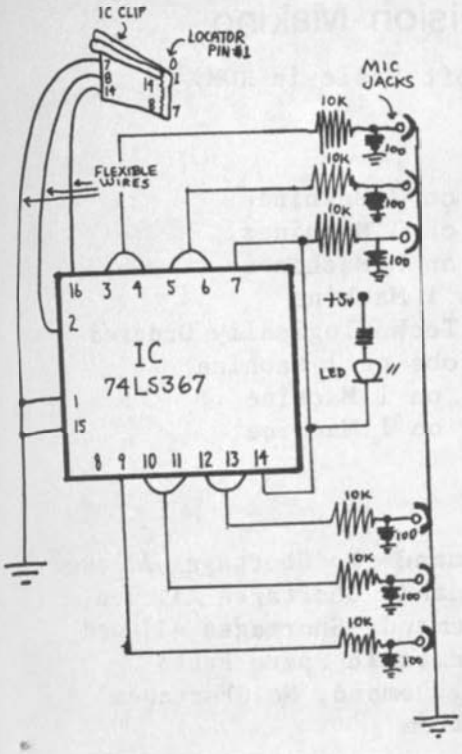
The LED monitors the actual dump and indicates if you are properly connected to IC (K13).

To connect the dumper to APPLE II an IC test clip was used; thus no hard or direct wiring is necessary. The test clip is attached to K13 inside the APPLE II by locating pin #1 and placing the test clip over the IC; this can be performed even if the computer is powered up.

With a program loaded in the computer put the recorders in record mode and type save; the LED will come on indicating the save has started, and it will go off when done.

Additional features can be added to the dumper. For example, You could add a MIC & MIC pre-amp, thus allowing you to narrate to all recorders simultaneously. And you could add a (6) receptacle (AC) strip controlled by a switch; this would allow you to leave all recorders in the record mode and allow simple control of many recorders.

In conclusion, any wire can be used between the dumper and the clip. Be sure to identify pin #1 on the clip. A good way is to burn a mark on the clip with a soldering iron.



THE 15¢ 15 SEC RESET FIX

BY KEN SILVERMAN

You are right at the end of a 16K program in memory and have not saved it to the disk yet or writing a text file and four pages later - you hit "reset".

Well you can try to retrieve it by many methods like "LAZARUS" program, or if you have the Auto Boot ROM you probably have not lost it.

No matter what it is a nuisance when we hit the RESET key, and we all hit it by accident once in a while. Now try this little 15 cent fix, and the only time RESET will be pressed is when you WANT to do it.

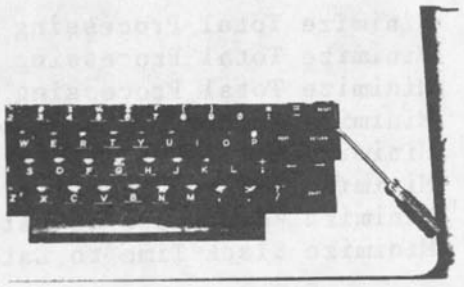
1. Remove the key cap for RESET, picture 1.

2. Get yourself a 7/16 in I.D.-5/8 in O.D. O Ring, Sears model 42-22517 - picture 2.

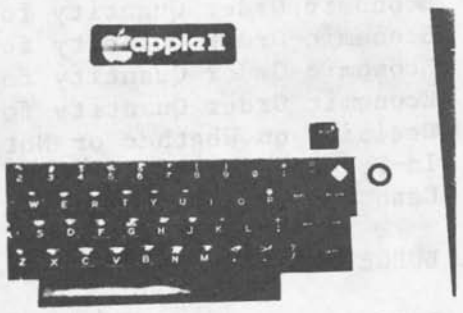
3. Place the O Ring over the switch under the cap, picture 3.

4. Put the key cap back on.

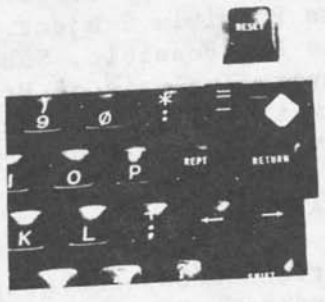
Now it will take some pressure to press the key. NOTE: This modification might void your Apple II Warranty.



(1)



(2)



(3)

QuikDirt *Product Information*

Management Planning & Decision Making

(for 32K or 48K APPLE II with AppleSoft Basic in ROM)

☆ PRODUCTION SCHEDULING

Minimize Total Processing Time, N Jobs on 2 Machines
Minimize Total Processing Time, N Jobs on 3 Machines
Minimize Total Processing Time, N Jobs on M Machines
Minimize Number of Late Jobs, N Jobs on 1 Machine
Minimize Total Processing Time, N Jobs Technologically Ordered
Minimize Weighted Completion Times, N Jobs on 1 Machine
Minimize Worst Case of Lateness, N Jobs on 1 Machine
Minimize Slack Time to Lateness, N Jobs on 1 Machine

☆ INVENTORY CONTROL

Economic Order Quantity for Constant Demand No Shortages Allowed
Economic Order Quantity for Constant Demand, Shortages Allowed
Economic Order Quantity for Uncertain Demand, Shortages Allowed
Economic Order Quantity for Stocking Strategic Spare Parts
Economic Order Quantity for Time-Varying Demand, No Shortages
Decision on Whether or Not to Stock an Item
In-Process Inventory Forecasting
Cash Replenishment Model

☆ CAPITAL BUDGETING (Project Selection)

Defining Projects
Defining Resources
Defining Payoffs per Project
Defining Amounts of Resources Reguired for each Project
Projects Possible Subject to Resources Available
Projects Not Possible, Subject to Resources Available
Sensitivity Analysis of Project Payoffs
Sensitivity Analysis of Resource Availabilities
Sensitivity Analysis of Resource Requirements per Project

☆ DISTRIBUTION PLANNING

Transportation Sources (Supply)
Transportation Destinations (Demand)
Transportation Shipping Costs
Optimal Selection of Routes
Warehouse Location
Generalized Network Flows

wyman associates
421 SEVILLE WAY
SAN MATEO CA 94402
(415)345-0380

☆ SUGGESTED RETAIL PRICE: **$100**

GRAPHICS

GRAPHICS

ANIMATION

BY CHRIS GARRIGUES

You may be aware of articles in the CIDER PRESS or CONTACT that mention 'Page 1' and 'Page 2' of display in connection with animation. I hope to make the use of these pages of screen a little more clear.

First of all, when you power-up you are viewing 'Page 1' and continue to do so unless you use HIRes graphics (which uses two pages of its own) or run a program like the one to be outlined.

But first, let's start with a program that only uses 'Page 1' and should be fairly clear:

```
 10 GR : REM GRAPHICS MODE
 20 CALL -936: REM CLEAR TEXT WINDOW
 30 FOR X=0 TO 19: REM LOOP X & Y
    COORDS
 40 FOR Y=X TO 19: REM ODD RANGE IS TO
    AVOID PLOTTING OVER ITSELF
 50 COLOR=((((ABS(20-X)-Z)*(ABS(20-Y)
    -Z)/25)MOD 16)+16) MOD 16: REM
    THIS HORRID FORMULA COMPUTES THE
    COLOR
 60 PLOT X,Y: REM PLOT ALL REFLECTIONS
 70 PLOT Y,X
 80 PLOT 39-X,39-Y
 90 PLOT 39-Y,39-X
100 PLOT 39-X,Y
110 PLOT 39-Y,X
120 PLOT X,39-Y
130 PLOT Y,39-X
140 NEXT Y,X: REM GO BACK FOR NEXT
    SQUARE
145 CALL -936: REM CLEAR TEXT ON PAGE
    PAGE#1
150 VTAB 22: TAB 20: PRINT Z: REM
    PRINT Z
160 Z=Z+1: REM MOVE ALONG Z AXIS
170 IF Z=183 THEN Z=-161: REM IF I
    DON'T STOP I WILL GET A >32767
    ERR
```

This program just plots colors determined in line 50 on the screen. Lines 64 through 130 plot all reflections so I can just compute 1/8 of the screen and plot it eight times. As it is, you can see the points as they are plotted, but suppose I wanted it to just flash from one step to the next.

First you must type: >LOMEM: 3072

(To help you visualize what this means, turn to the memory map on Page 136 in your red manual.)

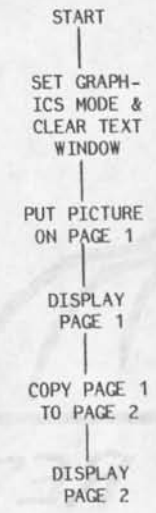
Before typing this, LOMEM equals 800 HEX, which puts your variables in 'Page 2' so your screen and variables interfere with one another. One or the other must be moved. Since 'Page 2' is there due to hardware it can't be moved, so obviously we have to move the variables. Typing 'LOMEM: 3072' moves the variables to C00 HEX (or 3072 decimal).

Then enter the following:

```
  0 REM LOMEM:3072
200 POKE-16300,0: REM FLIP TO PAGE #1
210 REM COPY PAGE #1 TO PAGE #2
220 POKE 60,0: POKE 61,4: REM START
    ADDRESS OF PAGE #1
230 POKE 62,255: POKE 63,7: REM END
    ADDRESS OF PAGE #1
240 POKE 66,0: POKE 67,8: REM START
    ADDRESS OF PAGE #2
250 CALL -468: REM DO THE COPY
260 POKE -16299,0: REM FLIP TO PG. #2
```

Line 0 is to remind you of the above.
Line 200 turns your view to 'Page 1'.
Lines 220 - 240 state where the pages are located.
Line 250 does the actual move.
Line 260 turns your view to 'Page 2'.

Please note my flowchart:



You should see that except for the first time you are looking at 'Page 2' while the picture is being put on 'Page 1'. This way the picture seems to jump from one picture to the next instead of being drawn on.

From this basic flow-chart all simple animation can be done.

Also, if you break the program while on 'Page 2' you need to flip back to 'Page 1' before you can see what's happening.

HIRES GRAPHICS

BY JOHN UHLEY

NOTE: JOHN WAS 13 WHEN HE WROTE THIS ARTICLE.

If any of you "Apple-Owners" are interested in how to plot points and how to create 'interesting' patterns using the super Hires graphics that you set up by typing 'POKE -16297,0 RETURN, and 'POKE -16304,0 RETURN', then this article is one that you might want to read.

To begin, get into Integer BASIC. Next, set up the HIRES screen. Do this by typing 'POKE -16297,0 RETURN', and 'POKE -16304,0 RETURN'. You will now see one of two things: either a screen full of little colored dots that look like they have been smashed together, or a screen that has white horizontal lines running across it. These lines may have a few green vertical lines running under them.

The next step is to get the cursor back. To do this, we will simply clear the bottom four lines for text. Just type 'POKE -16301,0' and press RETURN until you see the cursor.

You now want to clear the screen to black. To do this you must make every line number in the monitor starting at 2000 HEX (8192 in decimal) and ending at 3FFF (16383 in decimal) contain the value of zero. By simply typing in the clearing program, the screen will be cleared to black. What happens when you run this program is that the value of zero is given to location 2000, then 2001 (8193 in decimal), then 2002 (8194 in decimal), and so on until the for-next loop reaches 3FFF or 16383 in decimal. Now run this program (it takes 30 seconds to run).

You are now ready to plot your first HIRES point. Type 'POKE 8192,45' and hit RETURN. This is the same as typing 2000:2D in the monitor. In the upper left hand corner of the screen a line segment will appear. Actually you could have typed 'POKE (any number between 8192 and 16383) then a comma, and any number between 1 and 255'. For this example, however, we will use 'POKE 8192,45'.

A line segment appeared on the screen because you told the computer to put 2D(45 in decimal) into location 2000. In this mode, any value above zero and less than 255 that is in memory from 2000 to 3FFF will cause a dot or a line to appear on the screen.

Now that you understand how to plot HIRES points, there is something that you should know. After you get to the end of the top line you jump down to the 64th line. This is due to the way the screen is set up. Henceforth, when the computer plots your picture, it will plot line one, then line sixty-four, then line one hundred and twenty-seven, then line two, and so on.

If you plan to make pictures, then you should know that when you type 'POKE "X","Y"'that the computer will find "X" and then draw from "X" over "Y" units. These units number from 1 to 255(FF) and are divisions of "X" (see fig. 1). For example, a 'POKE 8192,255' won't draw from 8192 to 8447 (or 8192 + 255). It will draw to 8193. Therefore a 'POKE 8192,127' will draw a line about half way to 8193. Therefore, a 'POKE 8192,0' erases everything on line segment 8192. This is because the computer can't plot from zero to zero.

With this information you can create interesting pictures, and intricate designs (with the help of for-next loops).

```
        255 units
1.................1.................1
8192.............8193...........8194
```

ALL PROGRAMS ARE IN INTEGER BASIC:

```
        CLEARING PROGRAM
        -------- -------
10 FOR CLEAR = 8192 TO 16383
20 POKE CLEAR, 0
30 NEXT CLEAR
40 END
```

```
        RANDOM DOTS
        ------ ----
10 POKE -16297,0:  POKE -16304,0
20 FOR CLEAR = 8192 TO 16383
30 POKE CLEAR, 0
40 NEXT CLEAR
50 N1 = RND(8192) + 8192
60 N2 = RND(255)
70 POKE N1, N2
80 GOTO 50
90 END
```

```
        JUST FOR FUN
        ---- --- ---
10 POKE -16297,0:  POKE -16304,0
20 FOR CLEAR = 8192 TO 16383
30 POKE CLEAR, 0
40 NEXT CLEAR
50 G = 6:  REM BY CHANGING G YOU GET
   A DIFFERENT PICTURE
70 FOR TR = 1 TO 255
80 FOR TY = 8192 TO 16383 STEP G
90 POKE TY, TR
100 NEXT TY
110 NEXT TR
120 END
130 REM G = 100 AND G = 91 ARE
   INTERESTING
```

```
        WHITE OUT SCREEN
        ----- --- ------
10 POKE -16297,0:  POKE -16304,0
20 FOR T = 8192 TO 16383
30 POKE T,0
40 NEXT T
50 FOR SERT = 8192 TO 16383
60 POKE SERT, 255
70 NEXT SERT
80 END
```

SCREEN CLR

BY BILL SHRYOCK

```
>LIST

10 DIM C$(21)
20 C$ = '2001<2000.3FFFM E88AG'
30 POKE 8192,0: REM POKE 8192,255 TO
   WHITE OUT SCREEN
40 FOR I = 1 TO 21
50 POKE 511 + I, ASC(C$(I))
60 NEXT I
70 CALL -144
```

Just be sure to set 'HIMEM: 8192' before you run the clear routine.

Since this routine uses the system monitor move command to clear the screen it clears the screen much faster than the system in John's article.

HIRES STILL LIFE

BY JOHN UHLEY

In the monitor, memory locations 2000 through 3FFF stands for a very small line on the HIRES screen. This "line" is made up of 255 sections (Fig. A). Depending on the number (X) stored in a memory location between 2000 and 3FFF, a different "colored" dot, or group of dots, will be seen on the screen. To explain, let's assume that memory location 2000 contains FF (255 in decimal). If this occurred, a white line (actually many dots) would stretch from 2000 (0) to 2000 (255). If memory location 2000 contained 2D (127 in decimal) then a group of "colored" dots would appear from 2000 (0) to 2000 (127). In other words, a series of dots will appear on the screen starting at the memory location, "dot" zero, and ending at the memory location "dot" X.

In order to change the value stored in a memory location, from basic, you have to either 'POKE in' a new value or use the HPLOT command. To 'POKE in' a new value you need two things. First, you need the memory location you want to 'POKE into' (converted into decimal), and second, you need the decimal value you want to 'POKE in'. An example of this is 'POKE 8192,127'. This stores 2D (127 in decimal) in memory location 2000.

If you don't understand how to 'POKE in' values from basic, or how the values stored in memory locations 2000 through 3FFF affect the length of the "dots" or "lines" appearing in the screen, I suggest that you read HIRESOLUTION GRAPHICS, Page 46.

When you type 'HGR' AND HIT RETurn, the computer "GOSUBs" into a special machine language subroutine, via the command tables. This machine language subroutine performs several functions. One of its functions is to LoaD the Accumulator (LDA) with memory locations C050 and C057. This "turns-on" the HIRES screen. Another of its functions is to clear the screen to black. It does this by storing zero in memory locations 2000 thru 3FFF. As you know the computer draws from the memory location "dot" zero to memory location "dot" X(where X is the amount stored in that memory location). Obviously, it is not possible to draw from zero to zero. Therefore, any memory location containing zero will be a black space on the screen. If all the memory locations starting at 2000 and ending at 3FFF contain zero, then the entire HIRES screen will be black.

As you know, when you type "POKE num1, num2", num2 (converted into hex) is stored in num1 (converted into hex). As previously mentioned, you can "POKE in" a number from basic, or you can use the HPLOT command. If you use the HPLOT command "POKE num1" is replaced by one of the three versions of the HPLOT command (see Table A.). This leaves you the number that you want to store in your memory locations.

This number, in a way, is replaced by the value of HCOLOR. The command HCOLOR=X (where X is a number between 0 and 7) assigns the sequence that you start and step by. What I mean by this is that there are seven colors (see Fig. B). Each time the computer plots a dot, it steps seven times, skipping six colors each step (Fig. C). So there will always be a small "space" (the six unused colors) between all HORIZONTAL lines. This is NOT always true with vertical and diagonal lines because there is no space between each vertical line (Fig. D).

When you type 'HPLOT num1, num2', a single dot appears on the screen. Likewise, when you type 'HPLOT num1, num2 to num3,num4' a line, made up of dots, will appear on the screen starting at num1, num2 and ending at num3, num4. What the computer does to make these lines is to compute the correct values, and store these values in the correct memory locations. After it has done this it returns control to the program or programmer.

Hopefully, you now know how to use the HGR, HCOLOR, and HPLOT commands. If you are still not sure about how to use these commands, try the three programs given later in this article. After you have done those programs try making one of your own.

Now that you know how to make HIRES pictures you might want to know how to save "still-life" pictures you make. I do not mean making a program, saving it, and running it again. I mean saving the picture that is in memory from 2000 through 3FFF.

As you know when you tell the computer to HPLOT a line it computes the correct values to store somewhere between 2000 and 3FFF. Depending on the value, different colored dots and lines appear on the screen. Therefore if you save these values, you would be saving the "blueprint" to your picture, and if you loaded in this "blueprint" you would be reconstructing your picture!

If you don't have a disk, all that you have to do, after you make your picture, is to hit the key marked RESET, hook up your recorder, type 2000.3FFFW, start the recorder playing, hit the key marked RETURN, wait for the cursor to appear, and rewind the tape.

If you have a disk all that you have to do, after you have made your picture, is to type "TEXT" (if you are not in text mode), and type 'BSAVE (whatever you want to call your program) ,A$2000, L8192' and hit RETurn. When the disk stops your picture has been saved.

All that you have to do to see your picture, if you don't have a disk, is to hit the key marked RESET, hook up the tape recorder, type C050 C057 2000.3FFFR, turn on the recorder, and hit RETurn. When your picture is complete, turn off the tape recorder and rewind the tape.

If you have a disk, all that you have to do to load your picture is to type 'HGR' (RETurn) and 'BLOAD (whatever you called your program),V0'(RETurn). When the disk stops, your picture will be completed on the screen.

As a last word, I'd like to add that you can view your picture in Integer Basic by loading it in, and typing 'POKE -16297,0'(RETurn) and 'POKE -16304,0'(RETurn).

```
ALL PROGRAMS IN FP
--- -------- -- --
10 HGR
20 HCOLOR = X 30 HPLOT RND(1)*270,190
TO 100,
   RND(1)*190
40 X = X + 1
50 IF X + 1 = 8 THEN X = 1
60 GOTO 20
70 END


10 HGR
20 HCOLOR = 5
30 HPLOT X,100 TO 0,0
40 X = X + 2
50 GOTO 30
60 END


10 HGR
20 HCOLOR = 6
30 FOR T = 1 TO 20
40 X = X + Y
50 HPLOT X,Y
60 Y = Y + 2
70 NEXT T


TABLE A
----- -
HPLOT X,Y--HPLOT's a single HIRES
point
HPLOT X,Y TO E,R--HPLOT's from X,Y to
E,R
HPLOT TO X,Y--HPLOT's from the last
plotted point to X,Y.
```

FIG. A

FIG. B

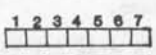


FIG. C

FIG. D

COLOR POKE

BY ART WELLS

For those of you who have any need to desire to POKE colors into a lo-rez program, you can do so by putting the color into location $30, which is decimal 48. The following chart shows which numbers to POKE in for which colors.

DECIMAL	COLOR
00	Black
17	Magenta
34	Dark Blue
51	Light Purple
68	Dark Green
85	Grey
102	Medium Blue
119	Light Blue
136	Brown
153	Orange
170	Grey
187	Pink
204	Green
221	Yellow
238	Blue/Green
255	White

PAGE-FLIP

BY GLEN HOAG

```
10 REM DEMONSTRATION OF THE PROCEDURE
   TO MOVE PAGE 1 TO PAGE 2 OF BASIC
20 REM 28 JANUARY 1979
30 REM BY GLEN C.  HOAG
40 REM PROGRAM FLOW
50 REM (1) SET UP PAGE 1 SCREEN
60 REM (2) SET UP POINTERS FOR PAGE 1
   START
70 REM (3) SET UP POINTERS FOR PAGE 1
   END
80 REM (4) SET UP POINTERS FOR PAGE 2
   START
90 REM (5) CALL MONITER MOVE ROUTINE
100 REM (6) POKE THE SWITCH TO DISPLAY
    PAGE 2
110 REM
120 REM
130 REM
140 REM STEP (2)
150 POKE 60,0: POKE 61,4: REM POKE
    VALUES FOR A1
160 REM STEP (3)
170 POKE 62,255: POKE 63,7: REM POKE
    VALUES FOR A2 (CONTAINS $07FF)
180 REM STEP (4)
190 POKE 66,0: POKE 67,8: REM POKE
    VALUES FOR A4 (CONTAINS $0800)
200 REM STEP (5)
210 CALL -468: REM MONITER MOVE
    ROUTINE LIVES AT $FE2C
220 REM STEP (6)
225 CALL -936: REM CLEAR PAGE 1
230 POKE -16299,0: REM TURN ON PG.2
235 PRINT "THIS IS PAGE 1"
240 FOR I=0 TO 1000: NEXT I
250 POKE -16300,0: REM TURN ON PG.  1
260 FOR I=0 TO 1000: NEXT I
270 GOTO 230
```

21 HIRES COLORS

The following program was on our August 1979 DOM (Disk Of the Month)

```
LIST
20 HGR 25 FOR C=39 TO 139 STEP 2
30 XCUR=5
50 FOR A=1 TO 6
60 FOR B=A TO 6
70 HCOLOR=A:HPLOT XCUR,C TO
   XCUR + 10,C
80 HCOLOR=B* HPLOT XCUR,C + 1
   XCUR + 10,C + 1
90 XCUR=XCUR + 12
100 NEXT B,A,C
110 END
112 REM BY ANDY HERTZFELD AFTER AN
113 REM IDEA BY D.  ALDRICH GIVEN
114 REM TO SF APPLE CORE 7/79
115 REM TYPED BY KEN SILVERMAN
116 REM AS READ TO HIM BY SCOT
117 REM KAMINS WHILE PAUL WYMAN
118 REM WAS TALKING TO THE AUTHOR
119 REM DURING THE TIME 21 MEMBERS
120 REM WERE PRESENT AT VILLAGE
121 REM ELECTRONICS.  THANKS TO ALL
122 REM INCLUDING MY MOTHER.
```

PS - The smaller the color TV screen the better this looks - if you have a color monitor around this looks great.

TEXT SCRN MAPS

ADDRESSES SHOWN ARE FOR THE FIRST CHAR IN EACH LINE - LINES ARE 40 CHRS. LONG

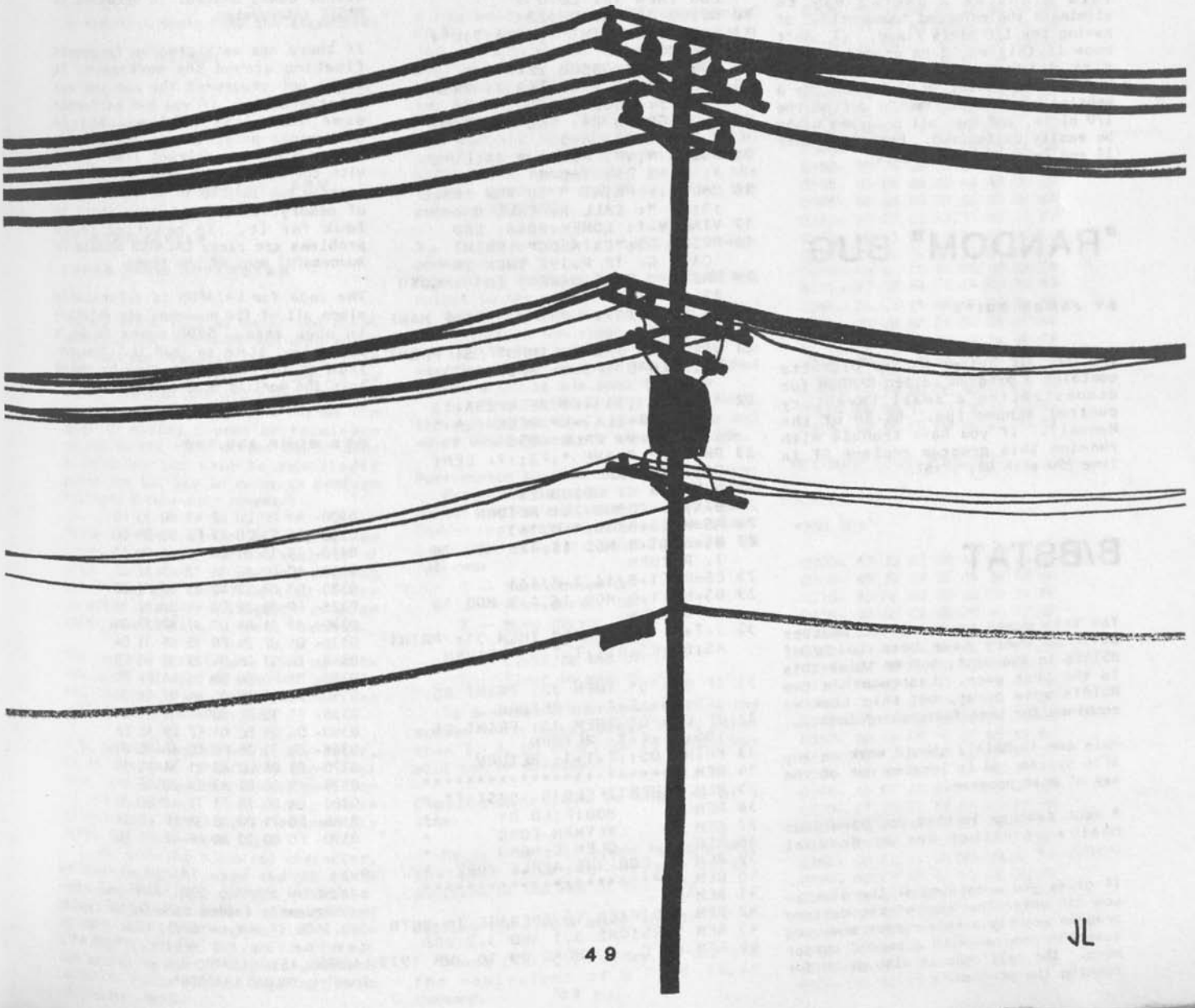
PAGE ONE

LINE #	POKE ADDRESS
00	1024
01	1152
02	1280
03	1408
04	1536
05	1664
06	1792
07	1920
08	1064
09	1192
10	1320
11	1448
12	1576
13	1704
14	1832
15	1960
16	1104
17	1232
18	1360
19	1488
20	1616
21	1744
22	1872
23	2000

PAGE TWO

LINE #	POKE ADDRESS
00	2048
01	2176
02	2304
03	2432
04	2560
05	2688
06	2816
07	2944
08	2088
09	2216
10	2344
11	2472
12	2600
13	2728
14	2856
15	2984
16	2128
17	2256
18	2384
19	2512
20	2640
21	2768
22	2896
23	3024

UTILITIES



JL

UTILITIES

SLOT #'S AS VARIABLES

BY FRANK FISHER

In/Out slots can be specified as variables! These can then be defined once, and then used anywhere in the program. In BASIC, the statements might look like this:

```
100 PR = 1:REM PR IS SLOT# OF PRINTER

110 PL = 3:REM PL IS SLOT# OF PLOTTER

                .
                .
                .

400 PR#PR:PRINT "THE PRINTER IS IN
    SLOT ";PR
```

This might be a useful way to eliminate the enforced "convention" of having the I/O slots fixed. (I don't know if this would be practical for disk drive I/O slots however). Program documentation might include a mention of what statements define the I/O slots, and then all programs might be easily customized. Both Applesoft II and Integer support this.

"RANDOM" BUG

BY JAMES SUITS

The DOS 3.2 System Master Diskette contains a program called RANDOM for demonstrating a small inventory control scheme (pp. 86-88 of the Manual). If you have trouble with running this program replace ST in line 290 with BW;",";ST.

B/BSTAT

Yes this month we bring you another BSTAT. There have been loads of BSTATs in the past, but we think this is the best ever. Last month's two BSTATs were great, but this time we combined the best features of both...

This one (B/BSTAT) should work on any size system and is located out of the way of most programs.

A neat feature is that you have your choice of either hex or decimal output.

It gives you a catalog of the disk so you can enter the name of the desired program exactly with a cursor move and save the program with a second cursor move. The call code is also given for running the program..

```
0 REM  [ "BSTAT" VERSION 7 ]
1 POKE 74, PEEK (76): POKE 75
  , PEEK (77)-8: CLR
2 M=150- PEEK (77):H=169:SL=181
  :LL=163:H=H-M:S=1:K=256:I=-
  384:N=-380:G=-198:D$=""
3 DIM F$(50),H$(16),Z$(50):H$
  ="0123456789ABCDEF": IF H<128
   THEN 4:S=-1:H=K-H:PTCH=0: IF
  PEEK (977)=191 THEN PTCH=189

4 SA=S*K*H+SL+PTCH:LA=S*K*H+LL+
  PTCH
5 PRINT D$;"NOMON C,I,O": TEXT
  : CALL -936: VTAB 3: TAB 8
6 CALL I: PRINT " (B)FILE STARTING
   ADDRESS ": PRINT : TAB 9: PRINT
  " AND LENGTH DELIMITATOR "
7 CALL N: PRINT : TAB 16: PRINT
  "[ BSTAT ]": VTAB 10: TAB 10
  : CALL I
8 PRINT " HEX OR DEC ADDRESSES "
  : CALL N: CALL G: PRINT : TAB
  15: PRINT "[H] OR [D] ?"
9 P= PEEK (-16384): POKE -16368
  ,0: IF P=196 THEN 10: IF P=
  200 THEN 12: GOTO 9
10 GOSUB 18: GOSUB 22
11 TAB T+8: PRINT ",A";A;",L";
  L;: GOTO 15
12 GOSUB 18: GOSUB 22:X=A
13 GOSUB 24:T=T+8: TAB T: PRINT
  ",A$";: GOSUB 30
14 X=L: GOSUB 24: TAB T: PRINT
  ",L$";: GOSUB 30
15 PRINT ",V": PRINT " CALL ";
  A: VTAB 23: TAB 11
16 CALL I: PRINT " TO RUN 'CALL "
  ;A;"' ": CALL N: CALL G
17 VTAB V-1: LOMEM:2048: END
18 PRINT D$;"CATALOG": PRINT :
   CALL G: IF P=196 THEN 20
19 PRINT " FILE NAME ? [H]": GOTO
  21
20 V= PEEK (37): PRINT " FILE NAME
  ?  [D]";
21 VTAB 14: TAB 7: INPUT F$: PRINT
  D$;"BLOAD";F$:V= PEEK (37):
   RETURN
22 A= PEEK (SA)+K* PEEK (SA+1)
  :L= PEEK (LA)+K* PEEK (LA+1
  ): VTAB V: CALL -958
23 PRINT " BSAVE ";F$;:T= LEN(
  F$): RETURN
24 B=X MOD K: GOSUB 28
25 B=X/K: GOSUB 26: RETURN
26 A$=H$(1+B/16,1+B/16)
27 B$=H$(1+B MOD 16,1+B MOD 16
  ): RETURN
28 C$=H$(1+B/16,1+B/16)
29 D$=H$(1+B MOD 16,1+B MOD 16
  ): RETURN
30 T=T+3: IF A$="0" THEN 31: PRINT
  A$;B$;C$;D$;:T=T+4: RETURN

31 IF B$="0" THEN 32: PRINT B$
  ;C$;D$;:T=T+3: RETURN
32 IF C$="0" THEN 33: PRINT C$
  ;D$;:T=T+2: RETURN
33 PRINT D$;:T=T+1: RETURN
34 REM ************************
35 REM * HERTZFELD'S  "BSTAT" *
36 REM *      MODIFIED BY      *
37 REM *      WEYMAN FONG      *
38 REM *      GLEN C HOAG      *
39 REM *  FOR THE APPLE CORE   *
40 REM ************************
41 REM
42 REM MODIFIED TO OPERATE IN BOTH
43 REM VERSIONS 3.1 AND 3.2 DOS
44 REM BY G. C. HOAG ON 10 JUN 1979
```

LAZARUS

BY ANDY HERTZFELD

This article describes LAZARUS, a short machine Language program which can resurrect inadvertently erased BASIC programs.

When a BASIC program is erased, it is usually not destroyed. It is probably still sitting around somewhere in memory; however, the BASIC interpreter no longer knows where to find it, so it is effectively lost. LAZARUS scans through memory trying to find the largest valid BASIC program that it can. If it finds one it resets the interpreter's pointers to point to the found program, thereby resurrecting it. If no valid program is found an error message will be printed. In either case, control is returned to BASIC interpreter.

If there are many program fragments floating around the workspace, it might not resurrect the one you are interested in. If you had performed some deletions of low-numbered statements prior to the erasure, it may restore the deleted lines along with the rest of the program. If your program was located in an unusual part of memory, it may not know where to look for it. In practice these problems are rare; LAZARUS should be successful most of the time.

The code for LAZARUS is relocatable since all of its branches are relative in page zero. $300 seems to be a convenient place to load it. You can load it for the first time by going into the monitor mode and typing:

HEX DUMP 300.390

```
0300- A9 FF D0 02 A9 00 85 F0
0308- 85 F1 20 89 F6 B0 3A E0
0310- 35 13 04 00 16 4A 00 17
0318- 4C 00 66 36 32 67 37 22
0320- D7 03 59 42 D5 07 F8 22
0328- F0 38 39 E8 48 D3 02 18
0330- A9 34 44 D5 07 12 27 D4
0338- 05 0E 24 F0 39 68 31 E4
0340- 64 D1 05 04 29 38 01 E3
0348- 22 F0 34 D9 06 D1 F4 F4
0350- F4 11 05 00 84 D1 06 0D
0358- E1 10 FF 00 D1 02 C0 24
0360- D6 05 BC 01 EF 29 32 E2
0368- B4 31 18 F0 00 68 06 04
0370- 21 DA 02 AB 21 3A 24 3B
0378- 29 3C 01 A3 2A 06 12 11
0380- CA 00 2B 71 11 4C 00 2C
0388- E0 71 00 20 3A FF 4C 03
0390- E0 00 20 2D FF 4C 03 E0
```

Once it has been loaded it can be saved by typing 300.3A0W and then subsequently loaded back in by typing 300.3A0R (for type use). For disk it can be saved with, "BSAVE LAZARUS,A$300,L$3A0" and of course for loading "BLOAD LAZARUS".

LAZARUS (CONT.)

LAZARUS is very easy to use. After losing a program, enter the monitor and load LAZARUS. Then return to BASIC and CALL 768 or CALL 772. The former will cause the longest program segment in workspace to be restored; the latter will cause the last to be selected. When it returns, do LIST to see what is found. If both entry points recover the wrong program, you can adjust LOMEM & HIMEM and try again.

QUICKIE TIP

BY GLEN HOAG

To access the assembler directly from basic:

CALL -2458
To return to basic from the assembler:

!$ (CTRL C) (RETURN)

EDIT +

WHO NEEDS AUTOSTART ROM

BY GARY LITTLE

TAKEN FROM APPLEGRAM

As all APPLE II PLUS users and other owners of the AUTOSTART ROM chip know, the editing features which that chip provides are tremendously superior to those found on the ordinary APPLE II system. In particular, it allows the user to easily freeze or terminate output to the video screen and it also eliminates the need to repeatedly press the ESC key in order to perform multiple pure-cursor movements.

Since I write a lot of BASIC programs I knew that these features would be of great use to me, but before rushing out to buy the chip I decided to try to write a machine language program which would duplicate these features.

The program that I have come up with not only allows for the 'AUTOSTART' editing features, but also for two additional useful features:

1. By pressing a control character, it is possible to quickly copy over text from the current cursor position to the end of the line, thus eliminating the need to repeatedly press the right-arrow key.
2. By pressing a control character, the screen width can be changed to 33 (i.e., a POKE 33,33 COMMAND RESULTS). this feature is useful when editing program lines which, when listed with the window width set at its normal value of 40, would contain unwanted blanks between string-variable quotation marks.

The program, which I call EDIT+, exits in two versions, each of which is listed below. The DISK version is to be used whenever DOS 3.2 has been booted and is active; otherwise, the TAPE version is to be used. The DISK version resides in memory from $300 to $3CF and the TAPE version resides from $300 to $3C3. In order to avoid unpredicable results, EDIT+ must be deactivated (by pressing RESET) before using these memory locations for any other purpose.

Either version of EDIT+ is to be loaded into memory starting at location $300. For details on how to enter machine language code into memory, see page 68 of the APPLE II Reference Manual (red book).

The detailed operating instructions for EDIT+ are as follows:

1. After EDIT+ has been loaded into memory, the TAPE version can be saved to tape by entering the command 300.3C3W from the system monitor. The DISK version can be saved to diskette by entering the command BSAVE EDIT+, A$300, L$D0 from BASIC immediate execution mode.

2. To activate the TAPE version of EDIT+, load it from tape via a 300.3C3R command from the system monitor, enter the command 300G, and then enter BASIC by entering CTRL C (or 0G for RAM APPLESOFT programs). To activate the DISK version, enter the command BRUN EDIT+ from BASIC immediate execution mode (or enter BLOAD EDIT+, press RESET to enter the system monitor, and then enter the command 300G3D0G).

3. To use EDIT+, follow these procedures:
Freezing, continuing, and halting output to the video screen:
   (a) Press CTRL S in order to freeze the output to the video screen.
   (b) Press any key in order to continue the output to the video screen after it has been frozen.
   (c) Press CTRL C in order to halt the output to the video screen and enter BASIC immediate execution mode.

Pure cursor motions and screen clears:
   Press ESC in order to activate the cursor-motion and screen-clear keys. The cursor-motion and screen-clear keys and their functios are as follows:

      I -- Move cursor up
      J -- Move cursor left
      K -- Move cursor right
      M -- Move cursor down
      E -- Clear to end of line
      F -- Clear to end of page

   To deactivate the cursor-motion and screen-clear keys, press a key other than I, J, K, M, E, or F. This key will not be displayed.

Copying over text to the end of a line:

   Press CTRL Z in order to copy over the text from the current cursor position to the end of the line.

Setting the window width to 33:

   Press CTRL Q in order to perform the equivalent of a POKE 33,33 command.

4. To deactivate EDIT+, press the RESET key. (EDIT+ will also be deactivated by PR# and IN# commands from BASIC).

Special note for RAM APPLESOFT users:

To use the TAPE version of EDIT+ in conjunction with a RAM APPLESOFT program, the last two bytes of the TAPE version must be changed from 03 E0 to 00 00. This will ensure a proper jump back into APPLESOFT immediate execution mode when CTRL C is pressed after the output to the video screen has been frozen.

How EDIT+ works:

The EDIT+ program uses the APPLE II input and output hooks in order to invoke the editing and stop list features. For details of the algorithms involved, see the annotated dissembled listing reproduced below.

HEX DUMP (DISK VERSION)
>CALL-151

*300.3CF

```
0300- A9 13 85 38 A9 03 85 39
0308- 85 37 A9 A7 85 36 A9 00
0310- 85 08 60 20 1B FD 24 08
0318- 30 0B C9 9B D0 59 A9 BF
0320- 85 08 A9 9B 60 C9 C9 D0
0328- 04 A9 C4 30 24 C9 CA D0
0330- 04 A9 C2 30 1C C9 CB D0
0338- 04 A9 C1 30 14 C9 CD D0
0340- 04 A9 C3 30 0C C9 C5 D0
0348- 04 A9 C5 30 04 C9 C6 D0
0350- 0B 24 08 70 0D 48 A9 FF
0358- 85 08 68 60 48 A9 00 85
0360- 08 68 85 09 68 85 06 68
0368- 85 07 68 A9 31 48 A5 07
0370- 48 A5 06 48 A5 09 60 C9
0378- 91 D0 07 A9 21 85 21 A9
0380- A0 60 C9 9A D0 20 A4 24
0388- B1 28 48 E6 24 E6 24 A5
0390- 24 C5 21 B0 0C C6 24 68
0398- 9D 00 02 E8 D0 E8 4C 62
03A0- FD 68 C6 24 C6 24 60 C9
03A8- 8D D0 1E 48 AD 01 C0 8D
03B0- 11 C0 10 14 C9 83 F0 14
03B8- C9 93 D0 0C AD 01 C0 10
03C0- FB 8D 11 C0 C9 83 F0 04
03C8- 68 4C F0 FD 68 4C D0 03
```

HEX DUMP (TAPE VERSION)

>CALL-151

*300.3C3

```
0300- A9 13 85 38 A9 03 85 39
0308- 85 37 A9 9B 85 36 A9 00
0310- 85 FE 60 20 1B FD 24 FE
0318- 30 0B C9 9B D0 4D A9 BF
0320- 85 FE A9 9B 60 C9 C9 D0
0328- 04 A9 C4 30 24 C9 CA D0
0330- 04 A9 C2 30 1C C9 CB D0
0338- 04 A9 C1 30 14 C9 CD D0
0340- 04 A9 C3 30 0C C9 C5 D0
0348- 04 A9 C5 30 04 C9 C6 D0
0350- 0B 24 FE 70 0D 48 A9 FF
0358- 85 FE 68 60 48 A9 00 85
0360- FE 68 85 FF 68 A9 31 48
0368- A5 FF 60 C9 91 D0 07 A9
0370- 21 85 21 A9 A0 60 C9 9A
0378- D0 20 A4 24 B1 28 48 E6
0380- 24 E6 24 A5 24 C5 21 B0
0388- 0C C6 24 68 9D 00 02 E8
0390- D0 E8 4C 62 FD 68 C6 24
0398- C6 24 60 C9 8D D0 1E 48
03A0- AD 01 C0 8D 11 C0 10 14
03A8- C9 83 F0 14 C9 93 D0 0C
03B0- AD 01 C0 10 FB 8D 11 C0
03B8- C9 83 F0 04 68 4C F0 FD
03C0- 68 4C 03 E0
```

FREE SPACE

BY ANDY HERTZFELD

This program calculates the amount of free space available on a given diskette. It prints out the answer in terms of kilobytes and also in terms of sectors (there are 256 bytes per sector).

The program performs its task by using the RWTS routine to read in track 17, sector 0, which contains the bit-map used by the DOS for sector allocation. It then simply loops through and counts the number of bytes that are set, thereby obtaining the number of free sectors.

One problem is that the program must jump back to BASIC ($E003) when it's finished or else it can't be BRUN from command level. Thus, to use it from a BASIC or machine language program, you should change 088C (4c 03 E0) to "RTS" instead of "JMP $E003".

The program lives at $800 and is about 250 bytes long. It should work on a system of any memory size but I've only tried it on a 48K machine. To use it, type "BRUN FREE SPACE" or BLOAD it, insert the disk you want the free space of, and type "CALL 2048". After entering the HEX DUMP save to disk by BSAVE FREE SPACE,A$800,L$900.

HEX DUMP (FREE SPACE)

```
*800.900

0800- 48 8A 48 98 48 A9 00 8D
0808- BF 08 8D C0 08 20 E3 03
0810- 84 1C 85 1D A9 00 A0 03
0818- 91 1C A0 05 91 1C A9 11
0820- 88 91 1C A5 4C A0 08 91
0828- 1C A5 4D C8 91 1C A0 0C
0830- A9 01 91 1C A9 00 C8 91
0838- 1C 20 E3 03 20 D9 03 A0
0840- 0D B1 1C F0 03 4C 2D FF
0848- A0 38 B1 4C A2 07 0A 90
0850- 08 EE BF 08 D0 03 EE C0
0858- 08 CA 10 F2 C8 C0 C4 90
0860- E9 A9 8D 20 ED FD A9 8D
0868- 20 ED FD AE BF 08 AD C0
0870- 08 20 1B E5 A0 00 B9 C1
0878- 08 F0 08 09 80 20 ED FD
0880- C8 D0 F3 18 A9 02 6D BF
0888- 08 8D BF 08 90 04 EE C0
0890- 08 18 6E C0 08 6E BF 08
0898- 18 6E C0 08 6E BF 08 AE
08A0- BF 08 AD C0 08 20 1B E5
08A8- A0 00 B9 E7 08 F0 08 09
08B0- 80 20 ED FD C8 D0 F3 68
08B8- A8 68 AA 68 4C 03 E0 7D
08C0- 00 20 46 52 45 45 20 53
08C8- 45 43 54 4F 52 53 20 52
08D0- 45 4D 41 49 4E 20 8D 57
08D8- 48 49 43 48 20 49 53 20
08E0- 41 42 4F 55 54 20 00 4B
08E8- 20 4F 52 20 53 4F 20 8D
08F0- 8D 00 08 14 16 01 42 28
08F8- 49 29 D0 49 C9 31 3A 4C
0900- 49
```

FREE SPACE FORMALIZED

BY LE ROY LARSEN

I came up with the following few lines that you may like to try...

```
 5 TEXT : CALL -936
 7 PRINT : PRINT : PRINT
10 PRINT "THIS PROGRAM WILL RUN FREE
   SPACE": PRINT
15 PRINT "BRUN FREE SPACE, A2048"
20 PRINT : PRINT
25 PRINT "INSERT DISK YOU WANT TO FIND
   FREE SPACE": PRINT
30 PRINT "CALL 2048"
35 PRINT : PRINT : PRINT : PRINT
   PRINT : PRINT
40 PRINT "GOTO 45"
45 VTAB 10
50 END
60 REM
65 REM FREE SPACE LOCATION IS
70 REM AT 2048 TO 3568
75 REM
80 REM TO LOAD USE
85 REM A2048, L1250
90 REM
95 REM BY LEROY W.  LARSEN
98 END
```

SON OF N

BY SCOT KAMINS

Last month's issue of the Cider Press semi-featured a disk "HELLO" program that did many nifty things. Here is a new version of the program that includes reset for himem and lomem.

```
 0 POKE -16298,0: REM CLEAR HIRES
   SCREEN
10 TEXT : REM SET TEXT MODE, CLEAR
   SCROLLING WINDOWS
20 D$="": REM CTRL D, ALLOWING EASY
   USE OF DISK COMMANDS
30 PRINT D$;"NOMON C,I,O": REM HIDE
   ALL DISK COMMANDS
40 CALL -936: REM CLEAR THE SCREEN
50 POKE 74,0: POKE 75,8: REM SET
   LOMEM TO 2048 (THE USUAL PLACE)
60 POKE 76,0: POKE 77,150: REM SET
   HIMEM TO 38400 (48K 'PUTERS ONLY)
70 PRINT D$;"CATALOG": REM SHOWS
   WHAT'S ON THIS DISK
80 PRINT D$;"BRUN FREE SPACE": REM
   SHOWS HOW MANY FREE DISK BYTES
   REMAIN
```

Note: Line 60 pokes will have to be changed if your system is less than 48K. To find out what numbers you should use, merely boot your DOS and print PEEK (76) and PEEK (77). The numbers that are returned are the ones you should POKE into their respective places.

Sibling of note: Line 80 assumes you have added to your disk Andy Hertzfeld's excellent "Free Space" program, the code for which is to be found elsewhere in this issue.

SPLIT CATALOG

FROM CALL -APPLE

This program will give you a split catalog when you boot your disk if you use it to INIT your disk. You can also run it and it will give you a split catalog. If you have a great deal of programs on your disk you will see them all with this utility.

This program has come from the January 1979, page 20 Issue of CALL -APPLE.

```
>LIST

 10 POKE 768,56: POKE 769,72: POKE
    770,132: POKE 771,0: POKE 772,160:
    POKE 773,0: POKE 774,201: POKE
    755,32: POKE 776,176: POKE 777,2
 20 POKE 778,105: POKE 779,192: POKE
    780,201: POKE 781,96: POKE 782,
    176: POKE 783,2: POKE 784,105:
    POKE 785,128: POKE 786,201: POKE
    787,128
 30 POKE 788,176: poke 789,2: POKE 790
    ,105: POKE 791,64: POKE 792,145:
    POKE 793,2: POKE 794,230: POKE 795
    ,2: POKE 796,208: POKE 797,2
 40 POKE 798,230: POKE 799,3: POKE 800
    ,104: POKE 801,164: POKE 802,0:
    POKE 803,96: POKE 804,0: POKE 805,
    0: POKE 806,0: POKE 807,0
 50 POKE 808,169: POKE 809,160: POKE
    810,164: POKE 811,36: POKE 812,
    145: POKE 813,40: POKE 814,96
100 B$=" ":Q=0:  DIM A$(250),A(50),
    B(50):D$="":LM=11+ PEEK (74)+
    PEEK (75)*256: POKE LM, PEEK (977)
    : POKE LM+1,PEEK (978):Q=Q+3037
110 PRINT D$;"NOMON C,I,O": TEXT :
    CALL -936: POKE 2,0: POKE 3,16:
    POKE Q,0: POKE Q+1,3: POKE Q+2,40:
    POKE Q+4,3
120 PRINT D$;"CATALOG": PRINT ""

130 PRINT D$;"PR#0":  PRINT D$;"IN#0"

140 PRINT " "
200 POKE 34,2:I=4096
210 J=1
220 POKE LM-6, PEEK (I):  IF ASC(B$) =
    133 THEN 400: IF ASC(B$) = 141
    THEN 240: IF B$=D$ OR J>19 THEN
    250: IF K<2 THEN 230: PRINT B$;
230 I=I+1:J=J+1: GOTO 220
240 PRINT :I=I+1:K=K+1: VTAB 3+(K-2)
    MOD 20: IF K=22 THEN GOSUB 800:
    GOTO 210
250 I=I+1: GOTO 230
400 TEXT : VTAB 23: END
800 POKE 33,19: POKE 32,19: CALL -936:
    RETURN
```

S.F. APPLE CORE
"Best of The Cider Press"
PROGRAM DISK

The **"Best of the Cider Press"** program disk contains most of the programs printed in this publication. (Checkbook, edit +, FRE(x) and a great deal more.)

Name _____

Street _____

City _____ State _____ Zip _____

Send $7.50 which includes handling and mailing in U.S.A.
Add $2.00 for Foreign Air Mail
Make Checks payable to: **"S.F. Apple Core"**
and send to: 1515 Sloat Blvd., Suite 2
San Francisco, CA 94132

INTERNATIONAL APPLE CORE TM

APPLE ORCHARD SUBSCRIPTIONS

P. O. BOX 2227 SEATTLE, WASHINGTON 98111, USA

The International Apple Core will make individual subscriptions to "The Apple Orchard" available commencing with Volume I, Number 2 to be published in September, 1980.

NAME _____

STREET _____

CITY _____ STATE _____ ZIP _____

COUNTRY _____

Annual Subscription Rate: $10.00 per year
First Class Postage: $5.00 per year additional (required for Canada, Mexico, APO, and FPO addresses)
Overseas and other foreign air mail postage (required): $10.00 per year additional

TOTAL REMITTANCE ENCLOSED: $(USA) _____

Make check or money order payable to "International Apple Core" and return with this form to:
Apple Orchard Subscriptions
P.O. Box 2227
Seattle, Washington, USA 98111

7/7/80

SAN FRANCISCO APPLE CORE LIBRARY

CONVENTIONS

BY LE ROY LARSEN

The following are conventions that are currently being used by the Apple Core. This is a first attempt toward establishing conventions with which we can live. If you have any input on this subject please write the Apple Core in care of the Conventions Committee.

PROGRAM REQUIREMENTS

1. All programs submitted must contain your name (either in REM or PRINT statements) within the program.

2. If at all possible the following Conventions should be used:

a) Programs should begin with a POKE -16298,0 to turn off the Hi-Res mode; it should be located in the first 5 executed lines.

b) The first thing to be displayed after RUN should be the title page with the program name, author, date, two or more lines of what the program does, and the choice of continuing or ending (see how below).

c) Timing loops in programs should be avoided. Use "PRESS SPACE BAR TO CONTINUE" wherever possible.

d) Pressing 'ESC' should return user to next higher level menu or, if no menu, exit (end) program.

e) Try to include REM statements in your program so that our members can learn by your programming. If for reasons of speed you can not do this please include written information about your program.

f) Put a REM statement in the last line of the program with the TITLE of the program.

g) At the end of a program the following should be reset: LOMEM to 2048, TEXT page 1 by POKE -16300,0, Hi-Res mode off by POKE -16298,0, and clear screen and set full text window by TEXT & CALL -936, clear keyboard strobe with POKE -16368,0.

The preceeding conventions are our attempt to put some standards into our programs within the library. If you have any input on this subject please write to us - good programming helps us all.

EXAMPLE OF TITLE PAGE PROGRAMMING:

```
 5 DIM C$(40)
10 POKE -16298,0: TEXT : CALL -936:
   GOTO 20
11 REM PRINT CENTER ROUTINE
12 HTAB 20 - LEN (C$)/2 (Use TAB in
   Integer)
13 PRINT C$: REM PRINTS LINE
14 PRINT " ": REM "CONTROL G"
15 RETURN
20 VTAB 5: C$="Title of program":
   GOSUB 11
30 C$="BY - authors name": GOSUB 11
40 C$="DATE - donated to library":
   GOSUB 11
45 PRINT : C$="A few lines about
   program": GOSUB 11
50 VTAB 21: PRINT "PRESS 'ESC' TO END"
55 PRINT "< PRESS THE SPACE BAR TO
   CONTINUE...>"
60 IF PEEK (-16384) = 155 THEN (To end
   of program)
65 IF PEEK (16384) <> 160 THEN 50
90 CALL -936: POKE -16368,0: REM
   RESETS KEYBOARD
```

(This would be the line to start program - lines 11 thru 15 could be located any place)

END OF PROGRAM LINES:
(Refer to part g above)

```
LINE ? POKE -16289,0: POKE -16300,0
LINE ? POKE -16368,0: TEXT : CALL -936
LINE ? VTAB 10: C$= "* * * THE END *
       * *": GOSUB 11: END
```

LIBRARY #1 PURCHASE

BY KEN SILVERMAN

The club now has completed its first book of contributed software. The book consists of over 350 programs on 20 diskettes. The diskettes are set up by category, i.e. utility, math, games, etc. The book is also made up of disk holders, Apple Core binder and about 30 pages of information.

This library #1 can be purchased by a member for $150.00. If you are not a member add $15.00 and mail with a membership form.

A complete listing of the contents of library #1 can be obtained for $1.00 at any Apple Core meeting. Out of town members can obtain it by sending the $1.00 and a 9" x 12" self addressed envelope with $1.00 postage on it to the Apple Core.

The club is now starting library #2 starting with the software on the November 1979 DOM.

LIBRARY ACCESS

The APPLE CORE LIBRARY of contributed programs is arranged by general categories. Members living in the S.F. Bay area may copy programs from the library at the following locations:

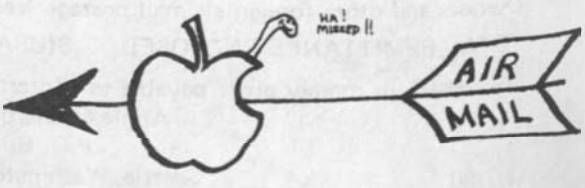
Computerland San Francisco	546-1592
Computerland of Belmont	595-4232
Computerland of Marin	459-1767
Computerland of The Castro	864-8080
Computerland of Los Altos	221-8500
AIDS	221-8500

Out of area members can get programs from the library through the mails in the following manner:
1. A member is required to donate at least one original or public domain program (not Copyrighted, please).
2. Donated programs must be sent on a disk or a computer tape placed in self-addressed, stamped proper mailer, suitable for returning the disk or tape. Please use a Program Submission form. Include a note indicating the desired volume from the library that you would like to have copied. Carefully package the mailer and Note:

CONTAINS LIVE COMPUTER PROGRAMS DO NOT EXPOSE TO X-RAYS OR ELECTRICAL FIELDS DO NOT BEND OR FOLD.

Please follow instructions as we do not want to see your disks or tapes ruined anymore than you do. Only one library disk or tape will be processed per month. The DOM (Disk of the Month) is considered separately.

The DISK OF THE MONTH is a group of recently donated programs or updated utilities, ect. It was originated to encourage newmembers to be able to write programs by having examples to study and enjoy.
Members unable to come to the meetings can send in $7.50(US) for the current DOM which covers the cost of the disk, mailing and handling. Three past months are also available for $7.50 each.
Members who come to the meetings can obtain the same DOM's for $5.00 each.
Prices are subject to change.
NOTE: All programs on the DOM's go into the library according to category. The stores do not have the DOM's on file.



LIBRARY BOOK #1

NOTES

1) PROGRAMS WITH (A) AFTER THEM REQUIRE THE PROGRAMMERS'S AID CHIP

2) PROGRAMS WITH A (P) AFTER THEM WILL OUTPUT TO A PRINTER (PR#1).

3) PROGRAMS WITH A (D) REQUIRE A DISK DRIVE.

4) ALL APPLE CORE DISKS ARE 3.2 MASTERS AND WILL 'BOOT' ON ANY SIZE SYSTEM.

PRESS ANY LETTER TO CONTINUE

PROGRAM INDEX PAGE 0

NO	CTGRY	DISK #	PROGRAM NAME

ENTER THE NUMBER OF THE FIRST PROGRAM TO BE LISTED OR PRESS RETURN:
?
ENTER THE CATEGORY TO BE LISTED (1-10) OR PRESS RETURN TO LIST THEM ALL
?
ENTER THE FIRST LETTER(S) OF THE PROGRAM(S) TO BE LISTED OR PRESS RETURN:
?

PROGRAM INDEX PAGE 1

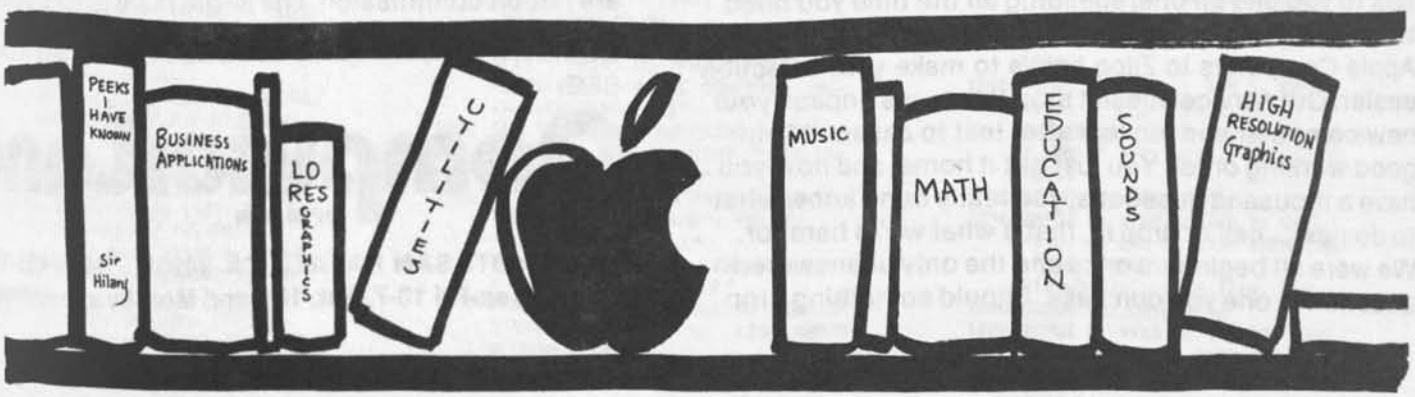
NO	CTGRY	DISK #	PROGRAM NAME
1	1	DISK 0A	PROGRAM LIST
2	1	DISK 0A	LIST INSTRUCTIONS
3	1	DISK 0A	LIBRARY POLICIES
4	1	DISK 1A	APPLE HELLO
5	1	DISK 1A	TEXTFILER
6	1	DISK 1A	TEXTCOPY
7	1	DISK 1A	SWEET 16 DISSEMBLER
8	1	DISK 1A	INTEGER RENUMBER/AP
9	1	DISK 1A	SLOWLIST
10	1	DISK 1A	LAZARUS.A768.L251
11	1	DISK 1A	CTRLFIND
12	1	DISK 1A	DISK RENUM APPEND
13	1	DISK 1A	TONY'S SUBROUTINE P
14	1	DISK 1A	APPLESOFT 1 TO 2 CO
15	1	DISK 1A	YES NO AND PAUSE
16	1	DISK 1A	CHR$ FUNCTION
17	1	DISK 1A	HEAPSORT
18	1	DISK 1A	ALPHABETIZE
19	1	DISK 1A	RANDOM SORT
20	1	DISK 1A	B/BSTAT
21	1	DISK 1A	FREE SPACE
22	1	DISK 1A	FIX CATALOG
23	1	DISK 1A	DISK MAP
24	1	DISK 1A	IMPROVED CATALOG
25	1	DISK 1A	LOCK DISK
26	1	DISK 1A	DISK PROGRAM ELIMIN
27	1	DISK 1A	DISK DUMP
28	1	DISK 1A	DISK AIDE [APPLE CO
29	1	DISK 1A	DISK AIDE [DOCUMENT
30	1	DISK 1A	DISK AIDE [MACHINE
31	1	DISK 1A	SUPERCATALOG.O
32	1	DISK 1A	SUPERCATALOG.DOC
33	1	DISK 1A	PASSWORD KEY
34	1	DISK 1B	DOS UTILITY #1
35	1	DISK 1B	MENU WRITER
36	1	DISK 1B	RANDOM TEST
37	1	DISK 1B	MULTICOPY
38	1	DISK 1B	CIDER SQUEEZER 3.1
39	1	DISK 1B	SLOW MEM TEST
40	1	DISK 1B	FAST MEM TEST
41	1	DISK 1B	HEX CODES LIST
42	1	DISK 1B	SHORTEN COMMANDS
43	1	DISK 1B	QUOTES IN INTEGER
44	1	DISK 1B	FORMAT-SUBROUTINE
45	1	DISK 1B	APPLETEST
46	1	DISK 1B	F
47	1	DISK 1B	RND NO FREQ
48	1	DISK 1B	MEMORY SPY
49	1	DISK 1B	DISC SPEED INFO
50	1	DISK 1B	DISC SPEED TEST
51	1	DISK 1B	DSPEED.OBJ
52	1	DISK 1B	BASIC TED PRINTER
53	1	DISK 1B	TED3
54	1	DISK 1B	TED START
55	1	DISK 1B	TED.DISK
56	1	DISK 1B	COPY FILE
57	1	DISK 1B	DISK COPY 3.2
58	1	DISK 1B	THE ALCHEMIST V2.7
59	1	DISK 1B	ASCII DECODE
60	1	DISK 1B	L O O P
61	2	DISK 2A	AVELAR'S FINANCIAL
62	2	DISK 2A	LINEAR PROGRAMMING
63	2	DISK 2A	PAYROLL/V4
64	2	DISK 2A	NETFLOW COST
65	2	DISK 2A	STOCK OPTIONS
66	2	DISK 2A	PAYROLL
67	2	DISK 2A	TEN KEY ADDER
68	2	DISK 2A	TEN KEY NOTES
69	2	DISK 2A	LISTS
70	2	DISK 2A	AUTO PURCHASE
71	2	DISK 2A	APT BLDG PURCHASE
72	2	DISK 2A	PORTFOLIO REVIEW DO
73	2	DISK 2A	PORTFOLIO REVIEW
74	2	DISK 2B	TAX 1040 '79
75	2	DISK 2B	DECISION MAKER
76	2	DISK 2B	SIMPLER INTEREST
77	2	DISK 2B	FUTURE VALUE
78	2	DISK 2B	PRESENT AND FUTURE
79	2	DISK 2B	HOME MORTGAGE
80	2	DISK 2B	INVESTMENT EVAL.
81	2	DISK 2B	TELEPHONE
82	2	DISK 2B	TRAVEL COSTS
83	2	DISK 2B	CRITICAL PATH ANALY
84	2	DISK 2B	CALL DEBT MARKER
85	2	DISK 2B	FED TAX PROGRAM
86	2	DISK 2B	FORECASTING
87	2	DISK 2B	COMPOUND INTEREST
88	2	DISK 2B	NEW CHECKBOOK
89	3	DISK 3A	MUSIC FOR CLOSE ENC
90	3	DISK 3A	JOHANN SEBASTIAN AP
91	3	DISK 3A	APPLEODIAN
92	3	DISK 3A	LUDWIG'S FANTASY
93	3	DISK 3A	BACH
94	3	DISK 3A	SMALL SMALL WORLD
95	3	DISK 3A	ALLEY CAT
96	3	DISK 3A	STING MUSIC
97	3	DISK 3A	MUSIC WRITER
98	3	DISK 3A	THE HART PIANO
99	3	DISK 3A	ODE TO JOY
100	3	DISK 3A	MUZAK
101	3	DISK 3A	SQUARE BACH
102	3	DISK 3A	SOUND EFFECTS
103	3	DISK 3A	MOZART 2 VOICE
104	3	DISK 3A	FLAG
105	3	DISK 3A	APPLE PIANO DOC

LIBRARY BOOK #1 (CONT.)

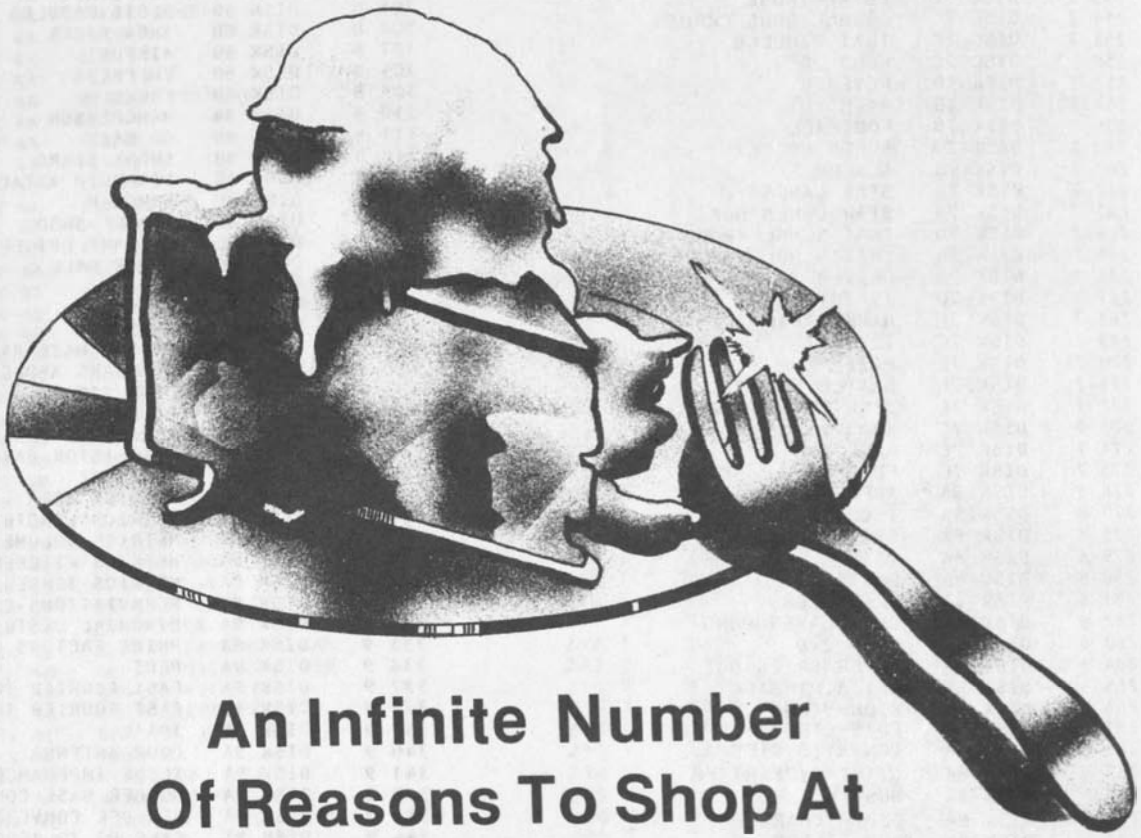
106 3	DISK 3A	APPLE PIANO
107 3	DISK 3A	ANDY'S TONES
108 3	DISK 3A	PHILA ORGAN
109 3	DISK 3A	MULTI TONE
110 4	DISK 4A	LOGO
111 4	DISK 4A	GET IT
112 4	DISK 4A	PLOT.3-D
113 4	DISK 4A	BESSEL DISPLAY
114 4	DISK 4A	PLOT1
115 4	DISK 4A	PLOT2
116 4	DISK 4A	PLOT3
117 4	DISK 4A	PLOT4
118 4	DISK 4A	PLOT5
119 4	DISK 4A	PLOT6
120 4	DISK 4A	PLOT7
121 4	DISK 4A	PLOT8
122 4	DISK 4A	PLOT9
123 4	DISK 4B	EASTER EGG(CALL5451
124 4	DISK 4B	MENSCHELL HI-RES
125 4	DISK 4B	STAR PATTERN
126 4	DISK 4B	VINCENT (D)
127 4	DISK 4B	HI-RES LOW LEVEL (D
128 4	DISK 4B	MT. FUJI (A)
129 4	DISK 4B	PAINTER (A)
130 4	DISK 4B	SAMPLER (D)
131 4	DISK 4B	MEMORY ORGANIZATION
132 4	DISK 4B	RANDOM ELEPHANT
133 4	DISK 4B	SHAPE MEDLEY
134 4	DISK 4B	DRAGON LOAD (D)
135 4	DISK 4B	DRAGON (D)
136 4	DISK 4B	TERMINAL-W
137 4	DISK 4B	SHTABLE LOC (D)
138 4	DISK 4B	CHARACTER SHTABLE (
139 4	DISK 4B	ALEX'S CALENDAR PAR
140 4	DISK 4B	ALEX'S CALENDAR PAR
141 4	DISK 4B	CHOICE FILE (D)
142 4	DISK 4C	CHARACTERS.PIC (D)
143 4	DISK 4C	SEE PIC (D)
144 4	DISK 4C	ROSE D
145 4	DISK 4C	HIRES.OBJ (D)
146 4	DISK 4C	WASHINGTON (D)
147 4	DISK 4C	LINCOLN (D)
148 4	DISK 4C	SQUARES
149 4	DISK 4C	HI-RES 21 COLORS
150 4	DISK 4C	ETCH-A-SKETCH
151 4	DISK 4C	IMPOSSIBLE FIGURE
152 5	DISK 5A	TYPING PRACTICE
153 5	DISK 5A	CAL DRIVER TEST 24K
154 5	DISK 5A	MORSE TRAINER
155 5	DISK 5A	MORSE CW
156 5	DISK 5A	FLASH CODE
157 5	DISK 5A	FLASH CARD
158 5	DISK 5A	NAME STATES
159 5	DISK 5A	STATES/CAPITALS
160 5	DISK 5A	COLOR MATH
161 5	DISK 5A	MATH TUTOR
162 5	DISK 5A	ECHOCARDIOGRAPH
163 5	DISK 5A	INTGER INSTRUCTION
164 5	DISK 5A	TITRATION
165 5	DISK 5A	TOP DOWN PROGRAMMIN
166 5	DISK 5A	CONVENTIONS
167 5	DISK 5B	QUIZBUILD (D)
168 5	DISK 5B	QUIZ (D)
169 5	DISK 5B	SEQUENCE.EASY
170 5	DISK 5B	SEQUENCE.HARD
171 5	DISK 5B	US.CAPITALS
172 5	DISK 5B	EUROPEAN.CAPITALS
173 5	DISK 5B	ASIAN.CAPITALS
174 5	DISK 5B	MISSPELL
175 5	DISK 5B	SHAKESPEARE
176 5	DISK 5B	BABY.ADULT
177 5	DISK 5B	VICTIM.KILLER
178 5	DISK 5B	CANADA.PROV
179 5	DISK 5B	AMERICAN.CAPITALS
180 5	DISK 5B	MALE.FEMALE
181 5	DISK 5B	PRESIDENTS
182 5	DISK 5B	AFRICAN.CAPITALS
183 5	DISK 5B	OPPOSITES
184 5	DISK 5B	ELEMENTS
185 6	DISK 6A	DOT WORLD
186 6	DISK 6A	COLOR SHOWS
187 6	DISK 6A	ROD'S COLOR PATTERN
188 6	DISK 6A	SUPER KALEIDOSCOPE
189 6	DISK 6A	GREAT AMER.PROB.MAC
190 6	DISK 6A	COLOR GRAPHICS
191 6	DISK 6A	COLOR SKETCH
192 6	DISK 6A	COLOSSUS
193 6	DISK 6A	T.V.PATTERN GENERAT
194 6	DISK 6A	BIT BUCKET
195 6	DISK 6A	TITLE PROGRAM
196 6	DISK 6A	COLOR STROBE
197 6	DISK 6A	ENTERPRISE
198 6	DISK 6A	PAGE 1&2
199 6	DISK 6A	FWORMS (D)
200 6	DISK 6A	FAST WORMS (D)
201 6	DISK 6A	WORMS (D)
202 6	DISK 6A	BEEP (D)
203 6	DISK 6A	A???
204 6	DISK 6A	ANIMATION
205 6	DISK 6A	OBJECT DRAWING
206 6	DISK 6A	DANCING BUTTERFLIES
207 6	DISK 6A	MOD GREETING
208 6	DISK 6A	RANDY'S PATTERN
209 7	DISK 7A	INFINITE NUMBER OF
210 7	DISK 7A	POETRY
211 7	DISK 7A	PLAYBOY BUNNY
212 7	DISK 7A	HAIKU POETRY
213 7	DISK 7A	TEXT BIORHYTHM
214 7	DISK 7A	APPLE POOP PLOT
215 7	DISK 7A	BUZZWORD
216 7	DISK 7A	LOVE
217 7	DISK 7A	HEBREW POSTER
218 7	DISK 7A	DAY OF WEEK
219 7	DISK 7A	MADLIB
220 7	DISK 7A	SLOT TEXT
221 7	DISK 7A	SLOT DICE
222 7	DISK 7A	PLANTMAN
223 7	DISK 7A	MADAME DUPRE
224 7	DISK 7A	BIORHYTHM FOR PRINT
225 7	DISK 7A	OREGON TRAIL
226 7	DISK 7A	PLANTS & LEMONADE
227 7	DISK 7B	STARTREK 8K
228 7	DISK 7B	LUNAR LANDER
229 7	DISK 7B	TEXTBLACKJACK
230 7	DISK 7B	COIN TOSS
231 7	DISK 7B	LUNAR
232 7	DISK 7B	GAMES SULLIVAN
233 7	DISK 7B	BAGELS
234 7	DISK 7B	WORLD POWER
235 7	DISK 7B	KINEMA
236 7	DISK 7B	CRYPTOGRAM
237 7	DISK 7B	DYNASTY
238 7	DISK 7B	MATCHGAME
239 7	DISK 7B	APPLE LEAF GAME
240 7	DISK 7B	DEPTH CHARGE
241 7	DISK 7B	ELIZA
242 7	DISK 7B	TEXT CHESS
243 7	DISC 7C	GUESS A BMP
244 7	DISC 7C	TIC TAC TOE
245 7	DISC 7C	BOXING
246 7	DISC 7C	BASEBALL
247 7	DISC 7C	INTERSTELLAR
248 7	DISC 7C	CHASE
249 7	DISC 7C	SWORDS & SORCER
250 7	DISC 7C	GOLF
251 7	DISC 7C	SEARCH

LIBRARY BOOK #1 (CONT. 2)

252	7	DISC 7C	FIBOSTRAT
253	7	DISC 7C	PETALS/ROSE
254	7	DISC 7C	DOUBLE-ODDS CRAPS
255	7	DISC 7C	TEXT OTHELLO
256	7	DISC 7C	KENO 20
257	7	DISK 7D	REVERSE
258	7	DISK 7D	ATOM 20
259	7	DISK 7D	FOOTBALL
260	7	DISK 7D	SUPER HOCKEY
261	7	DISK 7D	SLALOM
262	7	DISK 7D	STAR LANES
263	7	DISK 7D	STAR LANES DOC
264	7	DISK 7D	THAT'S HOLLYWOOD IN
265	7	DISK 7D	THAT'S HOLLYWOOD
266	7	DISK 7D	DRIVER
267	7	DISK 7D	TV TRIVIA
268	7	DISK 7E	NAME FREAK
269	7	DISK 7E	IQ TEST
270	7	DISK 7E	PIZZA
271	7	DISK 7E	BATTLE SHIP
272	7	DISK 7E	JIMMIES ODD-EVEN
273	7	DISK 7E	SNOOPY POSTER
274	7	DISK 7E	RACE CAR
275	7	DISK 7E	FLIPPER
276	8	DISK 8A	ACEY DUCY
277	8	DISK 8A	I CHING
278	8	DISK 8A	STAY AFLOAT
279	8	DISK 8A	TRAP
280	8	DISK 8A	COOTIE
281	8	DISK 8A	SKI RACER
282	8	DISK 8A	ONE PLAYER PONG
283	8	DISK 8A	PONG 2-D
284	8	DISK 8A	SUPER BRICK-OUT
285	8	DISK 8A	PHILA.PINBALL
286	8	DISK 8A	BIORHYTHMS
287	8	DISK 8A	ED'S LIFE
288	8	DISK 8A	CONWAY'S LIFE
289	8	DISK 8A	GIANT TYPEWRITER
290	8	DISK 8A	HUSTLE
291	8	DISK 8A	DEPTH CHARGE
292	8	DISK 8A	SUB KILLER
293	8	DISK 8A	GAME OF LIFE
294	8	DISK 8B	DRAGON MAZE
295	8	DISK 8B	BREAKOUT/B
296	8	DISK 8B	DUCK SHOOT
297	8	DISK 8B	BINGO CARD
298	8	DISK 8B	BINGO
299	8	DISK 8B	AUTOMATIC BINGO
300	8	DISK 8B	THEIF1
301	8	DISK 8B	MISSION
302	8	DISK 8B	TORPEDO RUN
303	8	DISK 8B	SKUNK
304	8	DISK 8B	NEV CRAPS
305	8	DISK 8B	SLOTS-PADDLES
306	8	DISK 8B	SNOW RACER
307	8	DISK 8B	AIRPORT
308	8	DISK 8B	CHECKERS
309	8	DISK 8B	FREUD
310	8	DISK 8B	HANGPERSON
311	8	DISK 8B	GO BACK
312	8	DISK 8B	SHOOT STARS
313	8	DISK 8C	STARSHIP ATTACK
314	8	DISK 8C	HANGMAN
315	8	DISK 8C	TARGET SHOOT
316	8	DISK 8C	INSTANT LOTTERY
317	8	DISK 8C	DODGE BALL
318	8	DISK 8C	D N A
319	8	DISK 8C	CARWASH
320	8	DISK 8C	JUGGLE
321	8	DISK 8C	ANDY'S MAZE RACE
322	8	DISK 8C	STARWARS APPLE
323	8	DISK 8C	LIFE #1000
324	9	DISK 9A	CALCULATOR
325	9	DISK 9A	REACTANCE CALCULATI
326	9	DISK 9A	TRANSISTOR PARAMETE
327	9	DISK 9A	SIN PLOT
328	9	DISK 9A	METRICS AREA
329	9	DISK 9A	METRICS LENGTH
330	9	DISK 9A	METRICS VOLUME
331	9	DISK 9A	METRICS KITCHEN
332	9	DISK 9A	METRICS TEMPERATURE
333	9	DISK 9A	PERMUTATIONS-COMBIN
334	9	DISK 9A	BINOMIAL DISTRIBUTI
335	9	DISK 9A	PRIME FACTORS OF IN
336	9	DISK 9A	PLOT
337	9	DISK 9A	FAST FOURIER TRANSF
338	9	DISK 9A	FAST FOURIER TRANSF
339	9	DISK 9A	TOTAL
340	9	DISK 9A	LOOP ANTENNA
341	9	DISK 9A	XLINE IMPEDANCE
342	9	DISK 9A	NUMBER BASE CONVERT
343	9	DISK 9A	HEX-DEC CONVERTER
344	9	DISK 9A	CALC PI TO 1000 DIG
345	9	DISK 9A	LONG DIVISION
346	9	DISK 9A	DIVISION
347	9	DISK 9A	MULTIPLY
348	9	DISK 9A	MATH PRACTICE
349	9	DISK 9A	EXTERIOR BALLISTICS
350	9	DISK 9A	HARMONIC ANALYSIS
351	9	DISK 9A	QUADRATIC SURFACE
352	9	DISK 9A	HEX CONV.
353	9	DISK 9A	NOTCH FILTER
354	9	DISK 9A	GREAT CIRCLE



APPLE][.



An Infinite Number Of Reasons To Shop At ComputerLand of Marin

Once you have made your decision to buy an Apple Computer, now comes the big question...*where*? There is a lot more to buying a computer than just taking it home and plugging it in - that's where *we* come in: We talk to you one on one, spending all the time you need to help you select a system. We stock everything from Apple Computers to Zilog books to make your selection easier. Our service doesn't stop there - we unpack your new computer and run test after test to assure it is in good working order. You just get it home, and *now* you have a thousand questions, you really don't know what to do next... call or drop in, that's what we're here for. We were all beginners once and the only unanswered question is one you don't ask. Should something stop working (it can happen), we have a full-time technician to repair your maching *in* and *out* of warranty!

Last... but not least, are the people you deal with. They are not on commission, and all are there to help you!!

COME IN... WE'RE SURE YOU WILL LIKE WHAT YOU SEE.

ComputerLand®
OF MARIN

1930 4th ST., SAN RAFAEL, CA. 94901 (415) 459-1767
Open Tues-Fri 10-7, Sat. 10-5 and Mon. by appointment.

CHECKBOOK

THIS CHECKBOOK PROGRAM IS BASED ON THE ORIGINAL FROM THE APPLE COMPUTER COMPANY. IT HAS HAD SO MANY PATCHES PUBLISHED IN SO MANY NEWSLETTERS THAT NO ONE SEEMS TO KNOW WHAT WORKS AND WHAT DOESN'T. THIS LISTING OF THE ENTIRE PROGRAM WITH ALL THE FIXES FOR DISK AND PRINTER WE HOPE WILL TAKE CARE OF THE PROBLEM.

The SF APPLE CORE wishes to thank Howard Gannes for the disk modifications, Ken Silverman for modifying it for proper printer output and last but not least to John Couch of Apple Computer Company for permission to print the program.

LIST

```
  0 A=0: DIM C$(12),D(13),N$(40
    ),RA(13),R(13),L$(40): PRINT
    "NOMON I,O,C": PRINT "BLOAD FAST
    SORT": GOTO 1000
  0 POKE 204,252: POKE 205,8: GOTO
    1100
  1 IF CM+S+S>HM THEN 20000: FOR
    J=0 TO S:A=D(J): POKE CM+J+
    J-0, PEEK (2052): POKE CM+J+
    J, PEEK (2053): NEXT J:CM=CM+
    S+S: RETURN
  2 FOR J=0 TO S: POKE 2052, PEEK
    (P+J+J-2): POKE 2053, PEEK
    (P+J+J-0):R(J)=A: NEXT J: RETURN
  3 BCM=CM:CM=P-0: GOSUB 0:CM=BCM:
    RETURN
  4 FOR Q=B TO B+ LEN(C$)/2-0: POKE
    2052, ASC(C$(Q-B+Q-B+0)): POKE
    2053, ASC(C$(Q-B+Q-B+2)):D(
    Q)=A: NEXT Q: RETURN
  5 FOR Q=B TO B+SL/2-0:A=R(Q):
    POKE 2059+Q-B+Q-B, PEEK (2052
    ): POKE 2060+Q-B+Q-B, PEEK
    (2053): NEXT Q: POKE 2059+Q-
    B+Q-B,0: RETURN
  6 CALL -936: TAB (20- LEN(N$)
    /2): PRINT N$: PRINT "CHK# MO/DA
    /YR TO:";: TAB 30: PRINT "AMOUNT
    CODE";:L=3: RETURN
  7 TAB E: IF C<0 THEN CALL -1008
    : FOR A=0 TO 4: IF ABS (C)>=
    10 ^ A THEN CALL -1008: NEXT
    A: IF I<Z AND E>15 THEN GOSUB
    36: PRINT C;: RETURN
 10 IF I<Z OR F<Z THEN POKE 50,
    127:C= ABS (I): GOSUB 7: PRINT
    ".";: IF ABS (F)<10 THEN PRINT
    Z;: PRINT ABS (F);: POKE 50
    ,255:SG=0: RETURN
 11 P=P+S+S:E=4:C=R(0): TAB 0: CALL
    -868: GOSUB 7:E=7:C=R(2)/256
    : GOSUB 7: PRINT "/";:A=R(2
    ) MOD 256: IF A<10 THEN PRINT
    " ";: PRINT A;"/";
 12 I= ABS (R(3) MOD 256): IF I<
    10 THEN PRINT " ";: PRINT I;
    " ";:B=4:SL=T: GOSUB 5: PRINT
    C$;:E=T1:I=R(10):F=R(11): GOTO
    10
 13 CALL -936: TAB (20- LEN(N$)
    /2): PRINT N$: PRINT "CHK# MO/DA
    /YR TO:";: TAB (27): PRINT
    "AMOUNT BALANCE";:L=3: RETURN
 14 I=Z:F=Z: INPUT "?",L$: IF NOT
    LEN(L$) THEN 14: IF L$(0,0)
    #"-" THEN 15: IF LEN(L$)=0 THEN
    14:L$=L$(2, LEN(L$)):SG=-0
 15 FOR J=0 TO LEN(L$):A= ASC(L$
    (J))-176: IF A=28 THEN A=1:
    A=(A<>31)*A: IF A<Z OR A>9 OR
    I>3275 THEN 16:I=I*10+A*SG:
    NEXT J: RETURN
 16 IF A#-2 THEN 14: IF LEN(L$)
    =J THEN RETURN : FOR K=J+0 TO
    LEN(L$):A= ASC(L$(K))-176: IF
    A<Z OR A>9 THEN 14:F=F*10+A*
    SG: NEXT K
 17 IF J+0= LEN(L$) THEN F=F*10
    : IF F>99 THEN 14: RETURN
 18 INPUT "?",L$:A= LEN(L$): IF
    A>=S THEN A=12: IF NOT A THEN
    18:C$=L$(0,A)
 19 IF LEN(C$)<12 THEN C$( LEN(
    C$)+1)=" ": IF LEN(C$)<12 THEN
    19:B=4: GOSUB 4:SG=0: IF C$
    (0,0)="#" THEN SG=-0: RETURN
 20 INPUT "?",C$: IF LEN(C$)>4 THEN
    C$=C$(0,4)
 21 IF LEN(C$)<4 THEN C$( LEN(C$
    )+0)=" ": IF LEN(C$)<4 THEN
    21:B=12: GOTO 4
 22 PRINT "";: CALL -958: RETURN

 23 IF PEEK (-16384)<128 THEN RETURN
    : POKE -16368,0: POP :L=L+0:
    GOTO 1835
 24 IF PEEK (-16384)<128 THEN RETURN
    : POKE -16368,0: POP :L=L+0:
    GOTO Q
 30 PRINT : PR#0: VTAB 22: PRINT
    "HIT SPACE BAR TO STOP LISTING"
    : VTAB L: PR#PS: RETURN
 36 CALL -1008: IF I=BALI THEN
    PRINT "-";: IF I#BALI THEN
    PRINT "#";: RETURN
 40 RETURN
 97 A=Q+2*(Q>0)+(Q>2)+12*(Q>3)+
    2*(Q>4)+4*(Q>5): RETURN
100 FOR C=0 TO S:RA(C)=R(C): NEXT
    C:I=P: RETURN
1000 O=1:Z=A:S=13:LM=3000:CM=LM-
    0: POKE 2052, PEEK (202): POKE
    2053, PEEK (203):HM=A: IF A<
    Z THEN HM=32767
1035 GOTO 1050
1040 DEL 0
1050 POKE 8,32: POKE 9,12: POKE
    10,253: POKE 11,133: POKE 12
    ,255: POKE 13,96
1090 D$="": REM  CTRL D
1100 TEXT : PRINT D$;"PR#0": CALL
    -936: PRINT "APPLE COMPUTER INC-
    -----CHECKBOOK II"
1105 TAB 12: PRINT "CHECKING ACCOUNT"
    : PRINT : TAB 6: PRINT "DATA BAS
    E MANAGEMENT SYSTEM"
1106 PRINT : PRINT "TO PRINT OUTPUT,
    TYPE 'P' BEFORE THE": PRINT
    "NUMBER 2,3,4 OR 6"
1110 PRINT : PRINT " 1.  ENTER DATA"
    : PRINT " 2.  BALANCE"
1115 PRINT " 3.  RECONCILE TO BANK ST
    ATEMENT": PRINT " 4.  LIST AND/O
    R CHANGE DATA"
1120 PRINT " 5.  SORT"
1125 PRINT " 6.  SEARCH": PRINT
    " 7.  DELETE RECONCILED RECORDS"

1130 PRINT " 8.  DELETE UNRECONCILED
    RECORDS": PRINT " 9.  CHECK FILE
     LENGTH": PRINT "10.  SAVE DATA
    TO DISK": PRINT "11.  QUIT"

1131 PR#0
1132 PRINT : PRINT L$
1133 A=0:Q=0
1135 PRINT : PRINT : PRINT "WHICH ?"
    ;:H=Z
1140 A= PEEK (-16384): IF A<128 THEN
    1140: IF A=208 THEN POKE -16368
    ,0: TAB 7: PR#0: INPUT Q
1144 PR#0
1145 IF Q<0 OR Q>11 THEN 1100:PS=
    0: IF A<>208 OR Q<2 OR Q=5 OR
    Q>6 THEN 1150: PRINT : INPUT
    "WHAT SLOT IS YOUR PRINTER IN "
    ,PS
1149 PR#0
1150 GOSUB Q*200+1000:PS=0
1155 PRINT D$;"PR#0": GOTO 1100
1200 CALL -936: PRINT : PRINT "DO YOU
     WANT TO ENTER DATA FROM THE"
    : PRINT "KEYBOARD, DISK, OR TAPE
    ?(K, D, OR T)?";: INPUT C$
1204 IF C$="D" THEN 1300: IF C$=
    "T" THEN 1320
1208 PRINT "IF YOU WANT AUTOMATICALLY
     ASSIGNED CHECK#'S ENTER THE STA
    RTING #, IF YOU WANT":S2=Z
1212 INPUT "TO ENTER THEM YOURSELF, E
    NTER A '0'",S1: IF S1<Z THEN
    1212: IF S1>Z THEN S2=0
1216 PRINT : PRINT "TO ENTER A DEPOSI
    T, TYPE A '#' SIGN FOR THE FIRST
     CHARACTER OF THE 'TO:' FIELD"
    : PRINT
1220 PRINT "WHEN YOU'RE THROUGH ENTER
    ING DATA, ENTERA '-1' FOR CHK# O
    R MO TO RETURN TO THE  MENU."
    : PRINT : GOTO 1228
1224 PRINT "READY TO GO TO THE NEXT P
    AGE ?"
1228 INPUT "HIT 'RETURN' KEY WHEN REA
    DY..",L$
1232 GOSUB 6
1236 IF NOT S2 THEN INPUT S1
1240 C=S1: VTAB L: CALL -958: IF
    C<Z THEN RETURN : IF C>9999
     THEN 1244:D(0)=C:E=4: GOSUB
    7: GOTO 1248
1244 PRINT "CHECK # TOO HIGH": GOTO
    1236
1248 INPUT C: VTAB L: TAB 5: CALL
    -958: IF C<Z THEN 1100: IF
    C=Z OR C>12 THEN 1252:E=7: GOSUB
    7: PRINT "/";:D(2)=C*256: GOTO
    1256
1252 GOSUB 22: GOTO 1248
1256 INPUT C: VTAB L: TAB 9: CALL
    -958: IF C<0 OR C>31 THEN 1260
    :E=10: GOSUB 7: PRINT "/";:
    D(2)=D(2)+C: GOTO 1264
1260 GOSUB 22: GOTO 1256
1264 INPUT C: VTAB L: TAB 12: CALL
    -958: IF C<Z OR C>99 THEN 1268
    :E=13: GOSUB 7:D(3)=C: GOTO
    1272
```

CHECKBOOK (CONT*)

```
1268 GOSUB 22: GOTO 1264
1272 GOSUB 18: VTAB L: TAB 14: CALL
     -958: PRINT " ";C$;: IF C$(
     1,1)#"#" THEN 1280
1276 IF S2 THEN S1=S1-1
1280 GOSUB 14: VTAB L: TAB 27: CALL
     -958:D(10)=I:D(11)=F:E=32: GOSUB
     10
1284 GOSUB 20: VTAB L: TAB 36: CALL
     -938: PRINT " ";C$;
1288 GOSUB 4
1292 L=L+0:S1=S1+0: GOSUB 0: GOTO
     1236-12*(L>22)
1300 CALL -936: PRINT "THE FOLLOWING
     CHECKBOOK DATA FILES ARE": PRINT
     "ON THIS DISKETTE:"
1301 PRINT "";"CATALOG"
1305 PRINT : POKE 54,37: POKE 55
     ,3: POKE 2052, PEEK (977): POKE
     2053, PEEK (978)
1315 PR#0
1320 PRINT : INPUT "REPLACE CURRENT D
     ATA OR APPEND TO IT     (R/A) ?"
     ,L$
1325 IF L$="R" THEN 1335: IF L$#
     "A" THEN RETURN
1330 BCM=LM:LM=CM+0: GOSUB 1335:
     LM=BCM: RETURN
1335 IF C$="T" THEN 1350: PRINT
     "WHAT IS THE NAME OF YOUR FILE?"
     : INPUT L$: PRINT "";"BLOAD"
     ;L$;".*,A2048":B=LM+A: IF B>
     HM OR A<Z THEN GOSUB 20000:
     CM=B
1338 CALL -936
1340 PRINT "LOADING IN FILE '";N$
     ;"'"
1342 PRINT : PRINT
1345 PRINT : PRINT "FROM CHECK #"
     ;D(5);" TO CHECK #";D(6): PRINT
     "";"BLOAD  ";L$;",A";LM: FOR
     WR=1 TO 500: NEXT WR: RETURN

1350 POKE 60,Z: POKE 61,8: POKE
     62,102: POKE 63,8: POKE Z,134
     : POKE 0,255: POKE 2,32
1355 POKE 3,253: POKE 4,254: POKE
     5,166: POKE 6,255: POKE 7,96

1360 INPUT "START PLAYING TAPE, THEN
     HIT 'RETURN'",L$: CALL Z:B=
     LM+A: IF B>HM OR A<Z THEN GOSUB
     20000
1365 CM=B: PRINT "READING IN FILE '"
     ;N$;"'": PRINT "FROM CHECK #"
     ;D(5);" TO CHECK#";D(6)
1370 POKE 60,LM MOD 256: POKE 61
     ,LM/256: POKE 62,CM MOD 256
     : POKE 63,CM/256: CALL Z: RETURN

1400 P=LM:T=8:T1=28:H=Z
1402 INPUT "ENTER STARTING CHECK #"
     ,C: PRINT "ENTER STARTING BALANC
     E";:SG=0: GOSUB 14:BALI=I:BALF=
     F
1405 PRINT "PR#";PS
1410 GOSUB 2: IF P>=CM THEN 1700
     :P=P+S+S: IF R(0)<C THEN 1410
     :P=P-S-S
1415 GOSUB 13
1420 PRINT "STARTING BALANCE---------
     --";:E=37:I=BALI:F=BALF: GOSUB
     10
1425 IF PS=0 THEN GOSUB 30: VTAB
     L+0
1427 IF PS THEN PRINT
1429 IF PS>0 THEN WW=940: IF PS<
     1 THEN WW=19
1430 FOR L=3 TO WW: IF P>CM THEN
     1700: GOSUB 2
1435 GOSUB 11
1440 BALI=BALI-R(10):BALF=BALF-R(
     11)
1445 BALI=BALI+ SGN (BALF)*( ABS
     (BALF)>99):BALF=BALF MOD 100

1450 IF ABS ( SGN (BALI)- SGN (BALF)
     )<>2 THEN 1460
1455 A= SGN (BALI):BALI=BALI+ SGN
     (BALF):BALF=BALF+100* SGN (
     A)
1460 IF H THEN RETURN
1465 E=37:I=BALI:F=BALF: GOSUB 10
     :Q=1470: GOSUB 24: IF PS THEN
     PRINT : NEXT L
1468 PR#0
1470 PR#0: PRINT : PRINT "HIT 'ESC' T
     O RETURN TO THE MENU, 'RTN'"
     : PRINT "TO CONTINUE BALANCING"
     ;: CALL 8:A= PEEK (255)-128
     : IF A=27 THEN RETURN
1471 IF PS>0 THEN GOTO 1425: GOTO
     1425
1600 P=LM:T=8:T1=29:H=0:BALI=Z:BALF=
     Z
1605 INPUT "ENTER STARTING CHECK #"
     ,C: PRINT "ENTER BEGINNING BALAN
     CE FROM BANK STATE-MENT";:SG=
     0: GOSUB 14:RBI=I:RBF=F
1606 PRINT "PR#";PS
1610 GOSUB 2: IF C>=CM THEN 1700
     :P=P+S+S: IF R(0)<C THEN 1610
     :P=P-S-S
1615 GOSUB 13
1620 PRINT "STARTING BALANCE---------
     --";:E=37:I=RBI:F=RBF: GOSUB
     10
1625 VTAB 20: TAB 0: PRINT "HIT SPACE
      BAR IF CHECK IS LISTED ON"
     : PRINT "STATEMENT, 'RTN' IF NOT
     , 'ESC' TO STOP": VTAB L+0
1630 FOR L=3 TO 16+16*(PS>0): IF
     P>=CM THEN 1665: GOSUB 2
1635 GOSUB 11:Q=R(3)<Z: IF Q THEN
     1645
1640 CALL 8:A= PEEK (255)-128: IF
     A=27 THEN 1665: IF A=13 THEN
     1695
1645 RBI=RBI-R(10):RBF=RBF-R(11)

1650 RBI=RBI+ SGN (RBI)*( ABS (RBF)
     >99):RBF=RBF MOD 100: IF ABS
     ( SGN (RBI)- SGN (RBF))<>2 THEN
     1660
1655 A= SGN (RBI):RBI=RBI+ SGN (
     RBF):RBF=RBF+100* SGN (A)
1660 A=- ABS (R(3)): POKE P-21, PEEK
     (2053): POKE P-22, PEEK (2052
     ):I=RBI:F=RBF:E=37: GOSUB 10
     :Q=1665: GOSUB 24
1661 IF PS THEN PRINT : NEXT L
1665 CALL -958: PRINT
1670 PRINT "SUM OF DEPOSITS NOT CREDI
     TED ON STATE-  MENT (+) & CHECKS
      STILL OUT (-)";:I=BALI:F=BALF:
     E=37: GOSUB 10
1675 R(10)=-RBI:R(11)=-RBF: GOSUB
     1440:I=BALI:F=BALF:R(10)=RBI:
     R(11)=RBF: GOSUB 1440
1680 PRINT "BOOK BALANCE AT CHK #"
     ;R(1);" IS----";:E=37: GOSUB
     10: CALL -958: IF P>=CM THEN
     1700
1685 PRINT : PRINT "HIT 'ESC' TO RETU
     RN TO THE MENU, 'RTN'": PRINT
     "TO CONTINUE RECONCILING";:
      CALL -958
1690 CALL 8: IF PEEK (255)-128=27
      THEN RETURN : GOTO 1615
1695 PRINT " -------";: GOSUB 1440
     : GOTO 1661
1700 PRINT : CALL -958: PRINT D$
     ;"PR#0": PRINT "THAT'S ALL THE C
     HECKS IN MEMORY."
1705 PRINT "HIT ANY KEY TO GET BACK T
     O THE MENU..";: CALL 8:H=Z:
      RETURN
1800 INPUT "ENTER THE CHECK NUMBER YO
     U WOULD LIKE   THE LIST TO START
      WITH",C
1805 P=LM: PRINT "PR#";PS
1810 GOSUB 2: IF P>=CM THEN 1860
     :P=P+S+S: IF R(0)<C THEN 1810
     :P=P-S-S
1814 IF PS<1 THEN WW=19: IF PS>0
      THEN WW=940
1815 GOSUB 6: GOSUB 30:T=12:T1=32
     : FOR L=3 TO WW: IF P>=CM THEN
     1825
1820 GOSUB 2: PRINT : IF PS<1 THEN
     VTAB L: GOSUB 11:SL=B:B=T: GOSUB
     5: PRINT " ";C$;:Q=1825: GOSUB
     24: NEXT L
1825 PRINT : IF P<CM THEN 1830
1826 PRINT "THAT'S ALL OF THEM!"

1827 PR#0
1828 INPUT "HIT 'RETURN' TO GET BACK
     TO MENU",L$
1829 RETURN
1830 PR#0: PRINT "THERE ARE MORE CHEC
     KS IN MEMORY, NOW..."
1835 PRINT "'M' MAKE CHANGES, 'L' LIS
     T FRM NEW CHK#": CALL -958
1840 PRINT "'ESC' RTN TO MENU, OR 'RT
     N' CONTINUE": PRINT "LISTING"
     ;
1841 CALL 8:A= PEEK (255)-128: PRINT
1845 IF A=13 THEN 1815: IF A=27 THEN
     RETURN : IF A=77 THEN 2405:
      IF A=76 THEN 1850: GOTO 1841
1850 CALL -936: GOTO 1800
1855 RETURN
1860 PRINT : PRINT "NO CHECKS IN MEMO
     RY": FOR N=1 TO 300: NEXT N:
      GOTO 1100
2000 CALL -936: VTAB 5: PRINT "YOU MA
     Y SORT ON ONE OF THE FOLLOWING:"
2005 PRINT
2010 PRINT "1. CHECK #": PRINT "2. DA
     TE": PRINT "3. 'TO:' FIELD"
     : PRINT "4. AMOUNT": PRINT
     "5. CODE": PRINT
2015 INPUT "WHICH ",Q: IF Q<0 OR
     Q>5 THEN 2015
2020 POKE 786,136+(Q=3 OR Q=5)*64

2025 GOSUB 97: POKE 2,A:Q=Q+(Q=3
      OR Q=5)-(Q#3 AND Q#5): GOSUB
     97: POKE 3,A+255* NOT A
2030 FOR P1=LM TO CM-S-S STEP S+
     S:I=P1: POKE Z,P1 MOD 256: POKE
     0,P1/256
```

CHECKBOOK (CONT. 2)

```
2035 FOR P2=P1+S+S TO CM-S-S STEP
     S+S:A=P2: CALL 768
2040 NEXT P2:I= PEEK (0)+ PEEK (
     1)*256
2045 P=I: GOSUB 2: GOSUB 100:P=P1:
      GOSUB 2:P=I: FOR C=0 TO S:
     D(C)=R(C): NEXT C
2050 GOSUB 3:P=P1: FOR C=0 TO S:
     D(C)=RA(C): NEXT C: GOSUB 3

2055 PRINT "SORTING ";: NEXT P1:
      PRINT "DONE!!!": FOR C=0 TO
     700: NEXT C: RETURN
2200 P=LM:H=0: CALL -936: VTAB 5
     : PRINT "YOU MAY SEARCH BY ONE O
     F THE FOLLOWING:": PRINT : PRINT
     "1. CHECK NUMBER": PRINT "2. MON
     TH": PRINT "3. DAY"
2205 PRINT "4. YEAR": PRINT "5. 'TO:'
      FIELD": PRINT "6. AMOUNT":
      PRINT "7. CODE"
2210 PRINT :BALI=Z:BALF=Z
2215 INPUT "WHICH FIELD BY NUMBER "
     ,F1: IF F1<0 OR F1>7 THEN RETURN
     : GOSUB 2260+F1*5: PRINT "PR#"
     ;PS
2220 GOSUB 6
2225 IF P>=CM THEN 2240: GOSUB 2
     : GOSUB 2295+F1*5: IF U THEN
     GOSUB 2230:P=P+S+S: GOTO 2225

2230 T=12:T1=32: PRINT : VTAB L:
      TAB 0: GOSUB 2: GOSUB 11:SL=
     B:B=T: GOSUB 5: PRINT " ";C$
     ;:L=L+0:P=P-S-S
2232 BALI=BALI+R(10):BALF=BALF+R(
     11): GOSUB 1445
2234 IF PS THEN RETURN
2235 IF L<23 THEN RETURN : PRINT
     "HIT 'ESC' TO GET BACK TO THE ME
     NU, 'RTN'TO CONTINUE SEARCHING."
     ;: CALL 8: IF PEEK (255)-128
     =27 THEN RETURN
2236 GOSUB 6: RETURN
2240 PRINT : TAB 21: PRINT "TOTAL"
     ;:E=32:I=BALI:F=BALF: GOSUB
     10
2245 PRINT : PRINT "THAT'S ALL OF THE
     M !"
2250 PRINT : PRINT D$;"PR#0"
2255 INPUT "HIT 'RETURN' TO GET BACK
     TO THE MENU",L$
2260 RETURN
2265 PRINT "CHECK NUMBER";: GOTO
     2335
2270 PRINT "MONTH";: GOTO 2335
2275 INPUT "INPUT MONTH,DAY,YEAR..."
     ,A,Q,D(3): IF A<0 OR A>=S OR
     Q<0 OR Q>31 THEN 2275:D(2)=
     A*256+Q: RETURN
2280 PRINT "YEAR";: GOTO 2335
2285 PRINT "TO FIELD LOOKING FOR"
     ;: GOTO 18
2290 PRINT "AMOUNT LOOKING FOR";
     : GOSUB 14:D(10)=I:D(11)=F:
      RETURN
2295 PRINT "CODE LOOKING FOR";: GOTO
     20
2300 U=D(0)=R(0): RETURN
2305 U=D(2)=R(2)/256: RETURN
2310 U=(D(2)=R(2) AND D(3)= ABS
     (R(3) MOD 256)): RETURN
2315 U=(D(3)= ABS (R(3) MOD 256)
     ): RETURN
2320 U=0: FOR J=4 TO 9:U=(U AND
     D(J)=R(J)): IF U THEN NEXT
     J: RETURN
2325 U=(D(10)=R(10) AND D(11)=R(
     11)): RETURN
2330 U=(D(12)=R(12) AND D(S)=R(S)
     ): RETURN
2335 INPUT " LOOKING FOR",D(F1-(
     F1>2)): RETURN
2400 GOTO 3600
2405 VTAB L: TAB 0: CALL -958: PRINT
     : PRINT "YOU MAY NOW :  1. CHANG
     E ABOVE DATA"
2410 PRINT "                2. DELETE
     BY CHECK #"
2415 PRINT
2420 INPUT "WHICH ",Q
2425 IF Q=2 THEN 3400: IF Q=1 THEN
     2430: GOSUB 22: GOTO 2405
2430 VTAB L+0: TAB 0: CALL -958:
      PRINT "TYPE 'CTRL' & DESIRED KE
     Y SIMULTANEOUSLY";
2435 PRINT "A=ADVANCE    B=BACK UP"
```

```
2440 PRINT "U=UP          D=DOWN";

2445 TAB 27: PRINT "E=EXIT TO MENU"
     : PRINT "*HIT SPACE BAR BEFORE E
     NTERING NEW DATA*";
2450 P=P-S-S:L=L-0: VTAB L:S1=0
2455 CALL 8:A= PEEK (255): IF A>
     154 THEN 2490: IF A=129 THEN
     2460: IF A=130 THEN 2465: IF
     A=132 THEN 2475: IF A=133 THEN
     2480
2456 IF A=149 THEN 2485: IF A=144
      THEN 1815: GOTO 2455
2460 IF S1=7 THEN S1=Z:S1=S1+0: GOTO
     2470
2465 IF S1=0 THEN 2455:S1=S1-0
2470 TAB S1+3*(S1=2)+5*(S1=3)+7*
     (S1=4)+9*(S1=5)+21*(S1=6)+29
     *(S1=7): GOTO 2455
2475 IF L=17 THEN 2455:L=L+0: VTAB
     L:P=P+S+S:SG=0: GOTO 2455
2480 RETURN
2485 IF L=3 THEN 2455:L=L-0: VTAB
     L:P=P-S-S:SG=0: GOTO 2455
2490 GOSUB 2: FOR J=0 TO S:D(J)=
     R(J): NEXT J:D(3)= ABS (R(3
     )): POKE 35,L: POKE 34,L-0:
      GOSUB 2500+S1*5: POKE 35,24
     : POKE 34,Z
2495 VTAB L: GOSUB 3: FOR J=0 TO
     S:R(J)=D(J): NEXT J: GOSUB
     11:SL=B:B=T: GOSUB 5: PRINT
     " ";C$;
2500 VTAB L:P=P-S-S: GOTO 2460
2505 INPUT D(0): RETURN
2510 INPUT A:D(2)=D(2) MOD 256+A*
     256: RETURN
2515 INPUT A:D(2)=D(2)/256*256+A:
      RETURN
2520 INPUT D(3): RETURN
2525 GOTO 18
2530 GOSUB 14:D(10)=I:D(11)=F: RETURN
2535 GOTO 20
2600 GOTO 3625
2605 PRINT "WHAT IS THE NAME OF THIS
     FILE?": INPUT N$
2610 P=LM: GOSUB 2
2615 D(5)=R(0):P=CM-S-S+0: GOSUB
     2:D(6)=R(0)
2620 A=CM-LM: IF A<Z THEN RETURN
     : PRINT "";"BSAVE ";N$;".*,A2048
     ,L103"
2625 PRINT "";"BSAVE";N$;",A";LM;
     ",L";A+0: RETURN
2800 A=(CM-LM+0)/S/2:B=(HM-LM-0)
     /S/2
2810 PRINT A;" RECORDS USED OUT OF "
     ;B;" TOTAL": PRINT "LEAVING "
     ;B-A;" UNUSED.": POP : GOTO
     1135
3000 GOTO 2605
3200 INPUT "DO YOU WANT TO SAVE YOUR
     DATA (Y/N)?",C$: IF C$="Y" OR
     C$="YES" THEN GOSUB 2605
3205 POP : PRINT : PRINT "OK": END

3400 VTAB 18: TAB 0: CALL -958: PRINT

3405 INPUT "START CHECK NUMBER",
     I: VTAB 20: INPUT "END CHECK NUM
     BER",F:P=LM
3410 GOSUB 2: IF P>=CM THEN 3440
     :P=P+S+S: IF I#R(0) THEN 3410
     :P=P-S-S:B=P
3415 GOSUB 2: IF P>=CM THEN 3440
     :P=P+S+S: IF F#R(0) THEN 3415
     :A=P
3420 POKE 60, PEEK (2052): POKE
     61, PEEK (2053):A=CM: POKE
     62, PEEK (2052): POKE 63, PEEK
     (2053):A=B: POKE 66, PEEK (
     2052): POKE 67, PEEK (2053)

3425 CALL -468:CM=CM-P+B: IF Q=2
      THEN 1100: IF Q=7 THEN RETURN

3430 VTAB 21: TAB 0: INPUT "MORE (Y/N
     ) ?",L$
3435 IF L$="Y" THEN 3400: IF L$=
     "N" THEN RETURN : GOSUB 22:
      GOTO 3425
3440 PRINT "BAD RANGE!!!": FOR N=
     0 TO 600: NEXT N
3445 GOTO 2405
3600 P=LM
3605 GOSUB 2: IF P>=CM THEN 3620
     :P=P+S+S: IF R(3)>Z THEN 3605
     :B=P-S-S
3610 GOSUB 2:P=P+S+S: IF P>=CM THEN
     3615: IF R(3)<Z THEN 3610
3615 P=P-S-S:A=P: GOSUB 3420:P=B:
      GOTO 3605
3620 PRINT : PRINT "DONE!": GOSUB
     22: FOR N=1 TO 300: NEXT N:
      RETURN
3625 P=LM
3630 GOSUB 2: IF P>=CM THEN 3645
     :P=P+S+S: IF R(3)<Z THEN 3630
     :B=P-S-S
3635 GOSUB 2:P=P+S+S: IF P>=CM THEN
     3640: IF R(3)>Z THEN 3635
3640 P=P-S-S:A=P: GOSUB 3420:P=B:
      GOTO 3630
3645 PRINT : PRINT "DONE!": GOSUB
     22: FOR N=1 TO 300: NEXT N:
      RETURN
```

```
20000 POP : POP : PRINT "  *** MEMORY
      FULL ***": FOR A=1 TO 200: NEXT
      A: GOTO 1100
30000 REM  PRINT ROUTINES MOD BY KEN S
      ILVERMAN
30001 REM  PROGRAM RE EDIDED BY MIKE N
      ADELMAN 8/79
65534 REM  MODIFIED BY H.J.CANNES 11/7
      8
65535 REM *** COPYRIGHT 1977 BY APPLE
      COMPUTER: WRITTEN BY R.WIGGINTON
       & A.C. MARKKULA
```

FAST SORT

```
*300.360
0300- A4 02 AD 04 08 85 04 AD
0308- 05 08 85 05 B1 00 D1 04
0310- D0 06 C8 C4 03 D0 F5 60
0318- 90 08 A5 04 85 00 A5 05
0320- 85 01 60 00 00 C9 8D F0
0328- 12 C6 1F 30 07 D0 04 C9
0330- C1 66 1E 60 24 1E 30 FB
0338- 4C F0 FD A9 08 85 1F A9
0340- 8D 24 1E 10 F3 60 A2 32
0348- A0 00 BD 00 08 4A 4A 4A
0350- 00

*300LLL

0300-  A4 02      LDY   $02
0302-  A4 04 08   LDA   $0804
0305-  85 04      STA   $04
0307-  AD 05 08   LDA   $0805
030A-  85 05      STA   $05
030C-  B1 00      LDA   ($00),Y
030E-  D1 04      CMP   ($04),Y
0310-  D0 06      BNE   $0318
0312-  C8         INY
0313-  C4 03      CPY   $03
0315-  D0 F5      BNE   $030C
0317-  60         RTS
0318-  90 08      BCC   $0322
031A-  A5 04      LDA   $04
031C-  85 00      STA   $00
031E-  A5 05      LDA   $05
0320-  85 01      STA   $01
0322-  60         RTS
0323-  00         BRK
0324-  00         BRK
0325-  C9 8D      CMP   #$8D
0327-  F0 12      BEQ   $033B
0329-  C6 1F      DEC   $1F
032B-  30 07      BMI   $0334
032D-  D0 04      BNE   $0333
032F-  C9 C1      CMP   #$C1
0331-  66 1E      ROR   $1E
0333-  60         RTS
0334-  24 1E      BIT   $1E
0336-  30 FB      BMI   $0333
0338-  4C F0 FD   JMP   $FDF0
033B-  A9 08      LDA   #$08
033D-  85 1F      STA   $1F
033F-  A9 8D      LDA   #$8D
0341-  24 1E      BIT   $1E
0343-  10 F3      BPL   $0338
0345-  60         RTS
0346-  A2 32      LDX   #$32
0348-  A0 00      LDY   #$00
034A-  BD 00 08   LDA   $0800,X
034D-  4A         LSR
034E-  4A         LSR
034F-  4A         LSR
0350-  00         BRK
```

BON VOYAGE

BY BRUCE TOGNAZZINI

We bid adieu this month to a friend of the Apple Core with whom we have all spent a great deal of time. But while I'm sure we shall long remember him, we are all certainly glad to be rid of him. No, I'm not speaking of the Apple Core librarian, and no ground-swells need apply; I am talking of course of good old ESC A.

"Going?", you ask incredulously, "ESC A?" "But however will we steam past those huge horrible holes in our listed quotes and rem statements?" Enter stage left our hero, POKE 33,33. Yes, our old friend POKE, helping us out of yet another jam. Try him out the next time you have to copy a listed print statement.

Press ESC (shift) P (return); then type "POKE 33,33" (return). Now LIST your line(s) to be edited. You will note that the print statement is all scrunched up* with no extra spaces on the left-hand side. Now COPY over the line, using the forward arrow. Since you have set a scrolling window 33 characters wide, the cursor will automatically jump to the next line as soon as it reaches the right margin of your text. How do you get out ot this mode? Just type "TEXT". This little trick will work in both Integer and Applesoft. And whom do we have to thank? Well, it appears everybody in the known world was aware of this one except the Apple Core, so I guess we have to thank J. Alfred Glitch for keeping us in the dark so long. Thanks a lot, Alf.

*This author has been asked to define the term "scrunch". In this author's opinion, those cretins who asked him to do so should spend a little more time studying their technical manuals and a little less time criticizing those who would utilize proper terminology. "Scrunch" is, of course, an historical term dating from the late 40's and the original Apple, ENIAC. It was short for, appropriately enough, ESC CRUNCH, a command which would cause a shift-right of all the electrons available in the entire five-story building. Needless to say ESC CRUNCH was the ENIAC's single most powerful instruction; so powerful it lead to the computer's eventual downfall, when an operator carelessly punched it up while using the full 30K of core and the right side of the building collasped from the sheer weight of the electrons.

POKE 33,33

BY LARRY DANIELSON

As we saw in last month's article, BON VOYAGE by Bruce Tognazzini, POKE 33,33 allows us to edit our listed programs much easier, without using the escape characters. When listing sections of a program with my printer, I discovered that by using POKE 33,33 first, the printer more efficiently uses its 80 character width capability. The POKE 33,33 feature enables the printer to list faster and use less paper as it does not have to perform as many carriage returns.

MAKE A BOX

BY MAX J. NAREFF

Placing program headings within borders adds to presentations. This is a simple routine which will create a box or border for headings.

The first line draws the top border.

```
10 VTAB4: HTAB9: FOR X=1 TO 23: PRINT
   "*";: NEXT
```

The second line draws two sides. Note how the ping-pong effect is achieved in the single line by the use of the semi-colon after the first PRINT command.

```
15 FOR X= 1 TO 13: HTAB9: PRINT "*";:
   HTAB32: PRINT "*": NEXT
```

The last line closes the box.

```
20 VTAB16: HTAB9: FOR X=1 TO 23: PRINT
   "*";: NEXT
```

Adjust the tabbing carefully for position, and the loop dimensions for size of the box. The three lines can readily be combined into a single line subroutine by adding a colon after each terminating NEXT.

The box can be entered for introduction and credit by succeeding lines using VTAB's and HTAB's to ascend the screen. One of caution: Using PRINT TAB (X) commands to enter the box will not be successful, because those necessary but nasty semi-colons suppress carriage returns. HTAB commands will perform, PRINT TAB (X) will not! Try it and see.

One final note: With 'ANDY'S KEYBOARD FILTER', some very striking borders can be created.

CHARTS & TABLES

HEXADECIMAL

BY ROD CARLISLE

While programming I often noticed the need to convert from hexadecimal to decimal. There are several programs available which will do this for you, but using them necessitates discontinuing the present program, running the converter, then resuming the program. This is an awkward process. I read with interest J.A. Backman's Poor Man's Hex-Decimal-Hex Converter in Peeking at Call APPLE Vol. I. However, that method suffered from the disadvantage of requiring several additions or subtractions. There had to be a better way.

I decided a larger table could cut down on the arithmetic. Also, I could take advantage of the fact that usually all that is required is 4 Hex digits. I wrote the program which generated the accompanying tables. With them one can convert from hex to decimal or negative decimal into either of the other number systems with just one addition or subtraction. That sure beats the method they teach in the books. Follow the examples which are included with the second table to see how they work.

The highest Hex digit is on the table's left side. Going from Hex to Decimal presents no problems. Just find the equivalent of the first two digits in the lower table and add it

to the equivalent of the last two digits in the upper table. When going from negative decimal to hex, first find the largest number in the body of the second table whose absolute value is smaller than the given number. Then subtract that number from the given number and find the difference in the upper table.

Assemble a notebook of handy charts and tables such as this and keep it by your Apple. You will find it cuts down on your programming frustrations.

EXAMPLES

HEX TO POSITIVE DECIMAL

```
A2C8 = ?
  A2 = 41472
  C8 =(+)200
       ------
A2C8 = 41672
```

HEX TO NEGATIVE DECIMAL

```
A2C8 = ?
  A2 = -23808
  C2 = (+)-56
       --------
A2C8 = -23864
```

POSTIVE DECIMAL TO HEX

```
  41672 = ?
(-)41472 = A2
---------
    200 = C8
  41672 = A2C8
```

POSITIVE DECIMAL TO NEGATIVE DECIMAL SUBTRACT 65536

```
  41672 = ?
(-)65536
---------
 -23864
  41672 = -23864
```

NEGATIVE DECIMAL TO HEX

```
  -23864 = ?
(-)-23808 = A2
----------
     -56 = C8
  -23864 = A2C8
```

NEGATIVE DECIMAL TO POSITIVE DECIMAL ADD 65536

```
  -23864 = ?
(+)65536
--------
  41672
 -23864 = 41672
```

THE APPLEHEADS — R. & E. HANCE

HEXADECIMAL TO POSITIVE DECIMAL

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	HEX
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	2
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	3
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	4
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	5
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	6
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	7
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	8
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	9
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	A
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	B
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	C
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	D
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	E
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	F

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	HEX
0	0	256	512	768	1024	1280	1536	1792	2048	2304	2560	2816	3072	3328	3584	3840	0
1	4096	4352	4608	4864	5120	5376	5632	5888	6144	6400	6656	6912	7168	7424	7680	7936	1
2	8192	8448	8704	8960	9216	9472	9728	9984	10240	10496	10752	11008	11264	11520	11776	12032	2
3	12288	12544	12800	13056	13312	13568	13824	14080	14336	14592	14848	15104	15360	15616	15872	16128	3
4	16384	16640	16896	17152	17408	17664	17920	18176	18432	18688	18944	19200	19456	19712	19968	20224	4
5	20480	20736	20992	21248	21504	21760	22016	22272	22528	22784	23040	23296	23552	23808	24064	24320	5
6	24576	24832	25088	25344	25600	25856	26112	26368	26624	26880	27136	27392	27648	27904	28160	28416	6
7	28672	28928	29184	29440	29696	29952	30208	30464	30720	30976	31232	31488	31744	32000	32256	32512	7
8	32768	33024	33280	33536	33792	34048	34304	34560	34816	35072	35328	35584	35840	36096	36352	36608	8
9	36864	37120	37376	37632	37888	38144	38400	38656	38912	39168	39424	39680	39936	40192	40448	40704	9
A	40960	41216	41472	41728	41984	42240	42496	42752	43008	43264	43520	43776	44032	44288	44544	44800	A
B	45056	45312	45568	45824	46080	46336	46592	46848	47104	47360	47616	47872	48128	48384	48640	48896	B
C	49152	49408	49664	49920	50176	50432	50688	50944	51200	51456	51712	51968	52224	52480	52736	52992	C
D	53248	53504	53760	54016	54272	54528	54784	55040	55296	55552	55808	56064	56320	56576	56832	57088	D
E	57344	57600	57856	58112	58368	58624	58880	59136	59392	59648	59904	60160	60416	60672	60928	61184	E
F	61440	61696	61952	62208	62464	62720	62976	63232	63488	63744	64000	64256	64512	64768	65024	65280	F

HEXADECIMAL TO NEGATIVE DECIMAL

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	HEX
0	-256	-255	-254	-253	-252	-251	-250	-249	-248	-247	-246	-245	-244	-243	-242	-241	0
1	-240	-239	-238	-237	-236	-235	-234	-233	-232	-231	-230	-229	-228	-227	-226	-225	1
2	-224	-223	-222	-221	-220	-219	-218	-217	-216	-215	-214	-213	-212	-211	-210	-209	2
3	-208	-207	-206	-205	-204	-203	-202	-201	-200	-199	-198	-197	-196	-195	-194	-193	3
4	-192	-191	-190	-189	-188	-187	-186	-185	-184	-183	-182	-181	-180	-179	-178	-177	4
5	-176	-175	-174	-173	-172	-171	-170	-169	-168	-167	-166	-165	-164	-163	-162	-161	5
6	-160	-159	-158	-157	-156	-155	-154	-153	-152	-151	-150	-149	-148	-147	-146	-145	6
7	-144	-143	-142	-141	-140	-139	-138	-137	-136	-135	-134	-133	-132	-131	-130	-129	7
8	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113	8
9	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97	9
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81	A
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65	B
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49	C
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33	D
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	E
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	F

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	HEX
8	-32512	-32256	-32000	-31744	-31488	-31232	-30976	-30720	-30464	-30208	-29952	-29696	-29440	-29184	-28928	-28672	8
9	-28416	-28160	-27904	-27648	-27392	-27136	-26880	-26624	-26368	-26112	-25856	-25600	-25344	-25080	-24832	-24576	9
A	-24320	-24064	-23808	-23552	-23296	-23040	-22784	-22528	-22272	-22016	-21760	-21504	-21248	-20992	-20736	-20480	A
B	-20224	-19968	-19712	-19456	-19200	-18944	-18688	-18432	-18176	-17920	-17664	-17408	-17152	-16896	-16640	-16384	B
C	-16128	-15872	-15616	-15360	-15104	-14848	-14592	-14336	-14080	-13824	-13568	-13312	-13056	-12800	-12544	-12288	C
D	-12032	-11776	-11520	-11264	-11008	-10752	-10496	-10240	-9984	-9728	-9472	-9216	-8960	-8704	-8448	-8192	D
E	-7936	-7680	-7424	-7168	-6912	-6656	-6400	-6144	-5883	-5632	-5376	-5120	-4864	-4608	-4352	-4096	E
F	-3840	-3584	-3328	-3072	-2816	-2560	-2304	-2048	-1792	-1536	-1280	-1024	-768	-512	-256	0	F

Chart by Rod Carlisle (Cider Press v2.6)

LOMEM

0000-03FF ZERO PAGE UTILITIES K_{00}	0400-07FF TEXT OR LO-RES 1	0800-0BFF TEXT OR LO-RES 2	0C00-0FFF	1000-13FF	1400-17FF	1800-1BFF	1C00-1FFF K_{08}
2000-23FF	2400-27FF	2800-2BFF	2C00-2FFF	3000-33FF	3400-37FF	3800-3BFF	3C00-3FFF
HIGH RESOLUTION GRAPHICS PAGE ONE K_{08}							K_{16}
4000-43FF	4400-47FF	4800-4BFF	4C00-4FFF	5000-53FF	5400-57FF	5800-5BFF	5C00-5FFF
HIGH RESOLUTION GRAPHICS PAGE TWO K_{16}							K_{24}
6000-63FF K_{24}	6400-67FF	6800-6BFF	6C00-6FFF	7000-73FF	7400-77FF	7800-7BFF	7C00-7FFF K_{32}
8000-83FF K_{32}	8400-87FF	8800-8BFF	8C00-8FFF	9000-93FF	9400-97FF	9800-9BFF	9C00-9FFF K_{40}
A000-A3FF K_{40}	A400-A7FF	A800-ABFF	AC00-AFFF	B000-B3FF	B400-B7FF	B800-BBFF	BC00-BFFF K_{48}
C000-C3FF	C400-C7FF	C800-CBFF	CC00-CFFF	D000-D3FF	D400-D7FF	D800-DBFF	DC00-DFFF
PERIPHERAL I/O HARDWARE FUNCTIONS K_{48}				PROM DØ		PROM D8 K_{56}	
E000-E3FF	E400-E7FF	E800-EBFF	EC00-EFFF	F000-F3FF	F400-F7FF	F800-FBFF	FC00-FFFF
INTEGER BASIC K_{56}					F.P., SW16, MINI-ASSEM.	MONITOR K_{64}	

HIMEM
16K
32K
48K

/RPW/

INTEGER BASIC TOKEN CHARACTER SET

BY LINDA SLOVICK

TOKENS

DEC	HEX	TOKEN	DEC	HEX	TOKEN
0	$00	[HIMEM:]	64	$40	$ <string>
1	$01	EOL	65	$41	$ <?>
2	$02	-	66	$42	(
3	$03	:	67	$43	,
4	$04	LOAD	68	$44	,
5	$05	SAVE	69	$45	;
6	$06	CON	70	$46	;
7	$07	RUN	71	$47	;
8	$08	RUN	72	$48	,
9	$09	DEL	73	$49	,
10	$0A	, <for DEL>	74	$4A	,
11	$0B	NEW	75	$4B	TEXT
12	$0C	CLR	76	$4C	GR
13	$0D	AUTO	77	$4D	CALL
14	$0E	, <for AUTO>	78	$4E	DIM <strings>
15	$0F	MAN	79	$4F	DIM <numbers>
16	$10	HIMEM:	80	$50	TAB
17	$11	LOMEM:	81	$51	END
18	$12	+	82	$52	INPUT <string>
19	$13	-	83	$53	INPUT <$ or "S>
20	$14	*	84	$54	INPUT <number>
21	$15	/	85	$55	FOR
22	$16	=	86	$56	= <FOR/NEXT>
23	$17	#	87	$57	TO
24	$18	>=	88	$58	STEP
25	$19	>	89	$59	NEXT
26	$1A	<=	90	$5A	,
27	$1B	<>	91	$5B	RETURN
28	$1C	<	92	$5C	GOSUB
29	$1D	AND	93	$5D	REM
30	$1E	OR	94	$5E	LET
31	$1F	MOD	95	$5F	GOTO
32	$20	^	96	$60	IF
33	$21	+	97	$61	PRINT <"S>
34	$22	(	98	$62	PRINT <X or X$>
35	$23	,	99	$63	PRINT
36	$24	THEN <line #>	100	$64	POKE
37	$25	THEN <stmt>	101	$65	,
38	$26	, <string>	102	$66	COLOR=
39	$27	, <number>	103	$67	PLOT
40	$28	, <beginning>	104	$68	,
41	$29	" <ending>	105	$69	HLIN
42	$2A	(	106	$6A	,
43	$2B	!	107	$6B	AT
44	$2C	!	108	$6C	VLIN
45	$2D	(	109	$6D	,
46	$2E	PEEK	110	$6E	AT
47	$2F	RND	111	$6F	VTAB
48	$30	SGN	112	$70	= <string>
49	$31	ABS	113	$71	=
50	$32	PDL	114	$72	)
51	$33	RNDX	115	$73	)
52	$34	(	116	$74	LIST <from,to>
53	$35	+	117	$75	,
54	$36	- <signs>	118	$76	LIST
55	$37	NOT	119	$77	POP
56	$38	(	120	$78	NODSP <string>
57	$39	=	121	$79	NODSP <number>
58	$3A	#	122	$7A	NOTRACE
59	$3B	LEN(	123	$7B	DSP <string>
60	$3C	ASC(	124	$7C	DSP <number>
61	$3D	SCRN	125	$7D	TRACE
62	$3E	)	126	$7E	PR#
63	$3F	(	127	$7F	IN#

CHARACTER

DEC	HEX	CHR	DEC	HEX	CHR	
128	$80	NULL	192	$C0	@	
129	$81	C-A	193	$C1	A	
130	$82	C-B	194	$C2	B	
131	$83	C-C	195	$C3	C	
132	$84	C-D	196	$C4	D	
133	$85	C-E	197	$C5	E	
134	$86	C-F	198	$C6	F	
135	$87	BELL	199	$C7	G	
136	$88	BS	200	$C8	H	
137	$89	C-I	201	$C9	I	
138	$8A	LF	202	$CA	J	
139	$8B	C-K	203	$CB	K	
140	$8C	C-L	204	$CC	L	
141	$8D	CR	205	$CD	M	
142	$8E	C-N	206	$CE	N	
143	$8F	C-O	207	$CF	O	
144	$90	C-P	208	$D0	P	
145	$91	C-Q	209	$D1	Q	
146	$92	C-R	210	$D2	R	
147	$93	C-S	211	$D3	S	
148	$94	C-T	212	$D4	T	
149	$95	advance	213	$D5	U	
150	$96	C-V	214	$D6	V	
151	$97	C-W	215	$D7	W	
152	$98	C-X	216	$D8	X	
153	$99	C-Y	217	$D9	Y	
154	$9A	C-Z	218	$DA	Z	
155	$9B	ESC	219	$DB	[	
156	$9C	CS-L	220	$DC	\	
157	$9D	CS-M	221	$DD	]	
158	$9E	CS-N	222	$DE	^	
159	$9F	CS-O	223	$DF	_	
160	$A0	space	224	$E0	`	
161	$A1	!	225	$E1	!	(a)
162	$A2	"	226	$E2	"	(b)
163	$A3	#	227	$E3	#	(c)
164	$A4	$	228	$E4	$	(d)
165	$A5	%	229	$E5	%	(e)
166	$A6	&	230	$E6	&	(f)
167	$A7	'	231	$E7	'	(g)
168	$A8	(	232	$E8	(	(h)
169	$A9	)	233	$E9	)	(i)
170	$AA	*	234	$EA	*	(j)
171	$AB	+	235	$EB	+	(k)
172	$AC	,	236	$EC	,	(l)
173	$AD	-	237	$ED	-	(m)
174	$AE	.	238	$EE	.	(n)
175	$AF	/	239	$EF	/	(o)
176	$B0	0	240	$F0	0	(p)
177	$B1	1	241	$F1	1	(q)
178	$B2	2	242	$F2	2	(r)
179	$B3	3	243	$F3	3	(s)
180	$B4	4	244	$F4	4	(t)
181	$B5	5	245	$F5	5	(u)
182	$B6	6	246	$F6	6	(v)
183	$B7	7	247	$F7	7	(w)
184	$B8	8	248	$F8	8	(x)
185	$B9	9	249	$F9	9	(y)
186	$BA	:	250	$FA	:	(z)
187	$BB	;	251	$FB	;	
188	$BC	<	252	$FC	<	
189	$BD	=	253	$FD	=	
190	$BE	>	254	$FE	>	
191	$BF	?	255	$FF	?	

APPLESOFT INTERPRETER SET

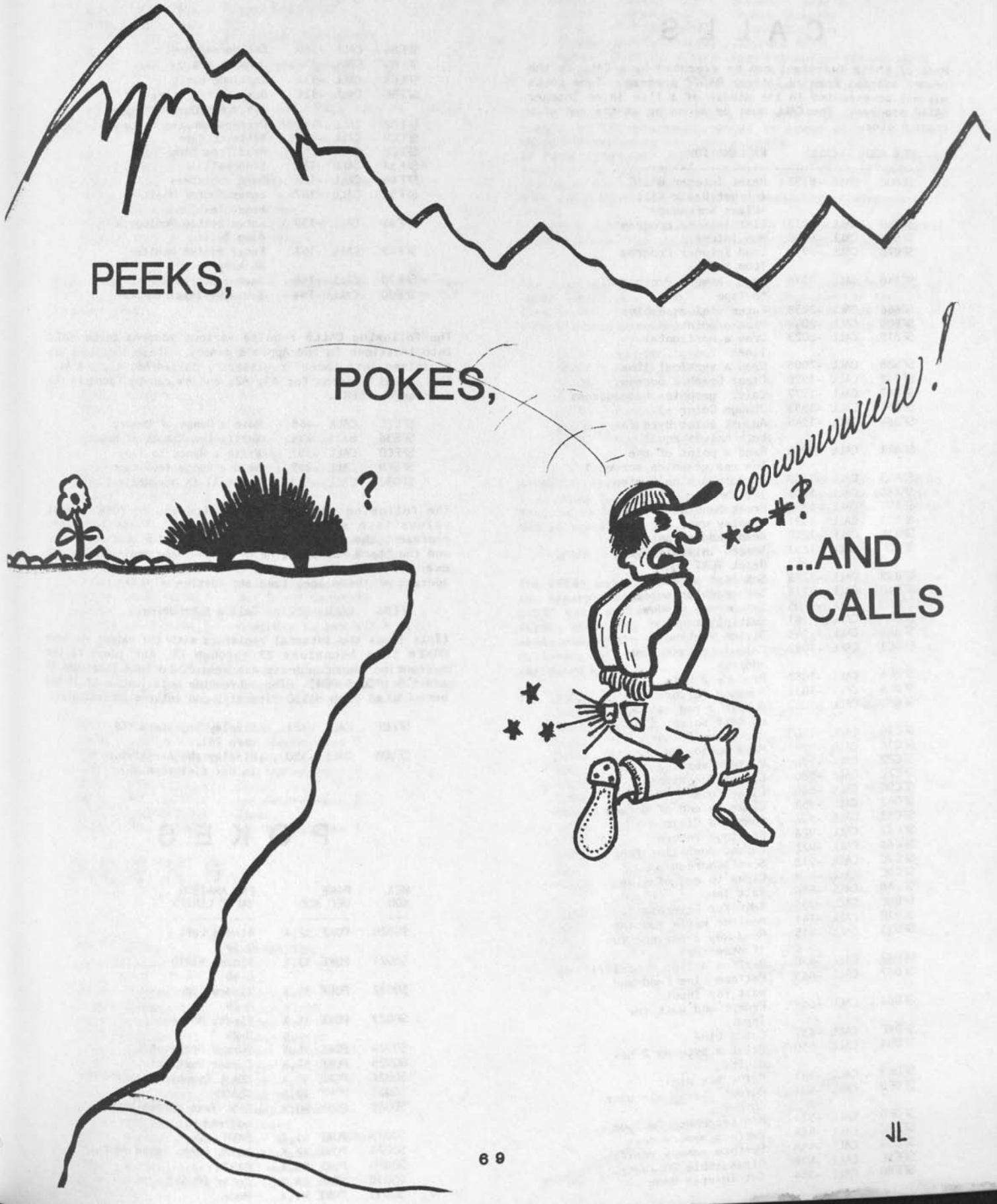
ROM ADDRESSES D000-F7FF

SUBROUTINES - ENTRY POINT LISTING:

NAME OF SUB	FUNCTION	ADDRESS
END	TERMINATE EXECUTION	$D86F
FOR	START OF FOR NEXT LOOP	$D765
NEXT	END OF FOR/NEXT LOOP	$DCF8
DATA	DEF OF A DATA STMT	$D994
INPUT	INPUT ROUTINE EXEC	$DBB1
DELETE	DELETE A LINE FUNCTION	$F32C
DIM	DIMENSION A VAR FUNCT.	$DFD8
READ	READ DATA ROUTINE	$DBE1
SETGR	SET GRAPHICS ROUTINE	$F388
SETTXT	SET TEXT MODE ROUTINE	$F394
PR#	SET OUTPUT PORT	$F1E4
IN#	SET INPUT PORT	$F1DD
CALL	CALL USER MACH SUB	$F1D4
PLOT	PLOT A POINT-LORES	$F224
HLIN	DRAW HORZ LINE-LORES	$F231
VLIN	DRAW VERT LINE-LORES	$F240
SETHR-2	SET HIRES PAGE 2	$F3D3
SETHR-1	SET HIRES PAGE 1	$F3DD
SETHRCOL	SET HIRES COLOR	$F6E4
LINE	DRAW LINE HIRES X,Y	$F6F9
DRAW	DRAW SHAPE SPECIFIED	$F764
XDRAW	EOR DRAW SHAPE	$F76A
HTAB	HORZ TAB X # SPC.	$F7E2
CLRSRC	CLEAR SCREEN	$FC57
SETROT	SET ROTATION SHAPE	$F71C
SETSCALE	SET SCALE FOR SHAPE	$F722
SHLOAD	LOAD A SHAPE TABLE	$F770
SETTRACE	TURN ON TRACE	$F26C
TRACEOFF	TURN OFF TRACE	$F26E
SETNORM	SET NORMAL TEXT	$F272
INVERSE	SET INVERSE TEXT	$F276
FLASH	SET FLASHING TEXT	$F27F
COLOR	SET LORES COLOR	$F24E
RETURN	RETURN FROM SUB	$D96A
VTAB	VERTICLE TAB	$F255
HIMEMSET	SET HIMEM POINTER	$F285
LOMEMSET	SET LOMEM POINTER	$F2A5
ONERR	SET ONERR FLAG	$F2CA
RESUME	CONTINE FROM ONERR	$F316
RECALL	RECALL VARIABLE-TAPE	$F387
STORE	SAVE VARIALBLES-TAPE	$F39A
SETSPD	SET SPEED FOR OUTPUT	$F261
LET	LET ASSIGN VARIABLE	$DA45
GOTO	GOTO BRANCH FUNCTION	$D93D
RUN	EXECUTE A PROGRAM	$D911
IF	IF TEST FUNCTION	$D9C8
DOREL	DO RELATION TEST	$DFF64

NAME OF SUB	FUNCTION	ADDRESS
RESTORE	RESTORE DATA PT	$D848
3F4 HOOK	BRANCH TO MACH	$03F4
GOSUB	GOSUB BRANCH FUNCTION	$D920
RETURN	RETURN FROM GOSUB	$D96A
REM	REMARK STATEMENT	$D9DB
STOP	STOP PROGRAM	$D86D
ONGOTO	ON GOTO	$D9E8
WAIT	WAIT FUNCTION	$E783
LOAD	LOAD TAPE PGM	$D8C8
SAVE	SAVE TAPE PGM	$D8AF
DEF	DEF USER FUNCTION	$E312
POKE	POKE MEMORY LOC	$E77A
PRINT	PRINT OUTPUT	$DAD4
CONT	CONTINUE EXECUTION	$D895
LIST	LIST PROGRAM	$D6A4
CLEAR	CLEAR VARIABLES	$D669
GET	GET A CHAR INPUT	$D89F
SGN	SIGN OF NUMBER	$EB90
INT	INTEGER FUNCTION	$EC23
ABS	ABSOLUTE VALUE	$EBAF
FRE	FREE MEMORY	$E2DE
PDL	GAME PADDLE	$DFCD
POS	POSITION IN HIRES	$E2FF
SQR	SQUARE ROOT	$EE8D
RND	RANDOM NUMBER	$EFAE
LOG	LOG X BASE 10	$E941
EXP	EXPONENT FUNCTION	$EFEA
SIN	SIN FUNCTION	$EFF1
TAN	TANGENT FUNCTION	$F03A
ATN	ARCTANGENT FUNCTION	$F09E
PEEK	PEEK MEMORY	$E764
LEN	LENGTH FUNCTION	$E606
STR$	CHAR-NUMERIC VAR	$E3C5
VAL	VALUE STRING CHAR	$E707
ASC	ASCII FUNCTION	$E6E5
CHR$	CHAR STRING FUNCTION	$E646
LEFT$	LEFT JUSTIFIED	$E65A
RIGHT$	RIGHT JUSTIFIED	$E686
MID$	MID STRING FUNCTION	$E691
FADD	FLOATING POINT +	$E7C0
FSUB	FLOATING POINT -	$E7A9
FMULT	FLOATING POINT x	$E981
FDIV	FLOATING POINT /	$EA68
FPWR	FLOATING POINT PWR	$EE96
AND	AND FUNCTION	$DF54
OR	OR FUNCTION	$DF4E
NEG	NEGATE FUNCTION	$EECF
NOT	NOT FUNCTION	$DE97

THE APPLEHEADS — E. HANCE


PEEKS,
POKES,
?
ooowwwwww!
...AND
CALLS
JL

PEEKS - POKES & CALLS

THANKS TO CHRIS ESPINOSA, PAUL WYMAN, & APPLE COMPUTER

CALLS

Most of these functions can be executed by a CALL to the proper address from an Integer BASIC program. Some CALLs may not be executed in the middle of a line in an Integer BASIC program: The CALL must be alone or at the end of a program line.

HEX ADD	CALL	EXPLANATION
$E000	CALL -8192	Reset Integer BASIC Integer Basic Kill -Clear workspace
$E04B	CALL -8117	List Integer program
$E836	CALL -6090	Run Integer
$F07B	CALL -3973	Load Integer Program from tape
$F140	CALL -3776	Save Integer Program to tape
$F666	CALL -2458	Enter mini-assembler
$F800	CALL -2048	Plot a point
$F819	CALL -2023	Draw a horizontal line
$F828	CALL -2008	Draw a vertical line
$F832	CALL -1998	Clear Graphic Screen
$F847	CALL -1977	Calc. graphics baseaddress
$F85F	CALL -1953	Change Color +3
$F864	CALL -1948	Adjust color byte for both halves equal
$F871	CALL -1935	Read a point of the low res graphics screen 1
$FA43	CALL -1469	Perform a single step
$FA86	CALL -1402	IRQ Handler
$FA92	CALL -1390	Break handler
$FAD7	CALL -1321	Display user reg
$FB1E	CALL -1250	Read paddle controls
$FB2F	CALL -1233	Screen initialization Reset TEXT mode
$FB39	CALL -1223	Set text screen
$FB40	CALL -1216	Set graphics screen
$FB4B	CALL -1205	Set normal window
$FB63	CALL -1181	Multiply routine
$FB84	CALL -1148	Divide routine
$FBC1	CALL -1087	Calculate text base address
$FBE4	CALL -1052	Produce a bell
$FBF4	CALL -1036	Advance Cursor
$FBFD	CALL -1027	Output a reg as ASCII on text screen 1
$FC10	CALL -1008	Backspace Cursor
$FC1A	CALL -998	Move cursor up
$FC22	CALL -990	Perform vertical tab
$FC2C	CALL -980	Escape functions
$FC9C	CALL -868	Clear to end of line
$FC42	CALL -958	Clear to end of screen
$FC58	CALL -936	Home and Clear
$FC62	CALL -926	Carriage return
$FC66	CALL -922	Cursor down-line feed
$FC70	CALL -912	Scroll screen
$FC9C	CALL -868	Clear to end of line
$FCA8	CALL -856	Wait loop
$FD0C	CALL -756	Wait for Keypress
$FD1B	CALL -741	Monitor keyin routine
$FD35	CALL -715	Read key & per esc fun if necessary
$FD62	CALL -670	Perform a line cancel(/)
$FD67	CALL -665	Perform Line Feed and wait for input
$FD6A	CALL -662	Prompt and wait for input
$FD6F	CALL -657	Get a Line
$FDDA	CALL -550	Print a byte as 2 hex digits
$FDE3	CALL -541	Print hex digit
$FDED	CALL -531	Output char. via user output
$FDF0	CALL -528	Monitor char. output
$FE2C	CALL -468	Perform memory move
$FE36	CALL -458	Perform memory verify
$FE5E	CALL -418	Disassemble 20 instr.
$FE80	CALL -384	Set Inverse Mode
$FE84	CALL -380	Set Normal Mode
$FEB0	CALL -336	Jump to basic
$FEB3	CALL -333	Continue basic
$FEBF	CALL -321	Display registers A,X,Y,P,S Current Values
$FEC2	CALL -318	Perform monitor trace
$FECD	CALL -307	Write to tape
$FEFD	CALL -259	Read from tape
$FF3A	CALL -198	Ring bell
$FF4A	CALL -182	Save registers
$FF59	CALL -167	Enter System Monitor, reset text mode
$FF65	CALL -155	Enter System Monitor & Ring Bell
$FF69	CALL -151	Enter System Monitor No beep
$FF70	CALL -144	Scan input buffer
$FF70	CALL -144	Scan the input buffer

The following CALLS require various address to be POKEd into locations in the Apple's memory. These locations are organized into three "registers", called A1, A2, and A4. The actual address for A1, A2, and A4 can be found in the section on PEEKs.

$FE2C	CALL -468	Move a Range of Memory
$FE36	call -458	Verify Two Ranges of Memory
$FECD	CALL -307	Write a Range to Tape
$FEFD	CALL -259	Read a Range from tape
$FD92	CALL -622	Print A1 in Hexadecimal

The following subroutines require you to POKE several values into some memory locations. These locations represent the microprocessor's A,X,Y and Status regusters, and the Stack Pointer (These are the same registers you can examine with the Monitor's CTRL E command). For the address of these locations see section on PEEKs.

$FEB6	CALL -622	Call a Subroutine

(This loads the internal registers with the values you have POKEd into locations 69 through 73, and jumps to the subroutine whose address has been POKEd into locations 58 and 559 (PCL & PCH). The subroutine acts just as if it had been CALLed from BASIC directly, and returns accordingly.

$FEBF	CALL -321	Display Registers (69 thru 73).
$FDDA	CALL -550	Display the A-register in Hex (location 69).

POKES

HEX ADD	POKE DEC ADD	EXPLANATION VALUE LIMITS
$0020	POKE 32,X	Window Left 0-39
$0021	POKE 33,X	Window Width 0-40
$0022	POKE 34,X	Window Top 0-24
$0023	POKE 35,X	Window Bottom 0-24
$0024	POKE 36,X	Cursor Horizontal
$0025	POKE 37,X	Cursor Vertical
$0026	POKE 38,X	GBASL Graphics baseaddress
$0027	POKE 39,X	GBASH
$0028	POKE 40,X	BASL Text screen base address
$0029	POKE 41,X	BASH
$002A	POKE 42,X	BAS2L Temp. base addr.
$002B	POKE 43,X	BAS2H
$0030	POKE 48,X	Color (0-15)
$0031	POKE 49,X	Mode

POKES (CONT.)

$0032	POKE 50,X	COUT text modes 255-Normal mode 63-Inverse mode 127-Flashing mode
$0033	POKE 51,X	Prompt Character
$0036	POKE 54,L	CSWL
$0037	POKE 55,H	CSWH:CHARACTER OUTPUT
$0038	POKE 56,L	KSWL
$0039	POKE 57,H	KSWH:CHARACTER INPUT

The POKEs below may be used before a CALL -327 to CANL a machine language program with parameters.

$003A	POKE 58,L	PCL
$003B	POKE 59,H	PCH
$0045	POKE 69,X	A-register
$0046	POKE 70,X	X-register
$0047	POKE 71,X	Y-register
$0048	POKE 72,X	Status register
$0049	POKE 73,X	Stack Pointer

POKEs below (60-67) can be used in conjunction with various CALLs to perform many System Monitor commands from BASIC (see CALL list).

$003C	POKE 60,L	A1L
$003D	POKE 61,H	A1H
$003E	POKE 62,L	A2L
$003F	POKE 63,H	A2H
$0042	POKE 66,L	A4L
$0043	POKE 67,H	A4H
C$C010	POKE-16368,X	Clear Keyboard Strobe
$C020	POKE-16532,X	Toggle Cassette output
$C030	POKE-16336,X	Toggle Speaker
$C040	POKE-16320,X	Strobe Game I/O
$C050	POKE-16305,X	Set Graphics Mode
$C051	POKE-16303,X	Set Text Modes
$C052	POKE-16302,X	Full-Screen Graphics
$C053	POKE-16301,X	Mixed Text and Graphics
$C054	POKE-16300,X	Display Primary Page
$C055	POKE-16299,X	Display Secondary Page
$C056	POKE-16298,X	Set Block Graphics
$C057	POKE-16297,X	Set Dot Graphics
$C000	POKE-16384,X	Read Key to see which key.

A POKE to one of the locations below specifies the state of the output on the pin of the Game I/O Connector associated with that Annunciator. Set is +5 volts; clear is 0 volts.

$C058	POKE-16296,X	Clear,
$C059	POKE-16295,X	Set Annunciator 0
$C05A	POKE-16294,X	Clear,
$C05B	POKE-16293,X	Set Annunciator 1
$C05C	POKE-16292,X	Clear,
$C05D	POKE-16291,X	Set Annunciator 2
$C05E	POKE-16290,X	Clear,
$C05F	POKE-16289,X	Set Annunciator 3

P E E K S

HEX	PEEK	EXPLANATION
$0020	X=PEEK(32)	Window Left
$0021	X=PEEK(33)	Window Width
$0022	X=PEEK(34)	Window Top
$0023	X=PEEK(35)	Window Bottom
$0024	X=PEEK(36)	Cursor Horizontal
$0025	X=PEEK(37)	Cursor Vertical
$0030	X=PEEK(48)	Color
$0032	X=PEEK(50)	Video mode
$0033	X=PEEK(51)	Prompt Character
$C000	X=PEEK(-16384)	Keyboard
$C010	X=PEEK(-16368)	Clear key. strobe
$C050	X=PEEK(-16304)	GRaphics mode
$C051	X=PEEK(-16303)	TEXT mode
$Co52	X=PEEK(-16302)	all TEXT or GRAPHICS
$C053	X=PEEK(-16301)	mix TEXT and a GRaphics mode
$C054	X=PEEK(-16300)	Primary page 1.
$C055	X=PEEK(-16299)	Secondary page 2.
$C056	X=PEEK(-16298)	LO-RES GRaphics
$C057	X=PEEK(-16297)	HI-RES GRaphics
$C060	X=PEEK(-16288)	Cassette Input
$C061	X=PEEK(-16287)	Read Pushbutton controller 0
$C062	X=PEEK(-16286)	Read Pushbutton controller 1
$c063	X=PEEK(-16285)	Read Game I/O Pin 4

X = PEEK (218) + PEEK (219) x 256 sets X equal to the line number of the statement where an error occurred if an ONERRGOTO statement has been executed.
IF PEEK (216)>127 THEN GOTO ----. If bit 7 at memory location 222 (ERRFLG) has been set true, then an ONERRGOTO statement has been encountered.
POKE 216,0 Clears ERRFLG so that normal error messages will occur.
Y = PEEK (222) Sets variable Y to a code that described type of error that caused an ONERRGOTO jump to occur.

The locations below may be PEEKed to determine what type of Apple Intelligent Interface card is installed in each slot. If X is 44 - Communications Card, 162 - Disk Controller, 72 - Heuristics Speechlab. Others not known at this time.

$C100	X=PEEK(-16128)	Slot 1
$C200	X=PEEK(-15872)	Slot 2
$C300	X=PEEK(-15616)	Slot 3
$C400	X=PEEK(-15630)	Slot 4
$C500	X=PEEK(-15104)	Slot 5
$C600	X=PEEK(-14848)	Slot 6
$C700	X=PEEK(-14592)	Slot 7

A PEEK at this location determines which Monitor ROM is installed in the Apple. If X is 0, then the Auto-Boot Monitor is in; if X is 1, then the original Apple Monitor ROM is installed.

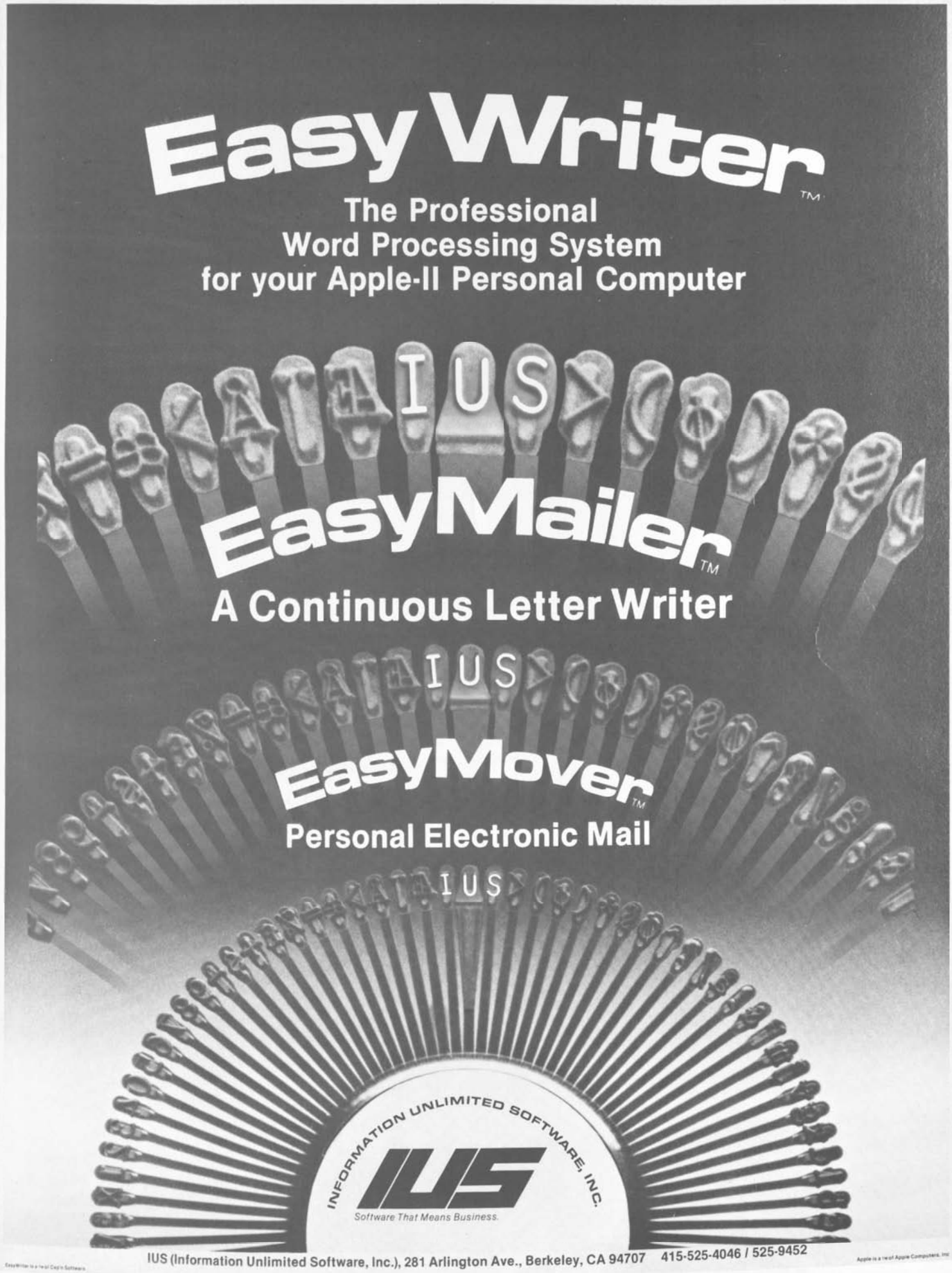
$FAFF	X=PEEK(-1281)	Monitor

The PEEKs below all require you to convert the values in two separate memory locations into one decimal number which BASIC can handle. This conversion involves two steps: first, obtaining the values in the locations; and second, amalgamating the two values into one decimal number. The following pairs of PEEKs give two values, stored in the two variables L and H.

$0028	L=PEEK(40)	BASL
$0029	H=PEEK(41)	BASH-Cursor Base Address
$0036	L=PEEK(54)	CSWL
$0037	H=PEEK(55)	CSWH-Output Subroutine
$0038	L=PEEK(56)	KSWL
$0039	H=PEEK(57)	KSWL-Input Subroutine
$004E	L=PEEK(78)	RNDL
$004F	H=PEEK(79)	RNDL-Random Number Seed
$004A	L=PEEK(74)	LOMEML
$004B	H=PEEK(75)	LOMEMH
$004C	L=PEEK(76)	HIMEML
$004D	H=PEEK(77)	HIMEMH
$00CA	L=PEEK(202)	PPL
$00CB	H=PEEK(203)	PPH-Program Pointer
$00CC	L=PEEK(204)	PVL
$00CD	H=PEEK(205)	PVH-Variable Pointer
$00DC	L=PEEK(220)	LINEL
$00DD	H=PEEK(221)	LINEH-Line Number
$00E0	L=PEEK(224)	STL
$00E1	H=PEEK(225)	STL-Beginning of Statement

NOTES

1068
1024
44


EasyWriter™
The Professional
Word Processing System
for your Apple-II Personal Computer
IUS
EasyMailer™
A Continuous Letter Writer
IUS
EasyMover™
Personal Electronic Mail
IUS
INFORMATION UNLIMITED SOFTWARE, INC.
IUS
Software That Means Business.
IUS (Information Unlimited Software, Inc.), 281 Arlington Ave., Berkeley, CA 94707 415-525-4046 / 525-9452