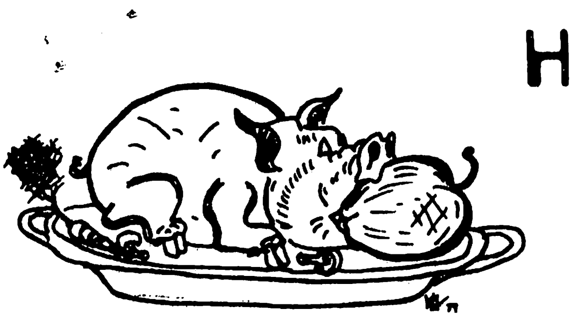
H.A.A.U.G.



HOUSTON AREA APPLE USERS GROUP

THE APPLE BARREL

< SINGLE COPY PRICE $1.00 >

==

VOLUME 3 NO. 7 SEPTEMBER/OCTOBER, 1980

President, Bruce Barber

Editor, Ed Seeger

==

<<< CONTENTS >>>

<<< CLUB NOTES >>>

Houston Area Apple Users Group
APPLE BARREL
4331 Nenana Drive
Houston, TX 77035

The HOUSTON AREA APPLE USERS GROUP is an Apple II user club, not affiliated with Apple, Inc., or with any retail computer store. HAAUG is a member of the International Apple Core and supports its purposes and publications. General membership meetings are held on the second Wednesday of each month in the rear chapel of Memorial Lutheran Church, 5800 Westheimer, right by the Jungman Branch Library. They start at 6:30 p.m. An additional software swap is held the last Saturday of each month at the clubhouse of the Houston Amateur Radio Club, 7011 Lozier Street, east of the Astrodome. These Saturday meetings begin at 2:00 p.m.

-----==*=====

OFFICERS / EXECUTIVE BOARD

President	Bruce Barber	469-5805
Vice President	(vacant)	
Treasurer	Ray Essig	497-7165
Secretary	James Odom	426-3970
Software Lib.	Dennis Cornwell	774-0671
Hardcopy Lib.	Leslie Doest	472-5485
Hardware Chair	David Marchand	497-7366
Business Uses	Rudge Allen	622-3979
Membership	Lee Gilbreth	342-2685
Newsletter Ed.	Ed Seeger	723-6919

-----==*=====

MEMBERSHIP INFORMATION

Dues are $18.00 per 12-month period for regular memberships, $6.00 for students through high school and where no adult member of the family is an Apple user. Please make checks payable to "Houston Area Apple Users Group," and mail to Lee E. Gilbreth, Membership Chair, 3609 Glenmeadow, Rosenberg, TX 77471. This includes a subscription to APPLE BARREL, which is published nine times a year. Newsletter exchanges with similar clubs are invited.

-----==*=====

APPLE BARREL REPRINT POLICY

Unless otherwise indicated within the program or article, any ORIGINAL material published herein may be reprinted without permission by any non-profit Apple club, group or newsletter, PROVIDED proper credit is given to the APPLE BARREL and the article or program author.

-----==*=====

SPECIAL INTEREST GROUPS

Members who share a common interest are encouraged to form Special Interest Groups to more fully explore their fields. Meetings may be arranged by common consent of the group and will ordinarily have one member who serves to coordinate or convene the meetings. If you would like to start a group around any given interest, please contact one of the club officers. If you would like to be in touch with others who share one of the following interests with you, please phone the coordinator.

Current groups are:

1) BUSINESS APPLICATIONS
   Coordinated by Rudge Allen,
   622-3979

2) PASCAL USERS
   Directory being assembled
   Pat McGee coordinating,
   663-6806
   This Special Interest Group is to meet and discuss aspects of Apple's Pascal language and to exchange programs.

3) MODEM USERS
Directory being assembled
Herb Crosby coordinating,
497-1061

4) HAM RADIO OPERATORS
Coordinated by Ed Seeger, WB5PTW
723-6919

5) NEW MEMBERS
Coordinated by Lee Gilbreth,
342-2685

6) EDUCATIONAL APPLICATIONS
Coordinated by Darrell Kachilla,
498-0186

7) BEGINNERS' PROGRAMMING
Coordinated by John C. Whiteman,
794-7267 (home)
This Special Interest Group is to meet and discuss Integer Basic and Applesoft.

8) FILE CABINET
Coordinated by Lee Gilbreth,
342-2685
Purpose is to understand, expand and enhance the File Cabinet program.

-----==*==----

APPLE BULLETIN BOARD SYSTEM

The Houston Area Apple Users Group supports an ABBS evenings and weekends, 6:00 pm through 8:30 am, and all weekend long. Feel free to sign-on and place your want-ad, meeting notice, request for help, Aggie joke, etc. Any ASCII terminal, Apple computer or not, with suitable modem or coupler, will give you ABBS capability. Call:

713/654-0759

SYSOP is Rudge Allen, 622-3979.

"No, Baxter, you're not being replaced by a computer, only a silicone chip."

Apple Fervor Puts Brokers On the Spot

By TIM METZ and PAUL BLUSTEIN
Staff Reporters of THE WALL STREET JOURNAL

Every speculator in hot new issues wants a bite of Apple—Apple Computer Inc.—but most will be lucky to get even a bit.

The personal computer manufacturer's first public sale of stock seems likely to become one of the hottest offerings of all time.

"Our phone has been ringing," a Minneapolis broker says. "Sometimes it'll be people who may have had accounts with us in the past. Sometimes it's people wanting to open new accounts. All of a sudden they want to be friendly. They want a couple of hundred shares of Apple."

Says Dan Mandresh, a securities analyst at Merrill Lynch, Pierce, Fenner & Smith Inc.: "Even my brother, who invests in the stock market only on Tuesdays in Leap Year, called the other day to ask what I know about Apple Computer. I said, 'My God, Marty, not you, too!' " Mr. Mandresh says he knows little about Apple.

A date hasn't been set for the Apple stock sale. Lately, share prices of nearly all companies in the personal-computer business have hit record levels. New issues of computer and other high-technology stocks sold publicly in the past 12 months have soared in price by as much as 50% or more above initial offering prices.

The demand for Apple is especially keen because the company ranks with Tandy Corp., maker of the Radio Shack's TRS model personal computers, as a leader in the industry. Some people expect Apple sales to reach $300 million next year from some $150 million this year and only $7 million two years ago.

All but a minority of would-be Apple buyers seem likely to come away from the public offering empty-handed. The supply is expected to be so scant that brokers already are devising allocation methods. At the Minneapolis broker's office, for example, customers' men will draw straws to determine who gets the office's allocation. The investors who do get to buy the stock are likely to be well-heeled customers of long standing.

Good Customers Favored

"Those who give us the business get the business," says Charles Ness, a broker for Shearson Loeb Rhoades Inc. in Seattle. "A client who's done a good bit of business with us is given first crack at a hot new issue."

Another broker insists that a customer's "style," not just the size of his account, will influence his chances to get Apple. The broker, Randy Estes, with E. F. Hutton & Co. in San Diego, says that if he gets any shares to sell, "I'll go to the people who'll buy any new issue. The ones who are with you through thick and thin."

Complaints Likely

Some unsatisfied customers are likely to complain. If they can't buy Apple in the public offering, they'll have to buy it afterward in the secondary market, presumably at a much higher price.

William M. LeFevre, investment policy vice president at Purcell Graham & Co., a smaller Wall Street securities firm, recalls some irritated customers following a hot new issue, Wang Laboratories, back in 1967. "I was allotted only five shares," he says, "and I decided to sell all five to one of my best customers. But he was a loudmouth. When the stock shot up to $50 from an issuing price of $10, he told people at the golf club that he had 500 shares. Word got around and my other good customers asked how I could get 500 shares for a simpleton like him and couldn't get any for them."

For big institutional investors, the jockeying for chunks of Apple won't begin until Apple files its preliminary prospectus describing the terms of its offering with the Securities and Exchange Commission. The filing could come any day. "It's safe to say that everybody is going to be able to find some money to buy Apple stock," says Manown Kisor Jr., senior investment officer at Detroit Bank & Trust Co.

Mum's the Word

Distinctly worried over the hoopla are managers at the prestigious investment banking firm of Morgan Stanley & Co., which is expected to become the lead underwriter of the Apple issue. Although Morgan declines to comment, the firm tacitly acknowledged that it is being besieged with inquiries about Apple. It sent its staffers a memo the other day pointing out that underwriters for the issue haven't yet been named, and that any comment about Apple is inappropriate. Morgan's fear is that all the chatter over Apple might smack to the SEC of unlawful touting, or blue-skying.

Veteran Wall Street securities men worry that demand could push Apple's offering price or later prices to unrealistically high levels.

"We're getting into the silly season," the Tucson broker says of the new-issue market. "It's really getting wild."

Mr. LeFevre, comparing the demand for Apple with other alluring things, observes that "it could turn out that the anticipation is so much better than the realization."

Reproduced from

The Wall Street Journal

October 10, 1980

THE WALL STREET JOURNAL, Wednesday, Oct. 1, 1980 45

Nautilus Fund Purchases More Apple Computer Stock

BOSTON—Nautilus Fund, a closed-end unit investing in so-called emerging companies, says it bought another 20,000 shares of Apple Computer Inc., expected to be a hot stock when its shares go public later this year.

The latest purchase, like the others was a private transaction. It increases Nautilus's holding in Apple to 180,000 shares. Price of the latest batch was $8.25 a share.

Nautilus, managed by Eaton & Howard, Vance Sanders Inc., said that it is boosting the carrying value of all 180,000 shares to $8.25 each from $2.625. Overall, Nautilus says, this will add about $1.25 a share to the fund's net asset value.

As of June 30, the fund's net asset value was $17.66 a share.

Because the Apple shares aren't publicly traded, Nautilus said, their value is based on the fund's "best judgment," rather than market price. Apple plans a $25 million offering in November or December.

FILE CABINET PARTIALLY EXPOSED

In the heart of FILE CABINET are two subroutines which, if understood, should dispell much of the mystery from this popular program found in our club Software Library. These routines are called upon sixteen times directly and countless times indirectly during a full running of the program. This is no small wonder, for they are the SAVE FILES and READ FILES of the data management system which has the disk drive hopping back and forth saving and retrieving text files.

Since both routines are mirror images of each other, they should be viewed together:

```
4280 REM * * * SAVE FILES * * *
4290 IF F$< >"INDEX" THEN FF = 1
4300 PRINT D$"OPEN"DB$" "F$"FILE"
4310 PRINT D$"WRITE"DB$" "F$"FILE"
4320 PRINT NR
4330 FOR J = 1 TO NR
4340 ON FF GOTO 4390
4350 FOR I = 1 TO NH
4360 PRINT N$(J,I)
4370 NEXT I
4380 GOTO 4400
4390 PRINT R$(J)
4400 NEXT J
4410 PRINT D$"CLOSE"
4420 FF = 0
4430 RETURN
```

```
4110 REM * * * READ FILES * * *
4120 IF F$< >"INDEX" THEN FF = 1
4130 PRINT D$"OPEN"DB$" "F$"FILE"
4140 PRINT D$"READ"DB$" "F$"FILE"
4150 INPUT NR
4160 FOR J = 1 TO NR
4170 ON FF GOTO 4230
4180 FOR I = 1 TO NH
4190 INPUT N$(J,I)
4210 NEXT I
4220 GOTO 4240
4230 INPUT R$(J)
4240 NEXT J
4250 PRINT D$"CLOSE"
4260 FF = 0
4270 RETURN
```

The titles and line numbers are naturally different and where one WRITES the file the other READS it. The act of writing is through the PRINT command and the act of reading is through the INPUT command. The varibles used above are:

F$ = Type of File (eg. BASENAME, HEADER, INDEX, etc)
FF = Flag for type of Array stored (eg. 0=one dimension,1=two dimension)
DB$ = Name of Data Base
NR = Number of Records (data elements following) in the Text File
NH = Number of Headers that make up a Record
R$(J) = Data Array (one dimensional)
N$(J,I) = Data Array (two dimensional)

All text files of FILE CABINET are of the Sequential type. (See DOS Manual.) The first informational element stored will always be the total number of Record elements expected to follow. Files therefore, graphically look like this:

TEXT FILE	NR	R$(1)	R$(2)	...	R$(J)	...	R$(NR)
HEADERFILE	7	H#1	H#2		...		H#7
DATABASEFILE	3	DB#1	DB#2				DB#3
RPTNAMEFILE	4	RN#1	RN#2		...		RN#4

Actual Record data is stored in the same manner. Illustrated below would be a three header file with four Records of information:

TEXT FILE	NR	N$(1,1)	N$(1,2)	N$(1,3)	N$(2,1)	...	N$(J,I)	...	N$(NR,NH)
INDEXFILE	12	R#1,H1	R#1,H2	R#1,H3	R#2,H1		...		R#4,H3

Even the REPORT FORMAT File follows the same pattern. It signals the total number of data elements to follow and then stores them in blocks of three. The example below would be for a Report Format File containing five headers:

NS	K(1)	K(2)	K(3)...	...K(I-2)	K(I-1)	K(I)...	...K(3*RH-2)	K(3*RH-1)	K(3*RH)	K(0)	K(NR)
17	No. of H#1	Tab for H#1	Flag total H#1		...		No. of H#5	Tab for H#5	Flag total H#5	Flag for TOTAL	Tab Headr TOTAL

The number "NS" states how many elements are in the file. The K(1) element contains the Header Number for the first column in the report. The K(2) element gives its Tab Location and the K(3) element determines if it is to be included in the Totaling scheme (0 = Not to be Totaled, 1 = Include in Totals). After all Headers are positioned in the report, the K(0) Flag triggers the Grand Totaling process (0 = Make no Totals, 1 = Make Totals). Element K(NR) is tacked on at the end to give the Tab Location for TOTAL in the report.

Of course there is a lot more to FILE CABINET than comprehending the basic structure of its Text Files. In time, we shall study other aspects of the program and expose all.

<<< Lee Gilbreth >>>

WANT TO BUY AN APPLESOFT ROMCARD for a low to reasonable price. Fred Fuchs, 749-3235 or 781-6968.

<<< WATCH THIS SPACE! >>>

Coming very soon in your NOVEMBER APPLE BARREL is more Pascal notes from Pat McGee; CCA Data Management System Version 5.2 Upgrade memo; information on the UCSD Pascal Users Group Library (which we have on disk ready for distribution!); and the usual assemblage of notes, code and ads that make life worth living.

In the DECEMBER APPLE BARREL, look for a full review of the "almost perfect" MAGIC WAND word processor, which is now implemented under CP/M on the Apple! This is a program which, like Visicalc, is by itself sufficient reason to own an Apple. We will also bring you a holiday gift of good programming from other Apple-oriented newsletters from throughout the country.

USING THE BACKSPACE AS A DELETE KEY

by Kevin Winter

The following program takes advantage of the zero page location $38-39, which contains the vector to a user's key-in routine (default = $FD1B). The program is locatable anywhere in memory and is only 26 bytes long. The simple format will allow anyone to extend the program to add any number of special functions.

I used the mini-assembler to enter the following code:

```
5000: BIT $C000    CHECK FOR KEY PRESSED
5003: BPL $300     IF NOT PRESSED GOTO $300
5005: STA ($28),Y  GOT KEY - PUT ON SCREEN
5007: LDA $C000    PUT KEY INTO ACCUMULATOR
500A: BIT $C010    CLEAR KEY STROBE
500D: CMP #88      IS KEY A BACKSPACE
500F: BEQ $312     IF NOT GOTO $312
5011: RTS          IF YES RETURN TO NORMAL INPUT
5012: PHA          PUSH BACKSPACE INTO STACK
5013: LDA #A0      LOAD ACCUM WITH A SPACE
5015: DEY          DECREMENT SCREEN POSITION
5016: STA ($28),Y  STORE SPACE ON TOP OF BAD CHARACTER
5018: PLA          PULL BACKSPACE FROM STACK
5019: RTS          RETURN TO NORMAL INPUT
```

To use routine with DOS you need:

```
5020: PHA          SAVE ACCUM TO STACK
5021: LDA #$00     STORE LOW BYTE ADDRESS
5023: STA $38      IN $38 (KEY-IN VECTOR)
5025: LDA #$50     STORE HIGH BYTE ADDRESS
5027: STA $39      IN $39 (KEY-IN VECTOR)
5029: JSR $03EA    GOSUB TO DOS HOOKS
502C: PLA          GET ACCUM FROM STACK
502D: RTS          RETURN TO MONITOR/BASIC
```

Or one can use this entry:

```
5000: 2C 00 C0 10 FB 91 28 AD
5008: 00 C0 2C 10 C0 C9 88 F0
5010: 01 60 48 A9 A0 88 91 28
5018: 68 60
        (To use with DOS)
5020: 48 A9 00 85 38 A9 50 85
5028: 39 20 EA 03 68 60
```

To activate the function, if you use code $5000-5019, just enter '*38: 00 50' into the Monitor, which is the address of the code. Then you can use DELETE in machine code or enter BASIC and it will work. If you have a disk, you will need the code $5020-502D, by entering '*5020G' if in Monitor, or 'CALL 20512', if in BASIC.

The idea for this article came from 'CP/M Backspace Mod' by Rod Hallen (pg 48 Aug 80 issue of Kilobaud/Micro).

A BRIEF REVIEW OF THE MOUNTAIN HARDWARE MUSIC SYSTEM:

Incredibly disappointing.

A SOMEWHAT LESS BRIEF REVIEW OF THE MOUNTAIN HARDWARE MUSIC SYSTEM:

It is pathetically obvious that this product was released before it was finished. I find it hard to imagine that a normally reputable company like Mountain Hardware could not know about the major bugs and shortcomings in the manual and especially the software. After buying this product because of their reputation, I will never again buy a Mountain Hardware product without examining it in detail first. Well, enough moaning, on with the review.

First, the hardware: Its great. It sounds excellent when compared with an ALF system. The system comes with several instruments preprogrammed. The organ really sounds like an organ. A real pipe organ sounds better, but the MusicSystem could hold its head up among moderately priced home organs.

Now, the software. This is really a mixed bag. If you were looking just at the specifications, it would look great: input from keyboard, light pen, or paddles; ability to input dynamics and accents; ability to input chords; ability to play different parts with different instruments; etc. It all sounds great. And, if you have a semi-infinite amount of patience, it is. And therein lies almost the entirity of my disappointment. It takes so long to do each and every little thing that it isn't fun. Even just putting in notes takes long enough to be annoying. The wait after you decide to play something until the music starts can be downright stultifying. When I had a set of ALF boards, I had to force myself to work instead of playing with the music stuff. Now, with the Mountain Hardware MusicSystem, I have to force myself to use the music stuff instead of working. And that makes for a lousy toy.

I won't mention the many bugs that I have found in the software and the manual, except to say that most are glaringly obvious, and show a total disregard for anyone who should ever have to actually use this product after they have bought it.

Why haven't I sold mine yet? Well, mostly because of faith. Faith in Mountain Hardware that they will fix the obvious defects (because they won't sell many more if for nothing else), and faith in the Users group that Mountain Hardware is starting and supporting. This is too good a piece of hardware to be saddled with such a lousy software driver for long. However, if someone offers me a good price now, I'd probably take it.

Recommendation: If you want a great sounding music system and think you have the patience of Job, think about getting one now; but try to do some real music on it before you buy. Or, wait six months and see what changes have come down the road on the software. If you can't wait six months and want a music system to have fun with rather that serious work, consider the ALF system. It is fun.

Pascal Problems

by Pat McGee
P.O.Box 20223
Houston, Texas 77025

This is a list of problems I have had using the Apple Pascal system. Some are outright bugs, while others are problems caused by poor documentation.

Long Integers:

I expected them to work just like regular integers, except hold bigger numbers. They don't. In some places they do, in others they cause compilation errors, and sometimes they just plain don't work.

They do work as expected in most arithmetic expressions and a parameters to functions and procedures.

Trying to have a function return a value of type long integer causes a compilation error. The Apple Hot Line said that this was a limitation that had not been documented, not a bug. Long integers are similar in internal format to strings, and strings cannot be used in this manner.

There are several bugs involving long integers.

1. Typing a 10 digit number when the system is executing
```
Read(input,I)  where I:Integer[9]
```
usually causes the system to crash. The only way to recover is to reboot.
2. Sometimes, keying in any number when the system is trying to read a long integer will cause it to *STK OFLOW* and reinitialize itself. I haven't found exactly what things work and what don't.
3. The expression   TRUNC(Adr - 32768) where Adr:Integer causes *STK OFLOW*, but TRUNC(Adr - 16384 - 16384) does not.

Mod Function:

This does not work properly. Jensen & Wirth (p13) state that
A Mod B = A-((A div B)*B).
However, in Apple Pascal, it is implemented as
A mod B = |A|-((|A| div B)*B).
This can be seen by looking at -1 mod 2. This is particularly bad when looking at the definition of modulo munbers from back in high school. I was taught that if A mod B = C then (A+B) mod B was also = C. The implementation does not match this.

Arctangent Function ATAN:

This function returns the wrong angle for tangents less than -1. Use the following code when you want to use this:

```
If Tangent < 0 then
  Angle := -Atan(-Tangent)
Else
  Angle := Atan(Tangent);
```

For Loops:

I was trying to time a for loop, so I typed in:

```
Writeln(output,'BEFORE LOOP');
For i := 0 to 32767 do {nothing};
Writeln(output,'AFTER LOOP');
```

The computer printed "BEFORE LOOP", then I waited, with cocked stopwatch. After a while, I decided an alarm clock would be a more appropriate instrument. Even later, I was considering a calendar. Well, back to the drawing board. Changing 32767 to 32766 produced a nice quick loop, but changing it back to 32767 caused another infinite wait.

Apparently, the compiler designers blew it. The value of I should have been checked against 32767 before being incremented, or the increment should have checked for overflow.

To avoid the problem, either use 32766 or do the following:

```
Const Max = 32767
Type LoopControlState = (looping,thru);
Var  State:LoopControlState;
     I:Integer
Begin
  I := 0; State := looping;
  Repeat
    { Whatever }
    If I < Max then I := I+1 else State := thru;
  Until State = thru;
```

I use this instead of any for loop, because it is more versitile, and because it works in all cases. There are other reasons involving the use of variables that do not go outside the specified range.

Filer W(hat Command:

This command tells you the name of the workfile and whether it has been saved or not. In a single drive system, it works file. But, in a multiple drive system if you G(et a file from a different disk drive than you booted from, do something to it, then S(ave it back to the other disk, the W(hat command thinks that the workfile has not been saved, when in fact it has been.

Filer T(ransfer Command:

If you have two disks in the system at the same time and they have the same name, DON'T USE THE T COMMAND!!!!!!! You will wipe out part of at least one disk!. The filer gets very confused under these circumstances, and is apt to wipe out the disk you are transferring from, as well as the one you are transferring to. Furthermore, you sometimes don't find out until later just which files are messed up. They will look just fine in the directory, but the contents will be so much garbage.

If you must to this, first change the name of one of the disks, do the transfer, then change the name back to the original. The manual says (once, in a very obscure place which I can't find again) not to put in two disks with the same name, but doesn't say why.

Another problem I had was in using the T command to transfer several files from one disk to another. When I keyed in

```
T  AMF:T.=.TEXT,AMFBACK:$
```

I got the messge DESTROY AMFBACK:? (Y/N)

I don't know what would have happened if I had said yes because I never had the guts to try it.

System Library:

Several times I have seen the message:

```
REQUIRED INTRINSIC(S) NOT AVAILABLE
```

when trying to R(un or E(xecute a program. I soon found out that SYSTEM.LIBRARY had to be in the system. However, this was not the complete answer as I found out when I put a disk with it in #4 and tried again. As it turns out you MUST boot from a disk that has the library on it. If you boot from a disk without it, then put in a disk with it, the system can't find it.

appropriate instrument. Even later, I was considering a calendar. Well, back to the drawing board. Changing 32767 to 32766 produced a nice quick loop, but changing it back to 32767 caused another infinite wait.

Apparently, the compiler designers blew it. The value of I should have been checked against 32767 before being incremented, or the increment should have checked for overflow.

To avoid the problem, either use 32766 or do the following:

```
Const Max = 32767
Type LoopControlState = (looping,thru);
Var  State:LoopControlState;
     I:Integer
Begin
  I := 0; State := looping;
  Repeat
    { Whatever }
    If I < Max then I := I+1 else State := thru;
  Until State = thru;
```

I use this instead of any for loop, because it is more versitile, and because it works in all cases. There are other reasons involving the use of variables that do not go outside the specified range.

Filer W(hat Command:

in a multiple drive system

This command tells you the name of the workfile and whether it has been saved or not. In a single drive system, it works file. But, if you G(et a file from a different disk drive than you booted from, do something to it, then S(ave it back to the other disk, the W(hat command thinks that the workfile has not been saved, when in fact it has been.

Filer T(ransfer Command:

If you have two disks in the system at the same time and they have the same name, DON'T USE THE T COMMAND!!!!!!! You will wipe out part of at least one disk!. The filer gets very confused under these circumstances, and is apt to wipe out the disk you are transferring from, as well as the one you are transferring to. Furthermore, you sometimes don't find out until later just which files are messed up. They will look just fine in the directory, but the contents will be so much garbage.

If you must to this, first change the name of one of the disks, do the transfer, then change the name back to the original. The manual says (once, in a very obscure place which I can't find again) not to put in two disks with the same name, but doesn't say why.

Another problem I had was in using the T command to transfer several files from one disk to another. When I keyed in

```
T  AMF:T.=.TEXT,AMFBACK:$
```

I got the messge DESTROY AMFBACK:? (Y/N)

I don't know what would have happened if I had said yes because I never had the guts to try it.

System Library:

Several times I have seen the message:

```
REQUIRED INTRINSIC(S) NOT AVAILABLE
```

when trying to R(un or E(xecute a program. I soon found out that SYSTEM.LIBRARY had to be in the system. However, this was not the complete answer as I found out when I put a disk with it in #4 and tried again. As it turns out you MUST boot from a disk that has the library on it. If you boot from a disk without it, then put in a disk with it, the system can't find it.

This is documented in the manual, but only in a discussion of making a new library file. This is a place a beginner would not look at, and I skipped it my first few times through the manual. It should be in the section on E(xecute also.

Assembler:

When doing a forward branch (not a jump), the listing does not properly reflect the contents of the code file. When the branch is processed, the listing reads, for example:

```
D3EA|F0**       BEQ    $1
```

A few lines later, when the lobel is defined, the listing reads

```
D3EA*00
```

It should read

```
D3EB*05
```

Both the address and the contents are wrong.

Editor:

When in D(elete mode and deleting off the bottom of the screen, the editor rewrites the screen starting with the next line to be deleted at the top. It then blanks out the first 3 characters of that line and positions the cursor to the first blanked out character. These three characters have not been deleted, but the editor makes it look like they have been. Until I found out that everything was OK, I used to panic and ESC out of the delete and start over. This is not necessary, as they have not been deleted.

Conclusion:

This is not all the complaints I have with the Apple Pascal system, but a all the others involve the poor documentation or things that I would have designed differently. Most of the documentation problems I expect to be cleared up when Jef Raskin and his crew write a manual. The current manual was copied mostly verbatim from the UCSD Pascal manual, and almost all of its problems stem from that source.

If you have encountered a problem not in this list, please tell me (and Apple) about it. Hopefully we can work out a way to avoid it and keep others from wasting much effort finding the same bugs over again.

```
(* ALWAYS WONDERED HOW YOU COULD GET TO THE SYSTEM DATE STORED ON THE DISK
   BY THE F)ILER D)ATE COMMAND?  WELL, HERE IT IS *)


{$C(C) 1979 by John Strait.  Copying for non-profit use OK}
(* Copyright 1979 by John Strait, Three Rivers Computer Corp.
May not be sold for profit.  Copying for nonprofit use OK.*)

(* ADAPTED FOR STAND ALONE USE BY PAT MCGEE, 5 SEPT 1980 *)

PROGRAM GETDATE;
VAR
   RAWDATE      : STRING[8];
   NICEDATE     : STRING[9];


PROCEDURE INITDATES;
    CONST
      BLOCKNR =  2;
      UNITNR  =  4;
      ELEMENT = 11;
      BYTES   = 22;

    TYPE DATE = PACKED RECORD
                MONTH: 1 .. 12;
                DAY:   1 .. 31;
                YEAR:  0 .. 99;
      END; { date }

  VAR
    TODAY: DATE;
    BUFFER: PACKED ARRAY [1 .. ELEMENT] OF DATE;
    MONTH:  STRING[3];   { Month name }

  BEGIN (* INITDATES *)

    RAWDATE  := 'YY/MM/DD';   (* ASSIGN ANY STRING, WILL *)
    NICEDATE := 'DD MMM YY';   (*BE REPLACED BY INDIVIDUAL CHARS *)

    UNITREAD (UNITNR, BUFFER, BYTES, BLOCKNR);
    TODAY := BUFFER [ELEMENT];
    WITH TODAY DO BEGIN
      RAWDATE[ 1]   := CHR((YEAR DIV 10) + 48);
      RAWDATE[ 2]   := CHR((YEAR MOD 10) + 48);
      RAWDATE[ 3]   := '/';
      RAWDATE[ 4]   := CHR((MONTH DIV 10) + 48);
      RAWDATE[ 5]   := CHR((MONTH MOD 10) + 48);
      RAWDATE[ 6]   := '/';
      RAWDATE[ 7]   := CHR((DAY DIV 10) + 48);
      RAWDATE[ 8]   := CHR((DAY MOD 10) + 48);
    END; { WITH TODAY }

    CASE TODAY.MONTH OF
    1: MONTH := 'JAN';
    2: MONTH := 'FEB';
    3: MONTH := 'MAR';
    4: MONTH := 'APR';
    5: MONTH := 'MAY';
    6: MONTH := 'JUN';
    7: MONTH := 'JUL';
    8: MONTH := 'AUG';
    9: MONTH := 'SEP';
    10:MONTH := 'OCT';
    11:MONTH := 'NOV';
    12:MONTH := 'DEC';
    END (* CASE *);

    NICEDATE[ 1] := RAWDATE [7];
    NICEDATE[ 2] := RAWDATE [8];
    NICEDATE[ 3] := ' ';
    NICEDATE[ 4] := MONTH [1];
    NICEDATE[ 5] := MONTH [2];
    NICEDATE[ 6] := MONTH [3];
    NICEDATE[ 7] := ' ';
    NICEDATE[ 8] := RAWDATE [1];
    NICEDATE[ 9] := RAWDATE [2];

  END (* INITDATES *);


BEGIN (* MAIN *)
  WRITELN;
  INITDATES;
  WRITELN(RAWDATE);
  WRITELN(NICEDATE);
END.
```

<<< SCREEN CREATE >>>

by Bruce Barber

SCREEN CREATE is the "poor man's graphics tablet." This program will create high resolution graphic screens for use as signs or as backgrounds for hires games. Existing hires graphics can be loaded and modified. The program is self-documenting. At any time press 'H' for Help on commands.

As it is listed here, much of the programming IS for documentation, It is well-worth taking time to key it all in, for it then becomes instantly available with the 'H' command. It takes a little while to learn the command language, so the Help feature is an assset that will bring faster and more satisfying results.

Although all the features of a full graphics pad are by no means included, you do find here the basics of coordinate plotting, area filling, color selection, line drawing, etc. With care and imagination, it is possible to generate graphics of surprising sophistication.

One thoughtful feature is the flickering Grid to indicate distances of 20 points. The esc-G command toggles this coordinate system on and off, enabling the plotter to find the way when needed. In addition, your X-Y location is always read out to you when you enter the Help command.

"Random Lady With Moustache," anyone?

SCREEN CREATE

```
2  LOMEM: 25000
3 D$ = "": DIM X1%(300),Y1%(300):
 DIM H%(10):C = 3:IC = 0: HOME
: GOSUB 62000: HOME
5 X% = 140:Y% = 96: HGR2 : TEXT :
 GOSUB 61000: HGR : TEXT
145  POKE  - 16368,0:GG = 0: GOSUB
10000
160  IF  PEEK ( - 16384) > 127 THEN
170
161  IF GG = 1 THEN  POKE  - 1629
9,0:GG = 2: GOTO 160
162  IF GG = 2 THEN  POKE  - 1630
0,0:GG = 1: GOTO 160
163  GOTO 160
170 A$ =  CHR$ ( PEEK ( - 16384) -
128): POKE  - 16368,0
171  IF ES% = 1 THEN  GOTO 300
173  IF A$ =  CHR$ (27) THEN ES% =
1: GOTO 160
175  IF A$ = "U" THEN  GOTO 5000
180  IF A$ = "D" THEN  GOTO 5030
185  IF A$ = "R" THEN  GOTO 5090
187  IF A$ = "H" THEN 6000
188  IF A$ = "O" THEN C = 5: HCOLOR=
C: GOTO 160
189  IF A$ = "X" THEN C = 6: HCOLOR=
C: GOTO 160
190  IF A$ = "L" THEN  GOTO 5060
191  IF A$ = "W" THEN C = 7: HCOLOR=
C: GOTO 160
192  IF A$ = "B" THEN C = 0: HCOLOR=
C: GOTO 160
193  IF A$ = "G" THEN C = 1: HCOLOR=
C: GOTO 160
194  IF A$ = "V" THEN C = 2: HCOLOR=
C: GOTO 160
195  IF A$ = "1" THEN  GOTO 5120
196  IF A$ = "2" THEN  GOTO 5170
197  IF A$ = "3" THEN  GOTO 5210
198  IF A$ = "4" THEN  GOTO 5260
199  IF A$ = "P" THEN  GOTO 30000

200  IF A$ =  CHR$ (8) THEN 5400
202  IF A$ = "M" THEN RE = 0: GOTO
25000
204  IF A$ = "C" THEN 26000
206  IF A$ = "#" THEN 24000
298  GOTO 160
300 ES% = 0
305  IF A$ = "L" THEN  GOTO 60000
307  IF A$ = "G" AND GG = 0 THEN
GG = 1: GOTO 160
308  IF A$ = "G" AND GG > 0 THEN
GG = 0: POKE  - 16300,0: GOTO
160
310  IF A$ = "S" THEN  GOTO 59000

320  IF A$ = "E" THEN  TEXT : HOME
: END
330  IF A$ = "C" THEN  HGR : HCOLOR=
C: POKE 49234,0: GOTO 160
340  IF A$ = "T" THEN  POKE  - 16
300,0:GG = 0: HOME : GOSUB 1
0000: TEXT : GOTO 160
350  IF A$ = "H" THEN  GOTO 4900
999  GOTO 160
2502  IF X > 279 THEN X = 279
4900  POKE  - 16304,0: HCOLOR= C:
 POKE 49234,0: GOTO 160
5000 Y% = Y% - 1: IF Y% < 0 THEN
Y% = 0
5010  GOSUB 20000: GOTO 160
5030 Y% = Y% + 1: IF Y% > 191 THEN
Y% = 191
5040  GOSUB 20000: GOTO 160
5060 X% = X% - 1: IF X% < 0 THEN
X% = 0
5070  GOSUB 20000: GOTO 160
5090 X% = X% + 1: IF X% > 279 THEN
X% = 279
5100  GOSUB 20000: GOTO 160
5120 X% = X% - 1:Y% = Y% - 1
5130  IF X% < 0 THEN X% = 0
5140  IF Y% < 0 THEN Y% = 0
5150  GOSUB 20000
5160  GOTO 160
5170 X% = X% + 1:Y% = Y% - 1
5180  IF X% > 279 THEN X% = 279
5185  IF Y% < 0 THEN Y% = 0
5190  GOSUB 20000
5200  GOTO 160
5210 X% = X% + 1:Y% = Y% + 1
5220  IF X% > 279 THEN X% = 279
5230  IF Y% > 191 THEN Y% = 191
5240  GOSUB 20000
5250  GOTO 160
5260 X% = X% - 1:Y% = Y% + 1
5270  IF X% < 0 THEN X% = 0
5280  IF Y% > 279 THEN Y% = 191
5290  GOSUB 20000
5300  GOTO 160
5400  INPUT A$
5410  IF  VAL (A$) = 0 THEN  GOTO
160
5420 X =  VAL (A$)
5422  IF X =  - 999 THEN 160
5425  HCOLOR= 0
5430  FOR Y = IC TO IC - X + 1 STEP
 - 1
```

```
5433  IF X1%(IC) = 999 THEN  GOTO
5475
5438  IF X1%(IC) > 299 THEN X1%(I
C) = X1%(IC) - 300:Y1%(IC) =
Y1%(IC) - 300: HPLOT X1%(IC -
1),Y1%(IC - 1) TO X1%(IC),Y1
%(IC): GOTO 5455
5440 X% = X1%(IC):Y% = Y1%(IC)
5450  HPLOT X%,Y%
5455 X1%(IC) = 999:Y1%(IC) = 999
5460 IC = IC - 1: IF IC = 0 THEN
IC = 300
5470  NEXT
5475  HCOLOR= C
5480  GOTO 160
6000  HOME
6010  HTAB 11: PRINT "SCREEN COMM
ANDS"
6020  HTAB 11: PRINT "===========
===="
6030  HTAB 5: PRINT "SCREEN PLOT
COMMANDS:"
6040  HTAB 5: PRINT "1) U = PLOT
UP"
6050  HTAB 5: PRINT "2) R = PLOT
RIGHT"
6060  HTAB 5: PRINT "3) D = PLOT
DOWN"
6070  HTAB 5: PRINT "4) L = PLOT
LEFT"
6080  HTAB 5: PRINT "5) 1 = PLOT
ANGLE UP/LEFT"
6090  HTAB 5: PRINT "6) 2 = PLOT
ANGLE UP/RIGHT"
6100  HTAB 5: PRINT "7) 3 = PLOT
ANGLE DOWN/RIGHT"
6110  HTAB 5: PRINT "8) 4 = PLOT
ANGLE DOWN/LEFT"
6115  HTAB 5: PRINT "COLOR COMMAN
DS:"
6120  HTAB 5: PRINT "1) W = WHITE
  2) G = GREEN"
6140  HTAB 5: PRINT "3) V = VIOLE
T 4) B = BLACK"
6160  HTAB 5: PRINT "MISC COMMAND
S:"
6170  HTAB 5: PRINT "1) H = HELP(
LIST COMMANDS)"
6180  HTAB 5: PRINT "2) <- = (LEF
T ARROW) DELETE PREV-"
6190  HTAB 14: PRINT "IOUS PLOTS.
 REQUIRES A ": HTAB 14: PRINT
"NUMBER BETWEEN 1 -300"
```

```
6200  HTAB 14: PRINT "FOLLOWED BY
 A RETURN."
6210  HTAB 14: PRINT "(I.E. <- 17
 <RET> )": HTAB 14: PRINT "D
ELETES LAST 17 PLOTS."
6212  HTAB 5: PRINT "3) P = POSIT
ION(I.E.P 2,4<RET>)"
6215  TEXT
6220  VTAB 24: INPUT "<RETURN>";A
NS$
6230  HOME
6240  PRINT "LINE AND BLOCK COMMA
NDS:"
6250  HTAB 5: PRINT "1) M = MAKE
A LINE. MUST BE"
6260  HTAB 8: PRINT "FOLLOWED BY
THE END OF LINE X,Y"
6270  HTAB 8: PRINT "COORDINATES.
 I.E. M187,122<RET>"
6280  HTAB 5: PRINT "2) C = COLOR
 AN AREA. MUST BE FOL-"
6290  HTAB 8: PRINT "LOWED BY A N
O. OF LINE REPEATS"
6300  HTAB 8: PRINT "AND A RETURN
. THEN SPECIFY THE"
6310  HTAB 8: PRINT "ENDING X AND
 Y COORDINATES AND"
6320  HTAB 8: PRINT "RETURN. I.E.
 C12<RET>140,50<RET>"
6330  HTAB 8: PRINT "IF THE LAST
POINT WAS AT"
6340  HTAB 8: PRINT "LOCATION X=8
0 AND Y=50, THE"
6350  HTAB 8: PRINT "ABOVE EXAMPL
E WOULD PLOT A"
6360  HTAB 8: PRINT "RECTANGLE FR
OM X 80 TO 140"
6370  HTAB 8: PRINT "AND Y50 TO 6
2."
6371  HTAB 5: PRINT "3) # = CREAT
E A RECTANGLE. USE"
6372  HTAB 8: PRINT "POSITION COM
MAND TO SPECIFY"
6373  HTAB 8: PRINT "UPPER LEFT A
ND LOWER RIGHT"
6374  HTAB 8: PRINT "COORDINATES.
 THEN '#' WILL DO"
6375  HTAB 8: PRINT "THE REST. I.
E. P10,20<RET>"
6376  HTAB 8: PRINT "P30,40<RET>#
 WILL DO A SQUARE."
6377  VTAB 24: INPUT "<RETURN>";A
NS$: HOME
```

```
6380  PRINT : PRINT "SHORTCUTS:(M
 AND C ONLY):"
6390  HTAB 5: PRINT "WHEN USING E
ITHER OF THESE"
6400  HTAB 5: PRINT "COMMANDS, TO
 DUPLICATE THE CURRENT"
6410  HTAB 5: PRINT "X OR Y COORD
INATE, ENTER A -1"
6420  HTAB 5: PRINT "INSTEAD OF T
HE ACTUAL LOCATION."
6430  HTAB 5: PRINT "I.E. M140,-1
<RET> WOULD DRAW A"
6440  HTAB 5: PRINT "HORIZONTAL L
INE. M-1,160 WOULD"
6450  HTAB 5: PRINT "DRAW A VERTI
CAL LINE."
6455  HTAB 5
6460  PRINT : PRINT "WHEN USING T
HESE COMMANDS YOU MAY"
6470  HTAB 5: PRINT "LOSE YOUR PL
ACE AND NOT BE SURE"
6480  HTAB 5: PRINT "WHAT RESPONS
E THE COMPUTER IS "
6490  HTAB 5: PRINT "WAITING FOR.
 IF YOU ENTER <RET>"
6500  HTAB 5: PRINT "-999,-999<RE
T> THE CURRENT COMMAND"
6510  HTAB 5: PRINT "WILL BE CANC
ELLED."
6900  VTAB 24: INPUT "<RETURN>";A
NS$
6990  GOTO 4900
10000  REM
10010  HOME : HTAB 11
10020  PRINT "LIST OF COMMANDS"
10030  HTAB 11
10040  PRINT "================"
10045  HTAB 11
10050  VTAB 4: PRINT "MASTER COMM
ANDS"
10055  PRINT
10057  HTAB 5
10060  PRINT "1)ESC L-LOAD OLD SH
APE"
10070  HTAB 5
10080  PRINT "2)ESC S-SAVE CURREN
T SHAPE"
10082  HTAB 5
10084  PRINT "3)ESC C-CLEAR CURRE
NT SCREEN"
10090  HTAB 5
10094  PRINT "4)ESC E-END PROGRAM
"
```

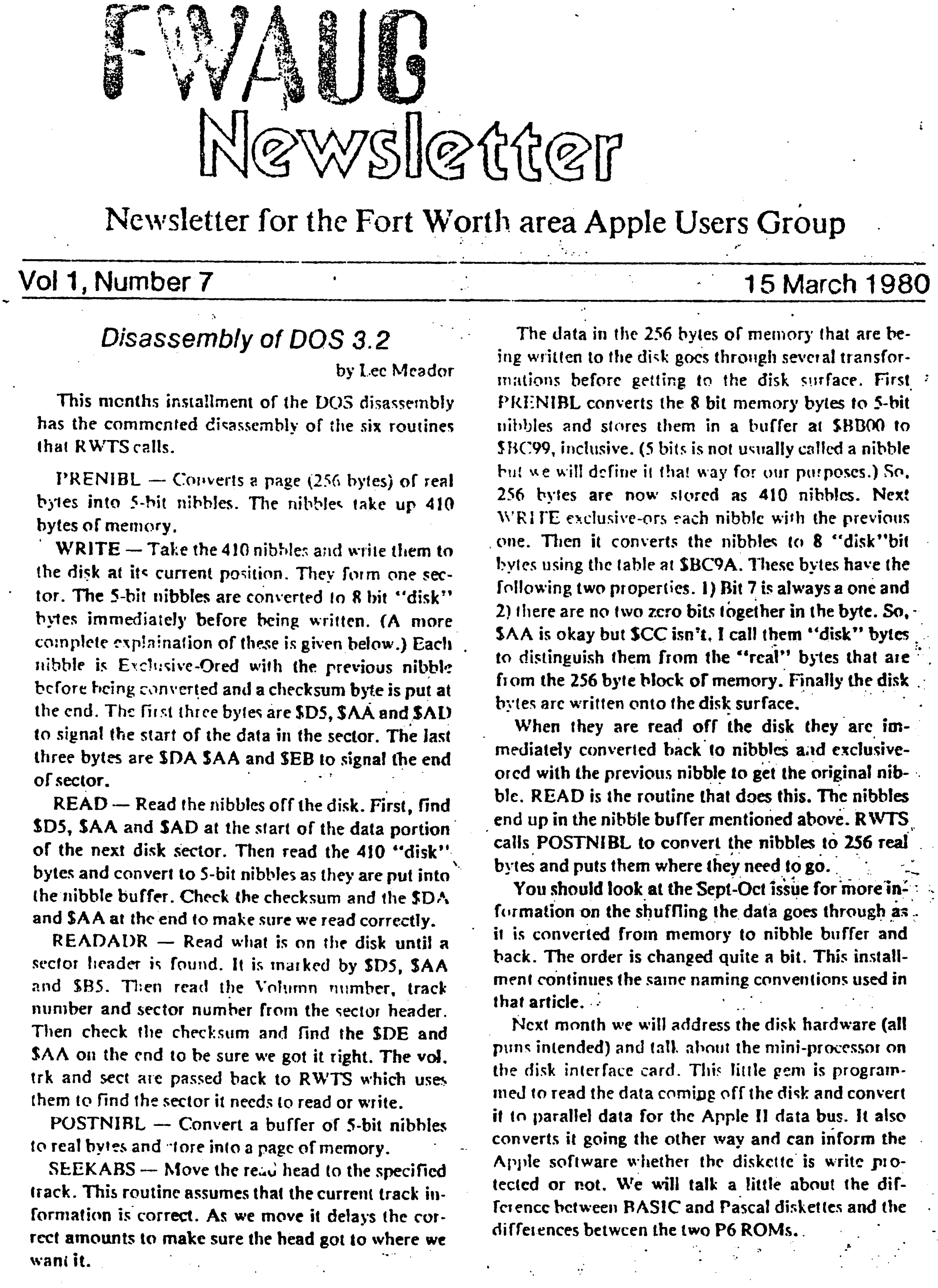
```
10097  HTAB 5
10100  PRINT "5)ESC T-TEXT MODE"
10110  HTAB 5
10120  PRINT "6)ESC H-HIRES MODE"

10121  HTAB 5: PRINT "7)ESC G-HIR
ES GUIDE GRID (ON/OFF)"
10122  HTAB 11: PRINT "(GRID IS E
ACH 20 PLOT POS'NS)"
10123  VTAB 23: PRINT "CURRENT PL
OT  POSITION  X=";X%;"  Y=";
Y%
10130  RETURN
20000  HPLOT X%,Y%
20003 IC = IC + 1: IF IC > 300 THEN
IC = 1
20005 X1%(IC) = X%:Y1%(IC) = Y%
20010  RETURN
24000  IF X1%(IC) =  - 999 THEN  GOTO
160
24010  IF IC = 1 AND X1%(300) =  -
999 THEN  GOTO 160
24020  IF IC = 1 THEN 24031
24023  IF X1%(IC - 1) =  - 999 THEN
160
24031 H%(1) = X1%(IC - 1):H%(2) =
Y1%(IC - 1):H%(3) = X1%(IC):
H%(4) = Y1%(IC - 1):H%(5) =
X1%(IC):H%(6) = Y1%(IC)
24033 H%(7) = X1%(IC - 1):H%(8) =
Y1%(IC):H%(9) = X1%(IC - 1):
H%(10) = Y1%(IC - 1)
24035  FOR Z = 2 TO 8 STEP 2
24036 X% = H%(Z - 1):Y% = H%(Z): GOSUB
20000
24037 RE = 1:X = H%(Z + 1):Y = H%
(Z + 2): GOSUB 25030
24038  NEXT
24090  GOTO 160
25000  REM  PLOT A LINE
25010  INPUT X,Y
25011  IF X =  - 999 OR Y =  - 99
9 THEN 160
25030  IF X > 279 THEN X = 279
25040  IF Y > 191 THEN Y = 191
25045 X% = X1%(IC):Y% = Y1%(IC): IF
X% > 299 THEN X% = X% - 300
25046  IF Y% > 299 THEN Y% = Y% -
300
25047  GOSUB 20003
25048  IF X >  - 1 THEN X% = X
25049  IF Y >  - 1 THEN Y% = Y
25060  HPLOT  TO X%,Y%
25070 X% = X% + 300:Y% = Y% + 300
```

```
25080  GOSUB 20003
25085 X% = X% - 300:Y% = Y% - 300

25088  IF RE > 0 THEN  RETURN
25090  GOTO 160
26000  REM  COLOR AN AREA
26010  INPUT RE
26011  IF RE =  - 999 THEN 160
26012 OX% = X%:OY% = Y%
26020  GOSUB 25000
26030 RE = RE - 1: IF RE =  < 1 THEN
 GOTO 160
26040 OY% = OY% + 1:Y% = OY%: IF
Y% > 191 THEN Y% = 191
26044 X% = OX%
26049 Y = OY%
26050  GOSUB 20000: GOSUB 25030: GOTO
26030
30000  REM
30010  INPUT X,Y
30011  IF X =  - 999 OR Y =  - 99
9 THEN 160
30020  IF X > 279 THEN X = 279
30022  IF X < 0 THEN X = 0
30030  IF Y < 0 THEN Y = 0
30040  IF Y > 191 THEN Y = 191
30050 X% = X:Y% = Y
30060  GOSUB 20000: GOTO 160
59000  REM  SAVE FILE
59010  TEXT : HOME
59011  REM
59020  VTAB 5: HTAB 7
59030  PRINT "ENTER SAVE FILE NAM
E"
59040  HTAB 7: INPUT "==>";ANS$
59050  PRINT D$;"BSAVE ";ANS$;",A
$2000,L$2000"
59060 A$ = "T": GOTO 340
60000  REM   LOAD
60010  TEXT : HOME
60020  VTAB 5: HTAB 7
60030  PRINT "ENTER INPUT FILE NA
ME"
60040  HTAB 7: INPUT "==>";ANS$
60050  PRINT D$;"BLOAD ";ANS$;",A
$2000"
60060 A$ = "T": GOTO 340
61000  COLOR= 7: HPLOT 19,0 TO 19
,189: HPLOT 39,0 TO 39,189: HPLOT
59,0 TO 59,189: HPLOT 79,0 TO
79,189
```

```
61010  HPLOT 99,0 TO 99,189: HPLOT
119,0 TO 119,189: HPLOT 139,
0 TO 139,189: HPLOT 159,0 TO
159,189: HPLOT 179,0 TO 179,
189: HPLOT 199,0 TO 199,189
61020  HPLOT 219,0 TO 219,189: HPLOT
239,0 TO 239,189: HPLOT 259,
0 TO 259,189: HPLOT 0,19 TO
279,19: HPLOT 0,39 TO 279,39
: HPLOT 0,59 TO 279,59
61030  HPLOT 0,79 TO 279,79: HPLOT
0,99 TO 279,99: HPLOT 0,119 TO
279,119: HPLOT 0,139 TO 279,
139: HPLOT 0,159 TO 279,159:
 HPLOT 0,179 TO 279,179
61040  RETURN
62000  VTAB 4: HTAB 5: INVERSE : PRINT
"

   ": HTAB 5: PRINT " ";: HTAB
34: PRINT " "
62010  HTAB 5: PRINT " ";: HTAB 3
4: PRINT " "
62020  HTAB 5: PRINT " ";: HTAB 3
4: PRINT " ": HTAB 5: PRINT
" ";: HTAB 34: PRINT " ": HTAB
5: PRINT " ";: HTAB 34: PRINT
" ": HTAB 5: PRINT "
                              "
62040  NORMAL : VTAB 6: HTAB 14: PRINT
"HIRES SCREEN";: HTAB 13: VTAB
7: PRINT "CREATE PROGRAM";: VTAB
8
62050  HTAB 10: PRINT "(C) BY BRU
CE BARBER";: VTAB 12: HTAB 7
: PRINT "NONCOMMERCIAL DISTR
IBUTION": HTAB 13: PRINT "IS
 ACCEPTABLE"
62060  VTAB 15: PRINT "THIS PROGR
AM WILL CREATE HIGH RESOLU-"
: PRINT "TION GRAPHIC SCREEN
S FOR USE AS SIGNS": PRINT "
OR BACKGROUNDS FOR HIRES GAM
ES.  IN"
62070  PRINT "AFFECT THIS IS THE
POOR MANS GRAPHICS": PRINT "
PAD.  THE PROGRAM IS SELF DO
CUMENTING.": PRINT "AT ANY T
IME PRESS 'H' FOR HELP ON": PRINT
"COMMANDS. PROGRAM MUST BE R
ELOADED"
62071  PRINT "FOR EACH EXECUTION
SINCE SOME CODE IS": PRINT "
DESTROYED BY RUNNING IT."
62080  FOR X = 1 TO 300:X1%(X) =
999:Y1%(X) = 999: NEXT : VTAB
24: INPUT "<RETURN>";ANS$
62090  RETURN
```

<<< DOS 3.2 DISASSEMBLY >>>

We continue in this issue our fifth installment of Lee Meador's excellent series on the Disk Operating System, as originally published in the "Fort Worth Apple Users Group Newsletter." Lee is thinking of preparing a technical booklet on Apple DOS, with these studies as the core. Comments, errors noted and suggestions can be directed to him at 1401 Hillcrest Drive, Arlington, TX 76010.



FWAUG Newsletter

Newsletter for the Fort Worth area Apple Users Group

Vol 1, Number 7 15 March 1980

Disassembly of DOS 3.2

by Lee Meador

This months installment of the DOS disassembly has the commented disassembly of the six routines that RWTS calls.

PRENIBL — Converts a page (256 bytes) of real bytes into 5-bit nibbles. The nibbles take up 410 bytes of memory.

WRITE — Take the 410 nibbles and write them to the disk at its current position. They form one sector. The 5-bit nibbles are converted to 8 bit "disk" bytes immediately before being written. (A more complete explaination of these is given below.) Each nibble is Exclusive-Ored with the previous nibble before being converted and a checksum byte is put at the end. The first three bytes are $D5, $AA and $AD to signal the start of the data in the sector. The last three bytes are $DA $AA and $EB to signal the end of sector.

READ — Read the nibbles off the disk. First, find $D5, $AA and $AD at the start of the data portion of the next disk sector. Then read the 410 "disk" bytes and convert to 5-bit nibbles as they are put into the nibble buffer. Check the checksum and the $DA and $AA at the end to make sure we read correctly.

READADR — Read what is on the disk until a sector header is found. It is marked by $D5, $AA and $B5. Then read the Volumn number, track number and sector number from the sector header. Then check the checksum and find the $DE and $AA on the end to be sure we got it right. The vol, trk and sect are passed back to RWTS which uses them to find the sector it needs to read or write.

POSTNIBL — Convert a buffer of 5-bit nibbles to real bytes and store into a page of memory.

SEEKABS — Move the read head to the specified track. This routine assumes that the current track information is correct. As we move it delays the correct amounts to make sure the head got to where we want it.

The data in the 256 bytes of memory that are being written to the disk goes through several transformations before getting to the disk surface. First PRENIBL converts the 8 bit memory bytes to 5-bit nibbles and stores them in a buffer at $BB00 to $BC99, inclusive. (5 bits is not usually called a nibble but we will define it that way for our purposes.) So, 256 bytes are now stored as 410 nibbles. Next WRITE exclusive-ors each nibble with the previous one. Then it converts the nibbles to 8 "disk"bit bytes using the table at $BC9A. These bytes have the following two properties. 1) Bit 7 is always a one and 2) there are no two zero bits together in the byte. So, $AA is okay but $CC isn't. I call them "disk" bytes to distinguish them from the "real" bytes that are from the 256 byte block of memory. Finally the disk bytes are written onto the disk surface.

When they are read off the disk they are immediately converted back to nibbles and exclusive-ored with the previous nibble to get the original nibble. READ is the routine that does this. The nibbles end up in the nibble buffer mentioned above. RWTS calls POSTNIBL to convert the nibbles to 256 real bytes and puts them where they need to go.

You should look at the Sept-Oct issue for more information on the shuffling the data goes through as it is converted from memory to nibble buffer and back. The order is changed quite a bit. This installment continues the same naming conventions used in that article.

Next month we will address the disk hardware (all puns intended) and talk about the mini-processor on the disk interface card. This little gem is programmed to read the data coming off the disk and convert it to parallel data for the Apple II data bus. It also converts it going the other way and can inform the Apple software whether the diskette is write protected or not. We will talk a little about the difference between BASIC and Pascal diskettes and the differences between the two P6 ROMs.

```
#B800L
002C B9A1 2C, 2D, 2E and 2F hold Vol, Trk, Sect and Chksum in RDADR
0478 BA20 $478 holds current track for SEEKABS
0478 BA2B
0478 BA39
0478 BA40
0478 BA50
0678 B875 $678 holds slot # of disk ($s0 format)
0678 B8C5   Used to take up one more cycle than $27 the page 0 value

B800 PRENIBL - CONVERT A SECTOR OF REAL BYTES TO RIGHT JUSTIFIED
                5 BIT NIBBLES ($19A - 5 BIT GROUPS, OR 410 DECIMAL).

B800-    A2 32        LDX   #$32        $33 bytes per section
B902-    A0 00        LDY   #$00        offset in real bytes (input)

B804 B858
B804-    B1 3E        LDA   ($3E),Y     form part 1, section 0
B806-    85 26        STA   $26          (part 2 is in $26)
B808-    4A           LSR
B809-    4A           LSR
B80A-    4A           LSR
B80B-    9D 00 BB     STA   $BB00,X     part 1, sec 0 is $BB00.BB32
B80E-    C8           INY               next real byte

B80F-    B1 3E        LDA   ($3E),Y     form part 1, section 1
B811-    85 27        STA   $27          (part 2 is in $27)
B813-    4A           LSR
B814-    4A           LSR
B815-    4A           LSR
B816-    9D 33 BB     STA   $BB33,X     part 1, sec 1 is $BB33.BB65
B819-    C8           INY               next real byte

B81A-    B1 3E        LDA   ($3E),Y     form part 1, section 2
B81C-    85 2A        STA   $2A          (part 2 is in $2A)
B81E-    4A           LSR
B81F-    4A           LSR
B820-    4A           LSR
B821-    9D 66 BB     STA   $BB66,X     part 1, sec 2 is $BB66.BB98
B824-    C8           INY               next real byte

B825-    B1 3E        LDA   ($3E),Y     form part 1, section 3
B827-    4A           LSR                (part 2 is spread out)
B828-    26 2A        ROL   $2A         bit 0 goes in $2A
B82A-    4A           LSR
B82B-    26 27        ROL   $27         bit 1 goes in $27
B82D-    4A           LSR
B82E-    26 26        ROL   $26         bit 2 goes in $26
B830-    9D 99 BB     STA   $BB99,X     part 1, sec 3 is in $BB99.BBCB
B833-    C8           INY               next real byte

B834-    B1 3E        LDA   ($3E),Y     form part 1, section 4
B836-    4A           LSR                (part 2 is spread out)
```

```
B837-    26 2A       ROL   $2A         bit 0 goes in $2A
B839-    4A          LSR
B83A-    26 27       ROL   $27         bit 1 goes in $27
B83C-    4A          LSR               bit 2 is in the carry
B83D-    9D CC BB    STA   $BBCC,X     part 1, sec 4 is in $BBCC.BBFE
B840-    A5 26       LDA   $26         add bit 2 to $26
B842-    2A          ROL
B843-    29 1F       AND   #$1F        keep only 5 bits
B845-    9D 00 BC    STA   $BC00,X     part 2, sec 0 is in $BC00.BC32
B848-    A5 27       LDA   $27
B84A-    29 1F       AND   #$1F        keep 5 bits here, too
B84C-    9D 33 BC    STA   $BC33,X     part 2, sec 1 is in $BC33.BC65
B84F-    A5 2A       LDA   $2A
B851-    29 1F       AND   #$1F        keep 5 bits again
B853-    9D 66 BC    STA   $BC66,X     part 2, sec 2 is in $BC66.BC98
B856-    C8          INY               next real byte
B857-    CA          DEX               back off 1 in each section
B858-    10 AA       BPL   $B804       if not to end of section - loop

B85A-    B1 3E       LDA   ($3E),Y     get "last byte"
B85C-    AA          TAX               save in X
B85D-    29 07       AND   #$07        keep 3 bits in part 2, sec 3
B85F-    8D 99 BC    STA   $BC99        (offset is 1)
B862-    8A          TXA               5 high bits are in "last byte"
B863-    4A          LSR
B864-    4A          LSR
B865-    4A          LSR
B866-    8D FF BB    STA   $BBFF
B869-    60          RTS               and we are done

B86A  WRITE - WRITE ALL THE NIBBLES ($19A OF THEM) ONTO THE DISK
              SURFACE. CONVERT EACH TO 8 BIT VALUE FIRST.

B86A-    38          SEC               set in case of error return
B86B-    BD 8D C0    LDA   $C08D,X     set Q6 high
B86E-    BD 8E C0    LDA   $C08E,X     and Q7 low to read write protect
B871-    30 7C       BMI   $B8EF        ... status (neg. means protected
B873-    86 27       STX   $27         X is the slot -- save in $27
B875-    8E 78 06    STX   $0678       and in Active Peripheral place
                                       ... used to take up cycles ($B8C5
B878-    AD 00 BC    LDA   $BC00       This is the first nibble of part
B87B-    85 26       STA   $26          ... save it for EOR-ing
B87D-    A9 FF       LDA   #$FF        Write an $FF on the disk (sync)
B87F-    9D 8F C0    STA   $C08F,X     set Q7 high (Q6 is already)
                                       ... to load ACC into Shift Regist
B882-    1D 8C C0    ORA   $C08C,X     set Q6 low to start writing on
                                       ... the disk surface. This reads
                                       ... $FF from the shift register,
                                       ... so the ACC is unchanged.
B885-    48          PHA               Waste some time to fall into loop
B886-    68          PLA               ... at the right time
B887-    EA          NOP               ... so Writes are 36 cycles apart
B888-    A0 0A       LDY   #$0A        Do this 10 times (that gives 11 $
B88A B890
```

```
B88A-   05 26       ORA  $26        Waste some time (no effect)
B88C-   20 F4 B8    JSR  $B8F4      Go write the byte in ACC ($FF)
                                    ... Writes are still 36 cycles apart
B88F-   88          DEY             One less to do
B890-   D0 F8       BNE  $B88A      ... and loop if any left
B892-   A9 D5       LDA  #$D5       Write a $D5 to signal start of data
                                    ... after 36 cycles
B894-   20 F3 B8    JSR  $B8F3      Same as $B8F4 (waits 2 cycles more)
B897-   A9 AA       LDA  #$AA       Write a $AA as second byte
B899-   20 F3 B8    JSR  $B8F3
B89C-   A9 AD       LDA  #$AD       Write an $AD as third byte
B89E-   20 F3 B8    JSR  $B8F3      ... $D5 $AA $AD are data header
                                    ... written 32 cycles apart

B8A1-   WRITE PART 2 BYTES $99 TO $00 IN THAT ORDER (EOR EACH BYTE
B8A1-   ... WITH THE NEXT HIGHER BYTE TO ALLOW ERROR CHECKING

B8A1-   98          TYA             Set ACC to zero (1st EOR)
B8A2-   A0 9A       LDY  #$9A       We will write $9A nibbles (part 2)
B8A4-   D0 03       BNE  $B8A9      Always taken - skip into loop
 B8A6 B8B9
B8A6-   B9 00 BC    LDA  $BC00,Y    ACC gets previous nibble
 B8A9 B8A4
B8A9-   59 FF BB    EOR  $BBFF,Y    EOR with current nibble
B8AC-   AA          TAX             Use this as offset into table
B8AD-   BD 9A BC    LDA  $BC9A,X    ... of disk bytes. The 5-bit nibble
                                    ... maps into an 8-bit byte that
                                    ... is suitable for writing.
B8B0-   A6 27       LDX  $27        X gets the slot
B8B2-   9D 8D C0    STA  $C08D,X    Write the byte!
B8B5-   BD 8C C0    LDA  $C08C,X    ... 32 cycles later (1st byte 33)
                                    ... (Disk IF writes 1 bit/4 cycles)
B8B8-   88          DEY             One less byte to do
B8B9-   D0 EB       BNE  $B8A6      Loop if any left

B8BB- WRITE PART 1, BYTES 0 TO $FF IN THAT ORDER

B8BB-   A5 26       LDA  $26        Get first nibble, part 2
B8BD-   EA          NOP             Wait 2 more cycles
B8BE B8D2
B8BE-   59 00 BB    EOR  $BB00,Y    EOR with current nibble
B8C1-   AA          TAX             Translate to disk surface byte
B8C2-   BD 9A BC    LDA  $BC9A,X    ... using X as offset
B8C5-   AE 78 06    LDX  $0678      Get the slot (use ABS addr to
                                    ... make it take 1 cycle longer)
B8C8-   9D 8D C0    STA  $C08D,X    Write the byte after 32 cycles
B8CB-   BD 8C C0    LDA  $C08C,X
B8CE-   B9 00 BB    LDA  $BB00,Y    Get current (soon previous) nibble
B8D1-   C8          INY             Do next byte
B8D2-   D0 EA       BNE  $B8BE      Loop if any left
B8D4-   AA          TAX             Change "last" nibble for writing
B8D5-   BD 9A BC    LDA  $BC9A,X    ... using X as offset
B8D8-   A6 27       LDX  $27        Get the slot
B8DA-   20 F6 B8    JSR  $B8F6      Write byte as checksum (Note:
```

```
                                                   ... the EOR of all the other
                                                   ... bytes gives this.)
B8DD-    A9 DE          LDA    #$DE                               ... 32 cycles later
B8DF-    20 F3 B8       JSR    $B8F3               Write $DE in data trailer
                B8E2-    A9 AA         LDA   #$AA  ... 32 cycles later    Write $AA
B8E4-    20 F3 B8       JSR    $B8F3
B8E7-    A9 EB          LDA    #$EB                and finally write $EB
                B8E9-    20 F3 B8      JSR   $B8F3          $DE $AA $EB is trai
B8EC-    BD 8E C0       LDA    $C08E,X             ... written 32 cycles apart
B8EF B871                                          Set Q7 low to end writing
                 B8EF-   BD 8C C0      LDA   $C08C,X     and Q6 low (thats
B8F2-    60             RTS                        end of Write routine

B8F3 B894 - ROUTINE TO WAIT A WHILE AND WRITE THE ACC TO DISK
B8F3 B899
B8F3 B89E
                B8F3 B8DF
B8F3 B8E4
B8F3 B8E9
                B8F3-    18            CLC                    wait 2 cycles
B8F4 B88C - ENTRY HERE DOESN'T WAIT AS LONG
                B8F4-    48            PHA                    wait 3 cycles
B8F5-    68             PLA                       wait 4 cycles

                 B8F6 B8DA - ENTRY HERE DOESN'T WAIT AT ALL
B8F6-    9D 8D C0       STA    $C08D,X            Write the ACC to the disk
B8F9-    1D 8C C0       ORA    $C08C,X            ... Q7,Q6 high then Q6 low
B8FC-    60             RTS                       return - delays 6 cycles too

B8FD- READ - READS THE SECTOR OFF THE DISK. FORMS $19A NIBBLES
             WHICH ARE LEFT JUSTIFIED

B8FD-    A0 20          LDY    #$20               We must find $D5 within $20 bytes
B8FF B909
B8FF-    88             DEY                       One less chance to find it
B900-    F0 61          BEQ    $B963              If no more chances, error return
B902 B905
B902-    BD 8C C0       LDA    $C08C,X            Keep Q6 low, read shift register.
B905-    10 FB          BPL    $B902              If positive, full byte not ready
                                                  ... since bit 7 is always a one.
                                                  ... Reads must be more than 12 an
                                                  ... less than 32 cycles apart.
B907 B913
B907 B91E
B907-    49 D5          EOR    #$D5               See if we got a $D5
B909-    D0 F4          BNE    $B8FF              If not, try again
B90B-    EA             NOP                       Wait 12 cycles before next try
B90C B90F
B90C-    BD 8C C0       LDA    $C08C,X            Read next byte
B90F-    10 FB          BPL    $B90C              ... and try until it is ready
B911-    C9 AA          CMP    #$AA               Is it an $AA
```

```
B913-   D0 F2       BNE   $B907       If not, try for a $D5 again
B915-   A0 9A       LDY   #$9A        We will read $9A bytes later
B917 B91A
B917-   BD 8C C0    LDA   $C08C,X     Read next byte
B91A-   10 FB       BPL   $B917       .. loop until ready
B91C-   C9 AD       CMP   #$AD        Is it an $AD
B91E-   D0 E7       BNE   $B907       If not, try for a $D5 again

B920- WE FOUND $D5 $AA $AD. THATS THE DATA HEADER. NOW READ PART
B920- ... 2 OFF DISK. NIBBLES $99 TO $0 IN THAT ORDER. (SEE $B915)

B920-   A9 00       LDA   #$00        We are ready-ing checksum
B922 B932
B922-   88          DEY               ready for current byte
B923-   84 26       STY   $26         Save offset (we use Y in between)
B925 B928
B925-   BC 8C C0    LDY   $C08C,X     Read the byte
B928-   10 FB       BPL   $B925       ... and loop until ready
B92A-   59 00 BA    EOR   $BAA8-$A8,Y Convert to left justified nibble
B92D-   A4 26       LDY   $26         Get offset into part 2
B92F-   99 00 BC    STA   $BC00,Y     Put nibble there
B932-   D0 EE       BNE   $B922       Loop if Y#0

B934- NOW READ PART 1, BYTES 0 TO $FF IN THAT ORDER

B934 B944
B934-   84 26       STY   $26         Set offset to 0
B936 B939
B936-   BC 8C C0    LDY   $C08C,X     Read the byte
B939-   10 FB       BPL   $B936       ... and loop until its ready
B93B-   59 00 BA    EOR   $BAA8-$A8,Y Convert to nibble
B93E-   A4 26       LDY   $26         Get offset back into Y
B940-   99 00 BB    STA   $BB00,Y     ... and store byte there
B943-   C8          INY               Next byte from disk
B944-   D0 EE       BNE   $B934       If any left, loop to read

B946- READ CHECKSUM BYTE TO SEE IF EVERYTHING SO FAR IS CORRECT

B946 B949
B946-   BC 8C C0    LDY   $C08C,X     Read the byte
B949-   10 FB       BPL   $B946       ... and loop until ready
B94B-   D9 00 BA    CMP   $BAA8-$A8,Y See if its the same as the last byte
B94E-   D0 13       BNE   $B963       If different, error return
B950 B953
B950-   BD 8C C0    LDA   $C08C,X     Read next byte
B953-   10 FB       BPL   $B950       ... yes, we still loop
B955-   C9 DE       CMP   #$DE        If it is $DE then we are at the
B957-   D0 0A       BNE   $B963       ... end,  If not, error return
B959-   EA          NOP               Wait 2 cycles
B95A B95D
B95A-   BD 8C C0    LDA   $C08C,X     Read next byte
B95D-   10 FB       BPL   $B95A       ... loop til its ready
B95F-   C9 AA       CMP   #$AA        If it is $AA (trailer is $DE AA EB)
B961-   F0 5C       BEQ   $B90F       ... then do successful return
```

```
B963 B900 - THIS IS THE ERROR RETURN PLACE. CARRY SET MEANS ERROR.
B963 B94E
B963 B957
B963 B96E
B963 B9AA
B963 B9B3
B963 B9BD
B963-    38          SEC                  Set it and leave
B964-    60          RTS

B965- READADR - READS ADDRESSES ON THE SECTORS OF CURRENT TRACK
                UNTIL IT FINDS A SECTOR. THEN IT RETURNS.
                $2C,$2D,$2E AND $2F HOLD CHECKSUM, SECTOR, TRACK AND
                VOLUMN, RESPECTIVELY. CARRY IS SET ON ERROR.

B965-    A0 F8       LDY    #$F8          Only $708 bytes will be read
                                          ... from $F8F8 to $10000
B967-    84 26       STY    $26           before error returning
B969 B977
B969-    C8          INY                  Count one try (low byte)
B96A-    D0 04       BNE    $B970         (this is for 16 bit increment)
B96C-    E6 26       INC    $26           Count one try (high byte)
B96E-    F0 F3       BEQ    $B963         If to zero, error return
B970 B96A
B970 B973
B970-    BD 8C C0    LDA    $C08C,X       Read a byte
B973-    10 FB       BPL    $B970         ... loop til it is formed
B975 B981
B975 B98C
B975-    C9 D5       CMP    #$D5          Is it a $D5 (Address header)
B977-    D0 F0       BNE    $B969         No? Count this as a miss
B979-    EA          NOP                  Wait 2 extra cycles
B97A B97D
B97A-    BD 8C C0    LDA    $C08C,X       Read next byte
B97D-    10 FB       BPL    $B97A         ... when its ready
B97F-    C9 AA       CMP    #$AA          Is it $AA
B981-    D0 F2       BNE    $B975         If not try for $D5
B983-    A0 03       LDY    #$03          We will read 0-3 later
B985 B988
B985-    BD 8C C0    LDA    $C08C,X       Read third byte
B988-    10 FB       BPL    $B985         ... at its leisure
B98A-    C9 B5       CMP    #$B5          Is it a $B5
B98C-    D0 E7       BNE    $B975         If not, see if its a $D5

B98E- WE FOUND ADDRESS HEADER ($D5 AA B5) NOT READ ADDRESS

B98E-    A9 00       LDA    #$00          We use this to form checksum
B990 B9A7
B990-    85 27       STA    $27           Keep the checksum in $27
B992 B995
B992-    BD 8C C0    LDA    $C08C,X       Read a byte (This is done 4 times
B995-    10 FB       BPL    $B992         ... and wait til its done
B997-    2A          ROL                  But this is just half of it
```

```
B998-    85 26       STA   $26        Save this half
B99A B99D
B99A-    BD 8C C0    LDA   $C08C,X    Read another byte
B99D-    10 FB       BPL   $B99A      ... keep trying!
B99F-    25 26       AND   $26        Put the halves together
B9A1-    99 2C 00    STA   $002C,Y    Store it away for the caller
B9A4-    45 27       EOR   $27        EOR to form checksum
B9A6-    88          DEY              One less to do
B9A7-    10 E7       BPL   $B990      do 3-0 then no more loop
B9A9-    A8          TAY              See if checksum EOR other stuff
B9AA-    D0 B7       BNE   $B963      ... is zero, If not, error return
B9AC B9AF
B9AC-    BD 8C C0    LDA   $C08C,X    Read next byte
B9AF-    10 FB       BPL   $B9AC      ... and so forth
B9B1-    C9 DE       CMP   #$DE       See if it is $DE
B9B3-    D0 AE       BNE   $B963      If not, error return
B9B5-    EA          NOP              Wait 2 extra cycles
B9B6 B9B9
B9B6-    BD 8C C0    LDA   $C08C,X    Read another byte
B9B9-    10 FB       BPL   $B9B6      ... you guessed it!
B9BB-    C9 AA       CMP   #$AA       See if it is $AA
B9BD-    D0 A4       BNE   $B963      If not, erro return
B9BF B961
B9BF-    18          CLC              Carry is clear for this, a
B9C0-    60          RTS              ... normal return

B9C1- POSTNIBL - CONVERT THESE LEFT JUSTIFIED NIBBLES ($19A-5 BIT GROUPS)
                 TO REAL BYTES ($100). $3E.3F POINTS TO BUFFER TO PUT THEM.

B9C1-    A2 32       LDX   #$32       X is number of bytes / section
                                      ... Start with last nibble in section
B9C3-    A0 00       LDY   #$00       Y is offset into out buffer
B9C5 BA10
B9C5-    BD 00 BC    LDA   $BC00,X    Do part 2, section 0
B9C8-    4A          LSR              ignore the three low
B9C9-    4A          LSR              ... order bits
B9CA-    4A          LSR
B9CB-    85 27       STA   $27        Keep rightmost bit in $27
B9CD-    4A          LSR              ... and dump it too
B9CE-    85 26       STA   $26        Keep new rightmost bit in $26
B9D0-    4A          LSR              ... and get rid of it
B9D1-    1D 00 BB    ORA   $BB00,X    Add part 2 to part 1, section 0
B9D4-    91 3E       STA   ($3E),Y    And put "real" byte into buffer
B9D6-    C8          INY              Get ready for next byte
B9D7-    BD 33 BC    LDA   $BC33,X    Now do part2, section 1
B9DA-    4A          LSR              First, ignore low order bits
B9DB-    4A          LSR              ... two
B9DC-    4A          LSR              ... three
B9DD-    4A          LSR              Put new low order in with bit
B9DE-    26 27       ROL   $27        ... already in $27
B9E0-    4A          LSR              And the next bit in with the one
B9E1-    26 26       ROL   $26        ... already in $26
B9E3-    1D 33 BB    ORA   $BB33,X    Add part 2 to part 1, section 1
B9E6-    91 3E       STA   ($3E),Y    Put new "real" byte into buffer
```

```
B9E8-    C8          INY                  Ready for next byte
B9E9-    BD 66 BC    LDA    $BC66,X       Do part 2,section 3
B9EC-    4A          LSR                  Again ignore 3 bits
B9ED-    4A          LSR
B9EE-    4A          LSR
B9EF-    4A          LSR                  Put new low order in with bits
B9F0-    26 27       ROL    $27           ... already in $27
B9F2-    4A          LSR                  Same again for the two bits
B9F3-    26 26       ROL    $26           ... in $26
B9F5-    1D 66 BB    ORA    $BB66,X       Add part 2 to part 1, section 2
B9F8-    91 3E       STA    ($3E),Y       Store into next spot in buffer
B9FA-    C8          INY                  as before
B9FB-    A5 26       LDA    $26           Now use the 3 bits in $26
B9FD-    29 07       AND    #$07          ... to go with part 1,
B9FF-    1D 99 BB    ORA    $BB99,X       ... section 3
BA02-    91 3E       STA    ($3E),Y       Store into buffer
BA04-    C8          INY
BA05-    A5 27       LDA    $27           And lastly use 3 bits in $27
BA07-    29 07       AND    #$07          ... with part 1, section 4
BA09-    1D CC BB    ORA    $BBCC,X
BA0C-    91 3E       STA    ($3E),Y       Store into buffer
BA0E-    C8          INY
BA0F-    CA          DEX                  Back up one byte in each section
BA10-    10 B3       BPL    $B9C5         If any are left, then loop
BA12-    AD 99 BC    LDA    $BC99         Get "last" nibble, part 2
BA15-    4A          LSR                  Ignore low order 3 bits
BA16-    4A          LSR
BA17-    4A          LSR
BA18-    0D FF BB    ORA    $BBFF         Add in "last" one, part 1
BA1B-    91 3E       STA    ($3E),Y       And put in into the buffer
BA1D-    60          RTS                  Finally, we're finished

BA1E- SEEKABS - MOVE HEAD TO TRACK SPECIFIED BY ACC. $478 IS CURRENT.
BA1E-           RWTS DOES PHASE OFF FOR ALL FOUR BEFORE CALL

BA1E-    85 2A       STA    $2A           $2A gets desired track
BA20-    CD 78 04    CMP    $0478         Compare to current track
BA23-    F0 59       BEQ    $BA7E         If equal, we are through
BA25-    86 2B       STX    $2B           $2B gets the current slot number
BA27-    A9 00       LDA    #$00          Count loop iterations in $26
BA29-    85 26       STA    $26           ... used to calculate wait times
BA2B BA75
BA2B-    AD 78 04    LDA    $0478         Get the current track
BA2E-    85 27       STA    $27           Save it for later use
BA30-    38          SEC                  Subtract the desired track
BA31-    E5 2A       SBC    $2A
BA33-    F0 42       BEQ    $BA77         If we are there we can leave
BA35-    B0 07       BCS    $BA3E         CS -> current > desired
                                          (ie. Result is positive.)
BA37-    49 FF       EOR    #$FF          Acc<0. Set Acc= ABS(Acc)-1
BA39-    EE 78 04    INC    $0478         Set for next track
BA3C-    90 05       BCC    $BA43         Carry is always clear, just skip
BA3E BA35
BA3E-    69 FE       ADC    #$FE          Carry is set. So, Acc=acc-1.
```

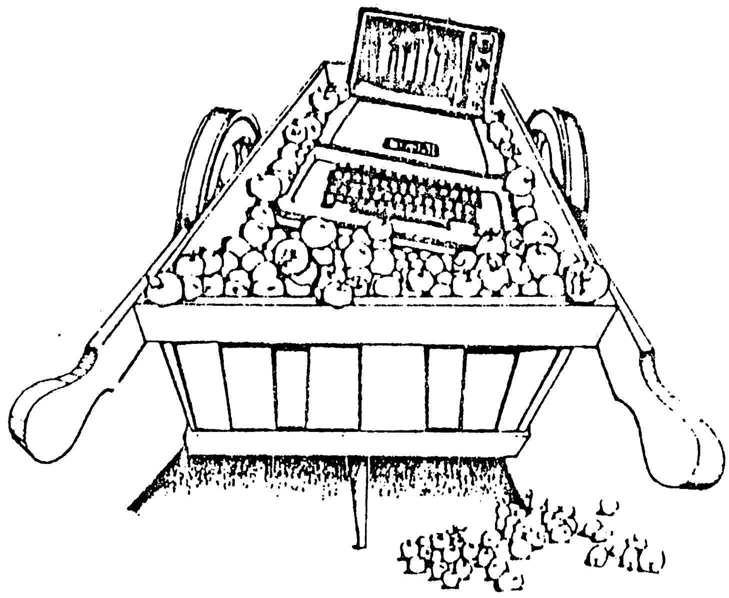
```
BA40-    CE 78 04    DEC   $0478       Set for next track.
BA43 BA3C
BA43-    C5 26       CMP   $26         Acc = min (Acc, ($26), #$0B)
BA45-    90 02       BCC   $BA49       ...
BA47-    A5 26       LDA   $26         ...
BA49 BA45
BA49-    C9 0C       CMP   #$0C        ...
BA4B-    90 02       BCC   $BA4F       ...
BA4D-    A9 0B       LDA   #$0B        ...

                                       Acc is now minimum of:
                                       ... A. # of tracks to move less 1
                                       ... B. # of iterations so far
                                       ... C. eleven (or $0B)

BA4F BA4B - TURN ON MOTOR WINDING TO STEP HEAD CORRECT DIRECTION

BA4F-    A8          TAY               Save Acc in Y for table offset
BA50-    AD 78 04    LDA   $0478       Get Next track number (xxxx xxxx)
BA53-    29 03       AND   #$03        Only keep 2 bits 0-3  (0000 00xx)
BA55-    0A          ASL               Shift left            (0000 0xx0)
BA56-    05 2B       ORA   $2B         Add in the slot number(0sss 0xx0)
BA58-    AA          TAX               That goes in X to reference right
BA59-    BD 81 C0    LDA   $C081,X     ... slot and PHASE-ON number xx
BA5C-    B9 90 BA    LDA   $BA90,Y     Get amount of time to wait
BA5F-    20 7F BA    JSR   $BA7F       Go wait that long
BA62-    A5 27       LDA   $27         Calculate PHASE-OFF by using

BA64 - TURN OFF LAST MOTOR WINDING TO ALLOW HEAD TO FINISH STEPPING

BA64-    29 03       AND   #$03        ... same formula as above.
BA66-    0A          ASL               ... Except use "current" track
BA67-    05 2B       ORA   $2B         ... as basis.
BA69-    AA          TAX
BA6A-    BD 80 C0    LDA   $C080,X     Phase-off
BA6D-    B9 9C BA    LDA   $BA9C,Y     Get correct amount of time
BA70-    20 7F BA    JSR   $BA7F       ... to wait and wait it out

BA73-    E6 26       INC   $26         Count iterations of loop
BA75-    D0 B4       BNE   $BA2B       Always taken

BA77 BA33 - WAIT SOME AND RETURN TO CALLER

BA77-    A9 FF       LDA   #$FF        Amount of time to wait (1/4 sec)
BA79-    20 7F BA    JSR   $BA7F       Long wait lets head settle
BA7C-    A6 2B       LDX   $2B         X gets the slot number back
BA7E BA23
BA7E-    60          RTS               And we are finished

BA7F BA5F - ROUTINE TO WAIT A LITTLE BIT. ACC HOLD THE LENGTH OF
BA7F BA70        THE WAIT. TIME IS IN ROUGHLY 100 MICRO SECOND UNITS
BA7F BA79
BA7F BA8D
BA7F-    A2 11       LDX   #$11        Do this little loop 17. times
```

```
BA81 BA82
BA81-    CA           DEX                Just count to waste time
BA82-    D0 FD        BNE    $BA81
BA84-    E6 46        INC    $46         Now count the total number of the
BA86-    D0 02        BNE    $BA8A       ... 100 microsecond units so we
BA88-    E6 47        INC    $47         ... know if disk is up to speed.
                                         ... (Called MONTIME in RWTS)
BA8A BA86
BA8A-    38           SEC                The Acc has the number of 100
BA8B-    E9 01        SBC    #$01        ... microsec. so one less to do
BA8D-    D0 F0        BNE    $BA7F       Loop if any left
BA8F-    60           RTS

BA90 - Table of Phase-on times to wait

BA90 BA5C
BA90- 01 30 28 24 20 1E 1D 1C
BA98- 1C 1C 1C 1C

BA9C - Table of Phase-off times to wait

BA9C BA6D
BA9C-             70 2C 26 22
BAA0- 1F 1E 1D 1C 1C 1C 1C 1C

BAA8- TABLE OF NIBBLES IN POSITION OF CORRESPONDING DISK BYTE
      (IE. AB->00, AD->08, AE->10, AC IS NOT VALID. IN FACT
      ANY BYTE WITH BITS 0,1 OR 2 SET IS NOT VALID) OFFSET
      FROM $BA00. (DISK BYTES --> NIBBLES)

BAA8 B92A
BAA8 B93B
BAA8 B94B
BAA8- 00 00 00 00 01 08 10 18
BAB0- 02 03 04 05 06 20 28 30
BAB8- 07 09 38 40 0A 48 50 58
BAC0- 0B 0C 0D 0E 0F 11 12 13
BAC8- 14 15 16 17 19 1A 1B 1C
BAD0- 1D 1E 21 22 23 24 60 68
BAD8- 25 26 70 78 27 80 88 90
BAE0- 29 2A 2B 2C 2D 2E 2F 31
BAE8- 32 33 98 A0 34 A8 B0 B8
BAF0- 35 36 37 39 3A C0 C8 D0
BAF8- 3B 3C D8 E0 3E E8 F0 F8

BB00 B80B - PART 1, SECTION 0 MEMORY BUFFER FOR NIBBLES
BB00 B8BE
BB00 B8CE
BB00 B940
BB00 B9D1
BB00-                   .DS   $33
BB33 B816 - PART 1, SECTION 1
BB33 B9E3
BB33-                   .DS   $33
```

```
BB66 BB21 - PART 1, SECTION 2
BB66 B9F5
BB66-                .DS   $33
BB99 BB30 - PART 1, SECTION 3
BB99 B9FF
BB99-                .DS   $33
BBCC B83D - PART 1, SECTION 4
BBCC BA09
BBCC-                .DS   $33
BBFF B866 - PART 1, "LAST" BYTE
BBFF B8A9
BBFF BA18
BBFF-                .DA #0  ONE BYTE
BC00 B845 - PART 2, SECTION 0 MEMORY BUFFER FOR NIBBLES
BC00 B87B
BC00 B8A6
BC00 B92F
BC00 B9C5
BC00                 .DS   $33
BC33 B84C - PART 2, SECTION 1
BC33 B9D7
BC33                 .DS   $33
BC66 B853 - PART 2, SECTION 2
BC66 B9E9
BC66                 .DS   $33
BC99 B85F - PART 2, "LAST" BYTE
BC99 BA12
BC99                 .DA #0  ONE BYTE

BC9A- TABLE OF BYTES FOR DISK SURFACE. USED TO CONVERT RIGHT JUSTIFIED
      NIBBLES (5 BITS IN FORM "000XXXXX") JUST BEFORE WRITING.
      (NIBBLES --> DISK BYTES)

BC9A B8AD
BC9A B8C2
BC9A B8D5
BC9A-       AB AD AE AF B5 B6 *  +-./56
BCA0- B7 BA BB BD BE BF D6 D7 *7:;=>?VW
BCA8- DA DB DD DE DF EA EB ED *Z.......
BCB0- EE EF F5 F6 F7 FA FB FD *........
BCB8- FE FF                   *..

BCBA- I DONT THINK THIS IS EVER USED. BUT HERE IT IS AS DATA AND CODE
      (WHERE IT MAKES CODE) FOR YOUR PERUSAL.

BCBA-       1C 1C 1C 00 00 00 *  ......
BCC0- A4 2D B9 D0 3C A0 05 4C *$-9P< .L
BCC8- 0A 3E 00 00 00 00 00 00 *.>......
BCD0- 00 05 0A 02 07 0C 04 09 *........
BCD8- 01 06 0B 03 08 00 00 00 *........
BCE0- 00 00 00 00 00 00 00 00 *........
BCE8- 00 00 00 00 00 00 00 00 *........
BCF0- 00 00 00 00 00 00 00 00 *........
BCF8- 00 00 00 00 00 00 00 00 *........
```

```
BCCO- THIS CODE MIGHT BE USED DURING MASTER BOOT OR RELOCATE

BCCO-    A4 2D       LDY   $2D
BCC2-    B9 DO 3C    LDA   $3CDO,Y    The byte loaded is a zero now
BCC5-    A0 05       LDY   #$05       ... its the same as $BCDO
BCC7-    4C 0A 3E    JMP   $3E0A      This is now $BE0A

C080 BA6A Phase On (beginning address of 4 spaced every other byte)
C081 BA59 Phase Off (similar to Phase On)

 Q6   Q7   Use of Q6 and Q7 lines in Disk Interface card
---- ----

 lo   lo - Read (disk -> shift register)
 lo   hi - Write (shift register -> disk)
 hi   lo - Sense write protect
 hi   hi - Load shift register from data bus.

C08C B882 Set Q6 low
C08C B885
C08C B8CB
C08C B8EF
C08C B8F9
C08C B902
C08C B90C
C08C B917
C08C B925
C08C B936
C08C B946
C08C B950
C08C B95A
C08C B970
C08C B97A
C08C B985
C08C B992
C08C B99A
C08C B9AC
C08C B9B6
C08D B868 Set Q6 high
C08D B8B2
C08D B8C8
C08D B8F6
C08E B86E Set Q7 low
C08E B8EC
C08F B87F Set Q7 high
```

FWAUG
Newsletter



<<< WANT AND DON'T WANT ADS >>>

EVER USED A COMPUCOLOR as an RS-232 terminal? Know who can repair one? Call Fred Gerlach, 981-4409, if you have or you do.

DI/AN PRINTER. Used and for sale in good condition with I/O device and software. Lewis Melton, 981-8866.

SELL HEATH H-14 DOT MATRIX PRINTER, tractor feed, 3 character sizes (80, 96, and 132 char/inch.), forms control, RS-232 or current loop. New cost from Heath is $900. Will sell for $600 firm. Call Mike Kramer, 358-6687 after 5:00 pm.

WANT TO BUY A D.C. HAYES MICROMODEM for the Apple. Call Pat McGee, 663-6806.

SELLING MY APPLE II+! 48K, disk, Integer Card, Atashi 19" B&W monitor, about 30 diskettes including the Muse Super.Text word processor. The works. $1550. Johnny Earl, 433-1339 after 6:30 pm.

SANYO MONITORS AVAILABLE IN GROUP PURCHASE. We need a minimum of 6 ordered if we are to get the special prices.

13" color	$430.	+ tax	(30-day delivery)
9" B & W	169.	"	(stock)
12" B & W	200.	"	(8-10 days delivery)
15" B & W	250.	"	(stock)

If you are interested contact Ray Essig, 493-9980 or 497-7165 (evenings).

BACK ISSUES OF APPLE BARREL are for sale in limited quantities! Many of you have inquired about their availability. The following back issues can be bought by mail for $1.00 each, postpaid:

vol. 2 no. 5	August, '79
vol. 2 no. 6	Sept/Oct, '79
vol. 3 no. 1	January, '80
vol. 3 no. 2	February, '80
vol. 3 no. 3	Mar/Apr, '80
vol. 4 no. 4	May, '80
vol. 5 no. 5	June/July, '80
vol. 6 no. 6	August, '80

This is a chance for newer members of HAAUG to catch up on programs, news, reviews, etc. Sorry, but there will be NO reprints when these are gone. Make checks payable to H.A.A.U.G. and send to Apple Barrel; Ed Seeger, Editor; 4331 Nenana Drive; Houston, TX; 77035. Please allow 10 days for delivery.

Houston Area Apple Users' Group
APPLE BARREL
Ed Seeger, Editor
4331 Nenana Drive
Houston, Texas 77035

(713) 723-6919

H.A.A.U.G

Postmasters:

Address correction requested:
Forwarding and Return Postage Guaranteed
