Issue #20/21 THE STAUNCH 8/89'er Sep-Dec 1990

THE ADDRESS BUS

PORT TO PORTAL -- Editorial

My apologies for the protracted delay in getting this "expanded" and final issue of the year to you. As I inferred on that postcard mailed well over a month back, the last half of the year has been rather distracting! Anyway, here I am and I have **some "heavy" stuff** to discuss.

First, it's **renewal time again! Check the issue number printed above your name and address on the manila envelope that this issue came in. That's the last** issue you've paid for. If it's identical with **this** issue number (a combined #'s 20 and 21), then you **must** mail me a check for the '91 calendar year. (You can make your check out either to me or to **The Staunch 8/89'er.**) Rates are the same as for the '90 calendar year: $12 in the U.S. and Canada, $16 overseas. I'll even take renewals for more than one year; as examples, some of you have re-upped through '93! I thank you for your confidence!

Also important is the present state of my "article" reserves. They're getting low! Besides the "regulars" (Hank Lotz, Peter Shkabara, Dan Jerome, and myself), I have some HDOS-oriented reprints from **REMark** and a few remaining articles that some of you have submitted. After those, however, I have **nothing** to run! I expect to run out of what I have about mid-year!

Believe it or not, **YOU** are my most important source of information about the H-8 and H/Z-89/90. So if you've been working on an interesting project recently, I'd like to hear from you. Of course, I'll take articles of **any** length. But if yours exceeds 1,000 words, I'll pay you **at least** $50 for it. That's better than **any** similarly-sized rag! If you feel uncomfortable with your writing style or spelling, I'll happily clean up your submission. So when you send your renewal check, you might also ask for an author's guide.

And speaking of articles, you'll see some interesting ones in '91. The first issue of the new year will have its customary hardware emphasis with Lee Hart's perusal of how to add automatic key repeat circuitry to the terminal logic board (TLB) of an H/Z-19 or '89/90. I've also noticed from your correspondence that **repair** is a area where some of you are having difficulties. With all due respect to Heath and its continuing repair service of our equipment, there are **still** some things you can do yourself **at less cost.** In the spring, I'll begin a series on hardware repair for the '19 and/or '89/90. There's a lot you can perform yourself using just some spare parts, replacement boards, and a minimal number of instruments. At minimum, when the series is complete, you'll be able to narrow the hardware problem to a specific board or area. I'd **also** like to see similar contributions for the H-8.

And although software listings have been a bit lean of late, I expect to have **major** contributions to announce next year. I have feelers out for some specific items and other packages have arrived recently which only await final preparation and packaging. So keep your eyes peeled for future software. Further, **past** software announcements have **finally** made it onto a pair of catalog disks. My thanks to Ralph Money, who volunteered his spare time to assemble the material from past issues. See the Software Listing overleaf!

Finally, I have to express my "undying" thanks to this rag's creator and first editor, Hank Lotz. When he heard I was pulling together a "double-issue," he made some quick enhancements to the custom utility he wrote for me to columnize and print **Staunch** from a MAGIC WAND/PEACHTEXT output file. And I used it to produce this issue (with, of course, the usual minor paste-up).

Kirk L. Thompson

THE EIGHT-BIT R/W -- Letters

Potpourri from Lee Hart (Continued). [From two letters begun in #19.] "HIDDEN DISK SPACE. Did you know Heath CP/M doesn't use all the sectors on a disk? Depending on the disk format, there is at least 1.25K of extra storage that is untouched by CP/M.

"Take the H17 hard-sector format for example. There are 40 tracks, each with ten 256-byte sectors. 256 x 10 = 2560 bytes = 2.5K per track. 40 x 2.5K is 100K total storage.

"So why can you only store 90K? First, CP/M reserves 3 boot tracks to hold the boot loader, BDOS, and CCP. These are hidden files installed by SYSGEN that don't appear in the directory. That leaves 37 tracks, or 37 x 2.5K = 92.5K. The CP/M directory takes 2K of this (64 entries of 32 bytes each), so 90K is actually available for your files.

"What happened to the extra .5K? It got ignored because CP/M can't handle blocks smaller than 1K! This means that the last two sectors on the last track (512K, or four CP/M 128-byte [logical] sectors) are available for special uses.

(Continued after the Software Listing on p. 3)

SOFTWARE LISTING

General Software Catalog

I'm **very** pleased to announce a catalog disk of **Staunch** software. This was initially prepared for me by reader Ralph Money of Largo, FL. Of course, it's available on either hard- or soft-sector, 5-1/4-inch. The catalog files are for **both** HDOS and CP/M and are squeezed to conserve disk space. An unsqueezer is provided to recover the information. This requires one (1) disk in any format. I expect to keep the catalog current as additions are made to **Staunch**'s library.

For CP/M Only

MUSIC EMULATOR
By Manfred Deffner

This is a music composition package written specifically for the H-8. It should also run on a suitably-equipped H/Z-89/90. It includes a music compiler; wave, Fourier, and spectrum editors; a Fourier analyzer; programs for wave revision, pitch generation on either the tempered or natural tone scales, a pitch and tuning activator; and a program to play your compositions. Tempo may be set within the play program. Included with the package are sample music files and a 15-minute audio cassette of compositions by J.S. Bach, John Williams, and Susato prepared with similar software by Manfred Deffner on a PC-clone. This package **requires** an analog-to-digital (A/D) board to reproduce the music through a speaker. It occupies 276K.

ANAPRO's Soft-Sector Formatting Utility
By Peter Shkabara

This is the formatting package Pete mentions in VENDOR.UPDATE. It's **only** for the H-37 controller and permits the following format combinations, either from the menu or command-line: 40-track (48-tpi) - SSDD, SSXD, DSDD, DSXD; 80-track (96-tpi) - DSXD; Z-100 CP/M. Be sure to read the documentation that comes with it before use for specifics on command-line use. It comes with NULU to recover the files from the .LBR file and occupies a mere 15K.

Family Ties (FT117)
By Computer Services, Provo, UT

This is the genealogy package Roger Dupuis briefly mentions in his letter in issue #16. Of the currently-available programs for CP/M, this is apparently the best of the lot. User documentation only comes when you register the package ($50.00) with the developer. What documentation (.INS) accompanies the distribution package is only enough for installation purposes (and I've had to clean it up some!). But registration gives you access to a genealogy BBS. An on-disk registration form, ready for printing, is included. This package occupies 130K.

MINIZAP and DENUM
By Hank Lotz

The first item is a disk **file** dump program, written in compiled BASIC. It is intended for the user who has no other such utility or doesn't want to make the time to learn the other more complicated programs available. Its limitations are that: it can only access **existing** files (by name), altering (patching) a file is done only one byte at a time, and to patch you must give row and column of the byte you wish to change. You may not move the cursor with the keypad as you can do with more sophisticated software. However, it does tell you the length (in 128-byte logical sectors) of a file and the location on disk of the sector being viewed or patched. A menu lets you move randomly or sequentially through the file. And you must **confirm** a patch before it's written to disk. Your input is error-trapped to prevent a premature abort of the program. BASIC source code is included.

The second item is a utility intended to strip unneeded line numbers from a Microsoft BASIC-80 source file when moving it to another BASIC (such as BASIC-E or CBASIC) that doesn't require line numbers except when needed for GOTOs and GOSUBS. Both interpreted **and** compiled versions are included. The BASIC source file must have been saved in ASCII. Besides the source, an input file (named TARGETS) is needed to supply those line numbers which are required by GOTOs and GOSUBs. You should use a cross-reference utility (such as that on HUG's #885-1231; CRG.BAS in **REMark**, June '85, p. 24; or B-XREF, first listed in **Staunch** #9--see further below) to determine these. The line numbers stripped may range from 0 through 65529. A help screen is embedded in the program. BASIC-80 source for both the interpreted and compiled version is included.

This entire package occupies 71K.

Updates to Prior Releases

HDOS 3.02 Now with Printed Manual

The printed manual for HDOS 3.02 is now available. Indeed, the first run is **already** sold out! I expect to make a second print run in mid- to late-January. The manual runs to 1,100 pages, comes in a 3-inch D-ring binder with chapter separators and preprinted index tabs, and includes system software and utilities. Because of the higher than expected print costs, I've had to raise the price for the package to **$75** (including U.P.S. shipping to the contiguous 48 states; HI, AK, and Canada please add $6 for air parcel; overseas please add $8 for surface mail or $30 for air parcel; write me about other shipping arrangements). Payment should be in U.S. funds. If you desire **both** printed and on-disk manuals (as described in **Staunch** #18), add $30. Be sure you indicate the media you require: standard hard-sector, soft-sector (the system is suppled on single-sided only, the on-disk manual may be on double-sided if your system supports it), or eight-inch. The system, as supplied, does not support Magnolia Microsystems' soft-sector controller, but Quikdata's DKFMDV4.DVD can replace the standard soft-sector driver if necessary; see issue #18, p. 12, and VENDOR.UPDATE in this issue. If you

need assistance setting up Quikdata's driver, let me know.

B-XREF for Both HDOS and CP/M

Originally listed in issue #9, both versions of this cross-reference utility by David Powers come as **stand-alone** (.ABS or .COM) programs. BASIC is not required to run them. I also still have a small supply of the hardcopy documentation for this package.

Placing an Order

With the exception of HDOS 3.02, your cost for this software depends on what you supply:

Formatted disk(s) and self-addressed, stamped return mailer ..... $2.00 per disk
Formatted disk(s) without mailer .... $4.00 per disk
No disk(s) or mailer ..... $6.00 per disk

Disk formats available are standard (SS/SD) or double-sided (DS/SD) hard-sector and single- or double-sided soft-sector for both HDOS and CP/M. **All** disks are 40-track (48 tpi) **only**. Please clearly indicate the format you are supplying or require. If you desire DS hard- or any soft-sector format, I will pack multiple items onto one disk. If you request it and there's space, I'll also pack the ANAFORM package on standard hard-sector. If your system **only** supports 80-track drives, I'll format **your disks** at 40-track for you before the software is transferred. I will **not subdivide a disk. Send mailorders to:**

Kirk L Thompson / **The Staunch 8/89'er** / P.O. Box 548 / West Branch, IA 52358

>----------<

THE EIGHT-BIT R/W (Continued from p. 1)

"The boot tracks have extra sectors, too. The BDOS is 3.5K, the CCP is 2K, and the boot loader is 1.25K. That totals to 6.75K of 7.5K available, leaving the last three 256-byte sectors on the third track unused.

"You can't access these extra sectors with BDOS calls because CP/M pretends they don't exist. But you **can** access them with BIOS calls (select disk, set track, set sector, read sector, write sector). You can prove to yourself they exist with DU, ZDUMP, or any other program that accesses the disk without using the BDOS. Note that DU calls the first sector 1; ZDUMP [and many other dump utilities] call the same sector 0.

"Double- and extended-density H37 disks have .875K free in the boot tracks; seven 128-byte [logical] sectors in track 1, sectors 21-27 for double-density, sectors 13-19 for extended-density.

"I don't know if anyone has put this space to use. One use would be for disk labels, an HDOS and MSDOS feature that CP/M lacks. Since almost 1K is available, there is enough room for elaborate sign-on screens like the Macintosh uses. Another would be to put some or all of the BIOS.SYS file [there] (which would have been in the boot tracks anyway if it had fit).

"Another hint. There is no reason not to use SYSGEN to make all your CP/M disks bootable. The boot tracks just go to waste if you don't, and this way you can put any disk in A:. BIOS.SYS is only necessary on the disk you cold boot from the H: prompt; warm boots and control-C don't need it. As long as you don't mix up different versions or sizes of CP/M, you can change the disk in A: freely (remember to type a control-C after changing any disk in CP/M)...

"H89 MEMORY EXPANSION. Did you know a stock H89 actually supports over 100K of banked memory? You can install **three** 16K RAM expansion boards simultaneously, for a total of 96K dynamic RAM, 2K static RAM, and 6K of ROM. The address decoder PROMs provide 8 different memory maps to allow the RAM to be switched in 8K banks. Too bad Heath never used this capability or even documented it. [Apple did a lot with RAM bank switching to keep its aging Apple II series alive. -Ed.]

Disk Packing Schemes. [Another extended discussion by Lee Hart from a letter following up my query in #19, p. 4.] "...Have you ever received a disk with dozens of files with bizzare names, and no hint of what they do, or how to use them?

"The CP/M User's group uses a method that's crude, but effective. They include a catalog number (a zero-byte file named something like -CATALOG.123), a READ.ME file that describes the files on the disk, and a checksum file CRCKLIST.CRC that has a checksum for each file so you can be sure you've gotten them error-free). If any files are packed, the unpacking program and instructions for its use are included...

Archived files are common in the PC-DOS world. But they use a scheme that makes it easy for the novice user. The disk has just one big .COM (or .EXE) file. When you run this program, it unpacks all the rest of the files.

"Here's a challenge for your readers: How can you do this in CP/M or HDOS? I'll donate a copy of **Write-Hand-Man** to the reader who submits the best solution. Judgement criteria:

"1. How GENERIC is it? Will it work on ANY H8 or H/Z89? Any number and type of disks (including single-drive systems)? Any amount of RAM (32K to 64K)? Any version of CP/M (2.202/3/4, Magnolia, CDR, Z-system) or HDOS 1/2/3)?

"2. How much USER INTERACTION is needed? Does he just type the program name, and watch everything happen automatically? Or must he configure the unpacking program, put up with excessive disk swapping, etc. How knowledgeable must the user be?

"3. How EFFICIENT is it? If the method is excessively slow or wasteful of disk space, nobody will use it.

"As an example to get you thinking, suppose a distribution disk has the following files:

READ.ME A text file to describe what is on the disk, and how to use it. It should be pure ASCII so it can be displayed, edited, or printed without difficulty (no TABs, characters with the msb=1, lines longer than 80 characters, etc.).

HDIR.CZM The directory listing programs and their
HDIR.DZC documentation, in squeezed form.
XDIR.CZM
XDIR.CZM
UNCR.COM A public-domain program to uncrunch files back to usable form.
H17.SUB A SUBMIT file to make 1st hard-sector copy.
H17-2.SUB A companion to above, for 2nd hard-sector copy.
H37.SUB same, for soft-sector copies.

"The READ.ME file contains the following:

Disk #123 - CP/M Disk Directory Programs

"This disk contains programs to list a directory of files on a disk. Unlike CP/M's built-in DIR command, these programs can list files alphabetically, show file sizes, space remaining on the disk, and other features.

"The programs have been crunched to save disk space (as indicated by the 'Z' in their file names). They must be uncrunched before use, and transferred to a working disk. This will require extra disks as indicated below. To do this:

1. Boot a CP/M disk that has the PIP, FORMAT, and SUBMIT programs.
2. Put the distribution disk in your B: drive.

3a. To uncrunch the files, and save them on H17 hard-sector disks, have two formatted H17 disks ready. Then type:

```
SUBMIT B:H17 B: C: (assuming B: is your source disk,
                    and C: is your destination disk).
```

3b. To uncrunch the files, and save them on H37 soft-sector disks, have one formatted H37 disk ready. Then type:

```
SUBMIT B:H37 B: C: (assuming B: is your source disk,
                    and C: is your destination disk).
```

"The .SUB files do all the work. The H37.SUB file is easy, because (I assumed) it all fits on one disk. The H17 version is more interesting, because I assumed the destination takes two disks. For example, H17.SUB contains:

```
; H17-SUB -- This SUBMIT file will automatically
; uncrunch the files from the designated distri-
; bution disk, save them on the first designated
; destination disk, then continue to the next.
;
; *** PUT 1ST H17 DISK IN DESTINATION DRIVE ***
; ***           AND HIT RETURN              ***
pip
$1:uncrunch $1:hdir.* $2
submit h17-2
```

"The line containing only "pip" is used to make the system pause while the user puts the first disk in the destination drive. Operation resumes when he hits the RETURN key, because it exits PIP and continues with the SUBMIT file.

"The 'uncrunch' command unpacks the first group of files; I'll assume here that they fill the first H17 disk. The last line ends the current SUBMIT file, and starts the next one, H17-2.SUB, which makes the second H17 disk:

```
; H17-2.SUB -- This SUBMIT file will automatically
; uncrunch the files from the designated distri-
; bution disk, and save them on the second desig-
; nated destination disk.
;
; *** PUT 2nd H17 DISK IN DESTINATION DRIVE ***
; ***           AND HIT RETURN              ***
pip
$1:uncrunch $1:xdir.* $2
```

"When this SUBMIT file ends, you will have two H17 disks with the unpacked versions of the distribution disk. It's rather slow, especially if you don't have 3 drives (it will use 'phantom' drives, which involve a lot of 'put disk C in drive A' disk swapping). Can you do better?" [Readers, let's hear about alternative approaches. Lee has laid down his gauntlet! -Ed.]

SamaritaWare. [From Stephen H. Kaiser, Cambridge, MA, concluded from #19] "Here are some thoughts on the issue of putting CP/M in the Public Domain. I *haven't seen so much legal boloney flying around* since the lawyers prevailed over a decade ago on software publishers to place those intimidating legal notices on software, saying in effect, You-Don't-Own-Anything-We-Don'tPromise-You-Anything-And-If-You-Think-Otherwise-Have-We-Got-A-Gulag-For-You. The real question is, how do we get the G@#ƆD&#@N lawyers to crawl back under a rock and never be heard from again? (At least on 8-bit software!)

"So let's review what we have:

1. Traditional up-front licenseware (typical commercial software)
2. Shareware (use first, pay later and get updates, manuals, help)
3. Freeware (pay nothing and get **no** updates, manuals or help)

"..... to which I would like to add category #4:

4. SamaritanWare ..... which means help without obligation, but you can say thanks if you want. With 8-bit programs, the original manufacturer (still the 'owner') could make a gift to the public domain, provided that there was no obligation or liability attached to any use of the software. Anyone satisfied or appreciative of this effort could send a contribution of any amount to the original manufacturer or a designated charity.

"Any disk or program so released to the public domain would be required to at all times carry a READ.ME text file describing the conditions of the release and the address for any contributions. There would be no obligation to provide updates, manuals or technical assistance.

"The disks could be released to the various 8-bit enthusiast groups and distributed from there. We would have a situation where there would be 3 types of disks in circulation. Original purchased software would still technically be owned by the company and licensed to the user. There would be

pirated copies and finally there would be user-owned SamaritanWare. The software manufacturers would be covered legally, would maintain good relations with user groups and would reap the benefits of contributions at minimal investment. The sounds like a win-win situation for everyone. (P.S. I know the Good Samaritan analogy is not perfect. The Biblical case involved an open-ended offer of help and no request for contributions, but .....)"

COBOL Tutorials and Resources. [In late July, I ordered the COBOL compiler/educational course from Lenny Geisler, editor of **SEBHC Journal**, that was listed in issue #18 (p. 8). After receiving it and working my way through part of the latter, I wrote to Lenny:] "Thanks for shipping the COBOL compiler and continuing-ed. package promptly. I've begun digging my way through the latter and found it somewhat disappointing. However, it appears to be the first home-study course on computer programming that Heath put together, dating as it does to 1977! COBOL has a lot of features not covered in the course and [the latter] also has a distinct mainframe flavor, probably to be expected given its age. But the course should still provide the fundamentals, although one topic definitely missing is how to key data into a program from the keyboard.

"Once I finish it, I plan to move on to H.W. Bauman's 17-part tutorial in **REMark** magazine (running between Oct., '83, and Apr., '86). This series has the advantage of dealing directly with the Microsoft compiler you provide and should supply a lot of 'hands-on' experience in the language and compiler/linker usage. Unfortunately, it too doesn't appear to cover data entry, although it does a lot with reading and writing data files (COBOL's strong points). So I plan to hunt for a suitable book. Once I locate one, I'll pass along the reference.

"Finally, when I set up the compiler and linker and used them for the first time last night, I stumbled across a quirk your readers should know about. The package includes a CRT driver for the '19/89 terminal (CDWH19.REL) and this should be copied to the disk containing the linker (L80) and runtime library (COBLIB.REL) as CRTDRV.REL. The driver will be automatically linked into programs. Further, one of the demos programs provided with the COBOL package is CRTEST, for checking out the terminal. The keys expected by the latter are **not** the 'conventional' ones (on the keypad) we've come to expect for on-screen editing! CRTEST simply beeps at you when you press the keypad keys, whether shifted or unshifted. The table of **expected** keys is near the beginning of the assembly language file, CDWH19.MAC, in the section titled 'Keyboard Code Definitions.' New users should print this table for their reference. Of course, the driver could be modified to bring up the keypad.

"Again, thanks for offering and shipping the COBOL package. I'm sure knowledge of this language will assist me at work. And I suspect COBOL programmers will continue to be in demand as long as IBM continues to manufacture mainframe computers!" [For new users of the COBOL compiler, I should add further items I've discovered about this compiler. First, be careful of the text editor you use when preparing source code. The compiler **gags** on embedded tab characters (ASCII 9) and garbage is sent to the screen or printer, whichever you've set to receive the line listing during compile. The fix is to use an editor that does **not** conserve disk space by including tabs in a file.

[MAGIC WAND's EDIT can be set to do this by keying "MBN" at the command screen prompt. You can also set tab positions to COBOL's (or your own) preferences.

[If you use Software Toolworks' PIE, you will have to change one byte in the program with a disk dump utility. Check the program's manual and distribution disk file, PATCHES.DOC, for details and patching location for the version you're running. PIE is probably the least convenient editor to use because you can't change tabbing from the system default of every eight. However, you could code against the left margin of the page, then use a macro to open up spaces where the compiler normally expects the optional line numbers to be.

[Another editor, Newline's TXTPRO, has a TABS function under the MODE (f2) key. Set the TABS to display as reverse-video I's so you can see them if you embed any. If you don't plan to use line numbering (not really required) in your source code, you can also set TXTPRO's left margin to column 7 permanently with the MODE and CONFIG keys. Further tabbing is at the system's default eight.

[In WORDSTAR, the non-document mode permits you to set tabs, provided you use CTRL-OV to turn on variable tabbing and CTRL-OI to set specific tabs where you want them. You might even set up a "ruler line" at the start of program development or code entry to automatically set all tabs with CTRL-OF. You **will** have to delete the ruler line before compile since COBOL does **not** recognize dot command comments. And whichever editor you use, you should use the **same** one to produce manually-keyed data files; compiled programs find tabs just an indigestible as the compiler!

[You might also try to locate an alphabetically-organized keyword reference book on the language. I have ones for BASIC and Pascal and they're much handier for quickly locating information on language specifics than thumbing through the language manual. For COBOL, I found Donald Sordillo's **The Programmer's ANSI COBOL Reference Manual** (Prentice-Hall, 1978, hardcover) at a used bookstore. It doesn't cover everything in COBOL-80 because Microsoft made some extensions to the 1974 ANSI language standard for programming on micros. But it includes information on those "standard" features that Microsoft **didn't** implement, so may be handy when converting mainframe-derived programs to COBOL-80.

[That book on data entry and screen handling I mentioned to Lenny I was hunting for is LeBert and Massoni's **Advanced Interactive COBOL for Micros: A Practical Approach** (Prentice-Hall, 1988). It isn't cheap at $30, but will provide information on interactive programming that neither Heath's continuing-ed. course nor the **REMark** magazine series supply. It **is** specific to Microsoft's COBOL for IBM-compatibles, but much of it is also applicable to Microsoft's earler COBOL-80; Microsoft seems to have rolled over many of the extensions originally developed for COBOL-80 to MS COBOL. Further, it's an intermediate-level book, so you should have some COBOL already under your belt

before digging into it. And almost all my remarks above apply to the somewhat later (now **very** hard to find) version of the compiler for HDOS if you should stumble across a copy of it.

[Finally, I've made contact with H.W. Bauman, author of **REMark**'s tutorial. Regrettably, he no longer has the on-disk material he prepared for that series. So anyone using it to learn COBOL will have to prepare their own input data files. -Ed.]

HDOS 3.02 Printed Manual, Etc. [Two letters by Al Bjorling, Circleville, NY] "Kirk, I think I can read between the lines ..... the HDOS 3.02 project has been a biggie for you, et al. Probably a lot more than was expected, too. If it is any consolation, my hat goes off to you all for this effort. The HEATH 8-bit community owes a debt of gratitude to you and the 'team'.

"In my request for printer assistance [in #19, p. 5], I had response from a number of great guys. Received much in the way of help that I needed and now I can proceed. It really is a pleasure to be associated with such a talented and well-meaning group of people...." [Al, putting together the HDOS 3.02 manual **was** a "biggie;" in size, it's about **three** times what I anticipated when I proposed the package in issue #13. A load of thanks are, indeed, due to system programmer Richard Musgrave, writer Dan Jerome, tech. advisors John Toscano and Bill Cordes, proofreader Terry Hall, and for the production assistance provided by Bill Lindley. Without their generous help, the package wouldn't have seen the light of day (to say nothing of the inside of a U.P.S. truck!). -Ed.]

"Received your card advising of some alleged missing pages in the HDOS 3.02 manual. Kirk, my Ch. 12 pages 1 - 49 are in manual. Decided to check the chapter index against the pages that I had in my manual and came up with a few deviations listed below for your reference--

Missing pages:
- 1-34 Appendix 1-C Port Assignment sheets
- 3-9 ? (Intentionally)
- 9-14 ? (Intentionally)
- 13-150 Chapter 13 Index sheets

Add to Table of Contents:
- Chapter 7 (after 7-56 two sheets)
  - Documentation
  - Background
- Chapter 11 Appendix 11-C, page 11-77
- Chapter 12 Index, page 12-103

Page # corrections to Table of Contents:
- Chapter 11 Index, page 11-79
- Chapter 12 Supplementary References, p. 12-101

"Kirk, I've been skimming through the manual and it is really quite a fine job of documentation, well organized, too. Very pleased with it. A side note, when first working with HDOS there is so much to learn and fully understand, it may be helpful for some of us to reference more examples of the uses for these new features/commands. Perhaps sell a booklet of such later?" [Thanks for the feedback on the manual, Al. Everyone who ordered the printed manual has already received the missing port assignment sheets for chpt. 1. The unnoted "MegaPIP" supplement to chpt. 7 was accidentally included; the same material is presented on pp. 7-7 thru 7-9. For the other flubbs, Dan Jerome is working up chapter indices where not currently present and corrections/additions where necessary. These will be mailed when available. This first print run (now sold out!) has forced to light a very few minor problems. See the "Software Listing" section for information on the second run of the monster! -Ed.]

Terminal Chips, Libraries, and Patches. [From Biff Bueffel, 19820 NW Metolius Drive, Portland, OR 97229] "This is a follow-up to my letter in ... [#19, p. 2] regarding replacing the TLB TTLs with CMOS ICs. Lee Hart's earlier article on cooling the H-89 [in #16] dealt mostly with the CPU and said nothing about the TLB. My machine is an H-89A, whereas I think Lee discussed the H-89 [in his letter in #19, p. 2-4]. There are differences in the numbering of the ICs (e.g., H-19 U421 is H-19A U430).

"H-89A users should not replace the CPU U564 with a 74C04, just as Lee pointed out [that] U501, also a 7404, cannot be replaced. The serial board does not work if it is replaced. Below is a list of H-89A TLB chips I have tried, with three that do not work also noted:

H-89A	old	new	notes
U406	74LS86	74HCT86	
U407	74LS245	74HCT245	
U413	74LS132	74HCT132	
U416	74LS157	74HC157	
U418	74LS74	74HCT74	
U419	74LS273	74HCT273	
U421	74LS166	74HC166	
U422	74S74	74F74	
U423	74LS08	74HCT08	
U424	74LS273	74HCT273	
U425	74S86	74F86	
U426	7404		Do NOT change!
U427	74LS161		74HCT161 gave no beep at 'power on' and sometimes no CRT filament
U428	74LS74	74HCT74	
U431	74LS132	74HCT132	
U432	74LS02	74HCT02	
7434	74LS00		74HCT00 gave no beep at 'power on'
U435	74LS138		Replaced by 74S138 in Flicker-Free mod
U442	74LS138	74HCT138	
U446	7404	74C04	
U447	74LS132	74HCT132	
U449	74LS244	74HCT244	
U450	74LS244	74HCT244	
U451	74LS74	74HCT74	

"The following ICs were not tested as I did not have the CMOS replacement in my inventory, but they should work:

U443	74LS244	74HCT244
U414-6	74LS157	74HC157

"I purchased all the replacement ICs from Jameco for around $25.

"My original letter contained two typos in section 7. The address in the 2.2.03 version of CONFIGUR should be 296E (not 296C). The address in the 2.2.04 version should be 2C67 (not 2C267). There is enough information in that paragraph so the correct sites should have been found by those who desire to try the patch.

"I think Lee Hart is wrong in his comments on the use of libraries in CP/M [packages] you distribute. Libraries save disk space and they are an easy way of grouping related files. Most users of plain CP/M do not use 'USER areas', an alternative he suggested. Most who use a modified CCP or BDOS, such as ZCPR3x and ZRDOS, have some understanding of LBRs. Perhaps you could offer to include NULU, UNSQ and UNCR on your distribution disks." [NULU and UNCRUNCH are already included where appropriate; UNSQ isn't because NULU has an internal command for unsqueezing. -Ed.]

"If any of your readers use WordStar version 4 (WS4) and have not been able to implement function keys and use [of] the keypad for cursor movement, I can supply a patch I wrote. Similarly, if they use ZCPR3 and have trouble using the '(R)un' a program option or using it in a 'multiple command line,' I can supply a patch."

PC89LINK. [From James H. Dummer, Libertyville, IL] "I have used the PC89LINK progrm from Lindley Systems to transfer my HDOS files to the MSDOS machine. It worked very well - very much better than the TF89 program from Heath that I used to transfer the CP/M files. Apparently Lindley Systems has since added the CP/M tranfer to their program. I wish I had had it sooner."

=====

A Professional Method for Program Testing:

Part 2 -- Background

By Kirk L. Thompson

In the first installment of this series on software testing (issue #18), I discussed module library construction, a "work bench" for testing your subroutines, and an assignment for you to think about and work through. Please bear in mind that each successive portion of this series **presumes** reading and doing the exercises given in the preceding one!

Just as you **cannot** acquire an adequate knowledge of a programming language from merely reading a book, so the **practice** of software testing is a skill you will acquire by reading my discussion and **working through** the exercises I give. This series will operate on the building-block principle; each installment is a course of bricks in a larger structure. You can't expect the structure to stand if you omit bricks in the foundation. So if you haven't done so before, go back and read Part 1. Then dig out your favorite high-level language (be it BASIC, FORTRAN, or Pascal) and explore the features of its integer function as I described on p. 11 of issue #18. If you're running some version of Pascal, also fiddle with ROUND and TRUNC. And as I wrote last time, think about "what kinds of tests you should use to ensure [these functions are] working properly." That question is the one I now address.

You Can't Test Everything! For the sake of argument, suppose I wish to test that INT function I asked you to look at in order to ensure it works for all one-decimal-place real numbers (that is, numbers with one digit to the right of the decimal point, such as 11.1 or 9900.5) that a typical home computer is capable of handling. Usually real numbers (that is, numbers with decimal points) on our machines range (in scientific notation) from +1E+38 through -1E+38. That is, these numbers extend from a maximum of 1 followed by 38 zeros to a minimum of **negative** 1 followed by 38 zeros, no matter what the language.

So if I were to use a brute-force method of testing I could set up a simple sample routine in Pascal similar to:

```
Y := 0;
FOR X := 1000 DOWNTO -1000 DO BEGIN
  Y := Y + 1;
  WRITE ((X/10):5:1, '-->', INT (X/10):5:1, ' ');
  IF Y = 5 THEN BEGIN
    Y := 0;
    WRITELN
  END {if}
END; {for}
WRITELN;
```

Of course, the execution limits of the FOR-DO loop would be expanded to something like "1E+38 DOWNTO -1E+38". (There're other questions to be addressed here, **precision** and **accuracy**; I'll turn to those below.)

But let's back up a minute to look just at **execution** time for such a routine. The above section of code (for **just** +100 to -100) takes 54 seconds to run after compiled (as a part of TBENCH from the last installment) by Turbo Pascal on my souped-up '89 and displayed on the screen. If the same routine is run under Lucidata Pascal (using the REALINT routine I give at the end of this article), the execution time is 3 min., 44 sec. (I mentioned last time that Lucidata is slower than Turbo. But Turbo's INT function is also built in **and** simpler in execution, as we'll see next time.) Further, if we **extrapolate** from the four orders of magnitude (powers of 10) tested here (+100 to -100) to those I proposed above (+1E+38 to -1E+38), we could expect to see an **increase** in execution time of about a factor of 19 ((38 * 2) / 4). For Turbo, such a routine would take about 18 minutes to run; Lucidata would require something like 70 minutes.

However, reviewing all that output on the screen would be extremely tedious; hardcopy would (undoubtedly?) be easier. But a suitably modified routine to dump the output to the printer will take much much longer simply because printers are **slower** than the CRT. My C.Itoh Prowriter dot-matrix requires almost 5 minutes when doing the +100 to -100 routine with Turbo; all possible "computer" real numbers could take about an hour and a half. Lucidata takes somewhat longer: about 6-1/4 min. for +100 to -100, an estimated two hours for all the numbers! And there's still the time and effort **you** would need to review the roughly 115 pages of output!

A professional programmer doesn't do testing this way! Usually he is under rigorous time and financial constraints (imposed by his employer) and

this "brute-force" approach is simply too time-consuming and machine-intensive. (You and I, as hobbyist programmers, are actually in the same boat when you think about it.) What the pro will usually do (if he does much testing at all) is select a small number of test cases.

But notice one implication the programmer makes when he selects only that small number of cases: trust in the language system to do the untested "intermediate" cases correctly. Generally, he (and we) can do this because the manufacturer has been reasonably thorough in the testing of his product. Bugs in language systems, particularly in better-known name-brands, are few. One such is the obscure bug Hank Lotz reported for MBASIC 5.21 under CP/M in issue #12, p. 5. Another is the problem with tabbing in Microsoft's COBOL I mention earlier in this issue. On the other hand, you should also know the **quirks** in the language you are using so you don't mistake wrinkles in it for bugs in your own code! (I've done that; more on it next time!)

Formulating Test Cases. For this series of articles, I'll be using "black-box" and "white-box" approaches to determining test cases for the various kinds of routines we'll be looking at. By "black-box," I mean that we don't know (or don't care) what the program code inside the routine is so long as the output from it is correct. For many subroutines, such as INT, ROUND, and TRUNC, this is more than adequate because **both** input and output are reasonably simple. In other situations, such as when designing tests for the "lookup" tables we'll be doing later on, we need to know what the code **inside** the routine is (or is supposed to be) in order to adequately test it. This is called a "white-box" approach because we can look **inside** the module to determine a good share of the tests we'll need.

Observe further that the tests I'll be discussing would be the **minimum** required to assure the correct functioning of a routine. Whether you're writing your own or keying and modifying one from some source, you may want to increase your confidence level by more intensive testing beyond these minimal tests. One definite example of such a situation would be testing a pseudo-random number generator, such as the one I expect to present in the next installment. In some situations, a loop, such as the one illustrated above to exercise the function over several powers of 10, might be in order. However, one thing you should do **as** you write or key in the function is decide exactly what tests you **plan** to use and **write these down**. A "shot-gun" approach to testing, that is, selecting a few arbitrary numbers to feed through a routine on the spur of the moment, is an extremely poor method for determining the quality of a module. One reason for this is that you seldom hit the really **critical** tests as I describe below. But I'll turn to test planning and execution in the next installment.

Actually, for simple routines, all we need do is test to be sure that the output is (within certain limits described below) correct. For numeric, single-input-parameter functions, these are few in number indeed. The tests we need are "extremes," "typical (or mid-ranges)," "boundary conditions," and "close calls." The next few paragraphs discuss these terms and two others of importance to testing.

Extremes are just that, the largest and/or smallest number(s) you expect the routine to process. For example the "extremes" for the short routine given above are +100 and -100. When considering all possible real numbers on a micro, the extremes would be +1E+38 and -1E+38. This is simple enough in concept, but there are two important catches to bear in mind.

When you start handling numbers with a large number of significant digits, you also have to know how many of those digits a computer language can reliably deal with. This is known as "precision" and is not to be confused with "accuracy." Precision is the **maximum** number of digits the computer can handle when representing a number. As a non-Pascal example, in Microsoft's BASIC interpreters and compilers, you have "single-" and "double-precision" variables at your disposal. The former have a precision of seven digits; the latter have 16. The choice is yours to select what precision you require in your program. You might well ask, why not do all calculations with double-precision? The reason is that calculations with double-precision numbers are significantly **slower** than with single-precision because the computer has to deal with more digits. Further, your program may not need all of those 16 digits of precision in its answers.

Accuracy, on the other hand, is the **error** in a calculation. How close is it to an absolutely correct answer? If your BASIC program requires the most accurate calculation your computer can produce, you will have to sacrifice speed of execution in order to use double-precision numbers. (Few programs actually require calculations of that accuracy.) When dealing with numbers with many significant digits, both precision and accuracy get jumbled together because of basic limitations in computing equipment, no matter how large or small the system might be.

You can get some feel for both of these issues on your own system by keying the following Pascal (or equivalent) code into TBENCH:

```
WRITELN;
WRITE ( 'Enter a positive REAL number: ');
READLN ( R );
WRITE ('The SQUARE of the SQUARE ROOT is ');
WRITELN ( SQR ( SQRT ( R ) ):18:9 );
WRITELN; WRITELN ( 'End of routine.' );
```

Notice the boldfaced section of code. This tells Pascal **how** to display the result of the calculation. Here, I'm asking for 18 **characters** in the answer (including the decimal point) with nine digits to the **right** of the decimal point. For you MBASIC programmers, this is equivalent to PRINT USING "########.#########". Run this inside TBENCH and, in response to the prompt, key in 10.666 and notice the result of the computation. Repeat the routine, but key 100.666. Continue repeating and keying, inserting another zero **before** the decimal point. At some point the last "6" to the right of the decimal point will round to "7". Count the number of digits in the calculation from the left **through** that "7" you now have. This count is the **precision** of your language. For example, Lucidata has a precision of 9 digits; Turbo has 11. Precision

also imposes limits on the accuracy of a computation. If the number of digits in an **intermediate** calculation exceeds precision, overall accuracy of the final result goes down. The language with the best precision I've seen on our systems is the ZBASIC compiler for CP/M (from Elliam Assoc.; see VENDOR.UPDATE in this issue), with a phenomenal 54 digits!

The above three-paragraph excursion was necessary to give you a better feel for some of the basic **limitations** in computing equipment. You must be cognizant of these during testing.

Now, to pick up were I left off, the second kind of minimal test is that of a "typical" or "mid-range" input value. Approximately what value do you expect the function will usually have to process? Or what is the value in the middle of the range you expect the routine to accept? Observe, though, that in the sample function for +100 to -100 above and similar ones, the mid-range is **not** zero (0). Zero is a boundary condition (as I'll discuss shortly) and we're interested in mid-range values **between** boundary conditions and/or extremes. For the above routine, there would be two mid-ranges to test, +50 and -50. For all real computer numbers, these could be +1E19 and -1E19. Of course, if your routine isn't supposed to handle positives or negatives, you need not worry about numbers with those signs. However, in your program you should also **ensure** that these numbers don't **reach** the routine. This can be done by trapping them someplace in the program. Locations could be during key-entry, external to the routine in question (such as part of the code immediately preceding a call to the routine), or inside the routine itself. Certain built-in functions (such as the square root) in most languages also expect **only** positive numbers. If using these, your program should include traps for negatives before these functions are called, otherwise the program may crash! As an example, I've had **commercial** software crash because of an untrapped division by zero. Very aggravating!

The third type of test is the boundary condition. A boundary condition marks a transition. For example, zero (0) is the boundary between positive and negative numbers. Indeed, one thing you'll have to watch for as you program is that a numeric function works correctly when processing numbers close to and equal to zero. For that reason, as you'll see, a professional will usually cluster **several** tests around zero.

Another boundary condition for routines that process real numbers is the highest and lowest **integer** numbers the computer is able to process. For most languages, these are +32,767 and -32,768. None of the material on testing I've perused treats these two as boundary conditions, but I strongly recommend you include them in your tests. As with zero, I suggest testing numbers **immediately contiguous** to them as well (these are "close calls"). One thing their inclusion checks is whether you have properly specified the **type** of variables (Pascal's integer or real, BASIC's integer or single-/double-precision, etc.) **inside** your routines. Of course, if your function only processes integers, these two numbers will be "extremes" rather than "boundary conditions."

The fourth type of test is the close call. You might consider this type as one that is intentionally "not quite on the money." In numeric routines, strange things can happen at a boundary, particularly around zero. If your routine is for positive and negative integers, you should test not only the boundary itself (zero), but the close-calls (+1 and -1). For real numbers, how you test close calls around zero depends on what the routine does and on how many decimal places you expect to display on the output (whether screen or printer). If the routine is processing decimal dollar amounts, you should at minimum include tests with inputs of +0.01, zero (the boundary), and -0.01. Beyond these examples, which ones to use is something of a judgment call. Actually, over-testing around zero is better than under-testing since strange things are **known** to happen here!

Close-calls around other boundary conditions and extremes don't need to be tested quite so thoroughly. If your routine uses positive and negative real numbers, you should include +32768.0 and +32766.0, -32766.0 and -32769.0 as close calls to +32767 and -32768, respectively. You should also test a number one less that the maximum positive and one greater than the minimum negative permitted by the precision of the language you're using. For example, since Turbo Pascal has a precision of 11 digits, the maximum and minimum numbers that **preserve** all decimal digits are +99,999,999,999 and -99,999,999,999. So include numbers that are one less than the former and one greater than the latter.

Assignment. But that's all I have space for in this installment. In the next one I begin guiding you through practical application of this professional program testing method. I had planned to discuss that quirk in Turbo's INT function here, but must postpone it till then. In the meantime, key in or adapt the function (REALINT) I give below. Although specifically written for Lucidata Pascal, it will run under Turbo as presented. Read the introductory comments so you know **how** the function is supposed to operate. Now test it as I've described above. Be sure you write down the tests you used, the results you got, and whether these were **what you expected**! This is **interactive** testing, so you will get **immediate** feedback from the routine to check against your expected answers.

Further, this function has one or more **bugs** I want you to find! If you apply what I covered above, you'll undoubtedly discover them. Next time I'll describe the tests I would use (to compare with your own) and a somewhat formal table-like layout I suggest you use for planning all testing you do. I'll also present fix(es) to the bug(s) in REALINT. Of course, if you have any questions about this or previous material, feel free to write.

LISTING

```
FUNCTION REALINT (X : REAL) : REAL;
{
  Rounds a real number (X), negative or positive;
based on Greg Davidson, PRACTICAL PASCAL PROGRAMS
(Osborne/McGraw-Hill, 1982), p. 199f.
  Input is a number of type REAL; output is a
number of type REAL.
```

For positive numbers, it rounds to the larger whole number when the fractional part is .500.. or larger (that is, 99.5 rounds to 100.0), to the smaller whole number when the fractional part is .499.. or smaller (that is, 9.49 rounds to 9.0). For negative numbers, it rounds to the larger whole number when the fractional part is -.499.. or larger (that is, -99.49 rounds to -99.0), to the smaller whole number when the fractional part is -.500.. or smaller (that is, -9.5 rounds to -10.0).

Input should be limited by Lucidata's precision to the range +/- 99,999,999.9.
}

```
FUNCTION REALTRUNC (NUM : REAL) : REAL;
 {truncates a real number (NUM) to zero (0) decimal
  places}

CONST   EPSILON = 1E-7;

VAR
   ACCUM              : REAL;
   NEGATE             : BOOLEAN;
   DIGIT, EXP, PLACES : INTEGER;

BEGIN
   NEGATE := NUM < 0;
   IF NEGATE THEN NUM := -NUM;
   EXP := 0; PLACES := 0;
   WHILE NUM >= 1 DO BEGIN
      NUM := NUM / 10;
      EXP := EXP + 1;
      PLACES := PLACES + 1
      END;
   IF PLACES <= 0 THEN REALTRUNC := 0.0
      ELSE BEGIN
      ACCUM := 0;
      WHILE PLACES > 0 DO BEGIN
         NUM := NUM * 10;
         DIGIT := TRUNC (NUM + EPSILON);
         NUM := NUM - DIGIT;
         ACCUM := ACCUM * 10 + DIGIT;
         EXP := EXP - 1;
         PLACES := PLACES - 1
         END;
      WHILE EXP > 0 DO BEGIN
         ACCUM := ACCUM * 10;
         EXP := EXP - 1
         END;
      WHILE EXP < 0 DO BEGIN
         ACCUM := ACCUM / 10;
         EXP := EXP + 1
         END;
      IF NEGATE THEN ACCUM := -ACCUM;
      REALTRUNC := ACCUM
   END
END; {realtrunc}

BEGIN {realint}
   REALINT := REALTRUNC (X + 0.5)
END; {realint}
```

=====

This 'n' That

by Hank Lotz / 2024 Sampson St. / Pgh, PA 15221

Color Me Puzzled: Steven W. Vagts described a perplexing problem in **REMark** of February 1987, p.52, where he continued his discussion of his **PAINT.ASM** program for CP/M. (For the original version, see **REMark**, March 1986.) The problem he noted was a failure to transfer properly to disk certain strings of graphic bytes if they contained too many graphic-mode-switching escape sequences. Many of the bytes didn't get written to disk. His article says he worked on the problem but finally decided to live with it, as it was minor, occurring only with complicated lines. Even so, if something doesn't work right I want to know why. I've since learned from Steven that "the problem seems to have gotten lost or modified" in the continuing development of the program. We may never know the cause, but if anyone knows any way such a phenomenon could be related to **hardware** or **system** software, please contact Kirk or me.

I typed in that whole thing (the one from March 1986), and modified it for my own purposes. It's a great program! And I think it was that educational March 1986 article that made me see how CP/M's BDOS function 6 could serve to **reliably** read the H-19/89 function keys, even in **MBASIC** programs! This led to my write-up ("**Function Keys...**" in **Staunch** #2, p.7) which, I think, satisfied a long-felt (and loudly proclaimed!) need among hobbyists, if they were willing to do the one-time preparation of my easily callable assembly-language subroutine for their libraries. So, since I failed to say it before: my very belated thanks to Steven Vagts for the revelation of function 6 and his use of it in context! In the said **Staunch** #2 article, I extended the idea to embrace also the 3-byte alternate-keypad keys, and I also showed how to **pass multiple parameters** back to the FORTRAN or MBASIC calling program.

But getting back to **PAINT.ASM**, I still want to add new features to my copy. One of my hopes is to implement a mod of it that will sign on with **no** screen output (except possibly the 25th line), the aim being that if the screen is thus undisturbed, it will be possible to copy any preexisting display to a disk file! The necessary presence of a command line on-screen (you need it in order to invoke PAINT) is no great blemish.

If you want the **PAINT** program (now "**PAINT89**" I believe), and you don't relish all that typing (it **is** a task, you know), Steven may still be distributing the COM **and source** files on disk. I suggest you write him to find out, and enclose an SASE. (I should mention he has an extremely busy schedule, but always gets back to you if you allow enough time!) His new address: Steven W. Vagts / 2409 Riddick Road / Elizabeth City, NC 27909.

Typos in the H-89A Op Manual: I noticed some typos in Section 11 of the H-89A Operation Manual. If you have the older "H-89" manual, I can't say whether these mistakes are present or not; and also, in the older manual, the same info may appear under a different section number. (Can you enlighten us on all that, Kirk, being that you have the Neanderthal? [The typos are in the H-88 manual, too, but in Section 12! -Ed.])

But in the H-89A manual, Section 11 is an APPENDIX. There are four errors in the ANSI escape sequences on page 11-18 -- the first **four lines** on

that page are wrong. Everywhere you see a lowercase j in those lines, make it an uppercase J. Likewise, change the lowercase k (fourth line) to an uppercase K. The same info appears **correctly** elsewhere in the same appendix.

LP Utility for CP/M: I have an H-14 printer I don't often use, but I do read of people still using them. If you use your H-14 and also run CP/M, you should be made aware of a few facts about my LP.COM program (in addition to what was said back in **Staunch** #7, p.8). LP.COM is available from Kirk, free but for shipping and handling. LP.COM does everything claimed for it, but I'd like to bring out a point or two in particular.

When you use the FEED FWD switch on an H-14 to advance the paper a few lines, the H-14 **forgets** where you had your top-of-form set, and you always have to realign it, which is a pain! Also, try holding in the FEED FWD button to move the paper up about 30 lines, and see how **long** it takes! But with LP.COM all you do is type "LP E30" (as a command line after a CP/M prompt) and your paper shoots right up! Moreover, when it stops, the H-14 still **remembers** where the top-of-form is! You can line feed any number of lines from 1 thru 99. The program takes only about **3 seconds** to load **and** run, and is thus the quickest (and **easiest**) way I have to handle the H-14, whether I'm setting horizontal or vertical pitches, sending those line feeds, ejecting forms, or flushing the printer's internal buffer. On that latter feature, I take it you've noticed how the H-14 retains characters sometimes when you aren't aware, and you get the tail end of the last thing you printed -- the **next** time you go to print! Solution: flush the buffer in a flash with LP.COM (command: "LP FL"). Also, thorough doc is supplied with this utility. LP.COM makes life easier for H-14'ers.

Old Floppies Never Die, They Just...?: In **Staunch** #17, Gary Appel brought up the question of the lifespan of data on floppy disks. The disks themselves may last, but could drop a few bits if magnetism weakens over time. I too am concerned about this problem -- I have been ever since I read Bob Ellerton's original Feb 1981 article in **REMark** #14, p.14, where he stated that data on "quality" diskettes has an "average shelf-life" of "about five years." And in the March 1986 **Pittsburgh HUG Newsletter**, I myself published a short piece on the subject. Basically all my article did was point to Mr. Ellerton's write-up, comment that 5 years is **no time at all** where stored data is concerned, suggest a procedure to refresh disks, and ask if anyone had further dope. At that time, I had never seen any of my distribution disks "fade," and I still haven't! Nevertheless, we're **all** in a position to make more meaningful observations now than we could four years ago, because more "decay time" has elapsed!

In my case I use only 48-TPI hard-sector diskettes and I store some distribution copies in a steel box, and some in a steel filing cabinet. The conditions of storage may be affording slight protection against long-term stray magnetic fields; however, as far as the lower **radial** density (48 TPI) is concerned, I've come to suspect (by roughly considering what I know of the parallel phenomenon of decay of the higher-frequency analog signals on audio tape) that **it** adds little to longevity.

I sometimes suspect that "panic time" is longer than 5 years for some disks, but I feel that since we **cannot know before the fact** when or if a given disk will fade, we **must** refresh, as there's no easy alternative **after** the fact.

I've corresponded with both Gary Appel and Kirk Thompson about this. Gary also feels he needs to do a refresh, but he has such a large number of disks, I think he'd like to rationalize putting it off "for just a bit longer."

Kirk finally put an end to my own speculation on whether disks will or won't fail. To quote from his letter: "I've certainly had disks, even old hard-sectors, 'fade.' Much depends, I think, on the **quality** of the media. For example, I've had no problems with the 3M's and Dysans I've bought over the years. Some of the really old commercial stuff I got...are also still good, such as the distribution set for HDOS 2.0 (which goes back **almost** ten years!). On the other hand, cheaper media have been a problem, including some 'no-name brands.'" End quote.

But if you glance back, you'll notice Bob Ellerton meant "quality" disks when he gave that **five-year** figure! Therefore, even though quality disks probably do **help** matters, I won't let that dissuade me from refreshing even my "quality" diskettes.

I already began my own preventative "refreshing" with a few disks in August 1987, and I've done a few more since that. (I should mention, I make DUP backups of my distribution disks at the time of purchase.) Here is my refresh procedure:

1. Verify dist. disk with existing DUP disk.
2. DUP from the dist. disk to a 3rd-party disk.
3. Verify the 3rd-party with dist. disk.
4. Verify the 3rd-party with old DUP disk.
5. Copy 3rd-party disk to old DUP disk. (This is the actual "refresh.")
6. Verify refreshed backup disk with dist. disk.
7. Release 3rd-party disk (or keep as 2nd backup)

You might stop after Step 3 or 4, labeling the "3rd-party" disk as the new backup DUP -- one reason for my "overkill" here is that the old backup disk has already stood the test of time, so why switch to an untried 3rd-party floppy for permanent storage? Another is, this checks things every step of the way.

Notice I do **not** rewrite to the distribution originals. This is on general principles; besides, a good backup DUP will suffice. (And keeping the D.D.'s as-is gives us a relatively risk-free barometer on when they **will** decay. You could even keep **two** backup DUPs. That way if the verify against the original (Step 1) some day **does** fail, the 2 DUPs can then be compared to check if it really was the **original** that changed!)

=====

VENDOR.UPDATE

ERRATUM. I misspoke (or rather, miss-wrote) when I added my bracketted comment about source code to Dan Jerome's review of Lindley's PC89LINK in issue #19.

H89/Z90 PARTS AND ACCESSORIES Dec 1990

Don't know what to give? How about a real Christmas "card"? It's a 6" high circuit board shaped like a Christmas tree, with colorful parts and blinking LEDs. It runs for weeks on a 9v battery, which doubles as a stand. Best of all, you get to say "I made it myself"! Manual $1, board and manual $3.95, complete kit $9.95.

Of course I still have plenty of H89/Z90 goodies ready for Christmas giving. Availability of used items is naturally on a first-come first-served basis, so call if you have any questions or to check availability. All prices include shipping, so order now!

NEW PRODUCTS

444+61 decoder PROM puts Z89-37 instead of H17	$10
in rightmost I/O slot	
LP001 LOW-POWER KIT includes MTR-90, cuts power	29
by 250mA. Specify if non-Heath ROMs at U516-U520	
LP002 MTR-90A (MTR-90 + 38400 baud support) lets ...	10
Superset boot at 38400 baud for 4X faster screen	
FF001 FLICKER-FREE kit eliminates screen flicker, ..	29
cuts power consumption for cooler operation.	
FF002 SUPERSET TLB upgrade adds prog.function keys,	49
38400 baud, screen-saver, on-screen time/date,	
help menus, white screen, interlace, and more!	
FF003 SUPERFONT adds 8 fonts including GTPROM, Z100,	29
VT-100, IBM-PC, 160x100 APA graphics, math, greek,	
bright/dim, double-wide chars, super/subscripts	
FF004 SUPERCLOCK adds battery backup for clock, ...	29
2nd page screen RAM, screen save/restore, user	
defined menus, fast animation, and windows!	
FF005 SUPERKEY uses CAPS-LOCK for "typewriter shift"	10
No need for SHIFT key in BASIC or CP/M commands	

USED EQUIPMENT

Magnolia 77316 soft-sector controller, complete	$90
Zenith Z89-37 soft-sector controller, complete	75
H89A 64K, 3 serial, H37, 1 40T DS drive, Superset ..	150
H89A 64K, 3 serial, H17+H37, 1 40T DS drive	125
H89A 64K, 3 serial, H37, 1 40T DS drive	100
H89 64K, 2 serial, H17, 1 40T SS drive, Flicker-free	65
H89 48K, 2 serial, H17, 1 40T SS drive	50
H77 with 2 40T SS drives and cables	50
H77 dual drive cabinet with cables, less drives	30
(free drive installation if you buy drives)	
non-Heath dual drive cabinet and power supply	25
for 2 FH 5-1/4" drives side-by-side horizontal	
Hayes 300 baud Smartmodem (the real thing!)	35

DISK DRIVES

FDD100-5 Siemens 40T SS FH 5-1/4" disk drive	10
TM100-2 Tandon 40T DS FH 5-1/4" disk drive	30
TM100-4 Tandon 80T DS FH 5-1/4" disk drive	50

SOFTWARE (specify disk format)

CP/M, Heath vers. 2.204	$29
Supercalc spreadsheet program for CP/M	29
WHM01 WRITE-HAND-MAN "sidekick" for CP/M. Hit BREAK	39
for calculator, notepad, calendar, phonebook, etc.	
WHM00 WHM ver. 2.4 update for Superset/Superclock ..	10
Full-screen VIEW/EDIT, 2xCLIPBOARD, screen dumps	

INTEGRATED CIRCUITS

4116 16K dynamic RAM 250 nSec (set of 8)	$5
4164 64K dynamic RAM 200 nSec (set of 8)	10
444-19 CPU H17 boot ROM	3
444-23 TLB character generator	1
444-37 TLB keyboard decoder	1
444-42 CPU memory PROM (48K, HDOS only)	1
444-43 CPU I/O PROM (H17, 3-port)	1
444-46 TLB program ROM	2
444-61 CPU I/O decoder PROM (H17/37/47/67, 3-port) .	6
444-62 CPU MTR-89 boot ROM (H17/47)	2
444-66 CPU "org-0" memory PROM (64K CP/M or HDOS) ..	4
444-81 Z89-37 I/O control PAL	5
444-82 Z89-37 interrupt control PAL	5
444-83 CPU bank 0 decoder PROM for MTR-90	4
444-142 CPU MTR-90 boot ROM (80T H17/37/47/67) ...	5

CABLES

134-1074 34-pin internal disk drive cable	$8
134-1144 34-pin internal/external drive cable	10
134-1158 16-pin Z89-37-to-CPU cable	5
134-1163 34-pin external drive cable	15

BOARDS

H19/89 Terminal Logic Board	$30
H19A/89A Terminal Logic Board	35
H89 CPU board (with 48K RAM and latest ROMs)	50
H89A CPU board (48K RAM, latest ROMs)	60
H88-1 H17 hard-sector controller	20
H88-3 3-port serial I/O	15
H89-16 16K RAM board (brings H89 up to 64K)	30
Z89-11 3-port I/O (2 serial, 1 parallel)	40
Z89-37 H37 soft-sector controller	50

MAIL C/O Lee Hart, 423 W. 18th Street, Holland, MI 49423, 616-396-5085

Bill Lindley asked me to print the following to correct the erroneous information I gave:

"Source code for PC89LINK is provided for the HDOS version only. This is the same as the source code for the CP/M version, but because of space limitations on the diskette, we could only fit one copy. Users wishing the CP/M source may simply transfer the HDOS files to CP/M, un-comment '#define CPM 1', and compile." The 8-bit versions of PC89LINK are written in Toolworks C/80; the MSDOS version is written in Microsoft C and Bill tells me that source for it is **not** available. My apologies for any inconvenience my misstatement may have caused you.

ANAPRO Moves, Again! "Extra! Extra! Flash news bulletin, read all about it. ANAPRO moves again! Well, if Lee Hart can do it, why can't we? Yes, we have moved a lot of times, but for the past 3 years, we have been in the same city. Isn't it time for a move?

"Kidding aside, I accepted a full-time teaching position at a community college in Blythe, California. Where is Blythe? It is a small town (8200 population) on the California/Arizona border. It is 100 miles from anything. I know that some of you out there are in similar situations, but it is the first time for this city-raised boy. Yes, it IS a real college. I am THE computer science instructor. The town is known for its HEAT. My wife says she likes the heat. We have had a hard time finding suitable housing. A new state prison was built near Blythe a couple of years ago and the influx of workers caused a housing crunch. After many frustrating false starts we have now found a house and are moving in. For the past 9 weeks I had been commuting each weekend (1000 miles round trip).

"So now you all know why there was silence from my end for a while. As soon as things get settled, more articles will appear. I do have an announcement to make. Triggered by comments which Lee Hart made in a letter to me, I created a new formatting program for the H89 under CP/M. It is a variation of the EMULATE EFORM program and allows selection of formats from a menu or entry of the selection from command line. No more answering awkward prompts. I have supplied Kirk Thompson with the package as ANAFORM.LBR - use NULU or LU to extract the members from the library. [Readers will find this described in the Software Listing. The package will also be available from **SEBHC Journal**. -Ed.]

"The development of the new format program uncovered a defect in the EMULATE EFORM program. EFORM has not yet been changed to fix the potential problem area, but a modification is in the works (just as soon as I have a home). If any EMULATE owners have had problems with EFORM, let me know (at the new address below) and describe the difficulties.

"That's all for now. The new address for ANAPRO (and for me) is: Peter Shkabara, Box 1987, Blythe, CA 92226 (619) 922-3919.

SEBHC Journal **Raises Subscription Rates.** Lenny Geisler, editor of the "friendly" competition, announced an increase in his subscription rate after August of this year. Regrettably, as he noted in his October issue (p. 6), declining circulation has forced a conversion from bulk-mail rates to first class. The new rate is: one year -- $24.00, two years -- $44.00; overseas, add $5 for one year, $9 for two. Of course, payment must be in U.S. funds and checks made out to "L.E. Geisler". Lenny also noted that actual expiration dates are now printed on the address label so subscribers know exactly where they stand.

Also worthy of note is that Lenny now carries a **functioning** version of Newline's old editor, TXTPRO ver. 4.1, for both HDOS and CP/M. This is a comparatively easy-to-use, WYSIWYG, ASCII editor. Lenny uses it to prepare the **Journal** and I've used it for certain projects myself. One thing it permits is creation and editing of files larger than working memory (a limitation in Software Toolworks' PIE, the other, simple, popular ASCII screen editor available for our machines). On-line help is available inside the program and documentation is on disk. Recommended if you don't already have a screen editor. Order codes, description, and prices are as follows:

HTXTH	For HDOS 2.0, 3.0x, on standard hard-sector (two disks), $32.00
CTXTH	For CP/M-80, on standard hard-sector (two disks), $32.00
HTXTS	For HDOS 2.0, 3.0x, on SS soft-sector (one disk), $29.50
CTXTS	For CP/M-80, on SS soft-sector (one disk), $29.50

To order subscriptions, software, or for information, contact:

Leonard Geisler / Editor, **SEBHC Journal** / 895 Starwick Drive / Ann Arbor, MI 48105 / 313-662-0750 (9 to 6 Eastern, weekdays)

HUG Software Sale. HUG is presently running its annual 30%-off software sale. And there's quite a bit of good stuff **not** listed in **REMark** that's still available. I ordered one myself in late November, HDOS W.I.S.E (#885-1038-37). If you'd like a list of what's available for both HDOS and CP/M from HUG, send **me** a long, stamped, self-addressed envelope and I'll return a three-page list compiled by reader Parks Watson. To order the software, contact:

HUG / P.O. Box 217 / Benton Harbor, MI 49022-0217 / 616-982-3463

And I might add that HUG is now officially a part of Zenith Data Systems; Heath is up for sale, again!

The Z-Letter. I noted the existence of this newsletter a year ago in issue #15. It continues to specialize in the Z-System, but recently underwent some changes. Its connection with Alpha Systems has been severed, the editor having started his own company (Lambda Software), and beginning in November this year, it shifted from an irregular to a monthly publishing schedule. Its format has also ballooned from a half-sized booklet to 20+ pages of 8-1/2 x 11. Subscriptions continue at $24 per yr.

As I remarked in #15, there's nothing specifically H/Z here, but if you're interested in or are running Z-System, I think you should at least give it a college try. This first monthly issue (#7)

included a thorough review of the Z-Festival, sponsored by the Connecticut CP/M Users' Group (CCP/M), held late last October. One of the references included in this article was to:

Lee Bradley / **Small Computer Support** / 24 Cedar St. / Newington, CT 06111 / 203-666-3139 (voice), 203-665-1100 (data)

as source for a $2 software catalog (actual disks go for $5 apiece) and a booklet of reprints from the CCP/M newsletter for $5.

And as a parenthetical paragraph, **H-SCOOP** #127 included information **also** from Lee Bradley about a new CP/M bimontly newsletter, **Eight Pieces & Change**, for $15 a year. If you belong to a users' group, you qualify for a 20% discount. Like **Staunch**, it pays for articles, though not so "munificently"! I haven't had a chance to check this new publication out myself, though. More later!

Anyway, **The Z-Letter** also included ads from **The Computer Journal**, Herbert R. Johnson (S-100- and SS-50-bus used equipment), Lambda Software, Davidge Corp., Logic Associates, and the "Socrates" Z-NODE 32 BBS. (Wow! Real 8-bit advertising!) For further information about **The Z-Letter**, contact:

David A.J. McGlone / **Lambda Sofware Publishing** / 720 S. 2nd St. / San Jose, CA 95112 / 408-293-5176

I found issue #7 much more impressive that the prior ones I'd received.

Quikdata. "...I would appreciate it if you would let your readers know that our new 7/90 catalog has been prepared and mailed out. If any of your readers are not on my mailing list, they can simply write to me at the ... address [below] or call and request a free no-obligation catalog. There is a hefty section on the H89, both hardware and software. A collection of some pretty good stuff, continuing our tradition of supporting the H/Z 8-bit machines. If they also request our liquidation list we will gladly send that also. It also has some 8-bit stuff being liquidated at good prices. Or they can call our Bulletin board at (414) 452-4345 and download the liquidation list. - Henry Fale" / 2618 Penn Circle / Sheboygan, WI 53081 / 414-452-4172

Micronics Technology. "I've enclosed a current catalog. The news for 89'ers is the price drop to $329 for the WIN89 [harddrive]. I need to clear some stock by the end of the year, so now's the time to buy. I've also got about 500 Speed Mod boards left. My last Speed Mod sale was six months ago. I am thinking of offering the kits again at $20 each or the boards with instructions and software for $10. Speed Mod works fine with HDOS 3.0 as long as you don't access the hard sector disks. I am sure that the H-17 drivers could be modified like the HDOS 2.0 versions, but I haven't attempted it. Guess I might have to finally break down and buy the Gibson assembler. I am still working on the next version of MTMDM and hope to have it completed by the end of September along with HDOS 3.0 support for the WIN89. I'll keep you posted on upcoming events."

For a catalog, contact:

Darrell C. Pelan / Micronics Technology / Suite 159, 54 Dalraida Road / Montgomery, AL 36109 / 205-244-1597 (voice, 6 - 8 pm Central, weekdays), 205-244-0192 (data)

PAINT+ CP/M Graphics Editor and HDOS 2.0/3.0 Clock or Ramdisk drivers. Some material from Lee Hart that appeared in **H-SCOOP** #128 is of interest to us. Lee announced a new graphics editor, intended for Superset users, by Steve Vagts. The editor is an enhanced version of the one Steve wrote about in a series of articles for **REMark**. With it you can create, edit, and print pictures using any of the Superset's character sets. You can also define and print your **own** character set. PAINT+ includes on-line help screens and uses the function keys. It requires TMSI's Superset and Superfont. No price for the software was given.

For the moment, the editor only supports the Panasonic KX-P1092 and KX-P1124 printers. But Lee and Steve are requesting information from users about printers in current use and the codes used to select graphics and other modes. To contact Steve, write to:

Steven Vagts / 2409 Riddick Road / Elizabeth City, NC 27909

Lee also announced two device drivers for HDOS 2.0 and 3.0x, provided by Terry Hall. One of these (CLOCK.DVD) is a real-time clock/calendar using TMSI's Superclock to set the current time/date. The other (GH.DVD) is a RAMdisk driver for users of the H-1000 CPU board: "It uses all RAM above 64K as a super-fast disk. An H-1000 with 1 meg of RAM has a RAM disk of over 900K." These two are **free** if you send an HDOS-formatted disk and postage-prepaid, addressed mailer to:

TMSI / c/o Lee Hart / 323 W. 19th / Holland, MI 49432

Elliam Associates. I mentioned this outfit a couple years ago (#9), but I ran into another ad from it this last summer. The line of CP/M software Elliam Assoc. carries has grown since my last reference to it. It now can supply a rather detailed catalog of its public domain (CP/M User Group, C Users' Group, Pascal/Z User Group, and SIG/M) holdings. This catalog costs $8.50 plus $1.50 shipping.

Elliam's commercial holdings catalog costs $1.00 and lists such things as WordStar 4, the MagicSeries laser printer package, T/Maker, Z-System, BackGrounder ii, Z80 and 8080 assemblers, SuperCalc2, dBASE II, Tarbell Database System, various language packages (including Turbo Pascal, Microsoft's BASIC interpreter and compiler, Toolworks C/80, BDS C, ZBASIC, LISP/80, and apparently the Nevada language set [COBOL, Pilot, BASIC, FORTRAN, and Pascal]), education programs for elementary through high school levels, and quite a bit more. But be warned that much of this software is **no longer** supported by the manufacturers. Further, better prices for some items are available elsewhere, especially for WordStar 4 directly from WordStar International (see issue #14, p. 12). Some commercial software is **still** outrageously priced,

too.

But this material is definitely worth the look. To order the catalogs or request further information, contact:

Elliam Associates / P.O. Box 2664 / Atascadero, CA 93423 / 805-466-8440

=====

PETE ON CP/M
By Peter Shkabara

Those readers that follow my contributions may have noticed that there has been a delay in my writing. I started writing this installment in May, but suffered writer's block. Perhaps it would be more correct to say that I had a form of writer's cramp since I did not run out of things to write about, but ran out of time to do it in!

Let's start off by correcting an error in the FORMAT patch listing which appeared in issue #17. Corky Kirk wrote to point out the transposition of the values 94 and D1. The corrected list of DDT operations should be:

```
DDT VERS 2.2
-S0267
0267 D1 94
0268 13 .
-S07A6
07A6 D1 94
07A7 13 .
-S0C6B
0C6B D1 94
0C6C 13 .
-^C
A>save 25 FORM.COM
```

This correction is particularly important if the abort with control-C patch was also installed. The later patch was placed at address 13D1 which is the old density logo location. Just goes to show that you can't trust anyone!

A note for those who may have seen the similar density display patch in issue #13 [p. 2]. That patch was correct, but it did not patch all locations where density is referenced. Without patching all locations, the new logo would appear only on the initial pass when running the program. Repeat questions for density would show the old logo.

In response to my last article, I got letters from Lee Hart and Hank Lotz. Since the items they mentioned should be of interest to your readers, I chose to respond to both of them herein.

Lee asked if my FORMAT patches were a reprint from somewhere else. Other articles had mentioned some of the patch locations, but the set of patches I supplied were never published before.

To continue on the subject, I noticed in my collection of assembly notes that there is still one more patch to be had for the FORMAT program. This one has to do only with the formatting of H17 hard sector disks.

In addition to extended double density, another hidden feature is the formatting of hard sector disks without the verify process. Pat Swayne revealed this option a long time ago in **Remark**. To activate the option, include an asterisk '*' following the FORMAT command. For example:

```
A>FORMAT *
```

At this point, it may be useful to discuss what exactly a verify is. When a disk is formatted, a set of identifying marks is written to the blank disk in order for the system to locate individual sectors of information. Just as in a tape recorder, until you play back the information, you cannot be sure that the write took place properly. In most cases there should be no problem, but if there is a defect in the recording media, the sector marks may be lost.

The verify operation simply rereads the track just written to see if all the sectors can be found. In some programs the verify is a separate option of the FORMAT program so that disks can be formatted and verified as separate operations. Verify can take place as each track if formatted, or it can be done after the entire disk is done. Instead of using the FORMAT verify sequence, you may wish to run FINDBAD instead. [FINDBAD is in the utilities package listed on p. 2 of issue #18. -Ed.] FINDBAD is a public domain utility which checks for bad data area on the disk as well as the sector marks.

Getting back to the Heath FORMAT program. I found that there was a significant speed improvement in formatting hard sector disks when the verify option was turned off. Since this was the most common way I would format the hard sector disks, I wanted the no-verify to be the default mode. Fortunately, this is a very simple thing to do if you understand even a bit of assembly language. Using DDT, I located the instruction which checks for the '*' option. Then I changed the result of the comparison test to be the complement of what it was. The following patch will do it for FORMAT version 03.

```
A>DDT FORMAT.COM

DDT VERS 2.2
-S01FE
01FE 20 28
01FF 04 .
-^C
A>save 25 FORM.COM
```

The change in code (with Intel mnemonics) is as follows:

```
original                    new
--------------------------- ---------
01FC CPI  '*'
01FE JNZ  0204h             JZ  0204h
0201 STA  170Eh
```

For those interested, the flag stored at 170Eh is used by a section of code located at address 0497h. So there you have a new patch for the FORMAT program!

Since I have been presenting a bit of information on the formatting of disks, it may be appropriate to continue on this theme till we are mostly done. In my last installment, I explained that formatting marks out the sector boundaries to be used later for writing information. The size of

sector is controlled by the format program as designed by the originator of the system. I will now add to your knowledge of what other details take place inside this mysterious (to most) program.

We first need to distinguish between the H37 soft sector and the H17 hard sector controllers. The H37 uses an industry standard Western Digital 1797 controller IC, while the H17 is a proprietary configuration of standard IC parts. Because of the H17 design, it is not capable of reading double density disks. [**Single-density** soft-sector is possible. See Lee Hart's "800K on a HARD-SECTOR DISK" in **SEBHC Journal**, vol. 1, #5, p. 10-13. -Ed.] Conversely, the H37 can't read the hard sector H17 disks either. Since most current work is with double density formats, I will provide information on that particular format. In general, the same would apply to the H17, but the hardware does not do as much and more needs to be done in the software.

Let us start off at the beginning with the design of the disk format. This means that we need to select the physical sector size, the block size, the number of system tracks and several other items. Before you start feeling lost, bear with me and I will try to explain it all in simple terms (at least to the degree of my own understanding).

The total capacity of the disk depends on the number of tracks, the number of sides and the number of bits per track (the recording density). The number of sides and tracks is usually dictated by the construction of the drive mechanism. Sometimes the innermost tracks of a drive are not used on purpose to improve reliability. This was done by C.D.R. in the high-density drive option they offered with their FDC880H controller. In addition to more reliable operation, the restriction to 77 of the 80 available tracks made the 5-1/4 inch configuration software compatible with 8 inch drives.

The number of bits per track are dictated by the rotational speed of the drive and the clock speed and design of the controller. Note that the system designer needs to consider the tolerances of the two parameters, drive speed and clock speed. If the drive is rotating slowly, more bits can be recorded in each revolution. On the other hand, if the drive is fast, then fewer bits will fit. I might mention that the reason for not running the drive more slowly is that signal degradation will result. The clock speed of the controller has a similar effect. If the clock is fast, more bits can be recorded per revolution and vice versa. What dictates the design concept of the hardware is the number of bits that can **reliably** be recorded per linear inch of recording medium. This is the BPI (bits per inch) rating for the floppy disk material. High density disks such as used in the IBM AT computer have a different recording material to allow greater BPI figures.

With the hardware considerations out of the way, we look at the software designers part in the disk format. If we consider the fastest allowable (within tolerance) rotational speed combined with the slowest controller clock speed we will have the worst case condition for safe assumption of available total bits per track. Let's go through some rough calculations. The nominal rotational speed of a standard disk drive is 300 RPM. This translates into 5 revolutions per second, or 200 ms per revolution. The controller system clock is normally 1 MHz and is divided by 2 internally in the Western Digital 1797 type controller. The data rate is thus 500 KHz or a 2 microsecond bit spacing. So in 200 milliseconds of revolution, we can potentially get 100,000 bits. Taking 8 bits per byte, the 100K bits translates into 12,500 bytes per track. For a sided 40 track drive this results in 40 x 2 x 12,500 = 1,000,000 bytes of raw recording capacity. If the rotational speed is 5% high and the clock 5% low, we can loose roughly 10% leaving 900K bytes. Since the average 40 track DS disk only holds 360K, you can appreciate the recording overhead needed to store the information!

There are different ways to record information on disk, but most of the CP/M world (and the IBM PC for that matter) conforms to the standard format developed by IBM for the 3740 (single density) and later the system 34 configuration. The Western Digital ICs allow formatting only in variations of the IBM standard. Apple [and Commodore] computers used a different controller (GRC - group code recording) and thus it is not physically possible to read those disks on Heath or IBM type computer. Incidentally, the GRC method is a technically superior method but is less supported in the microcomputer world. For those interested in calculating the true overhead in floppy recording, I would refer you to the Western Digital technical specifications for the 179x series of ICs.

I could go on and explain the specifics of the CP/M disk format parameters, which is what was originally planned. However, I am running short on time and if I continue this article will never get to Kirk in time for publication. So stay tuned for the rest of the story. [A somewhat technical introduction to soft-sector formatting and reading Kaypro disks on our systems with Pete's EMULATE package was presented by Ludo Van Hemelryck in issue #12.

[And speaking of EMULATE, Pete tells me he's working on a new version. It'll combine the disk read/write utility (EMULATE) with the foreign disk formatter (EFORM) in one program. I expect to have more information about it when Pete releases the package. -Ed.]

=====

C/80, Longjumps, and Control Character Interrupts in HDOS

By Gary Appel

The HDOS operating system supports the use of control character interrupt handlers for the control A, B, and C characters. These handlers, if defined, are executed whenever the corresponding control character is struck. These interrupt handlers can provide instant keyboard interrupts to the user, to an extent not offered even under MSDOS.

The HDOS version of C/80 incorporates the ctrl-C interrupt to provide an abort function. Not too exciting. Other than this, no control character interrupt processing is performed by C/80. And the ctrl-C abort may cause problems, if ctrl-C is struck while the display is in some unusual mode.

The addition of a few simple functions can enhance the capabilities of C/80 to handle the control A, B, and C interrupts under HDOS, using them as intended by the creators of HDOS.

When the user hits a control A, B, or C character, HDOS will attempt to execute a user defined service routine (the 'handler'). If the handler address is zero (the default), execution will continue as if no interrupt had occurred. If a routine address has been provided to the operating system, HDOS will enter the service routine with the interrupted PC value and the interrupted PSW on the stack, along with a return address into HDOS.

The user routine would normally save the other registers, execute its code, restore the saved registers, and finally return control to HDOS with a return instruction. HDOS will take care of restoring the interrupted PSW and return control to the interrupted program.

As an alternative, control can be transferred directly back to the executing program. This is a bit trickier, and requires the use of the setjmp and longjmp functions to be described later.

In the case of C/80, the start up code installs the address 'exic' into the ctrl-C handler address. When the ctrl-C character is struck execution will immediately be transferred to the point 'exic' in the runtime library. C/80 will first clear the console buffer, and then perform the standard C/80 exit. No return is ever executed back to the user program.

The simplest way to incorporate the control character interrupt handlers into a C/80 program is to provide a service routine which simply sets a flag when the control character is pressed. The C/80 program can test the flag as required, and take an appropriate action. While this may seem to defeat the use of interrupts, it does provide the user the ability to interrupt the program even if the program never looks at the keyboard. For instance, examine the following code:

```
do
{   calculations   }
while (!done and !control_C_Flag);
```

The calculations loop will continue to execute until done, unless the user strikes the ctrl-C character, which will cause an abort at the end of the present iteration. Crude, but effective. We require two functions to accomplish this level of handling:

```
set_int(which)  Used to enable the routine.
tst_int(which)  Used to test the flag.
```

Where 'which' is the character 'A', 'B', or 'C', corresponding to the desired control character. The function set_int is called initially to set up the service routine, and to clear the flag. It may be called at any later time, with the effect of clearing the flag.

The function tst_int is called to determine the condition of the flag. It returns a 1 if the control character has been struck, and a 0 otherwise. It also resets the flag before returning. The above loop now becomes:

```
set_int('C');
do
{   calculations   }
while (!done and !tst_int('C'));
```

The ctrl-C character will abort the calculations loop if struck. These functions could be used in this way to follow the HDOS practice of aborting a command whenever the ctrl-C character is struck, so long as the tst_int function is executed regularly.

In many cases we would like the control character interrupt to force the program to resume execution at some pre-defined location. In order to do this we must have the ability to mark this location, and later jump to it. This ability is provided by the long jump.

Two functions have been defined in C to enable a long jump. These functions are the 'setjmp' function, and the 'longjmp' function. C/80 does not support the long jump functions, so we'll have to write our own.

The setjmp function is used to 'mark' a point in the program. At a later time, a jump may be executed to the 'marked' point in the program, continuing execution. When the setjmp function is executed it will return a value of zero.

The longjmp function is used to accomplish the jump in the program flow. The long jump appears to resume execution within the setjmp function, which now returns a non-zero value which was passed as a parameter to the longjmp function. This returned value can be used to determine where the long jump originated. Consider the following example:

```
if (value = setjmp(here))
    printf("Long Jump number %d/n", value);
if (value == 20)
    exit();
for (;;)
    longjmp(here, value+1);
```

When setjmp is initially called, it returns a value of zero, which is stored in the variable 'value'. Because the result of the expression is zero, the printf statement will not be executed. Since value does not equal 20, the exit will not be executed. Control then passes to the infinite for-loop.

The only statement contained in the for-loop is a long jump back to the location marked with the setjmp function. But longjmp returns a value of 'value+1', which is equal to 1 on the first execution of longjmp. Although the value of 1 is returned by longjmp, it appears to be returned by the setjmp function. The returned value is assigned to 'value', and the printf statement is now executed, flagging a successful long jump.

Again the exit function is not executed, and control passes to the infinite for-loop which executes longjmp with a value of 2. This loop is repeated twenty times printing out the value of 'value' as it is assigned values from 1 to 20. Finally, with 'value' equal to 20, the exit function is executed, and the program terminates.

The infinite for-loop is actually not required, as the body of the loop is executed only once per long jump. It does, however, serve to show that a long jump can be used to execute from a loop at any time.

The pointer here passed to both the setjmp and longjmp functions is a pointer to an 'environment block'. This is a structure which is used to store values of certain CPU registers at the time when the setjmp function is executed. The long jump is

accomplished by having the longjmp function restore these values to the CPU registers, in effect allowing the program execution to continue just where it was when the setjmp function returned, but returning a non-zero value passed to the longjmp function. In our case we need only restore the contents of the program counter, and the stack pointer, and return a value which is left in the HL register.

There are a few restrictions to our version of setjmp. The setjmp function must be executed before attempting a long jump to that location. If setjmp had not yet been executed, the destination point would not yet be defined. A program crash would almost certainly result.

The function which called setjmp must still be active when longjmp is executed. If the function has already returned the stack will no longer be valid, including what was the return address from the setjmp function. A crash will again most likely occur.

Finally, we require that setjmp not be used in an expression, except as an expression by itself. This is required because any evaluation of the expression prior to the execution of setjmp may not be restored during the execution of longjmp, and even if it is, the values of any variables may have changed.

Now that we have examined the long jump functions, we can use them for our control character interrupt processing. It should be noted that the long jump is very similar to a goto statement in function, but is of a global nature. The destination of the goto statement must be within the same function as the goto statement. The long jump is not so restrictive. So long as we have not returned from the function in which the destination was defined using the setjmp function, we can jump to that destination from anywhere. The only requirement is that the longjmp function be passed the environment structure. We may execute the long jump from within the same function, or from a called function, at any depth, and even from a separately compiled function.

Our interrupt service routine can use a long jump to resume program execution at some predefined point. Consider the following code:

```
static  abort_menu();
main()
{    /* main */
    setjmp(restart);
    install('C', abort_menu);
    choice = master_menu();
    switch(choice)
    {
         case 0:
              sub_menu_0;
              break;
         case 1:
              sub_menu_1;
              break;
         etc.....
    }
}    /* main */

abort_menu()

{
    longjmp(restart, 1);
}
```

I have introduced a new function, called 'install'. The purpose of install is to install the address of a control character interrupt handler into HDOS (accomplished with the HDOS .CTLC System Call). Once installed, striking the selected control character will result in execution of the interrupt handler. In this code segment we have set up the function 'abort_menu' as the interrupt handler. Our handler will simply execute a long jump back to the start of 'main'.

Once the user has selected a choice from the master menu, he is dumped into a sub_menu function, and whatever happens beyond there. But at any time he can strike the ctrl-C key and be immediately returned to the master menu. That can be pretty useful.

The interrupt handler does not have to execute a long jump to exit. It can perform some interrupt time processing, then simply return, resuming execution at the point where the program was interrupted. Because this happens at interrupt time, we would have to be very careful with the coding of the function.

The first requirement would be to save the CPU registers immediately. The last step before exiting the function would be to restore the CPU registers. Although not specifically stated in the HDOS documentation, I assume that you cannot issue any system calls from within the interrupt handler. This restriction exists because the interrupt may have occurred during HDOS processing. Another call to HDOS would most likely destroy the state of the HDOS call that was interrupted.

It would be best to limit the use of the interrupt handlers to the first two cases, simply setting a flag, or executing a long jump back into the user program. While the ability to write a complex user interrupt handler function exists, it is a technique full of pitfalls.

Listing 1 is a listing of the functions written to provide control character interrupt handling under HDOS. These functions were compiled as a separate module under C/80, and assembled using M80. I would strongly recommend using the Microsoft assembler when developing programs. When the functions have been debugged they can be added to a library (as described later), and linked with the main program, greatly reducing the compile and assemble time.

The three globally accessible functions are set_int, tst_int, and install. These functions have already been defined. Also included in this module are the three local functions ctrl_A, ctrl_B, and ctrl_C. These are the interrupt handlers for setting a flag during a control character interrupt. The interrupt flags are contained in the three element array 'flag'.

The set_int function simply resets the interrupt flag and installs the appropriate handler function address. Although the handlers need to be installed on only the first execution, the repeated installation each time the function is called causes no problem.

The tst_int function returns the value of the appropriate flag, and resets it if required.

The install function interfaces directly with HDOS. The desired control character to install is passed as an 'A', 'B', or 'C' character, which is mapped to a binary 1, 2, or 3 in the HL register. This value is placed in the A register with the MOV instruction. The address of the handler function is then placed in the HL register with the 'addr;' statement. System call 41Q is executed to install the handler, and the function returns. HDOS will now recognize the corresponding control character interrupt.

The actual handlers simply set the appropriate flag and return. Upon examining the compiler output it can be seen that only the HL register is affected by the assignment statement. We must therefore save and restore only the contents of the HL register during the handler execution. Remember, the handlers are executed during interrupt time.

Listing 2 is a listing of the setjmp and longjmp functions. The setjmp function saves the PC and SP values into the environment block whose address was passed in the function call. Upon entry to the setjmp function the stack contains the return address, which is where longjmp will resume execution, and the address of the environment block.

The address of the environment block is retrieved with the 'blk;' statement, and placed into the DE register with the 'XCHG' instruction. The return address is then retrieved from the stack and saved in the environment block. Finally the value of SP plus 2 is saved as the stack location, which will be correct after the setjmp function returns.

The longjmp function again retrieves the address of the environment block and places it into the DE register. The return address saved in the environment block is then retrieved and placed into the 'JMP' instruction. The saved SP value is retrieved and placed on the stack for temporary storage.

Next the return value (passed as a parameter to 'longjmp') is obtained from the stack and placed in the DE register. Finally SP is restored with a 'POP' and 'SPHL' instruction, and the return value is placed into the HL register. The 'JMP' instruction is executed, returning control to the location where the setjmp function returned.

Upon returning SP is set two bytes too high. This will be taken care of by a 'POP' instruction supplied by C/80 following the setjmp call, used to pop off the address parameter passed to the setjmp function. The BC, DE, and AF registers are undefined, and therefore did not need to be saved in the environment block and restored.

The standard definition of longjmp states longjmp cannot return a value of zero. As implemented here, the return value is not tested, and a zero value may be returned.

Listing 3 is the header file "longjump.h", used to define the environment block structure. It must be included in the source code for the setjmp and longjump functions, and in any programs or modules calling them.

After compiling these modules, they are linked in with any program using them. The link process can be simplified by adding them to a library. This can be accomplished for HDOS by copying the .REL files and libraries over to CP/M using an HDOS to CP/M file transfer program. The relocatable modules can then be added to the libraries as desired using the CP/M librarian. The finished libraries are then copied back over to HDOS using a CP/M to HDOS file transfer program. Its awkward, but saves a lot of time later. I have incorporated all the C/80 functions including the long and floating point libraries into a library called 'CLIB.REL', and my own functions into a library call 'MYLIB.REL'.

The control character interrupt functions presented here should allow C/80 programs to be written to recognize user interrupts using the control A, B, and C characters. The ctrl-C interrupt can be used to abort commands in process, as implemented in many system programs. The long jump functions can be incorporated in programs either in conjunction with the interrupt functions, or simply to implement a global form of the goto statement.

I hope these functions prove useful to other programmers still programming under HDOS, and bring some of the benefits of HDOS into C/80 programs.

Listing 1

[Note: These listing have been edited to fit a 52-character column width and to conserve space vertically. If you'd like these listings on disk, just send me a formatted disk with postage-prepaid mailer and I'll transfer them for you free of charge. -Ed.]

```
/*interpts.c: Ctrl Character Interrupt Handlers */
static  char    flag[3];  /* Interrupt Flags */
static  int     ctrl_A(); /* Interrupt Handlers */
static  int     ctrl_B();
static  int     ctrl_C();
set_int(code)   /* Enable Interrupt Handler */
    char  code;         /* Control Character */
{   /* set_int */
    static  int i;    /* Index flag */
    static  (*loc)(); /* Store Handler Address */
    i = toupper(code) - 'A';
    flag[i] = 0; /* Reset Interrupt Flag */
    switch (i)  /* Set up Correct Handler Address */
    {
        case 0:
            loc = ctrl_A;
            break;
        case  1:
            loc = ctrl_B;
            break;
        case  2:
            loc = ctrl_C;
            break;
    }
    install(code, loc); /* Install Handler */
}   /* set_int */
tst_int(code)   /* Test for Interrupt 6/27/89 */
    char  code;         /* Control Character */
{   /* tst_int */
    static  int   i;    /* index into flags */
    i = toupper(code) - 'A';
    if (flag[i])     /* Has the flag been set ? */
    {   flag[i] = 0; /* Yes.Reset flag and return */
        return(1);
    }
    return(0);          /* No Interrupt */
}   /* tst_int */
install(code, loc)/*Install Intrpt H'dler 6/27/89 */
```

```
    int  code;      /* Control Character */
    int  (*loc)(); /* Pointer to Interrupt Handler*/
{   /* install */
    static  int  i; /* Code to pass to SCALL */
                    /* (1, 2, or 3) */
    static  int  (*addr)(); /*Pointer to Interrupt*/
                    /* Handler, Local */
    addr = loc;                 /* Save Address */
    i = toupper(code) - '@';    /* 1, 2, or 3 */
    i; /* Move Code (1, 2, or 3) into A */
#asm
    MOV  A,L
#endasm
    addr;               /* Address into HL */
#asm
                        /* And Call HDOS */
    RST  7
    DB   41q
#endasm
}   /* install */
static  ctrl_A()        /* Set Control A flag */
{   /* ctrl_A */
#asm
    PUSH HL             /* Must Save HL */
#endasm
    flag[0] = 1;        /* Set the Flag */
#asm
    POP  HL             /* And Restore HL */
#endasm
}   /* ctrl_A */
static  ctrl_B()        /* Set Control B flag */
{   /* ctrl_B */
#asm
    PUSH HL
#endasm
    flag[1] = 1;
#asm
    POP  HL
#endasm
}   /* ctrl_B */
static  ctrl_C()        /* Set Control C flag */
{   /* ctrl_C */
#asm
    PUSH HL
#endasm
    flag[2] = 1;
#asm
    POP  HL
#endasm
}   /* ctrl_C */
```

Listing 2

```
/*  longjump.c: C80 longjmp/setjump functions */
#include  "longjump.h"  /* Definition of env_blk */
setjmp(blk)
    struct  env_blk  *blk; /* Environmental Block*/
                           /* Address */
{   /* setjmp */
    blk; /* Get Address for Environment Data */
#asm
        XCHG    /* Save Address in DE */
         /* Get Address of Next Instruction */
        LXI     HL,0  /* Address is Last on Stack */
        DAD     SP
        MOV     A,M /* Move it into Envrnmt Block */
        STAX    DE
        INX     DE
        INX     HL
        MOV     A,M
        STAX    DE
        INX     DE
        INX     HL
        MOV     A,L     /* Save Stack Position */
        STAX    DE
        INX     DE
        MOV     A,H
        STAX    DE
        LXI     HL,0    /* Return 0 */
#endasm
}   /* setjmp */
longjmp(blk, val)
    struct  env_blk  *blk; /*Environment Block*/
                           /* Address */
    int     val;               /* Return Value */
{   /* longjmp */
    blk;       /* Get Address of Environment Data */
#asm
        XCHG    /* Save Address in DE */
        LDAX    DE /*Set up Jump to Old PC Address*/
        INX     DE
        MOV     L,A
        LDAX    DE
        INX     DE
        MOV     H,A
        SHLD    jmploc+1
        LDAX    DE      /* Get Old SP Value */
        INX     DE
        MOV     L,A
        LDAX    DE
        MOV     H,A
        PUSH    HL      /* And Save it on Stack */
        LXI     HL,4    /* Get Return Value */
        DAD     SP
        CALL    h.##
        XCHG            /* Save it in DE */
        POP     HL      /* Restore Old SP Value */
        SPHL
        XCHG            /*Return with Passed Value*/
jmploc: JMP     0
#endasm
}   /* longjmp */
```

Listing 3

```
/* longjump.h:  Environmental Block 6/22/88 */
struct  env_blk
    {
        int   *old_PC; /*Saved Pgrm Counter Value */
        int   *old_SP; /*Saved Stack Pointer Value*/
    };
```

=====

A Substitute for READ/DATA in MBASIC (CP/M)

by Hank Lotz

Consider, for a moment, a program using the READ/DATA feature:

```
10 DATA 65,66,67,68,69,70,71,72,73,74
20 FOR J=1 TO 10
30 READ I
40 PRINT I
50 NEXT J
60 RESTORE : GOTO 20
```

Nothing so remarkable here. When we run the program,

the DATA in line 10 is READ at line 30. Line 40 prints it out. Line 60 just illustrates a way you could reset (restore) the DATA pointer and loop the program, if you wanted to.

But now let's look a little deeper. When we originally enter line 10 into the computer, we have to type a "6" and then a "5" for 65 (the first DATA value). That ASCII representation uses up two bytes to describe the "65". And, in the same way, it needs two more bytes for each of the other data items in line 10. (Actually, counting all those commas, it eats up **three** bytes for most of the items.) But keep in mind that our 8-bit systems need only 1 byte to contain the value 65 once it's converted from the ASCII to the numerical ("binary") value. So I'm contrasting the 1 byte the computer **needs**, to the 2 bytes MBASIC **commandeers.**

If we put those same values into hexadecimal, line 10 would stretch out to:

```
10 DATA &H41,&H42,&H43,&H44,&H45,&H46,&H47,&H48,
                                        &H49,&H4A
```

The **computer** still needs only 1 byte to store the "binary" equivalents, but **we're** now eating up **4 bytes** to **express** each value, not counting commas! And did you know that **even if you compile** this MBASIC program, the object (COM) file **still** stores these values in the same 4-byte ASCII form? The compiler does **not** convert these to their numerical counterparts!

What all this means is that, when you have a large quantity of data, you'd save a lot of space (and probably some execution time) if you could store each data item as 1 "binary" byte instead of 4 (or the earlier 2) ASCII bytes. I concede that MBASIC always uses up **more** than 1 byte (even for a lowly 65 **in "binary"**) once it READs from the DATA area into a variable. But despite this we would still benefit **in the DATA storage area itself** by storing the data items as single "binary" bytes. And we **can indeed** do this!

Now, in the following assembly-language subroutine source you're "sure 'nuff" going to notice a bulky spread of ASCII values (in the DB lines near the end). **But**, after the assembly routine is assembled and loaded as an object file, those values will occupy only 1 byte each!

So, when you're using integers from 0 through 255, this method has an advantage over DATA statements. And of course, values within this range can also represent character-string data.

We now approach a setup/demo for this idea. In this case, making a COM file is actually the more **straightforward** way, so we'd learn little or nothing **new** by that route! Instead let's do an interpreted version; it's a trickier exercise and, as such, might teach us something extra along the way! We'll assemble the assembly-language subroutine, load it into memory, and then CALL it with an MBASIC program being run under the MBASIC interpreter!

To do this, we'll trot out Microsoft's M80 assembler and L80 loader. At last, **this** is the method I promised you back in **Staunch** #12, p.6, under the section headed "A By-product..."! (And **do** dig out that issue, because in the sentence **just above** that same heading, I made an error. "DEFINT CHAR" should read simply "DEFINT C". Please mark it in your copy.) But to return to the matter at hand, our MBASIC source, called RDEMO.BAS, will be:

```
100 REED=&HCC00 ' SETS UP ADDRESS OF ASM SUBROUTINE
110 I%=&HFF
120 PRINT "DECIMAL:  ";
130 FOR J=1 TO 10
140 CALL REED(I%) : PRINT I%;
150 NEXT J
160 I%=&HFF
170 PRINT : PRINT "ALPHA:    ";
180 FOR J=1 TO 10
190 CALL REED(I%) : PRINT CHR$(I%);
200 NEXT J
210 END
```

The variable I% is an integer variable; the % sign makes it so. This routine **will not work** unless you use an integer variable. (Another way to declare the "I" to be an integer is with the statement "DEFINT I", but be aware this makes **all variables** starting with "I" integers.)

The assembly source file, which you must call REED.MAC ("MAC" for the M80 assembler, later), is:

```
; REED.MAC (SUBROUTINE REED) by Hank Lotz
;   READS A BYTE FROM DATA BLOCK BELOW AND RETURNS
;   IT TO MAIN PROG IN AN INTEGER VARIABLE, SAY I%.
;   THE MAGNITUDE OF A RETURNED VALUE IS LIMITED TO
;   WHAT IS EXPRESSIBLE IN A SINGLE BYTE. THE MOST
;   SIGNIFICANT BYTE (MSB) OF I% IS ZEROED OUT IN
;   CASE IT WAS SET BY CALLING PROG. ONLY 1 VALUE
;   IS RETURNED PER CALL.
;
;   MBASIC CALL:  CALL REED(I%)
;
        ENTRY   REED
REED:   PUSH    PSW       ; Save accumulator, flags.
        PUSH    B
        PUSH    D
        MOV     A,M       ; See what was sent in I%.
        PUSH    H
        CPI     OFFH      ; Is it FF hex?
        JNZ     CONTIN    ; No, read next value.
        INX     H         ; Yes, but check hi byte.
        MOV     A,M       ; Look at MSB of CALL var
        CPI     0         ; Is it zero?
        JNZ     CONTIN    ; No, FF was not intended.
        LXI     H,RESTOR  ; Yes, restore pointer.
        SHLD    POINTR    ; Pointer restored.
CONTIN: LXI     B,POINTR  ; Put ADDRESS of
                          ; pointer-low-adr into BC
        LDAX    B         ; Put low pointer byte,
                          ; pointed to by BC, into A
        MOV     L,A       ; From there, put it into
                          ; L (low of HL).
        INX     B         ; Increment BC, point to
                          ; high pointer byte.
        LDAX    B         ; Put hi pointr byte in A
        MOV     H,A       ; From there, put it into
                          ; H (high of HL).
                          ; HL now has entire pointr
        MOV     A,M       ; Char pointed at by
                          ; pointer goes into A.
        INX     H         ; Move pointr to next char
        SHLD    POINTR    ; Store new pointr address
                          ; for next subroutine CALL
        POP     H         ; Get adr of CALL variable
        MOV     M,A       ; Transmit byte to var LSB
```

```
        PUSH    H       ; Resave HL so we can play
        INX     H       ; Point to MSB of CALL var
        MVI     M,0     ; Zero out hi byte of var,
                        ; as this routine returns
                        ; only 1-byte integers.
        POP     H
        POP     D
        POP     B
        POP     PSW
        RET
;
; START DATA BLOCK
;
RESTOR: DB      53H,55H,43H,43H,45H,53H,53H,21H,0DH
        DB      0AH
        DB      0FFH    ; Recognition byte (Reset)
POINTR: DW      RESTOR  ; Store currnt pointr value
        END
```

You can replace the data after the RESTOR: label with your own data -- and **as much of it** as you like. A "recognition byte", OFFH, comes last and is optional but recommended. When you type "RUN" in MBASIC everything is initialized, but the assembly subroutine's local data pointer is **not**. In other words, MBASIC doesn't restore our data pointer in the ASM subroutine the way it does for its own pointer in READ/DATA applications. You can use an FF hex byte to do this at any time, by setting I%=&HFF before the CALL; see lines 110 and 160 of RDEMO.BAS. When you do this, the **first** item in the data block is returned from the subroutine.

If your program doesn't reset the local pointer (i.e., doesn't have I%=&HFF), and you RUN the program multiple times, the pointer may eventually be left after the end of your formal data block, by a previous run. The next read would then be attempted past your last useful DB item. That's when the subroutine's built-in FF hex is encountered, and that FF **does** get returned! Your program can use it to test for end of data, and also thereby prevent using the FF itself as false data. The same FF hex will also serve as a pointer reset -- but only on the **following** CALL (and assuming your program doesn't reassign I%). The routine could be modified to do an automatic reset on the **same** CALL, and not return the FF. (Note that any value can be used in place of FF. Just go through and change the FF's to something you aren't using for anything else. But you must change them in REED.MAC as well as in RDEMO.BAS.)

Assemble the subroutine. For example:

B>A:M80 =REED

This gives REED.REL which you can now load with:

B>A:L80 REED/P:CC00/E

Here the hex CC00 needs no base-defining prefix because hex is the default base with the L80 loader's P: switch.

If you ever change that address, be sure to make corresponding changes in RDEMO.BAS (line 100) and in the following command line, which you should now type to invoke MBASIC:

B>A:MBASIC /M:&HCBFF

With MBASIC, **decimal** is the default base with the M: switch. Hence, the &H prefix is needed here to define the CBFF as hex. And according to the MBASIC manual, the number after the M: switch is the highest memory location **MBASIC will use**, so I chose CBFF because it is 1 less than CC00, where our assembly **subroutine starts occupying memory.**

By the way, I arbitrarily chose the CC00 as a pretty low starting address for the subroutine. But depending on your data block size, the size of your operating system, how much hardware memory you have, and what other memory-resident programs you may have installed, you might be able to set this higher -- or you may need to set it lower. But again, if you do change the CC00, make the associated changes mentioned a moment ago.

After MBASIC loads, type LOAD "RDEMO", and you're ready to RUN! Experiment and study the listings to get a better grip on what's happening.

If you want to make the COM file for this demo, the REED.REL file you already have will work, but remove or comment out line 100 in RDEMO.BAS before compiling it. And for that compile you should use the command:

B>A:BASCOM =RDEMO/O ("oh"--not zero)

When you link the relocatables, RDEMO.REL and REED.REL, omit the /P:CC00 switch this time. A good command is:

B>A:L80 RDEMO/N,RDEMO,REED,A:BASCOM/S/E

(For me, this results in about a 10K COM file.)

Depending on your MBASIC version, the relocatable library name and/or operation of switch(es) may differ from those in the above command lines.

=====

CONTACTS

(A Wanted/For Sale/Swap Column)

David A. Shaw (11059 Overrun Drive, Manassas, VA 22111, 703-368-8243 after 7 pm Eastern) "I guess this is a 'for sale' item, although I'm really getting depressed just thinking about it. Please allow me to explain.

"I've had my H8 since 1979 -- 11 years. It has, in the last few years, become somewhat unreliable. I've had a number of RAM chip failures. My H19 terminal was in the shop all last summer. The tractor on my Diablo 630 preinter recently broke. But I always managed to keep the machine going. People thought I was odd, and Ipve never denied it; I love that machine.

"Well, about two month ago (see how long it took to get the courage to write?) I 'smoked' the H8. The RUN lamp wouldnt' even light, and when I went about trying to track down the cause of the failure, I heard a loud 'snap!' and the power light went out.

"This problem is beyond me. And, in any case, I can no longer justify the cost and effort involved in keeping the old war horse running.

"I have some good equipment, some sort-of good equipment, and some questionable equipment. I would like to see any or all of it go to a good home where it might get some use. Here's the list:

- One H19 terminal, white screen, recently repaired by Heath (it got a new flyback transformer, whatever that is). It's got a "Super19" firmware upgrade in it. It's in real good shape.
- One Diablo 630 daisy wheel printer. The bidirectional form tractor is broken; I tried to get it fixed, but I can't seem to track Diablo down. (Did they go out of business?) [Diablo was

acquired by Xerox some years back. -Ed.] It also has a small and unimportant break on the paper-guide system on the top-back of the case It still prints good, though, and I have a number of new wheels and ribbons. [**Staunch** camera-ready copy is printed on a 630. -Ed.]

- o An original H17 with three Mitsubishi half-height, DS (200 Kbyte) drives. The drives are around six years old and running real well.
- o And, of course, the H8 itself, serial number 3815. It has a gold 50-pin Trionyx motherboard, original Heath 8080 CPU (very questionable condition since the 'smoke' job), H8-4 multiport serial I/O card, floppy disk controller card (hard sectored, modified to handle the dual-sided disks), extended configuration option (only installed to handle the dual-sided disks), two 16 Kbyte memory cards, and three 8 Kbyte memory cards (56 Kbyte static, total). There are also two cassette cards.
- o Original Heath manuals for all the above.

"I also have a lot of disks. I have unopened and partially-used boxes of Scotch 10 RH disks. I have a lot of disks with software on them; sofware written in assembler by me... [Dave sent his custom software to me at my request. I'll go through it early in the new year for distribution through **Staunch**. Dave **retained** the commercial software to accompany the system. -Ed.]

"And there's an EPROM burner, one that runs off a serial port. It can program 2708's, 2716's, and the like. I have software that can program EPROMs from .ABS files. (I used to do some contract programming on embedded controllers, using the H8 as a development system. It **has** been used.)...

"I'd take best-offer for the terminal, printer (with paraphernalia), disk drives (with blank disks), ... and the EPROM burner. The H8 can go to anyone who has the patience and expertise to determine what is usable and what is not; please pick up shipping..."

Mark Hunt "I'm looking for Microsoft's BASIC compiler under hard-sectored, 5" HDOS. Think you can help? Would need the documentation. Will obtain free & clear license from Microsoft before I use this program. (Purchased the CP/M version {soft-sector, 5"} - don't use CP/M, but it was all that I could find - still have it, new, unused. Interested?) If you can help - my name is Mark Hunt - I live at the PHS Hospital in Barrow, Alaska 99723 - My CompuServe number is 73770,2333. Thanks."

Dan Jerome (801 E. 132nd St, Burnsville, MN 55337) [I'm] "...preparing to sell some of my stuff. I have several shelves full of what may be called 'virgin' software and Heath Continuing Education volumes. Indeed, I have never opened some of the program manuals or disks. I thought I would send you a list and (1) give you first crack at them [and] (2) include a note in the next **Staunch**. I am asking only $30 for each book, plus $5 for U.P.S. shipping, and the programs include the master hard sector disk, and they are as follows:

BASIC Interpreter	Program	1 book	CP/M
Inventory Management	Program	1 book	CP/M
SuperCalc	Program	1 book	CP/M
SuperSort	Program	1 book	CP/M
WordMaster	Program	1 book	CP/M

Analog and Digital Meters	HEC	1 book
BASIC Programming	HEC	1 book
Digital Techniques	HEC	1 book
Electronic Circuits	HEC	1 book
Frequency Generation & Measurement	HEC	1 book
Oscilloscopes	HEC	1 book
Semiconductor Devices	HEC	2 books
Special Measuring Instruments	HEC	1 book

"If the customers need soft sector, I can convert to soft sector for free. In that case, I send them both hard and soft sector disks. It's a shame to waste those virgin hard sector disks."

Edward W. Davie (506 Sky Lane, Forest Grove, OR 97116, 503-357-3185) "I was referred to you by Paul Herman. I have a complete H8 system that is surplus. I would appreciate it if you could advertise it for me...

"The system consists of:

The H8 chassis with the original 2 MHz processor board.
A 4-port serial board.
64K memory.
The H17 disk drive controller.
An updated ROM.
The associated, H17, disk drive consists of 2 full-height drives (95K bytes, single side).
H19 monitor/keyboard.
All of the original documentation.

"I have a quantity of disks along with HDOS and CP/M operating systems. Some software including Big Eddy (a HUG editor) and dBASE II for CP/M. I am setting a price of $300.00 for everything above with the understanding that price is negotiable.
"In addition, I have an intelligent printer buffer, 64 Kbytes that I am asking $50.00 for. A 360K, DSDD floppy drive that was exchanged for a hard drive and never used. It ought to be worth $40.00. Finally a serial (A-B) data switch, that I will let go for $25.00."

Tony Durner (40 Westgate Dr., Edison, NJ 08820, 908-668-1047) "Sorry I am to say that I no longer use my H-89. Alas, I have been forced to an IBM clone because I need Autocad and other graphics software in my quest to make money as a teacher.

"I have therefore an H-89 with a single hard-sector disk drive, an installed fixed disk, and a separate double disk drive unit. These units are in top shape. My problem is that I cannot bear to drop them in the trash. I have tried to give them to many people, both computer and electronic types, with no luck.

"You are my last hope. Perhaps someone in your readership is willing to take my '89 and give it a good home. If you could put me in touch with such a person, I would love to give it to them free, no strings. I am even willing to deliver it to someone within fifty miles of Edison, NJ. Edison is about twenty miles from New York City...."

Dick Shotwell (546 Grandview Drive N., Twin Falls, ID 83301) "I am overloaded with '89 computers and components. We are moving and I have been instructed by you know who to reduce my inventory. So the following are available:

H89 with 64K and one hard sector drive on board. This computer has a card in it for the H/Z67 10Mb hard disk drive. The '67 has become inoperative so I will throw it in for next to nothing. I feel that the right tinkerer can resurrect this drive and have a real good system.

H89 with 64K and two 96-tpi soft sector, half height drives on board. This computer runs at 4 MHz and is in excellent condition and is currently in use.

H77 external drive cabinet with 2 hard sector drives. Fully operational, this drive system was used with the first '89 listed above. Excellent condition.

H77 external drive cabinet with 2 96-tpi drives. Fully operational. This was taken out of service when I installed the two 96-tpi half height drives in the 2nd '89 listed above.

Diablo 1640 daisy wheel printer. This is really a dandy, heavy duty printer which is in excellent condition and has a lot of life left in it.

"Drop me a line or give me a call and we can discuss prices. My daytime telephone number is (208) 733-7774 and you can call in the evening at (208) 733-7815. Due to the circumstances, I will make someone a very good deal and will throw in a lot of software."

Erven J. Buss (92-595 Palailai St., Makakilo, HI 96707) "...I have an H-89 with both hard sector and soft sector cards and drives that I wish to sell. Soft sector drives are 96 TPI. Also have an H-25 printer for sale..."

James H. Dummer (613 Wrightwood Terrace, Libertyville, IL 60048-3363, 708-362-3889) "After using a group of H89-90's with Anderson-Jacobson 830-832 printers for business until last year, I have had to upgrade to MSDOS with HP Laserjets. This leaves me with a surplus of three H89-90's (one with a soft-sectored controller), each with an internal hard sectored drive and there are 2 dual double-sided outboard double-density drives and 2 outboard dual single-sided drives. The computers all work, but some of the external drives need work.

"I also have 2 surplus Anderson-Jacobson 831 daisy wheel printers that need work and one Smith-Corona TP-1 unused daisy wheel printer.

"There are two unopened cartons containing a total of 10 10-packs of 5-1/4-inch, 10 sector, single-sided, double-density discs. In addition, there are a lot of parts, programs and manuals.

I would prefer for all of the items to be taken by the same party with a minimum of effort on my part. I do not expect any payment, but I also do not want to have to pay anything."

Pat Morrison (1003 Salem Dr., Las Vegas, NV 89107, 702-870-0131) "For Sale - Heath H-19-3 (H-19 converted to an H-88 computer), extra disk drive H-77, and 5-1/4 hard-sector disks. Best offer."

John S. Barford (Box 434, Hudson, NY 12534, 518-828-5994) "FOR SALE: H/Z89 with the following installed: 6 MHz speed up mod (Z80-B), 90K hard-sectored drive, 48-tpi 360K soft-sectored drive, 96tpi 720K soft-sectored drive, CDR 1 meg RAM drive with harddisk controller, Seagate 20 meg harddrive, MPI-88 dot matrix printer, Angel print buffer, mega software: CP/M, BASIC-80 compiler, BASIC-80 interpreter, MAGIC WAND, SUPERCALC, FORTRAN, COBOL, games, and much, much more. Make an offer."

Jack Wert (21 High Road, Levittown, PA 19056) "A recent change in my home computer hardware leaves me with several H-89's for which I would like to find a home - for a small monetary consideration, of course. As I have no way of packing any of them for shipment, my potential customer base will be limited to those within driving distance of Levittown, PA (northeastern Phila. suburb). My phone number is (215) 945-0397. Typical of long time users of a computer, I have accumulated enough goodies to allow me to supplement each machine with substantial software, documentation, etc.

"I am retired, and therefore available just about all of the time. Anyone interested, please give me a call and we can set up a visit."

Charles T. Huth (229 Melmore St., Tiffin, OH 44883, 419-448-007[sic]) "Wanted: Microflash M-89 Expansion Unit with H89-1, -3, -4, and -5 cards."

***** BE A STAUNCH RENEWER! *****

Renewals for 1991 are now being accepted!
Send $12 if you live in the U.S. or Canada.
Overseas, send $16 in U.S. funds, please.
Make checks payable to "Kirk L Thompson" or
to **The Staunch 8/89'er.**
Do it now before you forget!

THE STAUNCH 8/89'er, created by Hank Lotz, is a bimonthly newsletter on 8-bit H/Z computers. The editor is Kirk L. Thompson; P.O. Box 548; West Branch, IA 52358; home: 319-6437136. Subscriptions always start and end with the calendar year. Rate: $12.00/year. (Overseas, add $4.) Single copies: $2. Make checks payable to "Kirk L. Thompson". **Staunch** pays authors for their articles; write for an author's guide. It also accepts commercial ads for a modest fee; contact the editor. Neither this newsletter nor its editor is responsible for damages or losses resulting from use of any information presented herein. Info from **THE STAUNCH 8/89'er** may be reprinted only if this publication's name and address is included. Credit should also be given to authors and other sources of said material, if known. This publication is archived by the University of Iowa Libraries. CP/M is a registered trademark of Digital Research, Inc. REMark is a registered trademark of Heath/Zenith Users' Group. EOF