The Compass

International NorthStar Users Association

Volume V No. 1

INSUA News

RENEW!!!

Following INSUA custom, the first issue of the new volume of Compass is being sent to **all** INSUA members, those who have not renewed as well as those who have. If you have **not** renewed, kindly do so immediately, and take advantage of the **FREE DISK OFFER,** a flippy with Pavel Breder's DOSPOWER on the DOS side and an updated version of Ward Christenson's original CATALOGUE by Irvin M. Hoff on the CP/M side. Offer good until 1 August 1985.

DIMENSION and FAIRE NEWS

The **fifth** year of INSUA starts with interesting news. North Star has now actively joined the IBM-PC compatible world with the Dimension, which has been a long time a-comin', but now that it's here, it's turning out to be worth the wait for a lot of folks. Are you one of them? If you're a small businessman, or run a project or office with a staff of two or more, you should look into it. If you always use a single workstation, probably not, since the Dimension is designed to run up to twelve terminals simultaneously, and the entry price for a single terminal system would be above the price of any other single-user system.

The Dimension is good news and bad news for INSUA. We are running a major review article on the Dimension in this issue, and will publish any articles sent in to us. But should we announce our support of the Dimension in the same way as we support the Horizon and the Advantage? Should we try to support MS-DOS and IBM-PC software capable of running on the Dimension? We would be happy for advice on this question from the membership.

We got our hands on our first Dimension at the Computer Faire, which turned out nothing like our announcement in Volume IV, no. 6. Based on our experience at the '84 Faire, we had decided not to hire a booth (which had become priced out of the range of a mere users' group), nor to ask for a room for our annual meeting (we would rent a hotel room instead). About a month before the opening of the Faire, however, the management offered us and other users' groups a floor booth free, and also offered us a room for the annual meeting. These details were announced in a pre-Faire letter to INSUA members.

The Faire, which did not quite fill the Moscone Center in San Francisco, had more of the feel of the old fairs than those during the past two years. Users' groups were prominent once again, visitors to the Faire consisted largely of hobbyists and hackers, and the INSUA booth had more activity than we've been used to for some time. Whether the Faire management achieved this atmosphere by choice or by default, we congratulate them on their return to the excitement of personal computing!

Brillig Computers of San Francisco kindly loaned INSUA a Dimension for display purposes, and North Star sent Frank Maara, who gave an illustrated slide lecture on the Dimension. Thanks to the Faire management, we had a large room and an attendance of nearly fifty.

The Compass

The Compass is published every two months by INSUA, the Interational North Star Users Association, P. O. Box 2910, Fairfield, CA 94533.

Subscription Information:

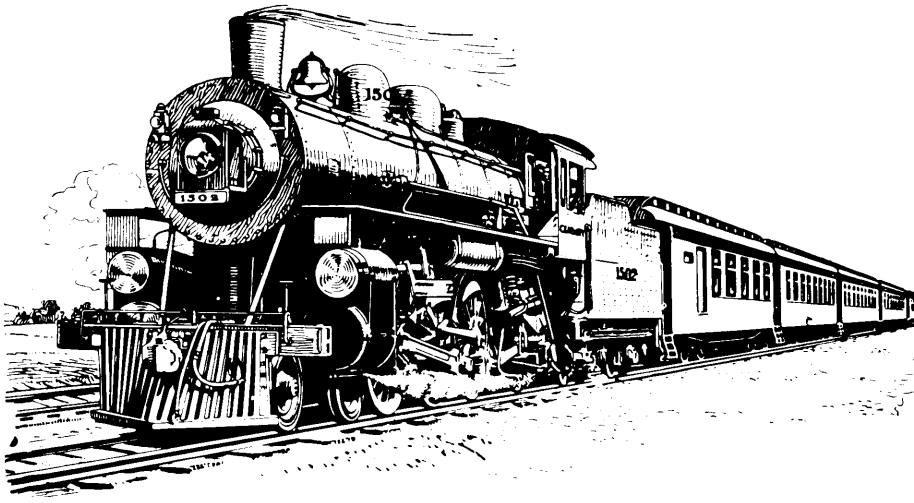
Subscription to The Compass is included in the $20.00 INSUA membership fee. Address subscription inquiries to INSUA, P. O. Box 2910, Fairfield, CA 94533.

DISCLAIMER

Programs printed in Compass and/or distributed through the INSUA disk library are offered to INSUA members in good faith. INSUA, however, is unable to guarantee the operation of any of these programs or to guarantee support. Users are advised to test the programs thoroughly for themselves in conditions under which they are to be used. Users who employ such programs in serious business or financial applications must do so at their own risk.

Facts or opinions published about manufacturers and dealers, and all opinions expressed in articles and letters, are the responsibility of the authors, and not of INSUA or the Editor of Compass. INSUA offers the right of reply to members and non-members alike.

Contents



Dimension: A Review

By Bob Cowart

North Star's Dimension system offers high performance, while breaking North Star tradition

(Bob Cowart, formerly an employee of North Star, is currently a free-lance writer. The Dimension used for this review was kindly loaned by North Star. --Ed.)

Leader of the Pack?

North Star computers has both gained and suffered from its 'rugged individualist' approach to high-tech research, development, and manufacturing. From the beginnings of the company, founders Chuck Grant and Mark Greenberg have made it clear that they intended to design computers and market them their way, deaf to the advice of many marketeers.

This attitude is, in part, what put North Star's truly advanced computer designs and innovations near the top of the relatively small heap in the late '70's, and which was later to steer their products out of the microcomputer mainstream and almost into oblivion.

In a fledgling industry, capturing a big piece of the pie often entails setting 'the standards' by which others are judged, and to which others must conform. North Star was well on its way to cutting its own slice when the microcomputer industry took off, leaving Grant and Greenberg (engineers by training) only guessing where their share (and the market) went.

It was, in fact, their dedication to, and unrelenting attachment to, the prospect of setting industry standards which almost sank the North Star ship. For example, the Horizon's inclusion of built in disk drives, internal serial and parallel ports, a "smart" mother-board, and their own disk operating system were terrific ideas when they were introduced. The North Star disk operating system and BASIC were in fact **the** standard for several years. With the correct marketing, and slight modifications, North Star DOS could have been the CP/M of today, and North Star could have outstripped Apple in sales. But continued emotional attachment to the then existing product line stagnated the company, and an attitude of isolationism set in. For example, North Star's proprietary accounting package, AccPak, was, at the insistence of North Star, rewritten by its authors (BSG) to run under a unique operating system rather than using CP/M for which it was originally designed.

Meanwhile Apple was strongly encouraging third-party development of software and hardware for the Apple II, an approach primarily responsible for their machine's incredible success.

Repeated requests for CP/M, soft sectored disks (a la Apple), 8 inch drives and true S-100 bus standards seemed to fall on deaf ears, only serving to buttress the appearance of North Star as an island unto itself, or at least one with few bridges.

Then came the Advantage. With its high resolution graphics capabilities, optional 8/16 processor and NorthNet system, the Advantage was a product superior to anything in its price range. But manufacturing and shipping delays, and its non-detachable keyboard, took their toll in a market known for its volatility. Meanwhile IBM began looming large with its foray into the micro market--the PC. With even Apple preparing to duke it out with big Blue, the nature of the microcomputer market began to change radically. Rugged individualism and American ingenuity were on the way out. Big advertising budgets were in.

To the chagrin of some, and applause of most, Greenberg and Grant have slowly matured in their marketing judgement, applying their substantial engineering talent to a more timely product, coming to terms with their realization that North Star is not, and probably will not ever be, as stellar a trend-setter as its trade-name implies.

The Great Beige Hope

In 1982, rumors of the development of a new machine, code named "C-5", were what kept many of us excited about North Star's future (i.e. the mere survival of the company, and for some of us our continued employment!). Not just another in the plethora of IBM PC clones, not just one more local area network to connect the clones to one another, and, thankfully, not another isolated North Star standard, the C5 promised to be a better idea which would sell, even in a world of business computing which was now dominated by IBM. Of course the machine was a long time coming and, as per industry tradition, was announced long before product availability. But in my view, the wait was worth it.

Some Background on the C5: LANS vs. MUMPS

As microcomputers become increasingly powerful, the use of older, dial-up time-sharing systems or in-house mainframes for simple, single-user tasks has become a relic of the past. The desktop microcomputer can often perform such tasks quicker and cheaper while providing the additional capabilities of spreadsheets, word-processing, and communications. This is well known by now.

But more recently, as increasingly capable microprocessors become available, and RAM prices plummet, micros have begun to usurp another domain of the mainframes, multi-user capability. For example, as complete offices switch over to micros, the question arises: How can numerous users share a common database, a printer, a tape-backup unit, or a hard disk? The kind of highly sophisticated and coordinated operations users have come to expect from expensive mainframes is slowly becoming available to smaller budgets.

One approach to this need for data and peripheral sharing among separate, stand-alone computers in the workplace is the Local Area Network (LAN). A LAN is simply the hardware and software necessary to interconnect anywhere from as few as two to as many as several thousand computers in an office, allowing exchange of data, as well as the sharing of expensive peripherals. LANS can cut operating time and expenses considerably, compared to the alternative of buying printers (and/or hard disks) for each user station. In an office with a large pre-existing base of PC's, a LAN makes good sense. And additional LAN capabilities such as electronic mail and simultaneous use of databases are icing on the cake.

However, studies reveal the average LAN to consist of only about four computers, not hundreds. And as it turns out, the purchase of four stand-alone PCs, with the necessary LAN hardware and cabling, can far exceed the outlay for a comparable, and often superior, multi-user-multi-processor system.

What is a MUMPS?

Until recently, multiuser systems (North Star's TSS-A and TSS-C are multi-user systems, as are most large mainframe computers) have relied on a technique called CPU-sharing or time-slicing to support several users on a single system. This arrangement apportions time on the microprocessor (CPU or Central Processing Unit) to each user on a rotating basis, giving the illusion of simultaneously running programs. But problems arise as more users log on to the system, causing all programs to 'slow down' since the CPU has more work to do.

Time-slicing multi-user system were the only logical choice until recently due to the high cost and unreliability of memory and CPU's. It was simply too expensive to give each user his own system. But now with prices so low, multiple CPU's have become possible. Adding a separate processor and RAM for each user, we have a MUMPS: Multi-User-Multi-Processor-System. For example, North Star's Horizon 8/16 TurboDOS system is a MUMPS, using either Z-80 or 8088 processor/RAM boards for each user.

The term 'multi-processing' can be a confusing one, since it can connote multi processORS or processES. Other types of multi-processing include parallel processing, array processing, and multi-tasking. Used here, the term means a number of CPU's servicing different users simultaneously.

For technical reasons, mostly relating to the speed at which data can be transferred between the computers and from the hard-disk to each user's RAM in a LAN, some MUMPS systems can outperform most LANS. In a MUMPS, since all the CPU's are contained in the same box, and connected to the same 'bus,' data can be transferred at very high speeds in 'parallel' fashion (8 bits at a time). LANs, on the other hand, transfer data 'serially' (one bit at a time), and require extensive data transfer protocols like 'collision detection' or 'token-passing,' to insure data integrity during transmission. This extra software overhead often acts to slow the overall system operation.

What is the Dimension?

North Star's new machine, officially dubbed the "Dimension," is IBM-PC-compatible, capable of running many popular off-the-shelf PC programs with little or no modification. But, in keeping with North Star tradition, there's a twist. Up to 12 people can use the system simultaneously, with programs executing considerably faster than either single IBM PC/XT's or any of the local area networks currently available for connecting stand-alone IBM PC's or IBM clones together.

Physical Design

Physically, the Dimension consists of one large box called the "Central Module", and up to 12 "workstations" (terminals) which are connected to the central module via thick, round cables of up to 1000 feet. The overall arrangement is very similar to the Horizon multi-user systems (TSS-A, TSS-C, and TurboDOS), falling into what is often called a 'star' system.

Like the Horizon TurboDOS system, each user on the system has his/her own microprocessor and RAM on a dedicated printed circuit board which plugs into an IBM-PC compatible connector slot inside the Central Module. Up to 12 user boards can be operative at one time, sharing up to two hard disks, and several printers. Each user board supports up to 512K RAM and contains an 8088-2 CPU running at 7.1 MHz.

Data traffic between user boards as well as disk and printer activity is handled by a main 'mother board' controlled by an Intel 80186 microprocessor running at 6 MHz, and up to 256K RAM. This RAM is used for extensive disk-caching techniques for the Dimension's operating system, speeding up hard disk activities. Intelligent disk access software using so called 'elevator algorithms' for stepping the hard disks heads further economize on time. Hard disks may be either 15 or 30 megabytes each, and an additional 45 meg streaming tape backup unit is available.

Work Stations

Each workstation consists of a beige colored keyboard (manufactured by Cherry) and green screen monitor, as well as a local serial port which can be used for connection to a modem, mouse, or printer. The keyboard has a pretty good feel to it, and is laid out in identical fashion to the IBM, except for a horizontal (but not large) Return key, the \ and shift keys on the left side being reversed, and the addition of num lock and caps lock LED indicators. The keys top are sculptured, and pop-up props for tilting the keyboard are also included.

The workstation monitor uses a 12-inch, long persistence green phosphor (P39), anti reflective CRT. The P39 phosphor eliminates any discernible 'flicker.' The monitor is a hi-resolution (18 MHz.) composite type, receiving its signal from the workstation board in the central module via a shielded cable. Text is displayed in the normal 80-character by 25-line IBM - PC format. Since the workstation board video driver hardware emulates the IBM PC's 'Graphics Adaptor' board, graphics programs like Lotus 1-2-3 can run without incident. But display of text is better than on a PC when using a color board and composite monochrome monitor, since the workstation board takes advantage of the monitor's horizontal scan 'interlace' capability, effectively doubling the clarity of text characters on screen. A new workstation, soon to be released, supports RGB and color video.

Peripheral Support

The Dimension has more printer connection options than any system I have

yet seen. On the back of the Central module are three connectors designed for up to three system printers - one parallel and two serial. Serial ports run as fast as 38.4 KBaud. In addition, each workstation has a 'local' serial port (addressed as COM1 in PC parlance) which can be used with a printer too. Under software control, each user may assign her or his output to anyone of these. What's more, four users, all assigned to different printers, may print simultaneously. All print jobs are sent to a 'spooler' (a temporary printer buffer on the hard disk), freeing up the workstation and its CPU for other tasks.

Another 'peripheral' feature, you might say, is the inclusion of electronic mail. Though not terribly sophisticated, it does allow editing, sending, and storage of mail between users.

According to North Star, there are plans for offering a sharable modem card which will plug into the Central Module's bus. Only one person could use the modem at a time, of course, but it beats buying a modem for each workstation.

Operating Systems

The current Dimension operating system is built around MS-DOS 2.1, which runs (with slight modification to closer resemble PC-DOS) in the workstations. The Central Module itself uses software created from several sources including MS-DOS and code written at North Star to coordinate the user boards, electronic mail, disk I/O, and printer spooling. The system emulates 3COM's LAN semaphore protocols for multi-user program operation. This means that programs written to run on the 3COM LAN (multi-user dBASE II for example) should run on the Dimension.

Dimension's operating system 8 signs on as "North Star DOS," but it is not the one WE know! Although pretty competent, it will soon be considered the bare bones version for the Dimension. To be released "mid year" for the Dimension is the Novell Corporation's "Netware". Netware is far and away the most sophisticated multi-user operating system available for microcomputers, and this will be a big plus if you are considering a Dimension.

Already available in over 13 versions and supporting many popular multi-user systems, including the Televideo PM, 3-COM, Quadnet 6 and Quadnet 9, and others, Netware is considered the leader in this area. Aside from speeding overall system operation considerably, it also supplies multiple levels of system security, advanced electronic mail, and the ability to run a wide variety of true multi-user programs and databases. It will also allow the Dimension to support the connection of external IBM PC's and XT's to the system as workstations. Eventually, using what has been dubbed 'Advanced Netware' by Novell, Dimensions may even be linked to other brands of existing networks as long as they are using a version of Netware.

Documentation and Installation

The manuals and instructions for the Dimension are (and some of you may be surprised at this one!) great. There's a technical manual with millions of schematics and specifications, a system manager's manual, and small users' manuals which may be kept by the workstations. A voluminous Applications Installation Notes Update describes how to install many of the popular IBM programs.

Installation of our four user system was relatively straightforward, with a few mishaps, but certainly simpler than some of the networks I have used. Expect about half a day to get things set up, above and beyond the time you need for stringing wire through walls, etc.

Performance

Having personally conducted tests on almost twenty local area PC networks for PC Magazine, and reviewed numerous portable and desktop PC clones for a variety of other publications, I was anxious to test-run the Dimension with an eye toward quantitative performance evaluations, using a benchmark test developed at Belmont Labs, a microcomputer testing facility in Belmont, CA.

This test was designed to make very heavy use of both the user station's CPU as well as server disk I/O (and

consequently the system's data path). Because of the amount of disk access necessary to perform the test it was a rare moment when the hard disk got a breather. Probably no "real-world" multi-user system or network would make such heavy demands on its resources as this program, so you can consider this a thorough challenge. In fact some of the LAN's tested at Belmont failed to execute this test on more than 3 workstations simultaneously.

For those of you familiar with some of the subtler details of programming, the following list shows the 13 steps the test's batch file executed. (IBM's Macro Assembler and Basic Compiler are used.)

1 - Assembles both sample files on the MASM disk

2 - Links them.

3 - Deletes the .OBJ files.

4 - Runs the program.

5 - Deletes the program.

6 - Compiles a version of PCTALK III.

7 - Links PCTALK III.

8 - Deletes both the .OBJ and .EXE file

9 - Assembles a file dumping program.

10 - Links it.

11 - Runs the program (dumping itself to the screen).

12 - Deletes it

13 - Gets the elapsed time and loops back to the beginning.

Since I administered this test incrementally, that is, starting with one workstation and then adding one at time, I was free to explore an important variable in the Dimension's performance--the amount of system degradation (overall slowdown) which would result from the addition of each extra station. This addresses the common question: 'How much slower is a four-user system than a two- or three-user one?' It appeared that only minimal system degradation occurs when adding users. (For a comparison of the Dimension to some other systems, all of which were tested in the same manner, see Fig. 1.)

In addition to these findings, a more recent report from Belmont labs, comparing various MUMPS's, showed the Dimension to outperform the Alloy and Corona and Televideo systems as well. As for a more real-world subjective evaluation, let me say this: this computer is FAST! Even the folks at Novell say it's the fastest implementation of Netware to date. It's also simple to use. You just sign on, enter your password, and you might as well be using an IBM XT, only faster. I was also pleased to see that it ran all my favorite programs: Crosstalk, Wordstar, Sidekick, Lotus 1-2-3, and Pacman--all without modification.

As for the details of DOS level usage, each user is assigned 'logical volumes' (C:, D:, E: etc) to use as his or her drives, just as on a single user system. These volumes may be private, public or shared. This way, two users can use the same data or programs without clobbering each other's files, or directories and files may be kept completely private.

Shortcummings (EE)

As you can gather, I don't have much negative to say about the Dimension! But there are a few things. The Dimension was not, nor was it intended to be, truly PC-Compatible. You cannot plug a PC-compatible memory, video, or modem board into the bus and expect it to work. It won't. The software isn't there to support it. Perhaps at a later date it will be, but not now. Also, though lots of programs seem to run fine 'off the shelf,' others may need modifications per North Star.

Secondly, there is only one floppy drive, making copying from floppy to floppy a problem. And if you want to use the drive, you have to issue a command to request it. If you forget to 'release' the drive using another command, nobody else can use it! Unfortunately, there is no easy way to solve these problems, as they are endemic to this type of system, and are

the price paid for its high performance.

Finally, keep in mind one central difference between MUMPS and LANS. If the central module or 'server' in a MUMPS fails for some reason, your entire office may grind to a standstill, awaiting repairs. Most LANS are based on fully functional microcomputers which can operate independently in a pinch.

Pricing

Now for the damage. A four user system will set you back about $11,600. But compare this to Corona's MUMPS at $14,345, Televideo's PM-16 at $18,769 (typically), or the Alloy PC/Slave/16 at $14,865! It's clear that you can't beat the Dimension. Of course, if you want to go bare bones, buying an XT clone and three PC clones might amount to about about $6500 if you really shop. Add a cheapie network for about $500 per station, and your bottom line could be around $9000. But it will run slowly and may be problem-prone. Buying one of the more popular LANS will run you significantly more, typically $8000 to $1000 per workstation plus cabling and software.

Conclusion

If North Star can keep up with IBM's price chopping, get Netware out soon, keep up a good list of applications which run on the Dimension, and market this box aggressively, they may survive the 'big shakeout.' In any case, if you need a fast, powerful alternative to a local area network for a fraction of the cost, this is it.

# # #

This chart shows the averaged times of many different LANs running the Belmont Labs 'development environment' benchmark test on four workstations simultaneously.

#	Network	Time in seconds
1	**North Star Dimension**	**195**
2	3COM 3 Server Network	310
3	Novell Netware /S	345
4	Quadnet 9	361
5	Davong using Novell Netware	367
6	Nestar Plan 3000	383
7	3Com Novell using an XT for file server	391
8	3Com Native using an AT for file server	400
9	Televideo PM-16	402
10	Corvus Omnishare using Novell Netware	415
11	Quadnet 6	438
12	Ungerman Bass Net-One	542
13	3COM using the 3COM OS and XT server	563
14	Davong using the Davong OS	632
15	Orchid PC Net	730
16	Ast PC Net	940
17	Corvus Omnishare using Corvus OS	1058

Alien Equipment

by
Warren Lambert

"I give up, how do you send alien-format files to and from a North Star Horizon using a wire and software." Our Alien gives a pedantically detailed account of how to transfer across disk formats. He sits on a back ward in Knoxville patiently waiting for Baudot to bring him a new disk controller and disk moving software. "Have faith and believe," he asks, "we'll do it with chips and PIPs

real soon now.

An important problem has been raised in recent issues of Compass by writers who need a convenient way to exchange N* files back and forth with other disk formats. In Knoxville, for example, there are only a handful of N* owners; at work, they use IBM PC's or PC contemptibles which offer a common disk format for passing texts around when collaborating.

Decent solutions

The two attractive solutions involve hardware or software. Software, in the form of advanced PIP or DISKCOPY programs like Xenofile, can copy files from one format to another. But current software handles only soft sectored formats, so N* owners are out of luck. Another hope is the smart disk controller hardware, such as the Morrow DMA (a Z80 microcomputer masquerading as a disk controller). The good news is that the DMA writes in a number of formats; the bad news is that the user has to rewrite the BIOS.

It is discouraging to see the lack of systems software cause good products to fail, even in the hands of an ace hardware hacker like Dr. Carnevale at SUNY (Compass Vol. IV, no. 5). If you spend $500 for a board and can't trust it after dozens of hours of effort, beware of tendencies to turn green, to grow eight feet tall, and to run around all night howling and tipping over mobile homes. "You wouldn't like me, Mr. Morrow, when I'm angry."

A workable solution for us wimps

For now I stick with a wire and Crosstalk. This unimpressive solution is practical and reliable; below I'll list each step for readers who haven't used a wire before. The process is not hard, but some of the details are tricky. It will work until Godot arrives. Like all methods I depend on, it takes no brains, effort, or machine language. Below, I'll list the exact steps needed to use CrossTalk and a wire, as things are sometimes awkward when you try to make the first connection.

CrossTalk

CrossTalk provides a shared serial protocol for two computers to communicate; the program is one the the popular classics, like WordStar, VisiCalc or dBase II, one of the first of its kind that actually worked. For clarity, I'll refer to two versions of Xtalk.com: Xtalk88 refers to Xtalk XVI version 3.4 for the IBM-PC (8088 CPU), and Xtalk80 refers to Xtalk 2.04s with I/O for the NorthStar Horizon (Z80 CPU). I use Xtalk88 often to make either computer act like an ASCII terminal for distant mainframes, such as IBM 4341 CMS or DECsystem-10 timesharing. Even now I can't use CP/M CrossTalk to interact with the IBM mainframe, since the IBM insists on a "break" signal, which is a 150-250 millisecond interruption in a status signal, rather than ASCII characters transmitted on pins 2 and 3 as God intended.

CrossTalk XVI could give me a break, so I can use IBM mainframe timesharing, but Xtalk80 had no break simulation. To use IBM timesharing with my N* I had to use Hawkeye Grafix COMMX, a program which had a good break but many other bugs on my N*. I like the COMMX people, and their program has gotten some good reviews, but COMMX was always freezing my N* at awkward moments. The problem might be related to the Horizon's strappable motherboard and serial port headers;

the serial ports on my N* may not act exactly like Hawkeye's. Hawkeye Grafix seemed like an honest hard working company who worked with me very responsibly. I wouldn't write them off forever.

Xtalk88 has numerous bells and whistles, like Xmodem file transfers. It can even emulate certain ASCII terminals, such as an ANSI standard VT100, the VT-52, or the ubiquitous TVI910/20. This emulation is so good that I once used an IBM-PC as a terminal on the North Star, running WordStar at 9600 baud. WordStar had been set up for an ANSI terminal, and the PC did all the cursor addressing, reverse video etc. correctly. A new version of Xtalk80 was recently announced (late in 1984) which would give a North Star many of the new features. Has anyone used the new Xtalk80 enough to review it?

An IBM Pussycat

My spayed housecat never cared for my Horizon--its case was too cool. But she was thrilled by the new IBM-PC! It had 768K of ram and a fan slower than a seasick flabellitist. The PC was always hot, the cat napped on it, yawning and stretching in the pleasant warmth. When the $695 Tecmar VDO board died after 3 weeks, the paper labels looked prematurely aged. The technician said, "The heat musta' gotten to it."

To test my hot new "16 bit" computer (with 8 data bits), I ran Eratosthenes' Sieve in BASIC and Pascal (Turbo and MT+). The PC stretched, yawned, and took nearly 2000 seconds in BASICA (against about 1400 for N* BASIC); in the two native code Pascals, the PC took 16 seconds (24 seconds for the Horizon). While the "16-bit" performance of the new computer wasn't really so hot, in terms of compatibility, I was getting warmer.

They had PC's at work, and the research computing center I used supported PC's, not N*'s; I needed a way to transfer my WordStar files from machine to machine conveniently. The following method worked:

On your mark

1. I bought some 25-wire flat ribbon cable at a local Radio Shack, along with some of their pitiful blue plastic solderless DB25's. (Be gentle, or you may rip the serial port off the back of your computer.) These plugs can be pressed on the ribbon cable; instructions for a similar plug appear in the N* kit assembly manual for the 34-wire disk drive cable. Be sure that the plug is exactly perpendicular to the ribbon cable. I wired the header of the second serial port on the Horizon to deliver a full set of 25 signals, rather than "simplified" RS-232; I can still use a simplified 3-wire (2,3,7: no handshake) or 4-wire (2,3,7,20: simple printer-ready handshake) cables with printers. I don't know whether Xtalk works when the serial port header is in the simpler of the two configurations given by N*.

2. Make sure the Horizon second serial port is configured as a modem, not a terminal (See N* manuals listed below: "Using the Serial I/O Interfaces"). Modem configuration is the normal one. RS-232 communication assumes two kinds of devices: terminals and modems, distinguished by whether they talk or listen on pins 2&3 of the DB25 plug. The PC is normally a terminal and the Horizon is a modem, so they can communicate perfectly on a simple (straight) 25-wire cable.

Get Set

1. Purchase two copies of Crosstalk: a CP/M version for the N* and a MSDOS version for the PC. Prices at Cutthroat Telephone Sales (one call does all) range down to $95 each.

2. Plug the IBM PC's COM1: into the Horizon's second serial port using the 25-wire cable.

3. Put a program disk with CP/M Xtalk80 in A: on the Horizon, and Xtalk88 in A: on the PC. Put text disks in B:. Be sure to put the correct text disk in the N*, as you can't change without aborting and restarting Xtalk. If you try it, CP/M will kindly say "BDOS ERROR ON B:" if you try to write on B:.

4. Power up the computers.

5. Execute CrossTalk on the PC:

A>XTALK NSTAR.XTK

then log in disk B: using CrossTalk:

"Esc DRIVE B:"

Only Xtalk88 has a drive command. On the N*, first logging in B: will prevent Xtalk80 from using the help files on A:, as it is too stupid to look for them on A: when B: is the default drive. Xtalk88 remembers the help files are on A:.

6. On the N*, log in disk B: and execute Xtalk:

B>**A:XTALK A:IBM**

The parameter "A:IBM" means there is a CrossTalk configuration file on A:. A configuration file contains the technical details for communication.

7. Make sure the two computers agree in their baud rates. I use 1200 baud because my printer can't handle any more, and I'm too lazy to rewire my baud rate header switch for 1200-9600.

8. Set the Horizon's mode answer, duplex half, BLksize 1.

9. Using Xtalk88, set the IBM-PC for SPeed 1200 (or whatever) baud, 8 bits, no parity, 2 stop bits, serial port 1, mode call, pm 1, duplex full.

PM 1 tells Xtalk88 that the other computer uses CP/M rather than MSDOS.

To transfer slightly faster, set the blocksize of both machines to its maximum value of 16 rather than the default of 1. Note that the Xtalk88 does this with the command "BK 16" where CP/M Xtalk80 does it with "BL 16". Sixteen-block transfers are only slightly faster, so using the default=1 256 byte block is the coward's way out. The default blocksize also lets you "see" problems and deadlocks more quickly; at 1200 baud it takes about 34 seconds for a 4K block to be transmitted and checked. So use the default blocksize of 1 until transfers become routine.

10. The XTALK command "LIst" causes each computer to display the technical options that Crosstalk is using. On the PC, the "home" key is also needed to display other options, so after setting options, check them on each computer's screen.

11. Test the protocol by typing on the keyboard of each computer. If the other displays the letters correctly, then your baud rate, stop bits, and parity are correct.

12. Save the options in a CrossTalk option file, e.g. on the N* tell CrossTalk:

"Esc SA A:ibm"

or on the IBM say

"Esc SA A:Nstar.xtk"

The "Esc" is a character that causes Xtalk to go to command mode and interpret what follows as a CrossTalk command rather than text to be transmitted. (Personally, I use ^E because the local DEC-10 mainframe hates people who can't send an "Esc" as text.)

Go

<u>To transfer files from N* to the PC</u>

1. On the N* give Xtalk80 the command

"Esc DIR"

to see the filenames.

2. Make sure the N* is in terminal mode rather than the command mode.

3. You must type all transfer commands only on the computer in the CALL mode, in this case the PC.

(Avoid this: Telling the answering computer (the N*) "XMIT B:ALIEN???.*" won't work; the text appears on the PC's screen, zero blocks transfer while an ever increasing number of errors accumulate. To stop this protocol deadlock, I had to reset the N*. The calling computer must lead and the answering computer must follow; either computer could take the leading role.)

4. To transfer my files to the PC, I sat at the PC and gave the command:

"RQ B:ALIEN???.*"

The PC then requests that the N* transfer all those files to the PC. A few seconds pause is normal at this point, just enough time to think "Protocol deadlock. Two locked computers. Heck. Darn. Golly gee." Then Xtalk displays scorecards, and you see it's working.

5. Watch the block and error count to make sure all is well.

6. The computers will tell you when they're done.

To transfer files from the PC to the N*

1. Make sure the N* is in the terminal mode.

2. Look at the PC directory to see what to send:

"Esc DIR"

3. Command the PC to send the files you want:

"Esc XMit B:BUNKBLA?.*"

4. Watch the block and error count to make sure all is well.

5. Go to the kitchen and open the refrigerator while the machines do the work.

DISCUSSION

WordStar files with their marked parity bits transfer fine with Xtalk protocol-checked transfers. You can write texts in WordStar using MSDOS or CP/M and transfer the texts by wire to other computers. The transfers are error free, and the text files are perfect. My wife is doing her homework on the N* at the moment, so I'm editing this copy on blue&yellow RGB WordStar on a PC, confident I can wire it back to the N* with no hassle. It was very pleasant to take my lecture handouts written in 1981 on the Horizon and transfer them all in one fell swoop to a PC disk, update them slightly, and then print them on the PC clone at school. So I'm not dependent on a single unique computer; if one machine blows up, I can go to another one and still get my work done on time.

Since Xtalk accepts ambiguous file names, such as ALIEN???.* or *.*, moving a whole disk takes over an hour at 1200 baud, but it is no trouble; just give the command "RQ B:*.*" and go relax while the machines do the work. Be aware, though, that utility programs will be transferred in a *.* transfer. If you accidentally execute one (like XDIR.COM), the alien machine code will probably freeze your computer. So it goes.

If you do a lot of full disk transfers, 9600 baud would be necessary.

I found the procedures listed above to work, and wire transfers ought to work for almost any computer on earth, but the details may differ for computers other than an IBM-PC with Crosstalk. Call me or send me these items so that we can all help each other with the tricky business of transferring data to different computers: (615) 584-1561 X7724, W. L., Lakeshore MHI, 5908 Lyons View Drive, Knoxville TN 37919.

Once you get wire transfers working, it is a practical workaday technique that is sturdy and reliable. Before I sent this note on a N* disk to our hard working editor, Alan Nelson, I copied it five times: from N* to PC-floppy (by wire), from PC-floppy to PC-XT fixed disk (by disk), from the PC-XT to Chameleon floppy (disk), from Chameleon floppy to PC floppy (disk), and from PC floppy to N* (wire), each time making a few minor changes with a different version of WordStar.

CrossTalk wire transfers may not be glamorous, but they'll get the job done until Baudot arrives with cross format chips and alien format PIPs. I believe Godot will arrive real soon now, so I'll just sit here and wait for him while I admire my brand new bridge some kid sold me a few minutes ago.

Chain letter:
1.This line typed on a N* Horizon.
2.This line typed on an IBM-PC.
3.This line typed on an IBM-PCXT.
4.This line typed on a Chameleon.
5.This line typed on an IBM-PC.
6.This line typed on a N* Horizon.

REFERENCES

1. North Star Computers, Inc. North Star Horizon Computer System Rev 1, 1977 (North Star kit) Port:p. 68; similar cable p39 step E7.

2. North Star Computers, Inc. North Star Horizon Computer System Double Density, HRZ-D-DOC undated, page 72; similar cable p39 step E7.

# # #

(Editor's note: Transfer between machines can be accomplished with less sophistication but also less money using the public domain MODEM7 and a three-wire cable (pins 2, 3, 7 with 2 & 3 reversed. MODEM7 may be purchased through the INSUA disk library, and is available through other users' groups for most other micros, including the IBM PC.)

A Graphics BASIC for the Advantage (CP/M)

Bruce De Rienze
24 Whip-o-will Lane
Milford, Massachusetts 01757

Graphics on the Advantage?

As of last fall, I had been using my NorthStar Advantage for about a year with virtually no graphics applications. The only things I could do were the pie charts and bar graphs in the demonstration BUSIGRAPH program that came with the Advantage.

Yes, I had NorthStar Graphics CP/M, but my attempts at using it were not very productive. The "graphics" is just some assembly language routines that do a few basic drawing functions. The functions were clear enough, circles, rectangles, polygons, etcetera, but the documentation was the pits. Some real technical screw-ups didn't make it any easier, either. For instance, some of the routines required putting parameters in the Z-80 IX and IY 16-bit address registers. But the standard CP/M assembler (ASM.COM) supplied by NorthStar can't do that, since it is designed for the 8080!

I am certainly not an assembly language programmer. However, I have muddled through some Z-80 assembly language programming projects, mostly for my old TRS-80 Model I. Anyway, I don't consider myself a total dummy about Z-80 assembly programming. But at the rate that I was going, it would have been ten years before I had even a crude program running. I really wanted a high level programming language to get to the Advantage graphics, but I also wanted to use CP/M, and NorthStar GBASIC does not work with CP/M.

Graphics CPM APCBASIC

Then last October I saw an advertisement for something called Graphics CPM APCBASIC in Compass, vol. IV, no. 3. For $299, this program promised high level BASIC language access to the Advantage 240 by 640 graphics. I was excited, but cautious. What if it was just a way to call the same low-level contorted routines that NorthStar had put in their CP/M? The APC in APCBASIC stands for American Planning Corporation, 4600 Duke Street, suite 425, Alexandria, Virginia, (703) 751-2574. I called them up and talked to someone named Louis about their product.

Louis was very nice, although he did not know the exact graphics capabilities. He assured me that it was not just a way to call the NorthStar primitives. I found out that I could buy just the manual for $50, plus an $8 shipping charge. So I took

a chance and ordered it. It arrived in less than a week.

Seeing the command list for myself, I was convinced that this would be just the thing to get graphics applications going on my system. I ordered the software, for an additional $250, since APC gives you credit toward the software purchase if you buy the manual. It came just as quickly. It also comes with a 30-day return guarantee if you don't like it. That is pretty enlightened in this era of "pig-in-a-poke" software.

There was a confusing situation about appending the NorthStar graphics manager software to APCBASIC. NorthStar Graphics CP/M does not contain the graphic routines. After you create a .COM file you run the program GMGRADD.COM which appends the routines to your existing .COM file. The APCBASIC instructions say to run GMGRADD on the APCBASIC.COM file. DON'T DO THIS. The graphics routines have ALREADY been added. You get an error if you try to add them again. Louis at APC straightened me out on this. Luckily, I NEVER do anything to the distribution copy.

The product consists of a typewritten manual on the APCBASIC language (150 pages), an addendum on graphics commands (100 pages), a loose-leaf binder for the manual, and a single disc of software. The disc has two flavors of APCBASIC in four numerical precisions. Regular APCBASIC has an interactive command interface, as you would expect. The other flavor, called APCRUN, has no interactive interface, but it is supposed to execute programs slightly faster. By naming the file RUN.COM, you can use an APCBASIC program from the CP/M command level by entering RUN FILENAME.

APCBASIC and APCRUN are supplied as separate .COM files on the disc for 8, 10, 12, or 14 digit precision. I just use the 8 digit version, since I rarely do anything requiring hairy calculations. APCBASIC.COM takes about 28K on the disc. APCRUN is 24K.

In addition to the actual interpreter, you get some utility programs. CONFIG is an APCBASIC program that sets some default parameters in your copy of APCBASIC. You can run it anytime and change things around. It lets you set the floating point board address, high memory address, patch in custom I/O drivers, disable CONTROL-C, and set end-of-file markers, among other things.

There is another APCBASIC program called NSB2ZBA which converts NorthStar Basic files to APCBASIC. I have not used it. It seems to convert certain NorthStar GBASIC graphics commands that are slightly different in APCBASIC. The instructions say it does "internal token mapping" and syntax alterations to certain system dependent operations.

The CRUNCH program will remove all comments and spaces from an APCBASIC program. This is reputed to reduce file and memory usage by 20 to 60 percent. It then makes the programs load faster and allows execution of programs that would normally be too large for memory. It preserves the initial program. I have not used this either, since I generally don't have performance-critical applications.

There is also a library of twenty-nine "handy" APCBASIC functions supplied. These are not particularly well documented; you have to read the comments in the code. The most useful routines would probably be the matrix inversion and multiplication, and the sorting. The others seem like it would take as much time to figure out how to use them as it would to "do your own thing."

A cross reference utility, called ZBIG.COM is provided for listing all of the references to variables, line numbers, and functions, in a program. That's useful.

Evaluation

I am generally pleased with APCBASIC. I think it has allowed me to really use the graphics capability of my Advantage. I have not done anything spectacular (yet), but this language at least allows me to get things on the screen without all the headaches and long debug time of assembly language programming.

A feature that APC didn't even mention is the CONTROL-T function. If you have an EPSON graphics compatible printer (like the FX-80) and you have an image on the screen, hitting CONTROL-T will print it on the printer. That's a heck of a great feature! I think this is part of the NorthStar Graphics Manager functions that get appended to the APCBASIC image.

I think you can sort of get the feel of a language by looking at a summary of the

legal commands and statements. A list of them is shown in Table A. Most of these have meanings similar to other BASICs or graphics interfaces that you may have seen. The APCBASIC manual implies that the commands are very similar to NorthStar GBASIC. These commands are for version 3.1.16. Some of the syntax and constructs seem cumbersome to me, but I am used to MicroSoft BASIC.

Features

Some noteworthy features are in APCBASIC. You can list and edit lines with a "search string" match. That's nice for finding all of the places to change a reference. There is a MERGE function that lets you load in another program and append it to the one you are working on. That makes standard subroutines easier to manage. The TRACE command is pretty sophisticated. You can turn it on and off based on variable values and send the output to a printer. There is a single step mode too.

File access seems quite fancy to me. There is byte-by-byte access to sequential or random access files. There are bit manipulation operations on string variables. Renumbering can be performed on a selected line range of your program. Also, there is a way to "list" a program to a file, thus storing it in ASCII, non-tokenized form. Then you can edit it with WordStar, if you want. There is an ENTER command which will take an ASCII program file and put it back in APCBASIC internal tokenized form.

APCBASIC is nice to you about saving programs. If you attempt to exit and have not saved your program it reminds you of that. It also asks for verification if you try to save a program with the same name as an existing file. If you don't give a filename with the SAVE command, it assumes the same file name that you used to load the program initially, if you did that.

One big bugaboo of BASIC programs is solved in APCBASIC. All subroutines and multi-line defined functions have their own local variables. You can make them global if you want. Also, you can preserve them on LINKing (CHAINing) to another program on disc.

Recommended improvements

There are two big and obvious areas for improvement in APCBASIC. The first one is the limitation of variable names to a single letter or letter followed by a number. This makes programs very hard to read and understand, since it is hard to convey meaning in the variable names. Good commenting is a must. The other problem is the line editor. It has a very cumbersome interface of special control character functions. I always get fouled up and end up typing in the line from scratch.

APCBASIC string functions seem awkward to me. I like to be able to take the right two characters off a string with a RIGHT$(A$,2) statement. APCBASIC assumes that you really want to take off those characters in order to compare them with something. So it gives you a MATCH function. Oh well. There are no time-oriented functions in APCBASIC, like a WAIT command or a timed INPUT. I have used these in Nevada BASIC.

More features

You can see from the command set that the graphics is fairly simple in APCBASIC. I was surprised to see the obscure PIESLICE and CHORDSLICE commands. The VIEWPORT command is a nifty way to put graphics in a small part of the screen. This allows you to combine separate graphics programs into larger ones, each working in its own part of the screen. You can't use text, though, since it does not get reduced in size.

I particularly like HYBRID mode, which leaves four lines at the bottom of the screen as regular scrolled text, while the remainder is graphics. This is really an Advantage hardware function, I think. It is nice for interactively typing in commands, and also for running in trace mode without scrolling all of your graphics off the screen.

Benchmark tests

On the subject of performance, I made up the following simple benchmarks for those who want to compare its speed. Benchmark 1 is just a 1000 iteration loop

with some calculations inside. It took 80 seconds with APCBASIC. Benchmark 2 lights 1000 pixels on the screen one by one. It took 10 seconds. Benchmark 3 draws 50 rectangles and 50 circles about the size of a quarter. Benchmark 3 took 8 seconds (the circles took most of the time).

BENCHMARK 1

```
100 FOR I=1 TO 1000
110  X=X+I
120  Y=23*I
130  Z=SQRT(5)
140  W=EXP(3)
150  M=SIN(7)
160 NEXT I
```

BENCHMARK 2

```
100 GRAPHICS
110 FOR J=1 TO 10
120  FOR I=1 TO 100
130   POINT I,J
140  NEXT I
150 NEXT J
```

BENCHMARK 3

```
100 GRAPHICS
110 FOR I=1 TO 100 BY 2
120  GMOVE I,30
130  CIRCLE 10
140 NEXT I
150 FOR I=1 TO 100 BY 2
160  GMOVE I,40
170  RECTANGLE I+10,50
180 NEXT I
```

Problems

The user interface to fill patterns is really hard to understand. Instead of a set of meaningful codes for fill patterns, APCBASIC uses the same bit-pattern orientation that the NorthStar system assembly routines have. There are basically 16 patterns that can be used to fill solid figures. Depending on whether the perimeter is solid, absent, part of the pattern, or alternately on and off, these get multiplied to 64 patterns. I still don't understand the use of the last perimeter. The patterns are numbered 0 to 63 as shown in Figure 1. I always have to look them up on this chart to get the right one.

I was disappointed that graphics text cannot be printed in any size other than the normal character size. Also, it cannot be rotated. NorthStar has a character set re-definition routine in its assembly-level interface, but there is nothing like this available at the APCBASIC level. You only get what is in the character ROM. However, you can place the text anywhere on the screen. That is, characters do not have to fall in the 24 line by 80 character standard character cells.

The promotional brochures on APCBASIC say that animation is possible with APCBASIC graphics commands. I don't think so. I am not a graphics whiz, but things just don't go fast enough for that. On the other hand, I have seen the speed of graphics using the direct system calls in NorthStar's CP/M, and they are not noticeably faster than APCBASIC.

There is one bug that I have found in APCBASIC. That is, string values are not concatenated unless the string is initialized before it is referenced in the concatenation statement. For example, in the following program Y$ prints as null (blank):

```
100 X$="E"
110 Y$=Y$+X$
120 PRINT Y$
```

However, in the following program Y$ prints as "E":

```
100 X$="E"
105 Y$=""
110 Y$=Y$+X$
120 PRINT Y$
```

Documentation

The documentation is good. It is not flashy, but the information you need is there. I would prefer some more examples in the BASIC part. There are better examples in illustrating the graphics commands. A notable feature is the 16-page index in the back. This is accurate, even though the manual has obviously been revised five or more times as new features were added. The index does not include graphics functions.

# # #

APCBASIC COMMAND SUMMARY
(per V3.1.16 of software, V2.4 of manual)

NOTE: Items in square brackets are optional

```
BIT (<string variable>,<bit address> [:<bit width>)=<expression>
CALL <address exprn>,<data register exprn$> [,<result register varbl$>]
CHANGE <line# range>,<search string>,<replacement string>
CLOSE [#<file number>]
CONT
CREATE <"new file name">, [<file size>] ,[<file type>]
DATA <data1> [,<data2>] [,<dataN>]
DEF <one-line function name> [(parameter list)]=<expression>
DEF <multi-line function name> [(<argument list>)]
DELETE [<line number range>]
DESTROY <"existing file name">
DIM <numeric array name> (<size1> [,<size2>] [,<sizeN>])
DIM <string array name> (<size1> [,<sizeN>] [,<length of element>])
DIR [#<device number>] [,<drive number>] [,"<file type>"]
DOS
EDIT [<starting line#>] [,<search string>]
EDIT <string exprn>
END
ENTER #<file number>
ENTER [<starting line#>] [,<stepsize>]
ERRSET #<numeric error type exprn>
ERRSET [<line# trap>] [,<error line varbl>] [,<error type varbl>]
EXAM <starting address>,<variable list>
EXIT [<line#>]
FILL <starting address>,<data list>
FNEND
FOR <index varbl>=<range1>,<range2> [,<rangeP>,<rangeQ>] BY <stepsize>
FREE
GOSUB <line#>
GOTO <line#>
IF <logical exprn> THEN <statement1> [ELSE <statement2>]
INPUT[1] [#<device number>] [<prompt1>],<varbl1> [,[<prompt2>],<varbl2>]
LINK <"program name"> [,<common varbl1>] [,<common varbl2>]
LIST [#<device>] [,<start line#>] [,<end line#>] [,<search string>]
LOAD <program file name>
LOCAL <varbl1>,<varbl2>,<varblN>
MERGE <program file name>
NEXT <index variable>
NOMARK <logical expression>
ON <exprn> GOSUB <line#> [,<line#>] ...
ON <exprn> GOTO <line#> [,<line#>] ...
ON <exprn> RESTORE <line#> [,<line#>] ...
OPEN #<file number> [%<file type>],<file name> [,<size varbl]
OUT <port number>,<8-bit data value>
PARAM(<exprn>)=<exprn>
PRINT [#<device number>] [,<data list>]
READ [#<device number>],<data variable list>
REM <descriptive text>
REN [<new start#>] [,<stepsize>] [,<old start#>] [,<old end#>]
```

```
REN$ <new start#> [,<old start#>] [,<old end#>]
RENAME <old file name>,<new file name>
RESTORE <varbl> [,<varbl>]
RESTORE <line#>
RETURN [<exprn>]
RUN [<line#>]
SAVE <program file name> [<file size>]
SIZE
STOP
SWAP <varbl1>,<varbl2> [,<varblP>,<varblQ>]
SWAPDEF <varbl1>,<varbl2> [,<varblP>,<varblQ>]
TRACE RET
TRACE [#<device>] [,IF <logical exprn>]
TRACE [#<device>] [,<line#>]
WHILE <logical exprn>
WRITE #<file number>,<data exprn list> [,NOMARK]
<numeric variable>=<numeric expression>
<string variable>=<string expression>
```

GRAPHICS STATEMENTS

```
ARC <x-radius>,<y-radius>,<start-angle>,<end-angle> [,<pattern>]
AXIS <x-tic>,<y-tic>,<x-zero>,<y-zero>
CHORDSLICE <x-radius>,<y-radius>,<start-angle>,<end-angle> [,<fill pattern>]
CIRCLE <radius> [,<fill pattern>]
CLEAR
DRAW <x>,<y> [,<pattern>]
ELLIPSE <x-radius>,<y-radius> [,<fill pattern>
GIN <x-variable>,<y-variable>
GINIT
GOFF
GON
GLINE <x1>,<y1>,<x2>,<y2> [,<pattern>]
GMOVE <x>,<y>
GPRINT <variable1> [,<variable2>] ...
GRAPHICS
HYBRID
MDRAW <x-point matrix>,<y-point matrix>,<number of lines> [,<pattern>]
MRDRAW <dx-point matrix>,<dy-point matrix>,<number of lines> [,<pattern>]
PIESLICE <x-radius>,<y-radius>,<start-angle>,<end-angle> [,<fill pattern>]
POINT <x>,<y> [,<pattern>]
POLYGON <x-point matrix>,<y-point matrix>,<elements> [,<fill pattern>]
RDRAW <dx>,<dy> [,<pattern>]
RECTANGLE <x>,<y> [,<fill pattern>]
RLINE <dx1>,<dx2>,<dy1>,<dy2> [,<pattern>]
RMOVE <dx>,<dy>
ROTATE <angle>
RPOINT <dx>,<dy> [,<pattern>]
SCALE <x-units>,<y-units>
TEXT
VIEWPORT <x-min>,<x-max>,<y-min>,<y-max>
WINDOW <x-min>,<x-max>,<y-min>,<y-max>
```

ARITHMETIC FUNCTIONS

INT (x)
CEIL (x)
TRUNC (x)
MOD (x,<mod>)
FRAC (x)
ROUND (x)
ROUND (x,<digits>)
ABS (x)
SGN (x)
SGN (x,y)
MIN (x,y,...)
MAX (x,y,...)
INDEX
RND (x)

SQRT (x)
LOG (x)
LN (x)
EXP (x)
PI
SIN (radians)
ASIN (x)
COS (radians)
ACOS (x)
TAN (radians)
ATAN (x)
POLY (x,<coeff-array>,<degree>)

STRING FUNCTIONS

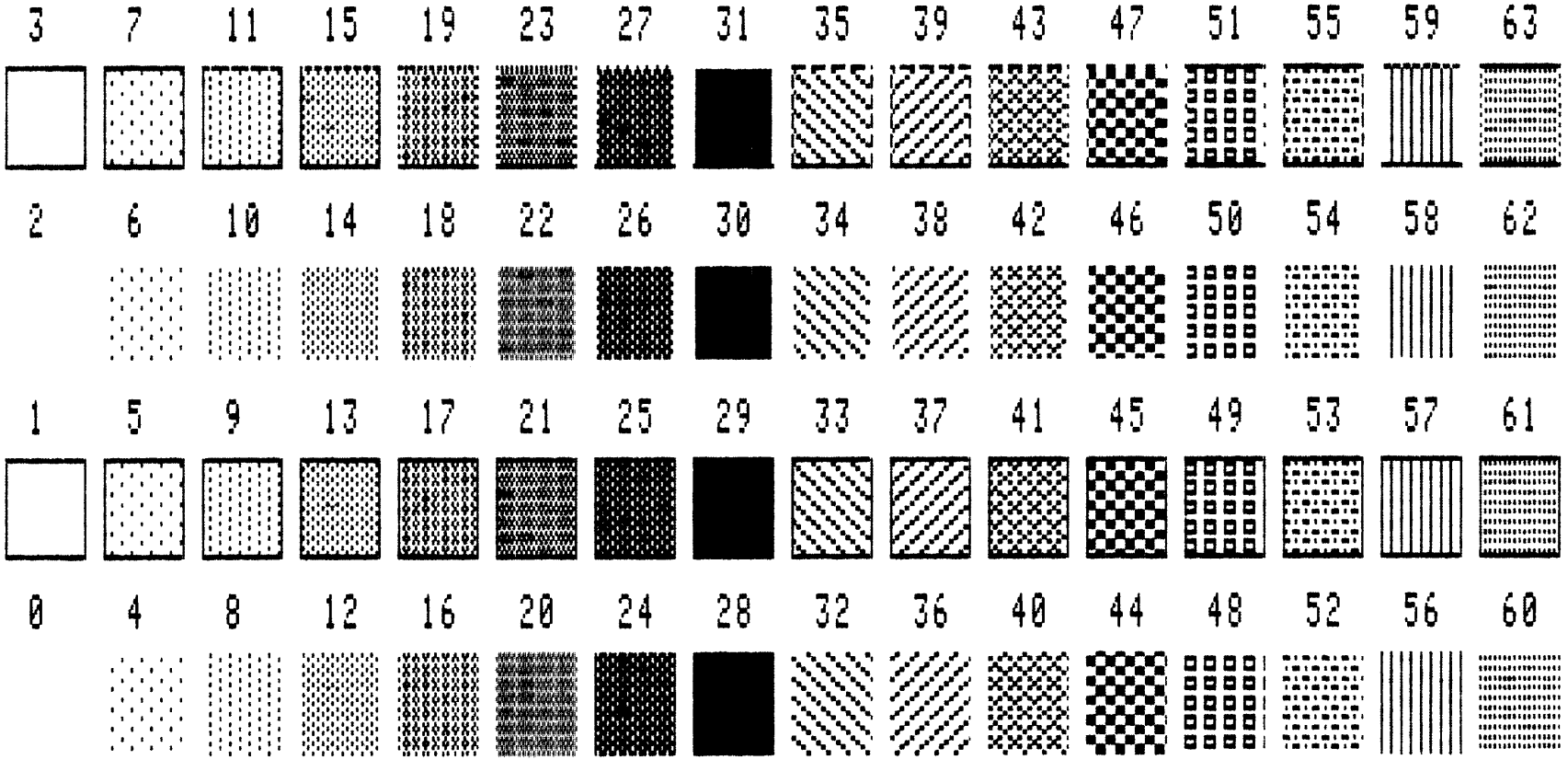
LEN (s$)
STR$ (x)
STR$ (x,"format")
VAL (s$)
CHR$ (x)
CHR$ (<begin code>,<end code>)
ASC (s$)
TRIM (s$)
REV (s$)
TRAN$ (s$,t$,u$)
MATCH (s$,<search string>)
MATCH (s$,<search string>,<begin position>)
FIND (t$<comparison operator>s$,<begin position>)
BIT (v$,<bit position>)
BIT (v$,<bit position>:<bit width>)
ROTAT$ (s$,<bit positions>)

FILE AND I/O FUNCTIONS

POS (<device number>)
LINES (<device number>)
INCHR$ (<device number>)
FILE (s$)
FILEPOS (<file number>)
FILESIZE (<file number>)
SPACE (<disc number>)

TYP (<file number>)
PARAM (x)
FREE (x)
EXAM (<address>)
INP (<port>)
CALL (<address> [,<param1>...])
[v]

APCBASIC GRAPHICS INTERIORS AND PATTERN NUMBERS



Installing TEAC Half-Heights

Bob Bloom
5 McCord Drive
Newark, Delaware 19713

SUBJECT: How to install the half-height 96tpi TEAC FD55f disk drives in a NorthStar Horizon. Installation includes both the hardware and software steps necessary to use two of the TEAC half-heights with one standard full height drive (SHUGART (SSDD) or TANDON (DSDD - ('quad')) at minimum cost.

This report contains all the the actions that I found necessary to get the drives on-line and should help any that are thinking about doing themselves. Although I managed the actual swap by myself, contributions by Frank Wancho and Al Plehn were invaluable.

This is the first of a number of articles on system changes for the NorthStar Horizon. This one deals primarily with the installation of the TEAC drives. The next will address about system patching and gives more details of the NorthStar CP/M configuration. A third article gives instructions on actually doing some system patching and addresses patches by Conn (ZCPR2), Peterson (Archive), Plu*Purfect (Public) and Plouffe (special disk formats). The last article (so far, anyway) talks about the actions necessary to install a full-up ZCPR3 system on a standard Horizon.

This entire exercise was started by a message I saw from Al Plehn on the virtues of the TEAC FD55F 96-tpi drives. I had been thinking of replacing the SSDD with the now NorthStar-standard QUAD (TANDON 100-2 DSDD) drives but had been stopped by cost. Then when I saw the ad in the November 84 Byte in which California Digital had the TEACs for a price that even I could afford, I finally was goaded into action. California Digital's price at the time was $139 (1), $135 (2-9), $129 (10+). The TANDONs were more expensive and had only half the capacity! (I've now noticed other places have dropped their prices on these drives - so look around.)

I ordered two of the FD55F drives on 14 Dec 84 by calling their 800 number and gave a MasterCard number. Received the acknowledgement copy via US Snail on 18 Dec and the actual drives on 28 Dec. That's a two week delivery in the middle of the holiday season - not bad at all.

The TEAC FD55F characteristics are, briefly: half-height, 96 tracks per inch, 80 tracks per side. Disks are secured with a twist lever, not a swinging door. Relative to the TANDON (or SHUGART) drives, the TEAC drive controller edge connector is slightly higher and upside down and physically the TEAC drives are slightly deeper. They are hardware compatible with the standard NorthStar disk controller, and software compatible with NorthStar's CP/M 2.2 Rev. 1.1.0 'CPMGEN'. CP/M formatted capacity shows 784k free space with the standard NorthStar FORMAT.COM formatter with a 4k block size. Plouffe's 'octal' patch BDOS modification yields 786k free space with a 2k block size. The drives came with a technical manual (~25 copied pages) which I had asked for (at no cost but I gather you have to ask for it) and two power connectors with detached plugs but no cables. Each drive had two 3mm and #6 mounting holes top and bottom, the 3mm holes being in positions that line up with the preexisting Horizon baseplate holes. The #6 holes are the same size as the old drives had but are in different positions.

Since I bought 2 of the TEACs, I decided to keep one of the old SHUGART SSDD drives in the Horizon so that I still could write 48-tpi formats. Because of the greater depth of the TEACs, they will only fit in the right side drive position without moving the power supply. (A power capacitor gets in the way on the left side drive - Al Plehn suggests moving all the offending parts of the power supply back an inch by drilling a few holes. The two new drives just barely fit in my standard configuration.) As the disk controller cable has only two connectors, I initially worked on getting just the two TEACs on-line - the old SSDD drive was

be just a dummy. Later in this message I tell how I got the SSDD drive on-line so that all three can worked together.

Interesting note: the manual gives the mounting configuration as lever up if mounted vertically, or the in-use light up if horizontally. (If mounted horizontally, having the disk drive motor up or in-use light down is called a no-no.) The vertical position as stated puts the R/O notch DOWN, opposite to what I'm used to. This would put the edge connector right-side up, at least compared to the TANDONs. As I have my Horizon vertically oriented on it's right edge (drives on top), I really have the drives horizontal as suggested. On the other hand, this means when I place the Horizon horizontal, the drives are now vertical and upside down as regards the instruction and edge-card connector, but right-side up as regards my 'instincts'.

Actually getting the drives to read and write disks was a bit more complicated than I thought, but straight forward once you realize that the TEACs can't write a 48-tpi disk or the 48-tpi drives a 96-tpi disk and CP/M has to be told about the 96-tpi drives before using them. What follows is the steps I took written as instructions. After switching to the two new drives one can still read all the old disks, but can only reformat them to 96-tpi, not do any other write operations on the 48-tpi disks. When one gets a longer controller cable (or cross fingers and stretch the old), you can put the old 48-tpi drive back on-line and then write 48-tpi formats too.

Before starting, inspect the drives for obvious problems. Program the drives via the mini-jumps for: one drive as '1' (to be used as the A: drive), and the other drive as '2' (to be used as the b: drive) and remove the terminating resistor strip from drive 2 (not 1! - when you're finished drive 1 will be on the end of the controller cable). I left the others as them came in the default positions. The manual does explain all of the options - my problem was understanding the explanation!

1. With power off, remove the right-hand side old drive and temporarily plug in a new drive (B) to the now-open power and controller cables. (The new drive is 'upside down' with R/O notch at the bottom.) It need not be secured in place as it will be moved later.

2. Power up and place a copy of the NorthStar CP/M distribution disk in the A: (old) drive and a blank disk in the B: (new) drive. Boot your old working version.

3. Run CPMGEN from the A: drive and create a new vanilla CP/M system with drive one as a SSDD (D) or QUAD (Q) drive and drive two as a nine (N) drive (fast stepping). Place the new system on the A: drive disk and cold boot. The 'N'ine option is not shown on the prompt, enter it anyway. If it does not 'take' you probably do not have a new enough version.

4. Format the disk in drive two as a 'Nine' disk. Do this by running FORMAT.COM and selecting either 'N' or '9' - it is not on the menu and you might need a one byte patch. (See the NorthStar-users message archive if you are not familiar with this patch - change '9' 39h to 'N' 4Eh at 0176h.) Or alternately, and probably preferred, use NEWFMT.COM and format as 'N' - (**NOT** 'O' for octal with special 2k blocks! NEWFMT.COM can be found in SIMTEL20 at MICRO: CPM.NSTAR NEWFMT.14COM which you should get anyway as it is needed later if you want Plouffe's modifications.)

5. Run CPMGEN again and create a new system with both drives one and two as N drives, and place it on the newly formatted disk in drive two (TEAC). Mark this disk as a 'N' format boot disk and power down.

6. Disconnect the power and controller cables from both drives. Place both both new drives in the right-hand slot in the correct orientation, i.e. R/O notch and in-use lights up. I left the old (disconnected) left hand drive in place. Before securing the drives to the chassis, attach power and controller cables. I found it easiest to run the controller cable to the right-most drive first, and then to the middle drive. Note that the connectors are now upside down relative to what they were. The controller cable is pressed directly between the drives and

a power connect strip with no clearance. (I've only 3 cards in the buss and the power supply runs cool - people with hot supplies should take Plehn's advice above or risk a melted cable.) Remember too that now drive #1 is on the end of the cable and should have the terminating resistor, not drive #2!

7. Secure the drives. The middle drive (which is the 1 or A drive) can be secured directly from underneath from the same holes as the old drive came from. I found the 3mm holes in perfect position - if you don't worry about buggering up the threads, a #4 screw works (or **gasp** - a short self-tapping screw), otherwise either use a correct 3mm screw or drill and tap the baseplate for the #6 holes. There is no easy way to secure the right-most drive - either drill the baseplate for another hole, fabricate a bracket to go across the top of the drives (my favorite), or just let flop. The latter is not as bad as it sounds, especially if the Horizon will be used vertically.

(One possible method of making a bracket: place all three drives in the chassis where they belong. Place a piece of paper over the drives where you want the bracket to go and punch holes in the paper where the screw holes are. I used all six of the #6 holes on the top of the three drives. The bracket will be nothing more than a flat piece of metal. Transfer punched holes in the paper to a piece of aluminum and drill - and voila! One custom-made NorthStar/TANDON/TEAC disk drive bracket! If you wish to get a bit fancier without any more work, make the plate large enough to press up against the front panel and turn the holes into slots with file or drill. Then one can use the plate to pull the drives tight against the front of the front panel while the plate is pressed against the back of the front panel.)

8. Power-up and boot with the 'N' drive boot disk. One can still read all the old 48-tpi disks but a write request to any of these will cause a "ILLEGAL ACCESS" and require a cold-boot to regain control. Try it, you'll like it!

That was the easy part - I did have some difficulty with the following. I don't recommend the following procedure with the old disk controller cable (it would better to buy or make a new cable) but if you have confidence in yourself and a restricted pocketbook you can try the following. (The old controller cable just barely fits in this procedure - I would recommend buying 3 drive connectors, one 34-pin connector and at least 24 inches of 34-wire ribbon cable and make your own. In general you can follow the procedure below but fit the new cable as you go - it's lots easier to wind it 'over around and through' the power supply if you have plenty to spare.)

1. Buy one or two 34-pin edge card connectors from somebody. (I got mine from Radio Shack.) Make sure that they are 'through' connectors, not the cable termination type.

2. Take everything apart again, i.e. remove all three disk drives and take the disk controller cable off of the controller card. My cable was 17.5 inches from controller connector to the end drive connector.

3. **VERY CAREFULLY** remove the middle drive connector from the cable. (This might not be possible without damaging the connector.) One will want to reuse the plug so don't break any pins. (I found myself needing to solder two of the little buggers back together after they broke. Sort of like trying to solder two wires together end-to-end WITHOUT overlapping them. Not the job one could do without **a lot** of patience, or substantial soldering skill. So either don't break it or buy the extra edge connector. My repaired connector failed a couple of days later, and had to be replaced anyway.) In any case, clearances are so tight the middle connector has to be removed one way or the other.

4. Crimp on the removed (if

salvaged) or a new connector 1-3/4 inches from the end drive connector. This moved connector and the end connector will connect to the two TEAC drives. I found it easiest to place the connectors in a vise to crimp them on. Face both connectors in the same direction.

5. Read steps 7 and 8 below and make a test fit of the cable on the computer. After you are familiar with how the cable has to twist and turn, crimp the new connector approximately 8 inches from the CONTROLLER end of the cable and facing the same direction as the other drive connectors. You should mark the exact location during the test fit.

6. This will leave the cable with the controller connector on one end, a drive connector on the other, and two drive connectors in between, all facing the same way, as in Fig. 1.

7. Place disk controller in the 4th or 5th buss slot from the rear or whatever is closest to the back of the disk drives. Attach cable to the controller and run the cable directly down to the SSDD drive. You will have to take out the old twists in order to get it into position. It is easier to do if the drive is not attached.

8. Make two 45 degree turns with the free end of the cable (between the first and second drive connectors) to flip it upside down, and attach the second connector to the third drive (TEAC). Finally, connect the end connector onto the middle drive (TEAC). This is practically impossible to do with the drives connected to the chassis, so leave them all loose. Fig. 2 is a "drawing" of the cable.

9. Solder or crimp one end of 4 short (4 to 6 inch) wires from your spare parts box to 4 supplied power connector pins. Try to use the same approximate sire gauge or larger - this is going to be a power cable, not signal carrier. It will be easier if you have one black, one white, one green and one red wire to match the current colors in the plugs. Using one of the drive power supply cables as a color guide, push the proper color-coded pins into the supplied power connector housing. Carefully remove the four pins from one of the drive power supply cables and solder on the free ends of the new connector cable, matching colors appropriately. Push the pins back into the housing, using the other cable as a color guide. NOTE: the connectors are keyed to their jacks to prevent misattachment. Therefore it is important that you install the pins in the housing correctly.

10. Attach the old, unaltered power cable to the old drive and the new, split power cable to the two new drives.

11. Double check that the power connections are right, and the controller cable is in the correct orientation. Ignore the markings on the cable connectors themselves (the connectors can and are often attached upside down), refer back to the controller to find which edge of the cable have the low numbers (pins 2 and 34 are marked on my controller card). Follow that edge to the three drives and make sure that the pin markings on the drive (circuit card) match up.

12. Providing everything is correct, cross fingers and power up.

13. Check out the three drives to make sure that all will read and write. Note that drive C: (the old one) will only read and write 48-tpi formats; drives A: and B: will read 48-tpi and read and write 96-tpi. If need be, generate a new system with the three drives identified appropriately.

14. Finally, attach drives to chassis with the screws underneath or your homemade bracket. Make sure that the cable connectors stay in place when the drives are pushed in, the old drive connector is especially difficult if it does not fit tightly.

P.S. After five days of use, one of the new TEAC drives died. The motor no longer started either with in parallel with the others or upon a drive select signal although the in-use indicator did light. California Digital was very cordial, gave me a return authorization number and said to send it back. I sent it back on Jan 9, and received a new replacement on Jan 31. For a while I was running one old SSDD and one of the TEACs, now all three

are working fine.

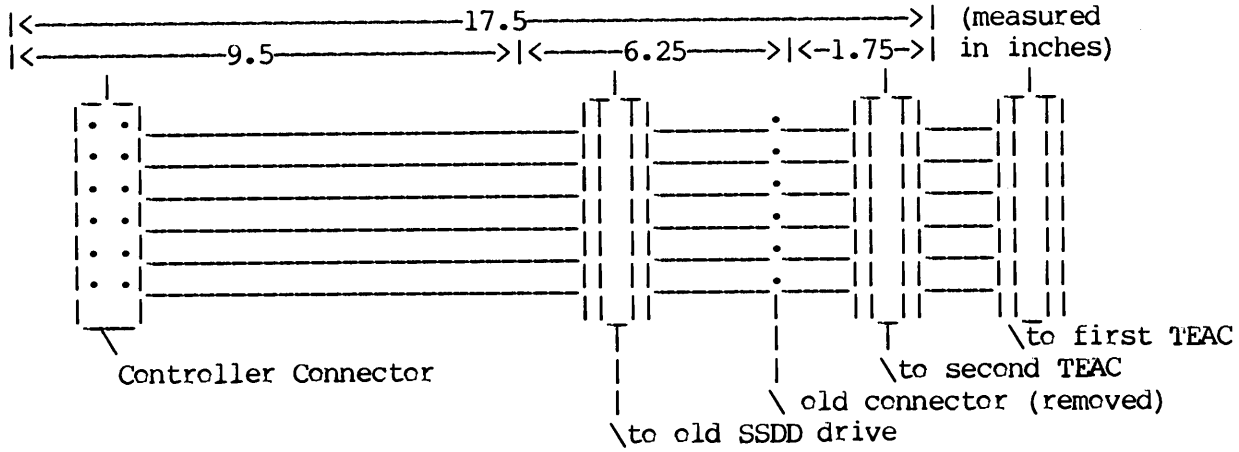
Coming attractions:

I'm currently working on getting ZCPR2/ZCPR3 working on the new drives. All of the ZCPR3 buffers fit very nicely above the disk controller address space, leaving continuous memory for a 58k CP/M.

I also am trying to do the same modifications to a 18Mbyte hard disk system, which is complicated because of the larger BIOS and multi-user operating system (TSS/C).

See later articles for the answers to these and other exciting questions.

# # #



NOTE: IF YOU ARE MAKING UP A BRAND NEW CABLE DO NOT USE THESE DIMENSIONS - THIS IS A **MINIMUM** LENGTH CABLE. With the luxury of cutting and fitting to length, one would probably want to add several inches before and after the connector to the old drive.

Fig. 1

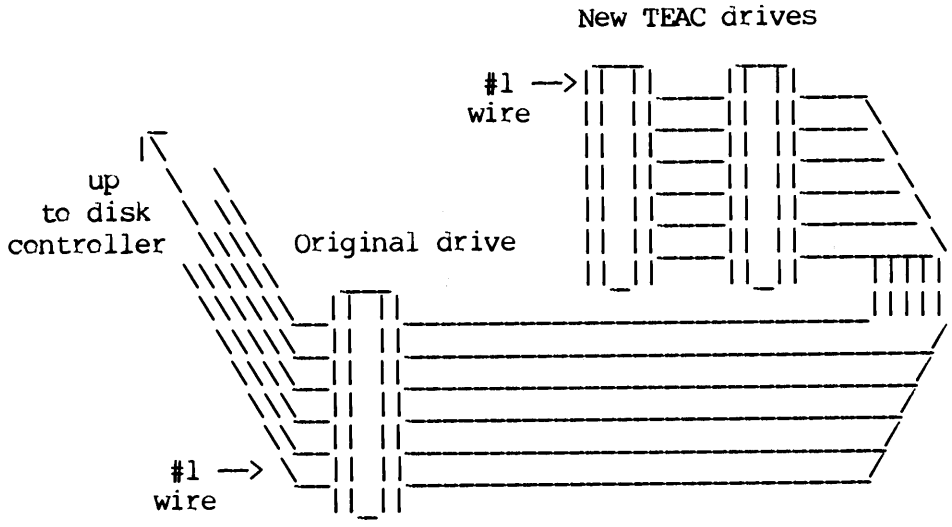


Fig. 2

Those Random Numbers!

Getting them where you want them.

By Steve and Joe Maguire
2321 Foxhall Drive
Anchorage, AK 99504

Recently, a company advertized for an experienced Basic programmer. Nineteen applicants were selected for interview. The interview consisted of the usual questions concerning experience but also a simple programming test. The test question was:

Write a one line Basic statement which will select a single random number out of a possible 69 which lies between the range of minus six and plus 11.

Of the nineteen applicants, not one could pass the test. Could you?

The RND function

The random number function of Basic is one of its most useful features. But surprisingly, even experienced programmers have difficulty using it. The reason seems to stem from the fact that the standard RND function returns a value between zero and one (0 n 1). Getting the desired amount of numbers into the proper range is not always easy.

The general expression for generating a number in the desired range is as follows:

```
N=INT(((U-L)/S+1)*RND(0))*S+L
```

where: U=largest number desired
L=lowest number desired
S=step value:
(U-L)/(# of values-1)
(or difference between successive numbers)

Let's use some examples.

1. We want to select from the integers between one and 100. Therefore:

U=100, L=1, S=1

```
N=INT(((100-1)/1+1)*RND(0))*1+1
```

which reduces to:

```
N=INT(100*RND(0))+1
```

(when reducing, don't forget Basic's order of operation: * / + -)
(crutch: My Dear Aunt Sally)

2. Select from only the odd integers between 3 and 21.

U=21, L=3, S=2

```
N=INT(((21-3)/2+1)*RND(0))*2+3
```

which reduces to:

```
N=INT(10*RND(0))*2+3
```

3. Our test question above.

U=11, L=-6, S=(11-(-)6)/(69-1)

```
N=INT(((11-(-)6)/.25+1)*RND(0))
                        *.25+(-)6
```

which reduces to:

```
N=INT(69*RND(0))*.25-6
```

At this point a few things are worth noting:

a. The multiplier in front of the RND function is always an integer and it is always equal to the total number of possible values which could be returned by the function.

b. The multiplier following the RND function is always the step value.

c. The value following the step value is the scaler. It is the amount of shift above or below zero which the series will have.

d. The RND function is always contained within the INT function unless a random step value is desired.

Extras for Experts

4. Randomly select a percentile value between 5% and 15% with a step value of .25%.

```
N=INT(((15-5)/.25+1)*RND(0))*.25+5
```

```
70 N=INT(41*RND(0))*.25+5
80 N=N*.01
```

Note that this example requires one additional step: converting the integer value to a decimal percentile. (assuming that you want to use the percentile value in a following calculation)

But also note that if you convert the percent to decimal first, and then use the general expression, you won't go wrong.

```
N=INT(((.15-.05)/.0025+1)
        *RND(0)).0025+.05
```

```
70 N=INT(41*RND(0))*.0025+.05
```

5. Randomly select among the values:

2, 4, 8, 16, 32, 64, 128, 256, 512, 1024

In this series there is no common step value so we cannot immediately apply the general expression. However, we notice that the desired values are powers of two. We can, therefore, randomly select an integer between one and ten and then use it as an exponent in another expression.

```
50 N=INT(10*RND(0))+1
60 Q=2 N
```

6. Randomly select from the even integers between 2 and 100 and the odd integers between 101 and 199.

This problem practically requires a separate program but it is a straightforward procedure. It requires one expression for the series between 2 and 100, a second expression for the series between 101 and 199, and a third to randomly select among the two.

```
120 N=INT(2*RND(0))+1
130 ON N GOTO 140,160
140 A=INT(50*RND(0))*2+2
150 GOTO 170
160 A=INT(50*RND(0))*2+101
170 ...
```

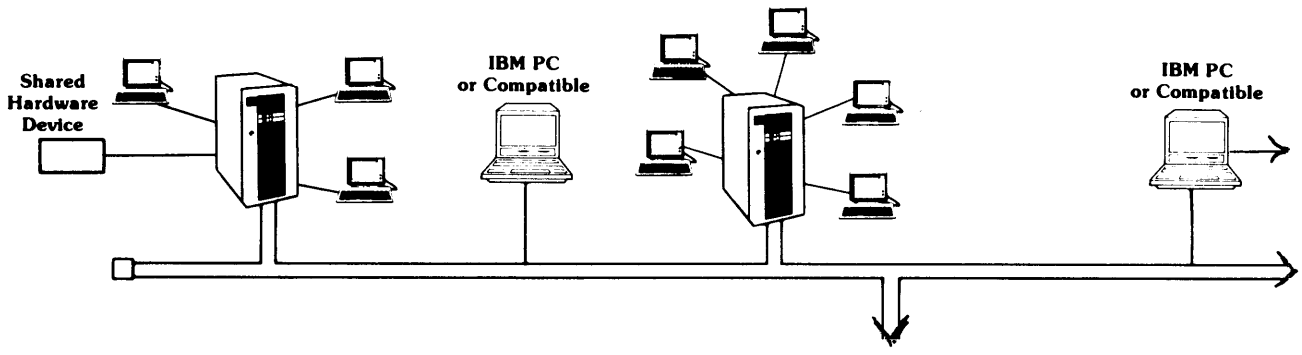
Any of the examples discussed above can be used within a FOR NEXT loop to obtain multiple values.

Practically any required random number series can be broken down to fit one of the examples given. If you really run into something unique, we would like to hear about it.

Oh yes. One of the job applicants was hired. He wrote the solution to the problem in APL!

# # #

DIMENSION Networking



North Star BASIC Random Number Generator

How good is it?

By Joe Maguire

Are all numbers created equal? When it comes to random numbers, definitely not! It's one thing to be blasted by an inordinate number of Klingons but quite another when my taxes might be raised because some government agency used a faulty random number generator in a demographic simulation model.

What this article will explore is just how good the random number generators of various Basics really are. You may be pleasantly surprised with North Star.

What's a Random Number?

The RND function of Basic is so convenient that we often take it for granted. But a truly random number is a very elusive thing. Trying to define a random number is like trying to answer the question--how high is up?

We usually say that a series of numbers are random if they are evenly distributed and non-repeating. But in order to determine these properties, we must compare one series with another, and another, and another, ad infinitum. The comparison is made by subjecting a series to a number of tests. If it passes most of the tests it is considered random. But as more tests of randomness are devised, a truly random series may become more difficult to obtain.

Pseudo Random Numbers

Generating truly random numbers is not simple. As mentioned above, one of the criteria is non-repetition. But as any game player in Basic is aware, the same number can come up often. This results from constraints which we place on the generation process. For a long series to have no duplicate members the numbers must be very large--on the order of a hundred digits or so. This can rapidly use up available memory in a computer! When we chop the numbers down to a manageable size, we are bound to have some repeats. The resulting series is then called "pseudo random."

Truly random number generators are almost always hardware devices. They count radioactive decay particles or molecular motion or other physical phenomena which have few constraints. If we were to take a list of numbers produced by such a device, tack it to the wall and then throw darts at it, we could feel reasonably sure that any number struck by the dart was random enough for our purpose. This is essentially what a computer does. The result is still a pseudo random number because there is a chance the dart may strike the same number twice.

The "list" stored by the computer is the software algorithm which generates the random numbers. The "dart" is the seed value with which the software starts its calculation. Using the same seed will always give the same result. This is why many Basics have a RANDOMIZE statement or other such function which "shakes the dice" before generating a random number.

North Star's Dice Shaker

North Star Basic does not have a RANDOMIZE statement but instead uses a hardware gimmick to establish a random seed value. Invoking the RND function with a minus one value (RND(-1)) seeds the random number generator with a value obtained from the status register of the disk controller. To my knowledge, North Star's Basic (and perhaps others specifically designed to be used with a North Star disk) is unique in this regard and is one reason why Basic produces good random numbers.

(An important thing to remember about

using this random starting seed is that it should be called only once at the beginning of the program. Thereafter, the function RND(0) should be used.)

But ultimately, the randomness of any series produced by software is determined by the size of the numbers. Various Basics allocate from two to ten bytes for each number generated which will give from 32,000 to over a billion values before a repetition occurs. The numbers are scaled into a range requested by the user by repetitively subtracting a fixed value (called a modulus) until the remainder lies within the requested range. Various other tricks are used to reduce near-neighbor correlation (successive numbers too close together). North Star Basic picks only every 23rd number, for example.

North Star Basic uses a random number algorithm devised by Eric G. Rawson. Rawson was involved in the field of data transmission security and realized that the technique used to generate maximal length bit sequences (m-sequences) also produced good random number series. The result is quite good as the following tests will show.

The Acid Tests

The proof is in the pudding and there are several tests we can apply to see just how good a random number generator really is. One time honored method, from the field of statistics, is the chi-square test.

We can request, say, 10,000 numbers which range from zero to nine and sort them into ten "bins." At the end of the run, if the numbers are evenly distributed, each bin should contain 1000 numbers. That rarely happens but we can measure the deviation from the expected value and then apply the chi-square test to find a statistical value of "closeness." By running the test ten times and averaging the results we can get a good idea of the distribution quality of the random number generator. See Listing 1. for the program.

A second test is the pi-generator. We make believe we are throwing darts at a circle which is enclosed by a square. After throwing, say, 4000 darts, we count the number which struck within the circle. The result should be pi, 3141. (area of circle) To simplify calculations we will use only the first quadrant and a unit circle. (radius=1) Again, we will average ten runs to see how good our random number machine is at generating pi. The program is in Listing 2.

```
100 N=RND(-1)
110 FOR I=1 TO 10
120   FOR J=1 TO 10000
130     N=INT(10*RND(0))
140     A(N)=A(N)+1
150   NEXT J
160   C=0
170   FOR J=0 TO 9
180     B=A(J)-1000
190     C=C+B*B
200   NEXT J
210   T=C/1000
220   PRINT "PASS ",I,
230   PRINT "CHI-SQUARE = ",T
240   T1=T1+T
250   FOR J=0 TO 9
260     A(J)=0
270   NEXT J
280 NEXT I
290 PRINT "AVERAGE = ",T1/(I-1)
300 END
```

Listing 1

Random number distribution tester by computing Chi-square statistic.

```
100 A=RND(-1)
110 K=4000
120 FOR I=1 TO 10
130   N=0
140   FOR J=1 TO K
150     A=RND(0) \ B=RND(0)
160     IF A*A+B*B>1 THEN N=N+1
170   NEXT J
180   T=(K-N)/1000
190   S=S+T
200   PRINT "PASS ",I," PI = ",T
210 NEXT I
220 PRINT "AVERAGE = ",S/(I-1)
230 END
```

Listing 2

Pi-generator. Counts darts thrown at circle enclosed by square. SQRT not needed since unit circle is assumed.

Results

The chi-square statistic produced by the program of listing 1. can be interpreted by referring to a chi-square distribution table. Our test has nine degrees of freedom.

A complete description of the chi-square distribution test is beyond the scope of this article but a brief explanation will be given.

The results of our random number test, as given in listing 1., could range from 10,000 "0"s, through a perfect distribution of ten "1000"s, to 10,000 "9"s. If the chi-square statistics of all the possible combinations were plotted as a "histogram" type of graph, it would form the shape of a somewhat skewed bell curve.

If we were to decide to discard all the values which fell within the first 5% and the last 5% of the area of our histogram as non-typical, then the remaining 90% could be considered typical or, in the case of our example, indicative of a random distribution.

Referring to a standard chi-square distribution table, under the heading of nine degrees of freedom, we see that the limits of such a distribution will have a lower statistical value of about 3.3 and an upper statistical value of about 17. If our test produces a result within these limits, we can say our random number generator is acceptable with a lower value somewhat better than a higher value.

The pi generation test can be considered acceptable if we hit the value within +/- .5%. (3.157-3.126)

# # #

CLIP TIP

The General Expression for Generating a Random Number

```
                N=INT(n*RND(0))*x+y
                  |  |  |   |     | |
The number--------|  |  |   |     | |
The INT function--|  |  |   |     | |
Total number of      |  |   |     | |
possible values------|  |   |     | |
The RNDom function-----|  |     | |
The step value-------------------| |
The scaler value-----------------|
```

Fig. 1 shows results from testing a number of Basics:

Basic Version		Chi test	Pi test
North Star 5.5.0		7.8764	3.14153
Microsoft 5.0	(1)	8.380	3.1429
Microsoft 5.21	(2)	8.7072	3.1385
Microsoft 100	(3)	9.5906	3.1486
Microsoft IBM	(4)		
ZBAS	(5)	7.547	3.1356
APCBASIC	(6)		
Cromemco	(7)		
S-Basic	(8)		

(1) CP/M version, circa 1979
(2) Supplied with KAYPRO computer
(3) Used in Radio Shack TRS80/M100
(4) Used in IBM-PC
(5) A public domain Basic
(6) American Planning Corporation
(7) Cromemco 32K Basic
(8) A compiled Basic

Feedback

Joe Maguire
2321 Foxhall Drive
Anchorage, AK 99504

Several of my articles appearing in The Compass have generated significant reader interest. What follows is a summary of comments.

LS-100

About 25 readers wrote to me concerning the review of the LS-100 disk emulator (from Digital Research Computers--see Compass Vol. 4, no. 2).

The majority requested the software patches and diagnostic routines on a N* disk, which I offered to supply. The offer is still good.

Of those reporting success in constructing the kit (not difficult) and patching their CP/M (more difficult), they were unanimous is stating that the LS-100 works very well in the Horizon.

Some important points

1. Be sure to make all the corrections to the manual listed in the READ.ME file.

2. Some jumper options have caused confusion. For the Horizon:

J3 advanced ready is OK
J7 CPU refresh is low true
J9 refresh on pin 66

3. The board is configured for battery backup but, if you want this option, you must supply the power source yourself. DRC does not offer one. Requirements are 8VDC at .8A per board.

4. One user tried testing his board with a disk test utility and it reported many errors. Remember, an emulator is not configured like a real disk. On the "system tracks" are stored checksum bytes which change as data are written to the "data tracks." Some test utilities will report this as an error condition. DRC provides a diagnostic utility which is designed for use with the LS-100.

Some good news: DRC has lowered the price of the board several times. At this writing, it is being offered for $259.00. See ad in Byte.

DRC may be able to supply patches for (Lifeboat) CP/M 2.21 on a N* disk (they sent one to me). Charles Prohaska can supply for CP/M 2.22 (see Compass Vol. 4, No. 4). I can supply for CP/M 2.21 and 2.23.

6MHz Horizon

This article (Compass Vol. 4, No. 4) has really generated interest! Letters and long distance phone calls breathlessly asked the same question. Does it work at 8MHz? (One reader even sent me a 16MHz crystal so I could rush the verdict back to him!)

The answer is yes/no. Yes, the CPU and the memory work OK; but no, the disk controller did not. While I have had flawless operation at 7MHz, at 8MHz many read/write errors appeared. I feel certain that adding an additional wait state to the controller board would cure the problem. A suitable circuit appeared in the S-100 Bus column in the November '84 issue of Microsystems. This is a hardware modification which I have not yet had time to try.

Some notes:

1. For reliability, use a Z80 version rated for the speed. While a Z80A might work at 6MHz (mine did), I wouldn't do the payroll with it. Bits can change here and there internally and really foul you up.

2. If your memory board has chips rated at 200ns, you should be able to use it at 6MHz. (provided it's not an early N*) With the wait state jumper enabled on the CPU board, slower memory works OK.

3. The higher speed tightens up the disk parameters. At 6MHz I can read/write in DD OK but I can no longer read my old SD disks. Two crystals and a switch could solve this problem.

4. If you are using the PROM option on the CPU board, it should work OK at the higher speeds.

5. The general consensus seems to be that operation is reliable at 6MHz, including operation with a hard disk, etc., but increasing speed above that is on a try-and-see basis.

# # #

Commentary

Ted Carnevale
Neurology Dept., SUNY
HSC T12 Rm020
Stony Brook, NY 11794

Since my article on using the Morrow DJ/DMA controller appeared in Compass, vol. IV, no. 5, pp. 22-23, several developments have occurred. First and most important, I found that my problem with FDJ.COM was in part due to hardware, not software. The board I initially used was a loaner that had old ROMs on it. The newer board that I am now using (Rev.3a, big ROM labelled "DJDMA 2.5") can create and read what it calls "Morrow Microdecision" DSDD floppies. I'm not sure these are really

Microdecision format, since nobody I know has such a machine, but it does reliably format, read, and write them. It may be capable of reading and writing other formats (e.g. Osborne, Kaypro) with appropriate modifications to the BIOS, but I haven't tried this yet. Does anyone have any details about various soft-sectored CP/M formats?

However:

1. SYSGEN does not enable the DJ/DMA to create a bootable soft-sectored "systems disk," i.e. a soft-sectored floppy cannot be used in the A: drive.

2. The "do-it-yourself" option in the FDJ menu apparently tries to format a disk, but hangs up during the "verify" cycle. So all is not perfect.

I now have a BIOS that runs with a full 5K track buffer in auxiliary memory. This is shared between reading and writing, so it's not as fast as a BIOS with separate read and write buffers. Most of the time I use a big RAM disk, so adding separate buffers isn't a high priority item right now.

Now that 5.25" drives are dirt cheap ($115 for half-height DSDD, and $150 or less for QD drives!) it might make more sense for interested users to hang onto their old North Star controller boards and buy one of the dedicated soft-sectored controller cards. True, the BIOS will enlarge somewhat, but you can tie two half-height drives permanently to each of the boards; some of those controllers are based on ICs that are used in machines like the Kaypro or IBM PC, for which there is a plethora of software to create/read/write alternate formats, and which are (unlike the secret inner workings of the DJ/DMA) well documented by vendors who actually support their products. That's probably the way I'd go if I hadn't already got something working with the Morrow board.

One final point--the Morrow DJ/DMA often fails to SYSGEN a hard-sectored disk properly, and it is quite finicky about reading disks that were created with my old N* controller. I never had ANY misreads etc. when using 35- or 40-track SS or DS formats with the N* controller--not ONCE in nearly four years, out of at least three hundred floppies that have seen lots of use.

# # #

Commentary

By Oliver C. Stokes

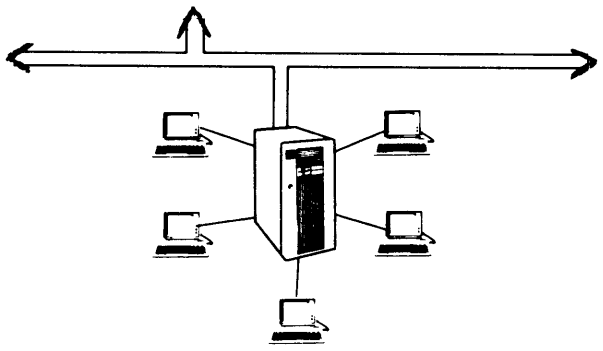
I have just received an announcement in the mail from Sol Libes in reference to his new magazine, Micro/Systems Journal. It was some of the best news I've gotten in the mail for a long time. You know what happened to Microsystems! Now Sol is coming back with a magazine which he promises will be like the old Microsystems when he was the editor.

Sol needs some help to get the word out. In view of the great support that Sol Libes gave the North Star computer when he was editor of Microsystems, I think the INSUA members should give him strong support in launching his new magazine. Besides, if it's like the original Microsystems, a subscription could be one of the best investments an INSUA member could make (besides his or her INSUA membership fee).

The following is from material I received from Sol Libes.

> Subscriptions in the U.S. are 1 year $18.00 ($24.00 first class) and 2 years for $32.00 ($44.00 first class). Canada and Mexico same as U.S. first class rates. Other foreign rates are $32.00 and $58.00 for 1 or 2 years respectively. Checks should be made payable to Micro/Systems Journal, P.O. Box 1192, Mountainside, NJ 07092.

# # #



Vendors Column

In this column we reprint vendors' descriptions of their own products. Of course, INSUA makes no claims about the quality or merchantability of any commercial software, hardware, or services. **Caveat emptor!**

TIS/APL for the ADVANTAGE/HORIZON

APL is a high level computer language that has in recent years become available for microcomputers. The language originated with professor of mathematics, Dr. Ken Iverson, who developed a concise notation originally for use in "blackboard" lectures. The notation proved so effective that when Dr. Iverson later became an employee of IBM the concepts were implemented into the computer language called simply APL.

Telecompute Integrated Systems of Toronto, Canada have developed powerful APL computer systems for many microcomputers and in particular for the North Star ADVANTAGE and HORIZON. The author has programmed many and varied applications on both computers using TIS-APL over the past 6 years.

Why Use APL?

It offers 10 to 20 times the application efficiency of other languages as BASIC, FORTRAN, etc.

Where Does APL Get Its Power?

1. Many very powerful computer functions have been implemented in terms of single keystrokes. In other languages one would have to "write" comparable programs which would require time and use up available work area.

2. APL is a universal language, equally good for Statistics, Word Processing, Business, Other applications.

3. APL can be used after just a few minutes instruction in its very powerful calculator mode which features "vector processing." With additional training the novice can write his own application programs.

Comparison of programs in APL, FORTRAN, BASIC for calculating the average of a set of numbers:

APL

```
(+/X)÷⍴X←⍞
```

FORTRAN

```
      REAL X(100)
      READ *,N,(X(I),I=1,N)
      S=0
      DO 10 I=1,N
10    S=S+X(I)
      PRINT *,S/N
      END
```

BASIC

```
10 S=0
20 FOR I = 1 TO 100
30 INPUT X
40 IF X=0 GOTO 70
50 S=S+X
60 NEXT I
70 PRINT S/(I-1)
```

APL uses many special characters such as the assign arrow

... ← ...

and quad

... ⍞ ...

shown above. All of the special characters are software generated automatically on the ADVANTAGE. (Use of the HORIZON requires a terminal capable of producing the APL characters.) The writer has developed an APROM that accomplishes this with an Heath H-19 terminal. APL terminals are available commercially, also.

APL programs written in TIS-APL for the ADVANTAGE or HORIZON, as well as files and variables, are easily transferred to APL environments on the IBM PC and vice versa.

References for further reading:

APL-An Interactive Approach (text), Gilman and Rose, 3rd ed., John Wiley.

PC TECH JOURNAL, Vol. 2 No. 3, September 1984, pp. 129-47.

PC Magazine, Vol. 3 No. 4, March 6, 1984, pp. 397-99.

PC Magazine, Vol 1 No. 11, March 1983, (3 articles) pp. 149-267.

(Submitted by:)

W.E. Claxton
431 Mishler Rd.
Mogadore, OH 44260

$ $ $

Novell NetWare on the Dimension

North Star will release a version of the Novell NetWare operating system for its high performance, multi-user DIMENSION system by mid-year.

The popular Novell Netware network operating system running on the DIMENSION will allow multiple PCs to be linked to a DIMENSION central module--and multiple DIMENSION systems to be linked together--via a variety of local area network hardware options. It will also significantly increase the performance of individual DIMENSION systems and ensure that the DIMENSION can utilize the new era of IBM compatible multi-user software that will be available this year for the MS-DOS 3.1 and IBM PC Network standards. In addition, it will give DIMENSION customers access to more than 800 software applications written in languages such as RM COBOL and SMC BASIC, and the wealth of software currently available for the single-user MS-DOS 2.1 environment.

DIMENSION NetWare, as the operating system will be called, combined with the DIMENSION's multi-user, multi-processor architecture, will make the DIMENSION an extremely high-performance "network" option in the IBM compatible marketplace.

The first DIMENSION NetWare release, by mid-year, will allow up to 50 IBM PCs or compatibles to be linked to a DIMENSION system using Omninet hardware. This release of DIMENSION NetWare provides compatibility with software written for the new multi-user MS-DOS 3.1 operating system and communications gateways to IBM minicomputer and mainframe systems.

The second release of DIMENSION NetWare, scheduled for this fall, will allow multiple DIMENSIONs to be linked into an IBM PC Network or Ethernet local area network configuration, using IBM or Ethernet hardware. DIMENSION users will also be able to plug IBM compatible add-in boards into a DIMENSION central module and use them in a shared mode throughout the network.

Since the DIMENSION began shipping in May of last year, it has utilized an operating system called DIMENSION DOS, based on MS-DOS 2.11 with extensive enhancements by North Star to provide multi-user capability. Under this operating system, up to 12 DIMENSION workstations can be connected to a DIMENSION central module, providing a dedicated 8088-2 processor for each user, an Intel 80106 server processor, one shared floppy disk drive, up to two 30 megabyte fixed disks and an optional streaming tape back-up.

DIMENSION DOS incorporates file sharing standards developed by 3Com Corporation so that multi-user software applications based on those semaphores can run on a DIMENSION cluster for up to 12 users. North Star will continue to offer DIMENSION DOS as a lower cost option for customers who need a high performance multi-user system, but don't require the advanced features of DIMENSION NetWare.

The suggested retail price for DIMENSION NetWare will be $1,000. DIMENSION DOS will continue to be sold for its current list price of $250.

$ $ $

FOR SALE

North Star Advantage with:

+ 15 Meg. hard disk
+ 8/16 board with 128K RAM
+ CP/M and MS/DOS
+ Tons of software

$1500 or Best Offer
Call (415) 527-0550

Insua

International NorthStar Users Association
Publishers of The Compass Newsletter
PO Box 2910 • Fairfield, CA 94533

Bulk Rate
U.S. Postage
PAID
Walnut Creek
Permit No. 203

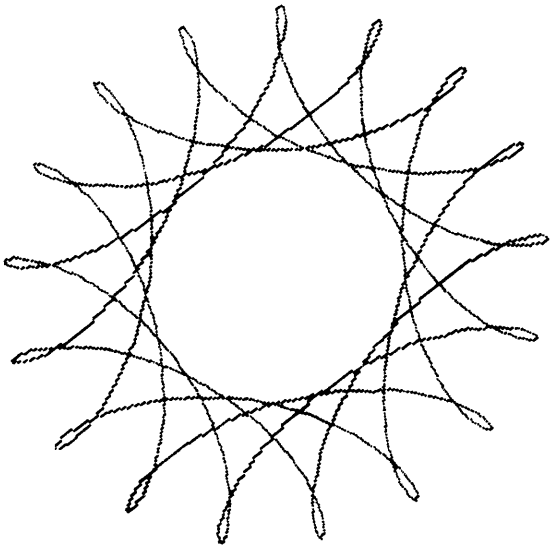
TATE 2761
JAMES TATE
23914 SPRING DAY LN.
SPRING, TX 77373

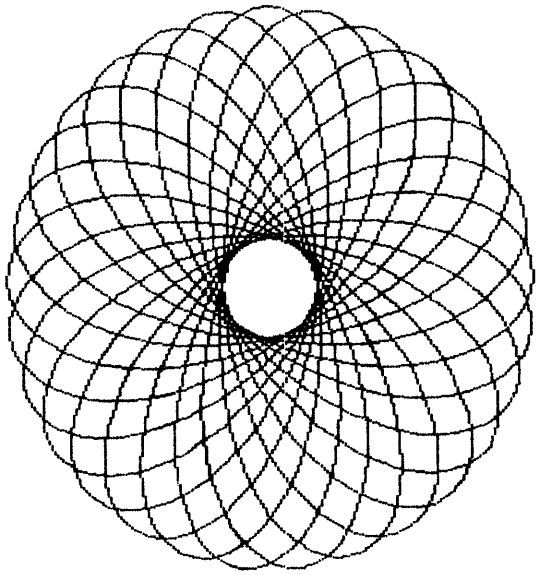
Postmaster: Address correction requested.
Please enter key number on form 3579.

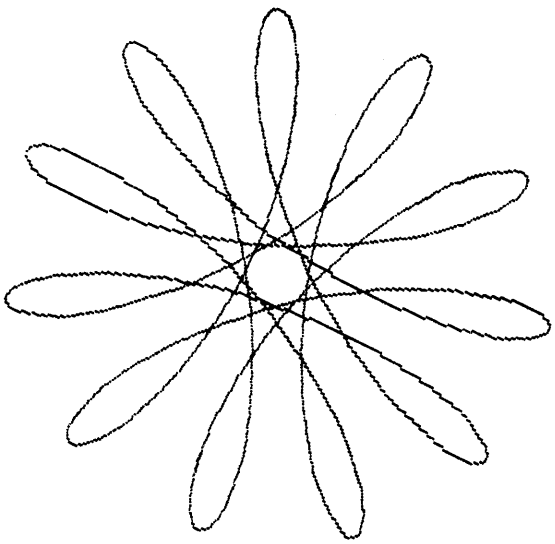
The Compass

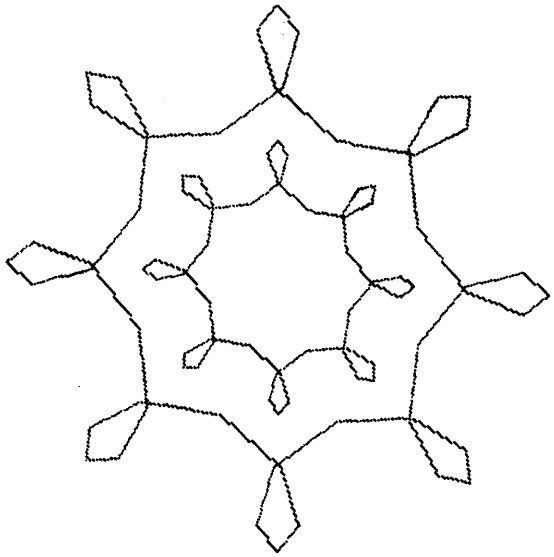
International NorthStar Users Association

Volume V No. 2





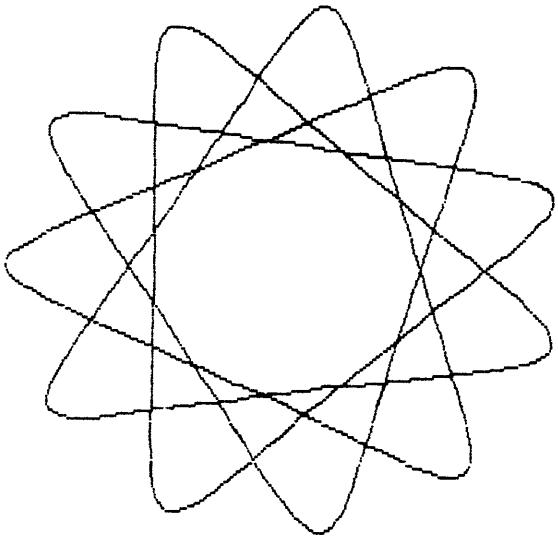


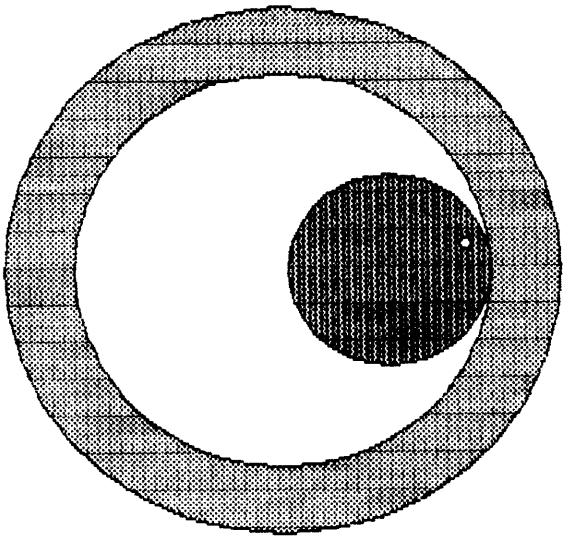


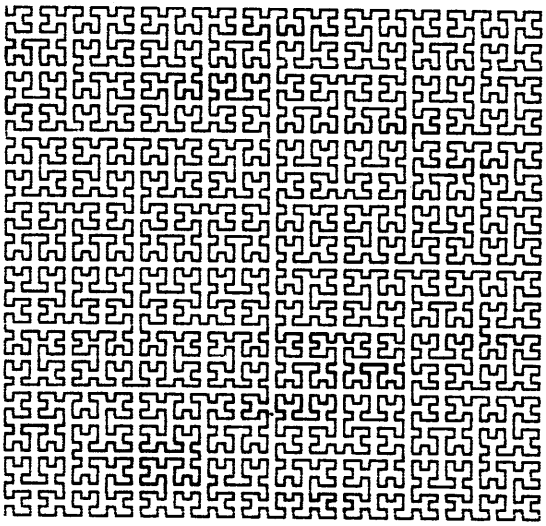
Note from the Editor

Our particular thanks to David M. Allen for the graphics on the cover, on this page, and scattered throughout this issue of Compass. Beginning on p. 24 he expands upon his article on "Graphics from Turbo Pascal" in Vol. IV, no. 3, which incidentally contained another set of Advantage-generated graphics by Steve Noll. We invite other members to submit additional computer-generated graphic designs, and will do our best to incorporate them in a forthcoming issue. The wish to display such graphics stems in part from our feeling that the graphics potential of the Advantage has never been tapped as it should have been.

Your editor wishes to extend his thanks to members who have submitted articles, especially on disk. We feel we are sitting on a wealth of resources with these not-yet-printed articles, and have every intention of getting them quickly into print. We have fallen behind in our schedule, but are determined that number 3 of Volume V will follow along shortly. (We are planning a campaign to return all disks within the next month.) Thanks for your patience.







The Compass

The Compass is published every two months by INSUA, the Interational North Star Users Association, P. O. Box 2910, Fairfield, CA 94533.

Subscription Information:

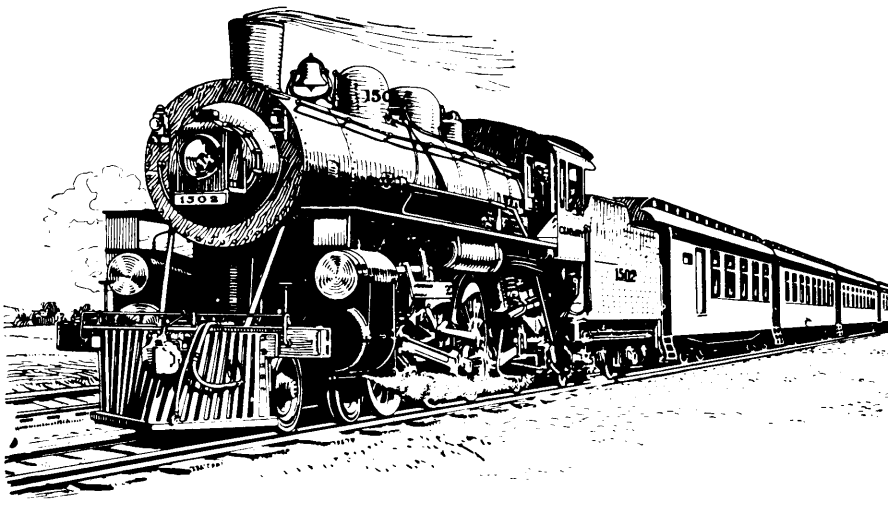
Subscription to The Compass is included in the $20.00 INSUA membership fee. Address subscription inquiries to INSUA, P. O. Box 2910, Fairfield, CA 94533.

DISCLAIMER

Programs printed in Compass and/or distributed through the INSUA disk library are offered to INSUA members in good faith. INSUA, however, is unable to guarantee the operation of any of these programs or to guarantee support. Users are advised to test the programs thoroughly for themselves in conditions under which they are to be used. Users who employ such programs in serious business or financial applications must do so at their own risk.

Facts or opinions published about manufacturers and dealers, and all opinions expressed in articles and letters, are the responsibility of the authors, and not of INSUA or the Editor of Compass. INSUA offers the right of reply to members and non-members alike.

Contents



A Look at a BASIC program for the Simulation Life

By Edgar F. Coudal
627 S. Crescent Ave.
Park Ridge, IL, 60068
(312) 823-3834

The purpose of this article is to teach some of the fine points of programming in BASIC by describing in detail how the simulation "LIFE" can be written and run on a North Star Advantage or Horizon with a Pleistocene terminal, Neanderthal Z80 and limited memory.

There are some bells and whistles included in this program for illustrative purposes. They slow it down, and at the end I'll tell you what to throw out to maximize execution. However, for the time, let's look at the whole thing with the idea that the concepts and even parts of the code can be picked up and moved to other programs of your own.

I copyrighted this program (whathehell, I even copyright my grocery lists because writing is my business), so don't try to sell it to anyone else without first making a deal with me ...

* * *

"LIFE" was created by John Conway of the University of Cambridge, England (Ha! And all along you thought life was created by God, or Henry Luce or Parker Brothers or someone like that) in the late '60s. "LIFE", to quote A. K. Dewdney, writing in Scientific American (May, 1985, p. 18), "is an infinite two-dimensional lattice of square cells whose states are influenced by the states of neighboring cells. Time is also discrete and from one tick of a cosmic clock to the next each cell is either alive or dead depending on a set of very simple rules:

1. If a cell is dead at time t, it comes alive at time t+1 if, and only if, exactly three of its eight neighbors are alive at time t.

2. If a cell is alive at time t, it dies at time t+1 if, and only if, fewer than two or more than three neighbors are alive at time t.

Put another way, living cells in the grid die of loneliness (if only 1 or 2 of the surrounding 8 neighbors are alive) and they also die of overcrowding (if 4 or more of their 8 neighbors are alive.). The "birth process" is similarly specific: If the cell is dead it comes to life if exactly three of its neighbors are alive. Apparently, it takes three to tango in that sex life.

* * *

The accompanying program is heavily REMarked with what are hoped to be useful comments. Don't write and tell me you can do it better; I **know** you can. But do write if I'm really wrong about something (which is not an infrequent happening). Remember, I wasn't trying to write the most efficient, elegant or fastest program; I wrote it this way so we could discuss some points in BASIC programming.

Let's start at the top and see what's going on. Refer to the program listing as we go through this.

LINES 100 to 170

Always litter the beginning of your programs with REMarks that tell who wrote it, when, why, and what the hell the program is supposed to do. If you die or run off to tour with the Stones or ride with The Outlaws or get a real job, some poor mope may have to pick up where you left off. It's also good to note what variant of what language was used and

what machine was used in development. Otherwise, everyone will presume it's IBM or some other equally crippled little beastie that should have been drowned at birth.

You'll notice a lot of blank lines with nothing more than REM after the line number. That's just to make the whole thing easier to read. The REM lines slow things down by milliseconds but we're not dealing with incoming missiles, so put them in.

LINES 180 TO 230

This is the initialization module. We're going to use three numeric arrays, each 20 deep by 38 characters wide. With the extra spaces we'll use in the screen display, that will give us room for a nice picture 18 lines deep by 72 columns wide. I'll explain that 20/38 18/72 discrepancy later; I don't want you to lose faith in Teach this early in the exercise.

If your terminal doesn't repond to some simple clear screen command (I have a really stupid 1978 Soroc 120), set up the clear screen as a simple string variable (note line 180). For the Advantage, set the C9$ equal to plain old CHR$(4) instead of the escape sequence shown.

In North Star BASIC, you have to DIMension an array before you can use it unless it's a one-dimensional array of 10 or less elements. We do that in lines 210 to 230. Each element of the C array -- hereinafter referred to as C() -- will hold either a 1, indicating the cell is alive, or a 0, indicating it is dead. Each element of the D() array looks at the eight neighbors of each element of the corresponding C() and holds the total live count. Thus if a 9-cell block looked like this:

```
0 1 1
1 0 1
0 0 0
```

the C() element (the one in the middle) would be 0 and the corresponding D() element would contain the value 4. Just keep that in mind.

The E() is really a bit of froo-froo. It's in here to see if the whole thing reaches a place where nothing occurs from one interation to the next. That's really boring. If nothing does occur, the program ends automatically. We'll see how that works later.

LINES 260 TO 280

These lines present an initial screen display to tell the user what's going on. You shouldn't be the only one who knows.

LINES 330 TO 410

Here's one way to handle a menu. I like this better than the standard 1., 2., 3., etc. method which usually leads to an ON X GOTO line of code. Granted those are shorter and faster, but why not be a bit more literate about things? The INCHAR$(0) in line 370 is a function the computer uses to just sit around and do nothing. Computers and Democrats love things like that. When the computer finds one of those INCHAR$(0)s, it just stops and waits for you to type something in from the keyboard, which is the device specified by the 0 in parentheses. The computer will wait until May 3, 1995 or until the power goes off, whichever comes first, for you to type a character.

It's thoughtful to put in a line such as 375, which tells the user the computer is doing something private and hasn't gone off to venerate ENIAC in the skies. Lines 380, 390 and 400 are conditional branches to other modules of the program. If the computer can't find somewhere to branch to in one of those three lines, it reaches line 400, which pops it back to the menu. Some call this an "error trap," though that's not strictly true. We'll see a real error trap later in this program. Line 400 is a "klutz trap."

LINES 420 TO 500

This module allows you to resume a program that was run previously, reading each cell's value into the C() from a data file. T is the variable that counts iterations, and is the first thing stored in the data file. You of course created the data file earlier from the DOS. Remember how you typed in:

CR LIFEDATA,2 10
(CReate a file named LIFEDATA on drive

2 and make it 10 sectors long)

TY LIFEDATA,2 3
(Make it a TYpe 3 -- Data -- file.)

I thought you remembered doing that ...

We might as well talk about Nested Loops here. You'll note that we have two FOR/NEXT loops, an I loop with a J loop inside it. You don't have to enter the I or J after the NEXTs, but it helps in understanding the program. Unlike DeRouchefauld, who said, "If we do not stop explaining, we shall never understand," I believe in programming clarity, even overkill clarity.

The I loop generates the rows (up and down the screen) and the J loop generates the columns (across the screen). Thus, C(3,15) would be the single element placed on the third row from the top and the 15th column in from the left side of the grid. C(1,1) would be the top left corner; C(1,36) the top right corner and so on. Nested Loops are used a lot when fooling around with multi-dimensional arrays.

LINES 540 TO 580

If you selected the Random option from the opening menu, the program jumped down here. Nested loops again uniquely identify each element of the array and assign a value of either 0 or 1 to each element. That's what all that stuff in line 560 means. Since BASIC gives you random numbers from .00000001 to .99999999, you have to do something with them to make such stupid numbers useful. So you multiply by a number -- in this case 2 -- and use the INT function to throw away the decimal part of the result of that muliplication. You get either a 1 or a 0, which is what we want. If you muliplied by 6, you'd get 0, 1, 2, 3, 4, or 5. (After a while, the discarded decimal fractions pile up under the space heater in the corner. Vacuum them up once a week. Neatness can be the difference between a C and a B. Honest. Especially if the prof is wired on coke or seeing the world through window pane.)

By the way, using a -1 as the argument for the RND function gives you a different start each time. I don't know why. I don't know how BASIC generates random numbers. All I know is that the manual ascribes random number generation to a "specific hardware condition." I don't know what hardware condition or why. It has nothing to do with your toaster. That much I know. But that's all.

LINE 590

Another bell or whistle. You can stop this program any time by typing a Control-C, the panic stop. Line 590, the true "error trap," will spot that Control-C right away, consider it an error, and leap to line 1110.

When using ERRSET in North Star BASIC, you must include the L and the E after the line number. The L is automatically assigned the number of the line that caused the error; the E is given a code number for the kind of error. Ignore them. We're just using the ERRSET 1110 as another type of conditional branch, and when we get down to line 1110, you'll see how valuable it is. ERRSET is useful in all kinds of programs, though I believe it was originally intended for use in program development and debugging, and not as a legitimate code device. Hell, use everything they give you; this is war!

LINES 640 TO 700

Straightforward screen display. Nested loops. IF/THEN construction. If a cell is dead (value 0), the program displays two blanks; if the cell as represented by the C() element is alive (value 1), a blank and an X are printed. The leading blanks just make it look nicer. You don't need them. And the T in line 700 tells you how many times you've watched cells live, love, flourish, take piano lessons, die, and be reborn.

Another thing to note is the way the stuff inside loops is indented. It doesn't make any difference to the program; it just looks better and is easier to read and understand.

LINES 720 TO 780

Well, we rang in a DEFined FuNction here. It's defined down there in 1580 to

1610. DEFined FuNctions are faster than actually putting the the code in the program. They also make you look like you know what you are doing. There isn't room for an explanation of the principles of DEFined FuNctions in this article, but there was a brilliant exposition of it a few issues back. Look it up. That was the first time I ever understood them. (See J. Burdeane Orris, "Multi--line User-Defined Functions," Compass, Vol. 3,, no. 2, pp. 3-10.)

FND9 adds up the totals of all the little neighbors to each C() element and assigns them to the complementary D() element in line 760.

LINES 800 TO 880

Remember that E() from up there in the DIMension statements? The program compares the entire C() array with the entire E() array, element by element, to see if there has been any change. As soon as it finds a difference in the corresponding elements of the two arrays, it EXITs from the nested loops. (It can only EXIT from one loop at a time, which is why we have the step effect in lines 840 and 860).

If the two arrays are precisely the same it runs into the END statement in line 880 and quits. The E() array is loaded each time through the program, so it always contains the data from the last iteration. Thus, this part of the program always compares the last iteration with the current iteration. Loading the E() array with current values takes place in the next part of the program, LINES 920 to 960.

LINES 1040 TO 1090

At last| The algorithm that makes it all work| Quite a letdown after all we've gone through to find that the whole thing is dependent on a couple of really simple IF/THEN statements.

Line 1060 takes care of making a dead cell come to life or stay dead. If the C() element is 0 (dead) and the corresponding element of D() is exactly 3, then the C() element becomes a 1 meaning it is alive. If the C() is 0 (dead) and the corresponding D() is anything else (0,1,2,4,5,6,7,or 8), the little mothering C() element stays dead.

Line 1070 takes care of the already living cells. If the C() element is alive (holds a 1), then if the corresponding D() element is 0 or 1, or more than 3 (it could be as much as 8) the cell dies, in the first case from loneliness and in the second case from overcrowding. If the C() element is alive (1) and the corresponding D() element is 2 or 3, the living C() element stays alive.

Really, very straightforward stuff.

LINE 1100

This line sends us back to 600 where the screen is cleared by printing the C9$ and the body count is neatly displayed with Xs and blanks.

LINES 1110 TO 1180

This is the target module for the ERRSET we talked about earlier. When an error occurs, such as the CONTROL-C, another menu is presented, this time offering a choice of either printing out the present status of the world to hardcopy, storing current values in that LIFEDATA file, or ending entirely with another CONTROL-C. Once an ERRSET is invoked, it is negated until reset. That's why a second CONTROL-C at this juncture will bring up the familiar READY after getting you out of the program. Using the ERRSET this way in almost any kind of program gives you great control over it, especially if it will run on for any length of time.

LINES 1200 TO 1290

A very simple WRITE of sequential data into a Type 3 data file, recording first the value of T, then using the Nested Loops to record the value of each C() element.

LINES 1330 TO 1410

Simple redirection of the screen display module to the printer, by using the |#1 command. Again, this could be done in the screen display module by assigning the output device to a variable (I like Z9 for that purpose ... I mean, if I ever write a program that exhausts all other

available variables and I really need Z9 for something, then I don't deserve to win $39 million in the lottery.)

LINES 1440 TO 1540

If you want to start with living cells in specific places for experimenting purposes, you would have selected that option up above and jumped down here. Lines 1460 to 1500 fill the world with 0s ... all dead cells. Then you specify which cells are to be alive by entering two numbers. You can enter as many live cells as you want, making little geometric patterns, writing your name, drawing the profile of Sher or Dolly Parton, etc. When you've got enough, enter 0,0 and it starts to run with your offspring. Gives one a powerful feeling.

LINES 1580 TO END OF LISTING

This is the DEFined FuNction we talked about before. I-1 identifies a cell in the row above the current row; J+1 identifies the cell immediately to the right; thus C(I+1,J-1) would identify the cell to the southwest of C(I,J).

If I-1 = 0 or J+1 = 37, those values are off the grid, which would ordinarily give you an OUT OF BOUNDS error. (Willie Gault makes his best catches OUT OF BOUNDS, if Jim McMahon can throw it that far ...). That's why we DIMensioned the arrays to be 2 units bigger in both directions than the actual arrays in use. When the I is either 0 or 19 or the J is 0 or 37, the program takes the value of the out-of-bound cell, which is always 0, and uses it, just as if it were a legitimate cell. That's really sloppy programming practice. Don't ever do it again. If I see you doing that, you're going to get a F in this course, Buster. People get grades of A if they make the sides wrap around and the top and bottom wrap around, so that the whole thing looks like a donut. The tosoidal effect, as it is called. But it's a pain to program, with all kinds of boring, repetitive statements such as IF J=37 THEN J=1, and you have to do that for every out-of-bound situation and combination of situations. Do it my way. But don't let me catch you.

LOOKING BACK ...

Let's see what we've done here. This relatively simple program gives you a slow but nice version of "Life." It also uses the following BASIC commands, statements, and techniques:

REM
CHR$()
STRING AND NUMERIC VARIABLES
DIM
ARRAY HANDLING
TAB
PRINT
|
OPEN
CLOSE
FOR/NEXT
EXIT
INT
RND()
ERRSET
DEF FNC
BOOLEAN OPERATORS
END
INPUT
INCHAR$
GOTO (the dreaded one|)
CR and TY
WRITE
RETURN
FNEND

and perhaps some others.

SPEEDING IT UP ...

If you want the program to run much faster, use the second listing, which doesn't give you any options ... it just runs the screen display. All we did was excise large pieces of the program, leaving only the heart. Or write the whole thing in APCBASIC and run it with the APC run-time semi-compiler. That runs even faster than the short version.

THINGS TO DO WITH "LIFE" ...

It's great fun to play around with this program. Change the size of the display by changing the size of the I and J loops throughout. Change the conditions under which cells die and are reborn by simply changing the logic in LINES 1060 and 1070. If you want to get really esoteric,

change the muliplier in the random generator to 3 or 4, so you'll have that many states for each cell, rather than just the 2 we've got here. Of course, adding more states will mean adding lines to the algorithm to take care of changing them and adding more symbols to the display modules. Four states for each cell really gets out of hand in a hurry. The eight neighbors can total anywhere from 0 to 32, for instance.

Once you've got this one down, it's a simple matter to to change everything to one-dimensional arrays and play with cell automatons, producing glider guns and other nifty stuff using totalistic tables. For a lot of lucid detail on those matters, see Dewdney's "Computer Recreations" column in the May, 1985 Scientific American.

SUMMING UP

Veteran North Star BASIC programmers probably have done "LIFE" long ago, in half the space and with twice the speed. God love them. But they weren't around when I was learning to program. For you guys just starting to fool around with BASIC, enjoy the above and realize that when the program YOU have written actually RUNS, it's the greatest feeling in the world ... well, maybe the second greatest.

# # #

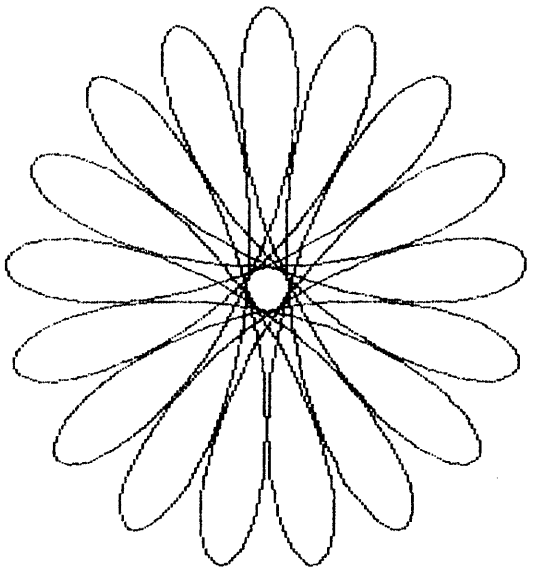
```
100 REM  THE SIMULATION "LIFE"--THIS VER. BY E.COUDAL  APRIL 1985
110 REM  WRITTEN  IN NORTH STAR BASIC 5.4 ON A HARD DISK NORTH STAR HORIZON
120 REM  DEAD CELLS (BLANKS) COME TO LIFE IF 3 OF 8 NEIGHBORS LIVE (1'S)
130 REM  LIVE CELLS (1'S) DIE IF LESS THAN 2 OR MORE THAN 3 NEIGHBORS LIVE
140 REM  THOSE RULES DESCRIBED IN SCIENTIFIC AMERICAN - MAY, 1985
150 REM
160 REM  INITIALIZATION ROUTINES FOLLOW
170 REM
180 C9$=CHR$(27)+CHR$(43)\ REM ASSIGN CLEAR SCREEN FOR SOROC 120 TO C9$
190 T=0 \ REM INITIALIZE THE ITERATIONS COUNTER
200 !C9$\!\!\!\REM CLEAR SCREEN AND THREE BLANK CELLS
210 DIM C(20,38)\REM DIM THE ARRAY THAT TELLS IF CELL IS ALIVE OR DEAD
220 DIM D(20,38)\REM DIM ARRAY THAT HOLDS TOTAL OF 8 NEIGHBORS
230 DIM E(20,38)\ REM DIM AN ARRAY TO COMPARE TO THE D() FOR NO CHANGE AND END
240 REM
250 REM
260 !TAB(21),"THE HISTORIC SIMULATION OF 'LIFE', WHEREIN"
270 !TAB(17),"INDIVIDUAL CELLS DIE OF OVERCROWDING OR LONELINESS"
280 !TAB(20),"--BUT ARE REBORN IF THINGS ARE JUST RIGHT..."
290 !\!\!
300 REM
310 REM  GET A NEW START, CONTINUE OLD ONE FROM STORED DATA,OR INPUT
320 REM
330 PRINT "How do you want to start:"\!
340 !"      --With a (R)andom world"
350 !"      --By resuming a stored (P)rogram"
360 !"      --By manually inputting the (L)iving cells"
370 !\!"Type R,P,or L to select: ",\H$=INCHAR$(0)\!H$
375 !"Initializing...."
380 IF H$="R" THEN 540
390 IF H$="P"THEN 420
400 IF H$="L"THEN 1460
410 GOTO 330\ REM  GO BACK TO QUESTION IF INCORRECT SELECTION
```

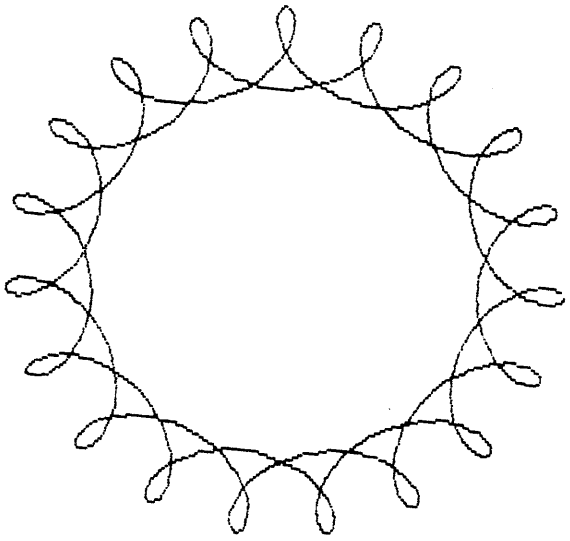
```
420 OPEN #1,"LIFEDATA,2"\ REM THE DATA STORAGE FILE
430   READ #1,T
440     FOR I=1TO 18
450       FOR J=1 TO 36
460         READ #1,C(I,J)\ REM READ THE STORED VALUES INTO THE C() ARRAY
470       NEXT J
480     NEXT I
490 CLOSE #1
500 GOTO 590
510 REM
520 REM FOLLOWING LOOPS GENERATE RANDOM LIVING AND DEAD CELLS TO START
530 REM
540 FOR I=1TO18
550   FOR J=1TO36
560     C(I,J)=INT(2*RND(-1))\ REM EITHER A 0 (DEAD) OR A 1 (ALIVE)
570   NEXT J
580 NEXT I
590 ERRSET 1110,L,E\ REM TRAPS CONTROL C AND JUMPS TO STORAGE OPTION
600 !C9$
610 REM
620 REM FOLLOWING LOOP IS SCREEN DISPLAY; AN X FOR LIVING, BLANK FOR DEAD
630 REM
640 FOR I=1TO18
650   FOR J=1TO36
660     IF C(I,J)=0 THEN !"  ",
670     IF C(I,J)=1 THEN !" X",
680   NEXT J\!
690 NEXT I\!
700 T=T+1\!T," ITERATIONS"
710 REM
720 REM FOLLOWING LOOP ADDS THE TOTAL OF ALL 8 NEIGHBORS AND ASSIGNS TO D()
730 REM
740 FOR I=1TO18
750   FOR J=1TO36
760     D(I,J)=FND9(I,J)\ REM A DEF FUNC LIKE THIS MAKES IT EXECUTE FASTER
770   NEXT J
780 NEXT I
790 REM
800 REM NEXT LOOP CHECKS TO SEE IF THERE HAS BEEN ANY CHANGE
810 REM
820 FOR I=1TO18
830   FOR J=1TO36
840     IF D(I,J)<>E(I,J)THEN EXIT 860\ REM EXIT ONLY ONE LOOP AT A TIME
850   NEXT J
860     IF D(I,J)<>E(I,J)THEN EXIT 920\ REM EXIT FROM THE TOTAL NEST WITH THIS
870 NEXT I
880 !"NO CHANGE AT ALL SINCE LAST ITERATION."\END
890 REM
900 REM NEXT LOOP STORES THE CURRENT NEIGHBOR TOTALS IN THE E() ARRAY
910 REM
920 FOR I=1TO18
930   FOR J=1TO36
940     E(I,J)=D(I,J)
950   NEXT J
960 NEXT I
970 REM
980 REM
990 REM THE ENGINE THAT MAKES IT GO FOLLOWS.  CHECKS WHETHER THE CURRENT CELL
1000 REM      IS ALIVE (1) OR DEAD (0) IN THE ARRAY C() AND THEN CHECKS WHAT THE
1010 REM      TOTAL OF THE NEIGHBORS IS IN THE ARRAY D().  CHANGES THE STATE OF
1020 REM      C() ELEMENT ACCORDING TO THE TWO RULES IN LINES 950 and 960
1030 REM
1040 FOR I=1TO18
1050   FOR J=1TO36
1060     IF C(I,J)=0 AND D(I,J)=3  THEN C(I,J)=1
1070     IF C(I,J)=1 AND D(I,J)<2 OR D(I,J)>3 THEN C(I,J)=0
1080   NEXT J
1090 NEXT I
1100 GOTO 600\ REM GOTO THE TOP AND DO IT AGAIN
```

```
1110 !C9$\!\!\!\!\!\!
1120 REM
1130 REM  OPTION TO PRINT HARDCOPY OF CURRENT POSITION OR STORE DATA FOR LATER
1140 REM
1150 INPUT "HARD COPY OR DATA STORAGE (H/S)? ",H$
1160 IF H$="S" THEN 1200
1170 IF H$="H" THEN 1330
1180 GOTO 1150\ REM TRAPS INCORRECT ANSWER AND ASKS QUESTION AGAIN; 2D ^C ENDS
1190 REM
1200 OPEN #1,"LIFEDATA,2"
1210 WRITE #1,T\ REM STORE THE ITERATIONS COUNT AS FIRST DATA ELEMENT
1220 FOR I=1TO18
1230   FOR J=1TO36
1240     WRITE #1,C(I,J)\ REM STORE THE C() ARRAY VALUES AS OF RIGHT NOW
1250   NEXT J
1260 NEXT I
1270 CLOSE #1
1280 INPUT "RETURN TO RUNNING PROGRAM? ",H$
1290 IF H$ = "Y" THEN 590 ELSE END
1300 REM
1310 REM FOLLOWING LOOP IS PRINTER DISPLAY; AN X FOR LIVING, BLANK FOR DEAD
1320 REM
1330 FOR I=1TO18
1340   FOR J=1TO36
1350     IF C(I,J)=0 THEN !#1,"  ",
1360     IF C(I,J)=1 THEN !#1," X",
1370   NEXT J\!#1,CHR$(13)
1380 NEXT I\!#1," "
1390 T=T+1\!#1,T," ITERATIONS"
1400 INPUT "RETURN TO RUNNING PROGRAM? ",H$
1410 IF H$ = "Y" THEN 590 ELSE END
1420 REM
1430 REM
1440 REM MANUAL INPUT OF LIVING CELLS
1450 REM
1460 FOR I=1TO18
1470   FOR J=1TO 36
1480     C(I,J)=0\ REM FILL THE WORLD WITH DEAD CELLS
1490   NEXT J
1500 NEXT I
1510 !\!\!
1520 INPUT "Location of living cells?. Input as maximum 18,36 (0,0 to end)",I,J
1530 IF I=0 THEN 590 ELSE C(I,J)=1
1540 GOTO 1520
1550 REM
1560 REM  FOLLOWING DEF FNC TOTALS THE EIGHT NEIGHBORS AND RETURNS TOTAL AS D9
1570 REM
1580 DEF FND9(I,J)
1590     D9=C(I-1,J-1)+C(I-1,J)+C(I-1,J+1)+C(I,J-1)\ REM DO THIS ON 2 LINES TO
1595     D9=D9+C(I,J+1)+C(I+1,J-1)+C(I+1,J)+C(I+1,J+1)\REM AVOID LENGTH ERROR
1600 RETURN D9
1610 FNEND
```

```
180 C9$=CHR$(27)+CHR$(43)
190 T=0
200 !C9$
210 DIM C(20,38),D(20,38),E(20,38)
540 FOR I=1TO18
550   FOR J=1TO36
560     C(I,J)=INT(2*RND(-1))
570   NEXT J
580 NEXT I
640 FOR I=1TO18
650   FOR J=1TO36
660     IF C(I,J)=0 THEN !"  ",
670     IF C(I,J)=1 THEN !" X",
680   NEXT J\!
690 NEXT I\!
700 T=T+1\!T," ITERATIONS"
740 FOR I=1TO18
750   FOR J=1TO36
760     D(I,J)=FND9(I,J)
770   NEXT J
780 NEXT I
1040 FOR I=1TO18
1050   FOR J=1TO36
1060     IF C(I,J)=0 AND D(I,J)=3  THEN C(I,J)=1
1070     IF C(I,J)=1 AND D(I,J)<2 OR D(I,J)>3 THEN C(I,J)=0
1080   NEXT J
1090 NEXT I
1100 GOTO 640
1580 DEF FND9(I,J)
1590     D9=C(I-1,J-1)+C(I-1,J)+C(I-1,J+1)+C(I,J-1)+C(I,J+1)+C(I+1,J-1)+
1600 RETURN D9                                                  C(I+1,J)+C(I+1,J+1)
1610 FNEND
```

Listing 2: LIFE cut short





ZCPR

Bob Bloom
5 McCord Drive
Newark Delaware 19713

(This is the second of four articles submitted by Bob Bloom. The first article, on installing TEAC half-height drives, appeared in the last issue. --Ed.)

This second article does not have that much about the TEAC drives, per se. It is more an extension of the first article to tell how to make the most of the new drives. Therefore, the main subjects all involve how to patch a CP/M system (all three parts), installing ZCPR, more detail on Plouffe's disk modifications, and lots of small details to get on started in the fine art of system hacking.

So this part II is not really directed to the person who already knows how to hack a BIOS - it's aimed at 'myself.' Let me clarify that - These are instructions (knowledge) that I wish I had two months ago. I did know what CCP, BDOS and BIOS meant but was (and still am) a weakling in assembly language programming. (The sum total of everything I've written in assembly is less complex than the first chapter in any CP/M programming book.) However, if you are still afraid of assembly language programming - find someone else to help you. You do have to learn how to read the stuff even if you don't write programs. Also, I was very ignorant of how the Horizon really was working, and what actually was happening in the various steps of the previous installation report.

Specific mention is made of system patches by Conn (ZCPR2), Peterson (Archive), Plu*Purfect (Public) and Plouffe (special disk formats). This are used as examples for the instruction. The next article (the third) will deal with the actual actions to install this patches.

My reference throughout assembly language has been Rodnay Zaks, How to Program the Z80. My copy has a Radio Shack cover but is exactly the same as the Sybex version (but cheaper). There is not much (or any in fact) about CP/M calls in this book but it can't be beat in its description of the internals of the Z80 and the individual opcodes. There is also an excellent Z80 to 8080 and 8080 to Z80 mnemonic equivalence chart included in the appendices.

So, if you were not frightened by the previous paper, maybe even went out and bought some drives, but maybe weren't quite sure of the steps needed to customize to your particular configuration, this is for you. I probably won't cover your exact configuration, but you probably will be able to figure out why you can't get there from here (or whatever).

* * *

As you should know, CP/M is divided into three parts: CCP, BDOS, and BIOS. The CCP is the part that Rich Conn has completely hacked with his ZCPR1, ZCPR2, and ZCPR3. The BDOS is the supposedly untouchable DRI-written module, common to all CP/M systems. I have two patches for the BDOS - ARCPATCH and PUBPATCH (which was just released to the Public Domain). The BIOS is where all the i/o action is - that's the code that interfaces the computer to the world. For instance, the code to input and output to the console in contained in the BIOS.

I'm sure most of you have read that BYTE's resident isoclast (Jerry Pournelle) has decried the lack of BIOS source code given to users. Unfortunately, North Star is one of those that don't give their source code out. However, they have done something that is almost as nice.

North Star has divided their BIOS into three areas: a disk i/o area for which source code is not given, a 'USER' area containing the code to the serial and parallel ports and initialization, and a uninitialized BIOS buffer area. The source code to the 'USER' area **is**

provided, and can be found on your North Star CP/M system disk under the name 'USER.ASM'. This is this code that is most often hacked.

The USER area starts with a jump table, somewhat like the one at the start of the BIOS. However, only the i/o related entries are in this table. The real BIOS jump table has the i/o related entries pointing at the USER table, so it's nearly like doing a direct patch. One other thing: the CPMGEN patches the list device output jump address in the BIOS table to the correct USER area jump depending on whether you picked a serial or parallel printer (there is code in the default USER area to support both a serial and parallel printer).

The other, non-selected driver is never used, and so it a prime candidate for code crunching. There are also some code in the North Star-supplied USER.ASM that is not even addressed anywhere. For example, there is code for parallel port input and second serial port input that don't have jumps pointing to them.

For an example of when it is necessary to patch the BIOS, consider ZCPR2: on cold boot it is necessary to set up a default path and to zero the external command line buffer. (If you don't know what that is, read the ZCPR2 documentation. If you don't care, I'm probably beyond you anyway.) The code to do that is simply tacked on to the end of the USER.ASM file and an appropriate call statement is placed in the cold boot initialization routine code. (This code was originally supplied by F. Wancho - thanks Frank, could have never done it without 'ya.)

At the top of the USER.ASM file put:

```
MCMDBUF EQU 0EC00h ; ZCPR2 Multi Command-line (MLC) Buffer
```

and patch in the appropriate origin address for the size system you are generating:

```
 ORG 0FA00H ;ORIGIN of USER area
```

Then insert and tack the following stuff on to the end of the file:

```
<insert this call to the initialization>
 ;
 ;DO ZCPR2 COLD BOOT INITIALIZATION TO
 ;    SET UP DEFAULT PATH AND ZERO MULTI-CMD LINE BUFFER:
  CALL ZINIT ;ZCPR2 INITIALIZATION

<before the original code>
 ;
 ;WRAP IT UP BY SENDING A CR TO THE PRINTER & EXIT
  MVI C,0DH ;A CARRIAGE RETURN
  JMP USERBASE-700H+15 ;GOTO PRINT JUMP IN BIOS VECTOR

<then add this>
 ;*****************************************************

  ORG 0FB30H ;EXTERNAL PATH
 ;
 ; External PATH is here:
 ; (my default path is $$ A0, but it may be as long as you want)
 ;
 pinit: db '$','$'  ; current disk/current user
  db '$',0  ; current disk/user 0
  db 1,'$'  ; A:/current user
  db 1,0  ; A0:
  db 1,10  ; A10: for finding NAMES.DIR
  db 0  ; end of list
```

```
;
; Rest of space is available for PATH info up to USERbas+200h
; So we use this space below for the one-time init of the MLC
; only done at cold boot
;
ZINIT LXI D,mcmdbuf     ;address of the buffer
 LXI H,clinit       ;default for startup
 LXI B,5            ;transfer 5 bytes
 DB 0EBH,0B0H       ;ldir (too lazy to write a separate routine)
 ret
clinit dw mcmdbuf+4
 db 200
 db 0               ;null to close off

;******* END OF INITIALIZATION CODE *********
```

Note the ending comment in the original USER code (not copied here)! The USER area is only 200h long in the best of cases and can be as short as 160h long if one has a non-standard PROM or hard disk. So watch the length. It does become a factor in the ZCPR3 initialization.

So that's it. During the cold boot this code is called and zeros the command line buffer and copies a default path into the correct location.

Now see Fig. 1 for a memory map. Note that the '64k' system is closer to 61k than really 64k. (That's why it's in quotes - Maybe North Star thought no one would notice.) The odd positioning of the disk controller reserved memory addresses at E800h causes much grief because CP/M expects continuous memory - CCP, BDOS, BIOS, all in a row. (The E800h is from ancient history, computer relative, when RAM came in 4k block and was *expensive*. There are available PROM sets to move it to the top of memory, but as it's working fine as is ... why change?)

The '64k' gets around this by duplicating the BIOS jump table directly below the PROM which directs the code to memory above the PROM. Quite clever, really.

The ZCPR2 system takes no added space on a standard Horizon - the buffers go in normally blank memory. The ZCPR2 utilities are nice, but I find the standard PD utilities (DU, SD, NSWP, NULU) more practically useful. Unless one has a hard disk or a RBBS, I find the named directories, the file protection, wheel stuff, and help files not worth it. On the other hand, all of this stuff takes NO MORE ROOM, so why not try it? Things like the built-in 'go' and 'list' commands, the search path, and multi-commands on one line are neat. I haven't got into the menus yet, as I by nature tend to dislike them. But if you are trying to set up a system for a computer novice who really doesn't care how 'the machine' works, it really deserves a closer look.

The full configuration ZCPR3 with all the bells and whistles yields a 58k CP/M system. To do this, I've backed the North Star USER area up against the E800h PROM and used all of the memory above the PROM for buffers. If one can manage the ZCPR2 installation, ZCPR3 is just longer, not harder. Having MAC is almost a must. On the other hand, one will be a bit slow in learning all of the ZCPR3 new utilities and assets, which are considerable advances over ZCPR1 or ZCPR2. I was happy with ZCPR1 - and guess that ZCPR1 was very similar to Plouffe's special CCP. Implementing all the various options keeps me off the streets however. But Rich Conn put so much obvious work into ZCPR3 I feel that you should at least try it. DU3 looks like a considerable expansion over standard DU and some of the new utilities are really neat (menus, rcp's, shells). Most ZCPR3 utilities will not work with standard CP/M as they require hooks to the ZCPR3 system.

Now it's time to talk about the actual patch procedure - It easily took me several months to really understand the 'offset' parameter of the 'r'ead command of DDT. If I can make it easier for someone else, so much the better. When you know why the ARCPATCH is read-in with a simple 'r', Z2USR.OVR (the Z80 version assembled with M80/L80) with 'r3100', NEWFRM.HEX with 'r3800' and ZCPR2 with 'r4400', you're in.

In general there is the way to patch the CP/M system image in DDT without saving it to disk. As far as I know, this method will only work with Horizon systems so is not applicable to other computers:

1. Run SYSGEN (or GENSYS if using plouffe's disk mods).

2. Read system from a disk.

3. Exit with an control-C (not return|)

4. Startup DDT without a argument. The sysgen CP/M image is in memory.

5. Patch with Ifilename, Roffset, etc. as required.

6. Exit DDT with an control-C or G0.

7. Re-Run SYSGEN (or GENSYS if using plouffe's mods).

8. Enter return for source (the source is in memory already.)

9. Specify drive number to store new system on.

10. ⟨cr⟩ to cold boot on new system, providing it's in drive 1

This does save some time if you have to create many systems before getting one right! It works because the CP/M system image loads at 1500h, which is higher than in 'normal' DRI systems, and this is ABOVE the last address of SYSGEN/GENSYS and DDT. Therefore, loading these DO NOT affect the image in memory. Between steps 4&5 is the place where one would normally save the sysgen image and re-read it in with a DDT CPM.COM. Seems so logical I wonder why DRI didn't think of it. Normally I save a vanilla CP/M system ('SAVE 51 CPM.COM' between steps 3 and 4 above) and do all the patches on it starting from a 'DDT CPM.COM'. Submits are good for multiple gens too. One can also save the patched image between steps 6 and 7 with a 'SAVE 51 NEWCPM.COM' if you want to keep it for posterity.

SYSGEN or Plouffe's GENSYS (all basically the same program) reads 'the system' or CP/M image into memory at some designated place. Also, one has to keep in mind the final system destinations of the memory map in Fig. 1. These destinations in the Horizon are as follows:

SYSGEN	58k	'64k'	
1500	C500	D100	CCP
1D00	CD00	D900	BDOS
2B00	DB00	F300	BIOS (disk)
3200	E200	FA00	BIOS (USER area)

(This information is actually included in the North Star CP/M documentation, but have fun finding it!) In the case of the '64k', (something) takes care of duplicating the BIOS jump table from F300h to E700h.

Explanation of the differing offsets:

ARCPATCH: If you look closely at the actual .ASM code you will see that it was written to load directly into the sysgen image and no offset is needed. The source to the actual ARCHIVE program also has the source to the BDOS patch which will load to the system location.

Z2USR: This was written in Z80 code, assembled with M80 and linked with L80 to .COM-like code. Therefore, no .HEX code was generated to indicate where it should be loaded and DDT, unless told otherwise, will load it at 0100h. But we want it to load at 3200h in the USER code area, so you have to use an offset of 3100h. i.e. 0100h (DDT default) + 3100h (given offset) = 3200h (location of USER area in SYSGEN image). If this was written in normal 8080 code and assembled to HEX, one would use an offset of 3800h. (user exercise: why?)

NEWFRM: This creates normal .HEX code to be loaded at F300h (it's a BIOS patch), which is where DDT will load it if not told otherwise. But we want it to load at 2B00h so one should use an offset of 3800h. i.e. F300h (DDT default) + 3800h (given offset) = 2B00h (address of the BIOS in SYSGEN image with wraparound).

ZCPR2: This also creates normal .HEX code to be loaded at D100h (it's a CCP replacement), which is where DDT will load it if not told otherwise. But we want it to load at 1500h so one should use an offset of 4400h. i.e. D100 (DDT default) + 4400h (given offset) = 1500h (address of the CCP in SYSGEN image with wraparound).

There, that wasn't too bad was it? (But there is worse to come.) "It is left as an exercise to the reader to figure the offsets for a 58k system."

All seems fine and good until the patches start to interfere with each other. For instance, I mentioned in the previous article that setting both alternate quad options true caused a problem. You might have experienced different as it depends on the structure of the USER area, order of patching, and options. Any which, setting 'quad' true will interfere with the Z2USR patch.

Suffice it to say, all of the patches work correctly by themselves patched into a plain vanilla CP/M image. But that's no fun - I wanted them ALL to work - ARCPATCH, PUBPATCH, NEWFRM, and ZCPR2/3, and with all options active! Well, you can't get there from here, but I got close:

58k system vs. '64k' systems: The '64k' system gives one 3k more TPA (49k vs. 52k) but some programs do fail under the split BIOS, ZEX and EX in particular. (They can be made to run with a patch-on-the-fly, however.)

Setting 'octal' true in NEWFRM: I've not noticed any problem in this patch interfering with any other of the patches mentioned here. This is where I was lucky when first setting up the system and not really knowing what I was doing.

Setting 'quad' true in NEWFRM: As part of this patch, the jump table at the start of the BIOS is changed to point directly into the USER area and the jump table at the start of the USER area is overlaid with other non-related code. However, the addresses in NEWFRM to patch the BIOS table with are only for the 'standard' North Star configuration. If any other patches are made to the USER area, these addresses probably will change. It is not hard to change them in the NEWFRM code, just be sure that you do! (I must have read that section in NEWFRM ten times before realizing what it was trying to tell me. One has to assemble th USER area and create a .PRN file to see the locations. Then use this locations to patch the NEWFRM.ASM file.) Secondly and potentially worse, the 'QUAD' true patch puts code at USER+115h, so if your modified USER area is larger than that, something has to be done. (What should be done I really don't know, I haven't got quad drives, but do have enough other problems without worrying about this one. You can if you want!)

ARCPATCH and PUBPATCH seem to be be exclusive - both cannot be active at one time. (PUBPATCH uses the high bit of the second character of the filename as a 'public' attribute - if it is set and the patch is installed, the file is called 'public' and accessible in all user numbers, INCLUDING OVERLAY FILES! Thus, for example, WordStar and its overlays can all be marked public and will run correctly from any USER area. Wonderful patch for hard disks - see ⟨CPM.PUBPATCH⟩PUBPAT.LBR in simtel20.)

In placing any patch into the CP/M SYSGEN image, be knowledgeable of where the patch should load both in the working system and the SYSGEN image so that the correct offset is used. Hint: providing your USER area code is less than 200h bytes, DDT should never show a number larger than 3400 for highest address. This address is the nominal top of the USER area in the SYSGEN image. In the split BIOS '64k' system some patches will load above the PROM, some below and the same offset can not be used for both!

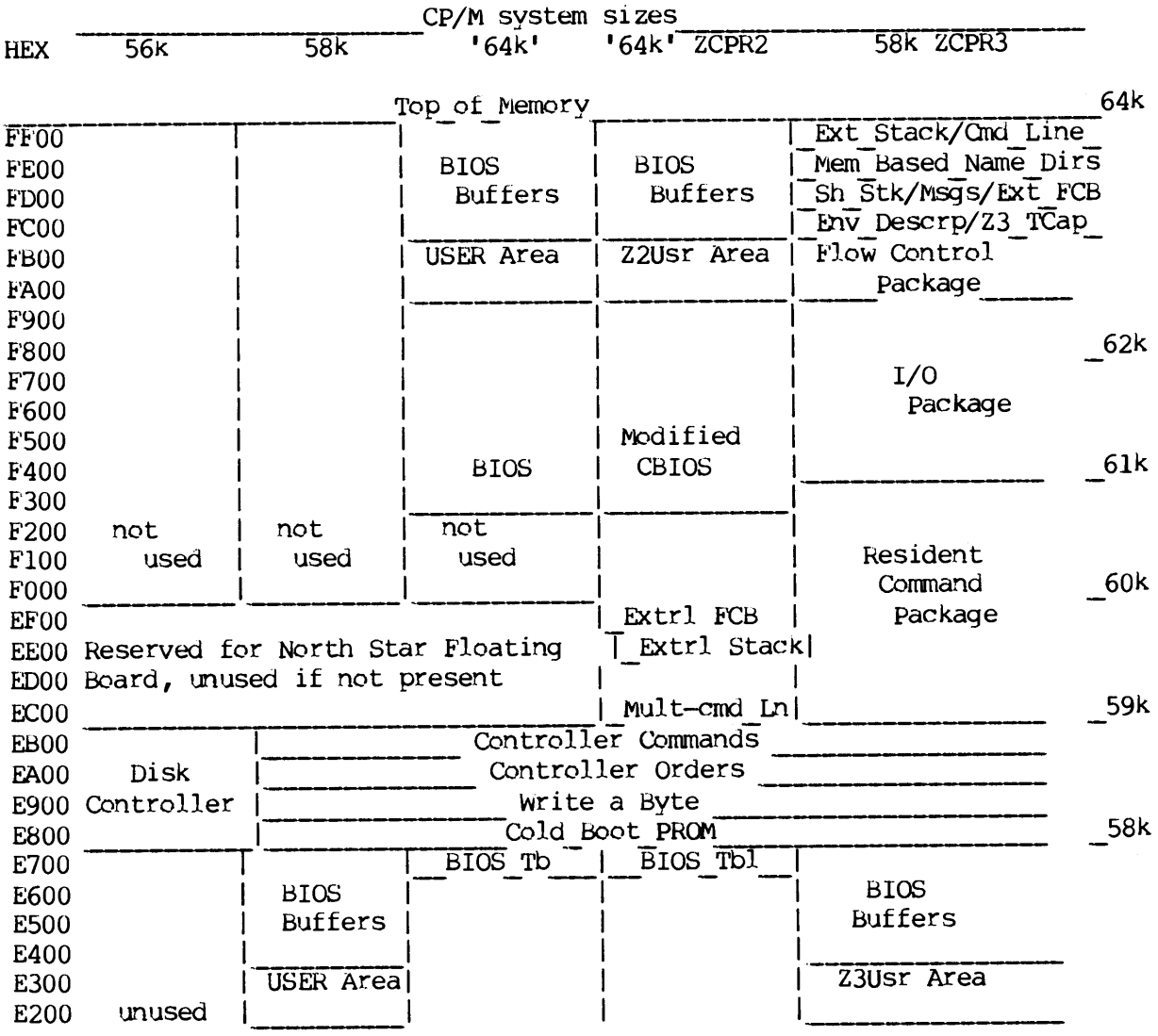
This part has gone on long enough, I'm up to over 20k and still have lots to say about ZCPR3. Therefore, I am going to cut this thing off now and continue in a separate part specifically on ZCPR3 and the Horizon. If I have at all turned you on to patching new and exciting things into your old stodgy plain CP/M - stay tuned. If I haven't - well, go back to sleep.

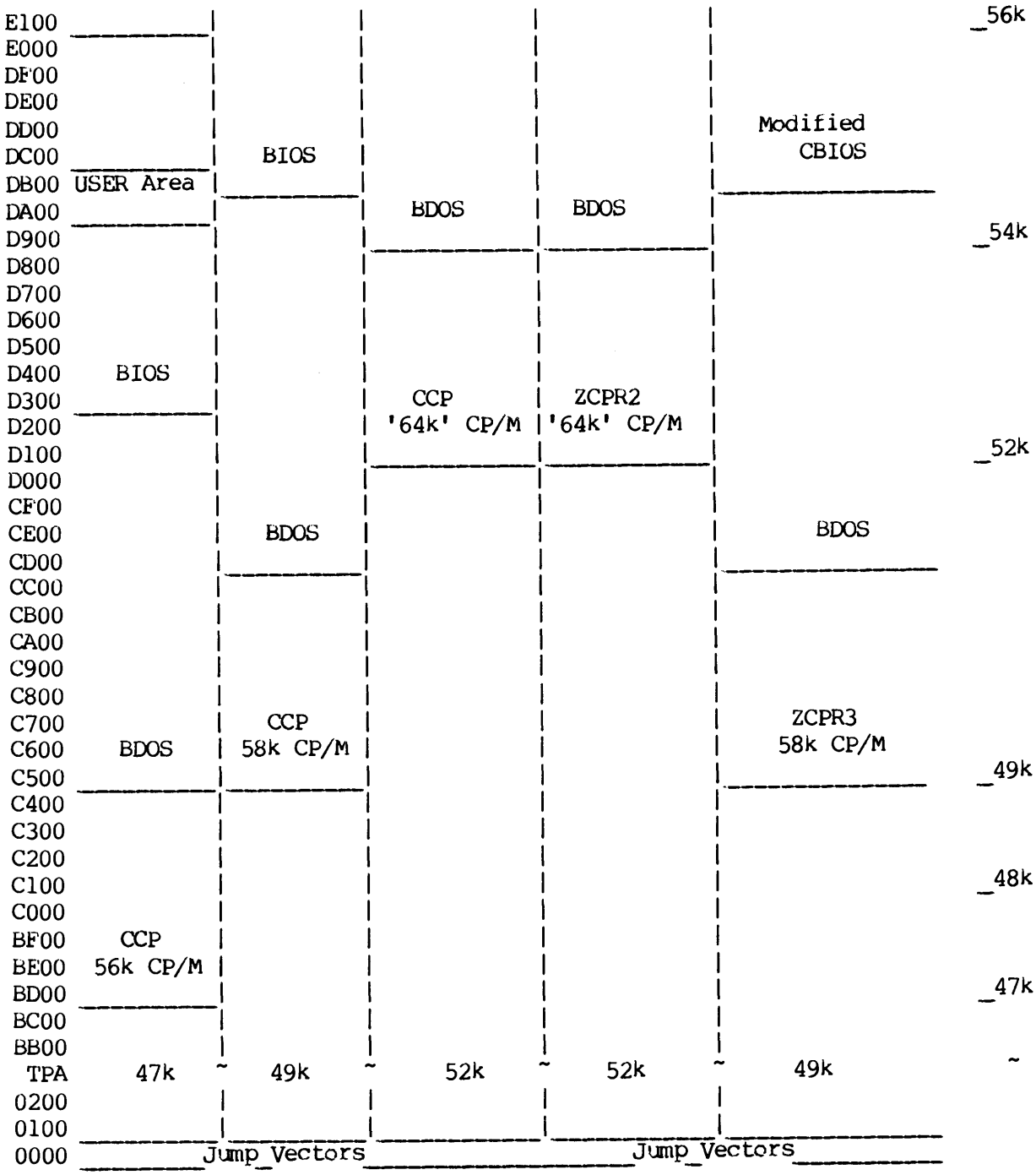
# # #

(See Vendors Column for more on ZCPR - Ed.)

Fig. 1

CP/M MEMORY MAPS FOR THE HORIZON





Bottom of Memory

A Fable

By a Friend of INSUA

Once upon a time, many eons ago, there dwelt in the land of Og a tribe of Little People. And also in the land roamed many fierce beasts; among them: the wolf, the bear and the sabre-toothed cat. The Little People were sore afraid of the beasts and huddled together in their caves at night least they be devoured by them.

Now, there also lived in the land a giant of a man called Ig Ba Mu. Mu was gentle and kind to the Little People and they, in turn, looked upon him as their protector.

Because of his great size, Mu feared nothing and wandered freely through the forests and beside the rivers of Og. When he encountered the beasts, they did not attack him but cowered and hid; and some even ran. Mu tolerated the beasts and did not harm them. Instead, he fed them scraps of food and talked softly to them.

Now, Mu was keen of mind and also observant. He observed that as he fed and talked to the beasts they became tame; and he had an idea.

Mu selected some members from a pack of wolves and carried them to his cave. There he fed them and talked softly to them. And when they had become tame, he clipped some of their hair and set his brand upon their flanks. And then he offered them to the Little People to act as guardians of their caves.

And the Little People accepted these animals from Mu because they trusted him. And they found that with regular feeding and soft words the animals obeyed commands and did indeed guard the entrances to their caves. And in honor of Mu the Little People named these animals dogs. And the Little People of Og traded with Mu for many of his dogs, as fast as he could tame them. And for the first time the Little People slept soundly in their caves at night and were not afraid. And they came to love their dogs even as they still feared the wolves.

And Mu observed all these things and wondered about them. How could the Little People have such affection for their dogs and such fear of the wolves when, as he knew, there was little difference between them? But Mu kept these things in his heart and only smiled.

Now, a few of the Little People observed that sometimes their dogs became lazy and needed prodding to remain alert. And so they began to search for a better animal. And they went out into the forests of Og and encountered the bear and the sabre-toothed cat. And they discovered what Mu had discovered: that with feeding and soft words they became tame. And they brought these animals to their caves and found that they were better guardians than the dog. The cat, in particular, seemed never to sleep.

And the few tried to tell the others what they had found, but the others refused to believe and would not listen. How could the bear and the cat be superior to the dog, they asked, when they had not the brand of Mu upon their flanks?

And so the Little People (except for the few) continued to seek the dogs from Ig Ba Mu. And Mu became famous throughout the land; and his cave was filled with gold and ivory which the Little People brought to him to trade for his dogs.

And Mu marveled at the trust the Little People had in him and for his dogs, for he knew what the few had found: that the bear and the cat were superior, but Mu said nothing for he was becoming famous and wealthy.

And the Little People continued to praise Mu and his dogs. And each night, as they lay in their caves preparing for sleep, they gazed upon their dogs guarding the entrances and they felt safe and comforted. For they could clearly see the animals sleek, clipped, coats (which disguised the fact that they were really wolves) but most of all they could see the brand of Mu upon their flanks which read: IBM.

# # #

How to Copy Files Between North Star Disk Formats

By Peter Midnight

One of the most confusing aspects of the North Star disk system is its multiplicity of diskette formats. There are single and double density diskettes. There are single and double sided diskettes. And there are various diskette formats for each of North Star's operating systems: Pascal, DOS, CP/M, and MSDOS. And that's not to mention North Star's application software packages, which don't run under any of those systems, or the experimentation some people have done with 96 track per inch drives.

For some people this confusing array of formats has become a serious problem, because they have files on diskettes of one operating system's format that they want to copy onto diskettes of another operating system's format and there is no obvious and easy way to do that.

Fortunately, North Star never came up with more than two ways to write a sector on a diskette: single density and double density. All of their operating systems on the Horizon use just these two formats. And the Advantage is even simpler because it uses only double density sectors. This means that, in fact, all of North Star's operating systems--Pascal, DOS, CP/M, and MSDOS--can be coerced into reading each other's diskettes. They just don't know how to find each other's files.

Some people have taken advantage of this consistency among North Star's operating systems by creating programs that will run under one system and use the drivers in that system to read and copy the files of another system. For example, there have often been programs under CP/M which used BIOS calls to read the directory of a DOS format diskette in one drive, find all the type 2 files on that diskette, and copy them into CP/M files on the diskette in the other drive, adding .BAS to the name of each file. Such programs can be a great convenience if they happen to copy the right group of files from a diskette of the right format to a diskette of the right other format and can do so on the machine that is at hand.

Unfortunately, not everyone has just the programs they need to copy the files they want to the format they want on the machine they have. But the real problem is that the very existence of such programs has led some people to think that they must be necessary. That isn't true at all. There are several good and easy ways to move files around among diskette formats without the aid of any special tools other than those that were supplied with the operating systems involved.

* * *

The easiest of these techniques also happens to meet the most common need, that of transferring programs between North Star BASIC, under DOS, and any of the many imitations of North Star BASIC, which all seem to run under CP/M. The reason this is so easy is that it does not require any understanding at all of either of the diskette formats involved. It works because BASIC programs are always small enough to fit in RAM. And the same technique can be used on any file of which this is true.

A little preparation is needed before you can use this technique. First, you will need to prepare two boot disks, one for DOS and the other for CP/M. It is important that neither of these disks performs a hard disk boot, as doing so wipes out RAM under either operating system. It is also important that the DOS be loaded somewhere near the top of whatever memory you have. If you are using an Advantage, this is already taken care of. If you have a double density Horizon (all quads are double density) and any of the later versions of DOS, then you

have a BASIC program, called MOVER, which will help you prepare a high loading DOS for your system. If you have a single density Horizon, this technique will only work for you on files of less than 8 Kbytes or 32 blocks, because your DOS loads at 2000H. The best way around this problem is to upgrade to double density. You may find that the requirements for one or both of these disks can be met by the boot disks you normally use.

The procedure for this technique is to read the file into RAM with one operating system and write it onto another diskette with the other. Wasn't that easy? Let's go through it again in a little more detail.

First, boot up the appropriate operating system for the diskette on which your file already exists. Then put that diskette in a drive.

Second, load your file into RAM at 100H. If it is a DOS file, just use the command

LF filename 100

If it is a CP/M file, change its name to something.COM and type

DDT something.COM

Third, make a note of the size of the file. If it is a DOS file, use the LI command. Of the three numbers listed for your file, the one in the middle is the size. If it is a CP/M file, you should see two hexidecimal numbers from DDT, one of which is 100. Subtract 1 from the other number. Then knock the 7F or FF off the end and convert what's left from hexidecimal to decimal.

Fourth, put the other boot disk in drive one and reboot the computer. A word of caution here: You can be more certain of getting an accurate copy with this technique if you do not use a reset to cause your computer to reboot at this point. If you are in DOS, type

IL

If your DOS is too old for that to work, type

JP E800

If you are in DDT on a Horizon, type

GE800

If you are in DDT on an Advantage, go ahead and reset it. (You can't really reset an Advantage. That's really just a nonmaskable interrupt. That's why it's safe to use it in this case.)

Fifth, before doing anything else that might disturb the contents of your RAM, save the file on another diskette. To do this you will need the size number you found in step three. If you are now in DOS, type

CR something size

Then type

SF something 100

If you are now in CP/M, type

SAVE size something

(The size number used here is not consistent with any of the other ways in which CP/M expresses the size of files. Instead, by a strange coincidence, the SAVE command requires that the file size be expressed in terms of North Star DOS blocks.)

* * *

A more interesting technique is to create a diskette which is simultaneously both a DOS diskette and a CP/M diskette, a Pascal diskette, or an MSDOS diskette. Such a diskette can contain one or more files which are equally accessible to both operating systems. This technique requires careful planning. And there are some limitations to what can be done with it. But once this technique is mastered, it can be extremely useful.

The biggest limitation to this technique is that for CP/M files of more than one sector it only works in single density. The reason for this limitation is that all of the double density CP/M diskette formats are skewed. That means that the sectors on the diskette are not used in numerical order but in a different sequence. Only the single density CP/M format uses the sectors in the same sequence as DOS, Pascal, and MSDOS.

The result is that the largest CP/M file that can be shared in this way is 78 Kbytes or 312 blocks.

To make a diskette for two operating systems, first format the diskette under CP/M, Pascal, or MSDOS. The formatting program under each of these systems creates both the sectors and the data structures for its respective operating system. In addition, they each also create a DOS directory on the same diskette. The purpose of this extra directory is to allow the format of any diskette to be identified by simply listing its DOS directory.

The second step is to create the desired file or files on the diskette using the same operating system under which it was formatted. Care must be taken at this step to ensure that no files are deleted during this process. If any file is deleted, the dynamic allocation schemes used by CP/M, Pascal, and MSDOS will attempt to reuse the deleted file's space on the diskette. This can make the locations of files on the disk difficult to predict and can make either CP/M or MSDOS break up a file into noncontiguous pieces. The surest way to avoid this difficulty is to create the files on this diskette by copying existing files of exactly the correct size from some other diskette or hard disk.

The hard part of this technique is to finally create entries in the DOS directory which refer to exactly the same areas of the diskette that are, in fact, the files described in the other directory. This is done with the CR command in DOS by specifying both the length and the starting disk address with the command. Getting all the numbers right in the DOS directory really does require an understanding of the other file system, as described in the manuals for that system. It also usually involves at least a little bit of trial and error. If the other system is CP/M, the DIRDUMP program can be used to determine the starting address of each file and to verify that each file occupies a contiguous group of sectors.

Once the two directories on a diskette match, and as long as neither of them is changed, the files on that diskette can be both read and written under both DOS and the other operating system that was used. One use for such a diskette is for copying the same file repeatedly from one system to the other, as you would when developing a program to run under one operating system with an assembler or a compiler that runs under another system.

Copying from DOS to the other system is easy because the DOS allows a whole new file to be copied into an existing file without disturbing the directory. And none of these operating systems disturbs its directory when a file is read. To copy another system's files into DOS files through one of these diskettes is a little harder. The trick is to erase all the files in the other system's directory and then copy another file or set of files of exactly the same sizes and in exactly the same sequence as the files that were there before. When done correctly, the result is a new directory which, again, exactly matches the carefully constructed DOS directory on that diskette.

* * *

Finally, if you have a hard disk in your system, you should know that HDOS, the hard disk version of DOS, is the underlying partitioning scheme for Pascal, CP/M, and MSDOS. That means that the spaces on the hard disk that are set up for use by any of these systems are really just HDOS files. What you may not have known is that CP/M units on the hard disk are like single density CP/M diskettes in that they are not skewed.

This is, perhaps, the most difficult consistency in the North Star system to take advantage of. Although an HDOS file can be much larger than a diskette, it cannot contain a DOS directory along with the structures used by another operating system and it cannot be accessed by HDOS as a diskette image.

However, if you do your own programming, you can do whatever you want with an HDOS file. Let's suppose, for example, that you need to transpose a large data base from the obscure internal structure used by one data base system into the truly bizarre internal structure used by some other system. And let's suppose that the obscure structure is in a Pascal file while the bizarre structure needs to end up in a CP/M file.

If the size of this data base is 79 Kbytes or more in both forms, then you

couldn't use either of the techniques described above to move it in one piece from Pascal to DOS and from there to CP/M. But if the program you develop to perform the transposition runs under HDOS, it can very easily take its input from a Pascal volume and store its output in a CP/M unit, thus completing the whole job in one operation. All you have to do is copy a file of the right size into the empty CP/M unit under CP/M, and then make your program start reading and writing at the right places in the two HDOS files.

Here, then, are three different techniques for moving files among the different operating systems on one computer. As you can see, there are cases where one or another technique won't work. But with all of these tricks up your sleeve, the lack of a program for making such copies should never be a problem for you.

# # #

Commentary

By Roy Chadwick
P.O. Box 212
Robbins, CA 95676

I would like to compliment you on the Compass Newsletter. I have all of the issues and I have found information that is useful & valuable to me in every one of them. I think that one of the most important things that you do with Compass is to publish articles & tips that are of interest to both novice & advanced computerists. Additionally, I like its current physical form. Keep up the good work.

In regard to future issues, here is a list of topics that are of particular interest to me.

1. Modular programming with BASIC.

2. Hardware & software needed to "Read" & "Write" alien "Disk Formats" with North Star computers.

3. Miscellaneous alien hardware that will work with North Star computers.

4. APC BASIC & MEGABASIC.

5. Operating systems. (I prefer NSDOS because it is fast, requires less memory than other systems and is easy to understand.)

6. Plotters.

7. Tips on "Old" as well as "New" hardware. (New isn't always better.)

* * *

I have been using a Televideo 950 terminal with my Horizon for several years and I have a few tips for anyone else using this terminal.

TIP #1

If you have written a program that sends instructions to the programmable function keys & you find that the keys don't always accept these instructions, you should be able to cure the problem by having your program execute a time delay loop immediately after it programs each key. Apparently, the TV-950 cannot perform this job as fast as a program can send instructions. A BASIC For-Next loop that counts to 50 usually does the trick.

TIP #2

Televideo 950 terminals come from the factory with 2 pages of screen memory installed. If you like, you can install 2 additional pages. Additional page kits are available from the factory for about $90 each. Now, guess what comes in each kit. You guessed it. One memory chip & one DIP socket.

If you would like to add this enhancement to your TV-950 without making a payment on your computer dealers' new sports car, perform the following steps:

1. Obtain two HM6116P (150 nanosecond) memory chips (TMM2016P chips will also work). These are available from most computer parts suppliers for about $5 to $7 each. Also, obtain two 24 pin DIP sockets. You must install both chips because one chip by itself will not do anything.

2. Remove the terminal cover (four screws). Next, ground your body to insure that you don't "ZAP" any chips with a static discharge. Then, slide the main circuit board rearward, remove any cables that may be plugged in and then remove the board completely.

3. Find locations A36 & A37 and solder the DIP sockets firmly into these locations. Plug the new memory chips into these sockets with the indexing notch oriented in the same direction as the notches on the memory chips in locations A34 & A35.

4. Re-assemble the terminal and you are in business.

TIP #3

Now that you have 4 pages of screen memory, what do you do with them? Here is a way to use them to reduce program execution time.

Imagine that you are writing a program that makes use of a "Program Initialization Menu", a "Main Menu" and several "Sub Menus". First, have your program clear all four pages and park the cursor in the HOME position on the first page. This can be done with the following BASIC statements:

```
PRINT CHR$(27)," 3",
      CHR$(26),CHR$(27)," 1",
FOR X=1TO100
NEXT X
```

The For/Next loop is needed to insure that the terminal has had time to complete the clearing procedure before any new information is sent to the screen.

Next, have your program fill the first page with the "Program Initialization Menu" format. When your program has proceeded to the point where it is time to display the "Main Menu" format, have your program advance the screen to the 2nd page before sending the "Main Menu" format to the terminal.

The following BASIC statement will advance the screen memory to the next higher numbered page:

```
PRINT CHR$(27),"K",
```

When your program has proceeded to the point where it is time to display a "Sub Menu" format, have your program advance the screen memory to the 3rd page.

When your program has proceeded to the point where it is time to display the format of a "Sub Menu" choice, have your program advance the screen memory to the 4th page.

At this point in the program, no reduction in execution time has been achieved but here comes the speed.

The formats for the "Program Initialization Menu", the "Main Menu" and the "Sub Menu" are still in the 1st, 2nd and 3rd pages of the screen memory even though they are not being displayed. If at this time the user of the program decides to recall the "Sub Menu", have your program back-page the screen memory to the 3rd page. This will display the "Sub Menu" instantly because the 3rd page did not need to be refilled. Have your program repeat the this procedure to recall the 2nd page and again for the 1st page.

The following BASIC statement will back-page the screen memory to the next lower numbered page:

```
PRINT CHR$(27),"J",
```

It is, of course, up to you as a programmer to be sure that your input routines match the page being displayed.

Additional information regarding the screen memory can be found in your TV-950 Operators' Manual.

# # #

XGRAF

By David M. Allen
Department of Statistics
University of Kentucky
Lexington, KY 40506

In Compass, Vol. IV, no. 3, pp 5-6, 30-31, I gave some procedures for doing Advantage graphics from Turbo Pascal. Those routines are adequate for simple annotated charts and graphs. This article is much more ambitious; it gives procedures that allow access to nearly all of the Advantage's graphic capabilities from Turbo Pascal. The routines work with Turbo Pascal, Version 3.0. To use the routines with Version 2.0 you will need to replace all occurrences of ">GRAF" with "GRAF" and all occurrences of ">BDOS" with "BDOS". The use of these procedures is illustrated with some fun programs.

To run graphics programs in Turbo's memory mode, first execute GMGRADDD with TURBO.COM on the command line. Then operate the Turbo system in the usual way. To produce a command file that can ported to other Advantages that don't have the Turbo system do the following. Execute the Turbo system, use the options command to have the compiled program placed in a command (.COM) file, compile your program, and return to CP/M. Execute GMGRADD with your programs name on the command line. Your program is now ready to run.

The graphics routines are placed in a file XGRAF.PAS (Listing 1) which can be included in application programs using the include directive {$I XGRAF}. The procedures RECTANGLE through PCHAR are the Pascal equivalents to the routines RECTANGLE through CHAR described in "NorthStar Graphics CP/M Preface". The name change from CHAR to PCHAR is due to CHAR being a reserved word in Pascal. The arguments of the Pascal procedures are closely related to to registers described in the CP/M Preface. You need to have the CP/M Preface to understand how the Pascal procedures work.

The first argument of RECTANGLE is a variable parameter A of type byte. The content of register A is placed in the variable A just before returning from RECTANGLE. The CP/M Preface does not say so explicitly, but I presume the error code mentioned is returned in register A. I have not bothered to to use the error codes. The second argument is a value parameter C of type byte. It is used to specify a pattern code as described on page 6-15 of the CP/M Preface. The next four arguments are integers. The first two are the (Y,X) coordinates of a corner and the last two are the (Y,X) coordinates of the opposite corner. For the technically inclined, I will describe how this relates to the instructions in the CP/M Preface. Others may skip to the next paragraph. Pascal places the arguments of a procedure on a stack that goes from high memory to low memory. Thus, when we are asked to put the address of a table in the register pair HL, we have to put the elements of the table in the argument list in reverse order and the put the address of the last element in HL.

The A and C arguments of POLYGON are the same as they are for RECTANGLE. B contains the number of vertices. Because the table used by POLYGON is not of fixed length, one cannot use the method used in RECTANGLE. You have to declare an array of integer, define its elements as X1, Y1, ... XN, YN, and then use the ADDR function to pass the address of the array to the HL parameter. This is illustrated in the program BANNER. The IX and IY parameters are rarely used, and one would usually pass 0 to them.

The B, C, D, E, IX, and IY arguments of ELLIPSE are as described in the CP/M Preface. The Y and X arguments are the coordinates of the center point. The C argument of SPECLINE is as described in the CP/M Preface. The Y and X arguments are the coordinates of the end point of the line. CLEAR has no arguments. The arguments of BLOCK and PCHAR correspond exactly to those of the

CP/M Preface.

In this paragraph I will describe the remaining procedures in XGRAF. VIEWPORT establishes a window for making graphs in hybrid mode. Its boolean argument FRAME is set to true if you want a frame and is otherwise set to false. The procedure LINE draws a line from the current position to the coordinates (X,Y) with COLOR=1 for a bright line and COLOR=0 for a dark line. Note that LINE is just SPECLINE with the arguments in a different order to be more natural. DOT is just a special call to POLYGON. However, DOT does illustrate two points you need to keep in mind as you write additional "user friendly" procedures. First, some procedures require addresses rather than values. Fortunately, the Turbo ADDR function gives a simple solution to this problem. Secondly, arguments of Pascal functions are put in memory from high memory to low memory. This is why X and Y had to be interchanged. Finally, SCRPRINT, of which I am very proud, is a procedure that takes whatever is on your screen and puts it on your C. Itoh Prowritter. Its arguments are described in the comments. It illustrates the use of the procedure BLOCK. The work done in SCRPRINT compensates for the fact that the bits in a byte of screen memory are displayed horizontally while bits in a byte sent to the printer must be displayed vertically. Since this should be true for all makes of dot matrix printers, SCRPRINT should be easy to modify to work with other printers.

HILBERT.PAS (Listing 2) draws a very interesting geometric pattern on the screen. The algorithm used is very popular for illustrating recursive procedures. It also provides examples for the use of VIEWPORT, LINE, and SCRPRINT.

BALL.PAS (Listing 3) is a program for animation of a bouncing ball. It is a modified version of a program in Paul A. Sand's Advanced Pascal Programming Techniques, Osborne/McGraw-Hill. It illustrates some rules of motion that might be presented in a high school physics course. BALL.PAS uses the procedure ELLIPSE. Refer to page 6-15 of the CP/M Preface for the pattern codes that may be used with ELLIPSE.

SPIRO.PAS (Listing 4) is a program that emulates a Spirograph (Kenner Products, Cincinnati, Ohio). A Spirograph consists of a ring and a disk that is smaller than the ring. The disk has a small hole near its edge. The ring is held stationary on a piece of paper, the disk is placed inside the ring with a pen in the small hole, and then the disk is rolled around the inside of the ring. In the process, the pen draws a pleasing geometric design. A Spirograph and some figures it drew are included in the illustrations.

BANNER.PAS (Listing 5) is a program that draws banners. It uses SCRPRINT. Recall that SCRPRINT may need to be modified for printers other than the C. Itoh Prowritter. A novel aspect of BANNER is that it writes a character to the screen and then reads it back into an array of bytes using BLOCK. Each byte is expanded into an array of pixels. Each pixel is then expanded into a polygon designed to blend nicely with it neighbors. The polygons are displayed on the screen using POLYGON. SCRPRINT is used to print each large character.

If you distribute programs using these procedures or modifications of them, I ask only that you cite this article. I will provide a disk with the five listings for $10.00.

# # #

```
                    {XGRAF.PAS, LISTING 1}

{XGRAF, A COLLECTION OF PROCEDURES FOR MAKING MOST OF THE }
{NORTHSTAR ADVANTAGE GRAPHICS CAPABILITIES AVAILABLE FROM }
{TURBO PASCAL.                                            }
{AUTHOR: DAVID M. ALLEN                                   }
{DATE LAST REVISED: MAY 29, 1985                          }
{PAGE REFERENCES ARE TO "GRAPHICS CP/M PREFACE", COPYRIGHT}
{1981, 1982, BY NORTH STAR COMPUTERS, INC.                }

PROCEDURE RECTANGLE(VAR A:BYTE;C:BYTE;Y1,X1,Y2,X2:INTEGER);
{PAGE 6-16, THE PARAMETER A IS NOT USED ON INPUT, BUT IS  }
{USED ONLY TO RETURN THE ERROR CODE.                      }

CONST GRAF=12;
BEGIN
INLINE(
      $3A / C /           { LD   A,(C) }
      $4F /               { LD   C,A   }
      $21 / X2 /          { LD   HL,X2 }
      $3E / $02 /         { LD   A,2   }
      $CD />GRAF /        { CALL GRAF  }
      $32 / A );          { LD   (A),A }
END;

PROCEDURE POLYGON(VAR A:BYTE;B,C:BYTE;HL,IX,IY:INTEGER);
{PAGE 6-17, THE PARAMETER A IS NOT USED ON INPUT, BUT IS  }
{USED ONLY TO RETURN THE ERROR CODE.                      }

CONST GRAF=12;
BEGIN
INLINE(
      $3A / B /           { LD   A,(B) }
      $47 /               { LD   B,A   }
      $3A / C /           { LD   A,(C) }
      $4F /               { LD   C,A   }
      $3E / $01 /         { LD   A,1   }
      $2A / HL  /         { LD   HL,(HL) }
      $DD / $2A / IX /    { LD   IX,(IX) }
      $FD / $2A / IY /    { LD   IY,(IY) }
      $CD />GRAF /        { CALL GRAF  }
      $32 / A );          { LD   (A),A }
END;

PROCEDURE ELLIPSE(B,C,D,E:BYTE;Y,X,IX,IY:INTEGER);
{PAGE 6-19 }

CONST GRAF=12;
BEGIN
INLINE(
      $3A / B /           { LD   A,(B) }
```



```
      $47 /               { LD   B,A   }
      $3A / C /           { LD   A,(C) }
      $4F /               { LD   C,A   }
      $3A / D /           { LD   A,(D) }
      $57 /               { LD   D,A   }
      $3A / E /           { LD   A,(E) }
      $5F /               { LD   E,A   }
      $21 / X /           { LD   HL,X  }
      $DD / $2A / IX /    { LD   IX,(IX) }
      $FD / $2A / IY /    { LD   IY,(IY) }
      $3E / $03 /         { LD   A,3   }
      $CD />GRAF );       { CALL GRAF  }
END;

PROCEDURE SPECLINE(C:BYTE;Y,X:INTEGER);
{PAGE 6-21 }

CONST GRAF=12;
BEGIN
INLINE(
      $3A / C /           { LD   A,(C)     }
      $4F /               { LD   C,A       }
      $3E / $06 /         { LD   A,6       }
      $2A / Y   /         { LD   HL,(Y)    }
      $5D /               { LD   E,L       }
      $54 /               { LD   D,H       }
      $2A / X   /         { LD   HL,(X)    }
      $CD />GRAF);        { CALL GRAF      }
END;

PROCEDURE CLEAR;
{PAGE 6-23 }

CONST GRAF=12;
BEGIN
INLINE(
      $3E / $00 /         { LD   A,0       }
      $CD />GRAF);        { CALL GRAF      }
END;

PROCEDURE BLOCK(C,D,E:BYTE;HL,IY,IX:INTEGER);
{PAGE 6-24 }

CONST GRAF=12;
BEGIN
  INLINE(
      $3A / C /           { LD   A,(C)   }
      $4F /               { LD   C,A     }
      $3A / D /           { LD   A,(D) }
      $57 /               { LD   D,A }
      $3A / E  /          { LD   A,(E) }
```

```
      $5F /              { LD   E,A }
      $DD / $2A / IX /   { LD   IX,(IX) }
      $FD / $2A / IY /   { LD   IY,(IY) }
      $2A / HL /         { LD   HL,(HL) }
      $3E / $04 /        { LD   A,4 }
      $CD />GRAF );      { CALL GRAF }
END;

PROCEDURE PCHAR(VAR A:BYTE;B:CHAR;C,E:BYTE;HL,IX:INTEGER);
{A IS TO RETURN ERROR CODE }
{PAGE 6-25 }

CONST GRAF=12;
BEGIN
  INLINE(
      $3A / B /          { LD   A,(B) }
      $47 /              { LD   B,A   }
      $3A / C /          { LD   A,(C) }
      $4F /              { LD   C,A   }
      $3A / E  /         { LD   A,(E) }
      $5F /              { LD   E,A   }
      $DD / $21 / IX /   { LD   IX,IX }
      $2A / HL  /        { LD   HL,(HL) }
      $3E / $05 /        { LD   A,5   }
      $CD />GRAF /       { CALL GRAF  }
      $32 / A );         { LD   (A),A }
END;

PROCEDURE VIEWPORT(FRAME:BOOLEAN);
VAR A:BYTE;
BEGIN
IF FRAME THEN RECTANGLE(A,$81,0,0,199,639)
         ELSE RECTANGLE(A,$80,0,0,199,639);
END;

PROCEDURE LINE(X,Y:INTEGER;COLOR:BYTE);
BEGIN
SPECLINE(COLOR,Y,X);
END;

PROCEDURE DOT(X,Y:INTEGER;COLOR:BYTE);
VAR A:BYTE;
    Z:INTEGER;
BEGIN
Z:=X;
X:=Y;
Y:=Z;
POLYGON(A,1,COLOR,ADDR(Y),0,0);
END;

PROCEDURE CIRCLE(X,Y:INTEGER;R,PAT:BYTE);
```



```
BEGIN
ELLIPSE(0,PAT,R,R,Y,X,0,0);
END;

PROCEDURE SCRPRINT(MARGIN:REAL;X,Y:INTEGER;WIDTH8,HEIGHT:BYTE);
{THIS PROCEDURE IS SPECIFIC TO THE C. ITOH PROWRITTER }
{ MARGIN IS LENGTH OF LEFT MARGIN IN INCHES }
{ ( X , Y ) IS COORDINATES OF UPPER LEFT CORNER OF BLOCK TO PRINTED}
{ WIDTH8 IS THE WIDTH OF BLOCK IN 8-BIT COLUMNS }
{ HEIGHT IS THE HEIGHT OF BLOCK IN BITS }
CONST
  GRAF=12;
  BDOS=5;
VAR
  H:BYTE;
  I,K,Y0,PTR:INTEGER;
  BUF:ARRAY[1..560] OF INTEGER;
  S:STRING[4];
BEGIN
K:=ROUND(MARGIN*17);
WRITE(CHR(25));
I:=WIDTH8*8;
STR(I:4,S);
FOR I:=1 TO 4 DO IF S[I]=' ' THEN S[I]:='0';
WRITE(LST,CHR(27),'Q',CHR(27),'T14');
Y0:=Y;
WHILE Y-Y0<HEIGHT DO
  BEGIN
  IF (Y+7)-Y0 <= HEIGHT THEN H:=7 ELSE H:=HEIGHT-Y;
{PROCEDURE BLOCK(C,D,E:BYTE;HL,IY,IX:INTEGER);}
  BLOCK(4,WIDTH8,H,ADDR(BUF),Y,0);
  IF K>0 THEN WRITE(LST,' ':K);
  WRITE(LST,CHR(27),'S',S);
  INLINE(
    $21 / BUF /          { LD   HL,BUF }
    $22 / PTR /          { LD   (PTR),HL }
    $3A / WIDTH8 /       { LD   A,(WIDTH8) }
    $57 /                { LD   D,A }
    $0E / $08 /          { LD   C,8 }
    $2A / PTR /          { LD   HL,(PTR) }
    $3A /  H  /          { LD   A,(H) }
    $47 /                { LD   B,A }
    $AF /                { XOR  A }
    $CB / $16 /          { RL   (HL) }
    $1F /                { RRA }
    $23 /                { INC  HL }
    $05 /                { DEC  B }
    $20 / $F9 /          { JR   NZ,$F9 }
    $F5 /                { PUSH AF }
    $3A / H   /          { LD   A,(H) }
    $47 /                { LD   B,A }
```

```
    $3E / $08 /          { LD   A,8 }
    $90 /                { SUB  B }
    $47 /                { LD   B,A }
    $F1 /                { POP  AF }
    $A7 /                { AND  A }
    $1F /                { RRA }
    $05 /                { DEC  B }
    $20 / $FB /          { JR   NZ,$FB }
    $E5 /                { PUSH HL }
    $C5 /                { PUSH BC }
    $D5 /                { PUSH DE }
    $5F /                { LD   E,A }
    $0E / $05 /          { LD   C,5 }
    $CD />BDOS /         { CALL BDOS }
    $D1 /                { POP  DE }
    $C1 /                { POP  BC }
    $E1 /                { POP  HL }
    $0D /                { DEC  C }
    $20 / $D3 /          { JR   NZ,$D3 }
    $22 / PTR /          { LD   (PTR),HL }
    $15 /                { DEC  D }
    $20 / $CB );         { JR   NZ,$CB }
  Y:=Y+7;
  WRITELN(LST);
  END;
WRITE(LST,CHR(27),'A',CHR(27),'N');
WRITE(CHR(24));
END;
```

{HILBERT.PAS, LISTING 2}

```
PROGRAM HILBERT;
CONST XDOTS=639;  {Maximum number of x dots.}
      YDOTS=199;  {Maximum number of y dots.}

VAR H,I,X,Y:INTEGER;
    C:CHAR;
{$I XGRAF}

{$A-}
PROCEDURE A(I,T:INTEGER);
BEGIN
IF I>0 THEN
CASE T OF
0: BEGIN
   A(I-1,3);X:=X-2*H;LINE(X,Y,1);
   A(I-1,0);Y:=Y-H;LINE(X,Y,1);
   A(I-1,0);X:=X+2*H;LINE(X,Y,1);
   A(I-1,1)
   END;
1: BEGIN
   A(I-1,2);Y:=Y+H;LINE(X,Y,1);
   A(I-1,1);X:=X+2*H;LINE(X,Y,1);
   A(I-1,1);Y:=Y-H;LINE(X,Y,1);
   A(I-1,0)
   END;
2: BEGIN
   A(I-1,1);X:=X+2*H;LINE(X,Y,1);
   A(I-1,2);Y:=Y+H;LINE(X,Y,1);
   A(I-1,2);X:=X-2*H;LINE(X,Y,1);
   A(I-1,3)
   END;
3: BEGIN
   A(I-1,0);Y:=Y-H;LINE(X,Y,1);
   A(I-1,3);X:=X-2*H;LINE(X,Y,1);
   A(I-1,3);Y:=Y+H;LINE(X,Y,1);
   A(I-1,2)
   END
END;  {CASE}
END;

BEGIN {HILBERT}
WRITE(CHR(18));
VIEWPORT(FALSE);
REPEAT
WRITELN('Enter order (1 THROUGH 7) OR 0 TO QUIT');
READLN(I);
IF (I>=1) AND (I<=7) THEN
BEGIN
CLEAR;
H:=YDOTS DIV (1 SHL I-1);
X:=(XDOTS+H*(1 SHL I -1)) DIV 2;
Y:=(YDOTS+H*(1 SHL I -1)) DIV 2;
DOT(X,Y,1);
A(I,0);
WRITE('Print graph? ');
READLN(C);
IF C IN ['y','Y'] THEN SCRPRINT(1.5,0,0,80,200);
END;
UNTIL I=0;
WRITE(CHR(19));
END.
```

{BALL.PASS, LISTING 3}

```
PROGRAM BOUNCE;
CONST
    XDOTS=639;  {Maximum number of x dots.}
    YDOTS=199;  {Maximum number of y dots.}
    R=5;
TYPE
    COORD=ARRAY[1..2] OF INTEGER;
```

```
VAR
    I:INTEGER;
    BELL:BOOLEAN;
    DT:REAL;
    POS,NEWPOS,MIN,MAX,VEL,ACC:COORD;

{$I XGRAF}

PROCEDURE BALL(POS:COORD;ON:BOOLEAN);
BEGIN
IF ON THEN ELLIPSE(0,29,R,R,POS[2],POS[1],0,0)
      ELSE ELLIPSE(0,66,R,R,POS[2],POS[1],0,0);
END;

BEGIN {BOUNCE}
WRITE(CHR(18));
VIEWPORT(TRUE);
MIN[1]:=2*R+1;        MIN[2]:=R+1;
MAX[1]:=XDOTS-2*R-1; MAX[2]:=YDOTS-R-1;
ACC[1]:=0;            ACC[2]:=1;
VEL[1]:=30;           VEL[2]:=10;
POS[1]:=MIN[1];       POS[2]:=MIN[2];
{
WRITELN('Enter acc1 acc2');
READLN(ACC[1],ACC[2]);
WRITELN('Enter vel1 vel2');
READLN(VEL[1],VEL[2]);
WRITELN('Enter pos1 pos2');
READLN(POS[1],POS[2]);
}
DT:=1.0;
REPEAT
BELL:=FALSE;
FOR I:=1 TO 2 DO
  BEGIN
  NEWPOS[I]:=ROUND(POS[I]+DT*(VEL[I]+ACC[I]*DT/2.0));
  IF (NEWPOS[I]<=MIN[I]) THEN
    BEGIN
    BELL:=TRUE;
    NEWPOS[I]:=MIN[I];
    VEL[I]:=-VEL[I];
    END
  ELSE IF (NEWPOS[I]>=MAX[I]) THEN
    BEGIN
    BELL:=TRUE;
    NEWPOS[I]:=MAX[I];
    VEL[I]:=-VEL[I];
    END
  ELSE
    VEL[I]:=ROUND(VEL[I]+ACC[I]*DT);
  END;
```



```
BALL(POS,FALSE);
BALL(NEWPOS,TRUE);
IF BELL THEN WRITE(CHR(7));
POS:=NEWPOS;
UNTIL (ABS(VEL[2])<=1) AND (POS[2]=MAX[2]);
WRITE(CHR(19));
END.
```

{SPIRO.PAS, LISTING 4}

```
PROGRAM SPIRO;
CONST INCREMENTS=48;
      PI=3.141593;
      XDOTS=639;  {Maximum number of x dots.}
      YDOTS=199;  {Maximum number of y dots.}

VAR C:CHAR;
    X0,Y0,I,K,L,M,N:INTEGER;
    ON:BOOLEAN;
    SCALE,R,X,Y,SUM,RS,RM,RH,S1,S2,AS,AM:REAL;

{$I XGRAF}

BEGIN {SPIRO}
X0:=XDOTS DIV 2;
Y0:=YDOTS DIV 2;
WRITE(CHR(18));
VIEWPORT(FALSE);
SCALE:=1.0;
REPEAT
  WRITELN('Enter inside radius of ring.');
  READLN(RS);
  WRITELN('Enter radius of disk.');
  READLN(RM);
  WRITELN('Enter distance from center of disk to hole.');
  READLN(RH);
  S1:=(2.0*PI)/INCREMENTS;
  S2:=1.0-RS/RM;
  R:=RS-RM;
  SUM:=ABS(R)+ABS(RH);
  R:=Y0*R/SUM;
  RH:=Y0*RH/SUM;
  DOT(X0+ROUND(2.0*SCALE*(R+RH)),Y0,1);
  I:=0;
  REPEAT
    I:=I+1;
    AS:=S1*I;
    AM:=S2*AS;
    X:=R*COS(AS)+RH*COS(AM);
    Y:=R*SIN(AS)+RH*SIN(AM);
    LINE(X0+ROUND(2.0*SCALE*X),Y0+ROUND(SCALE*Y),1);
```

```
  UNTIL (ABS(R+RH-X)<2.0) AND (ABS(Y)<2.0)
        AND (I>=INCREMENTS);
WRITE('Print graph? ');
READLN(C);
IF C IN ['y','Y'] THEN SCRPRINT(1.5,0,0,80,200);
WRITE('Another graph? ');
READLN(C);
IF C IN ['Y','y'] THEN
  BEGIN
  SCALE:=1.0;
  WRITE('Reduce figure? ');
  READLN(C);
  IF C='Y' THEN
    BEGIN
    WRITELN('Enter scale factor. ');
    READLN(SCALE);
    END;
  WRITE('Clear screen? ');
  READLN(C);
  IF C='Y' THEN
    BEGIN
    CLEAR;
    VIEWPORT(FALSE);
    END;
  C:='Y';
  END;
UNTIL C<>'Y';
WRITE(CHR(19));
END.
```

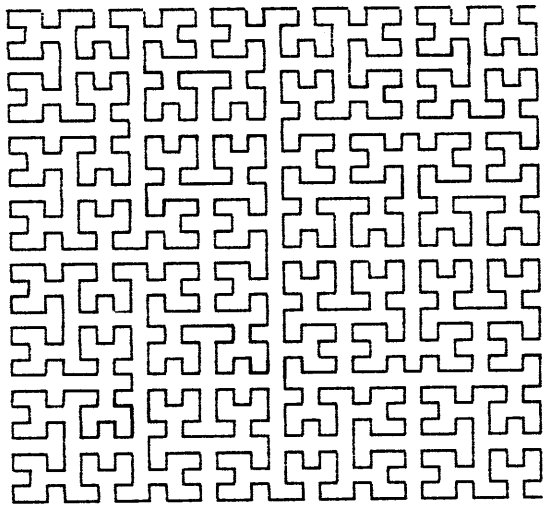
{BANNER.PAS, LISTING 5}

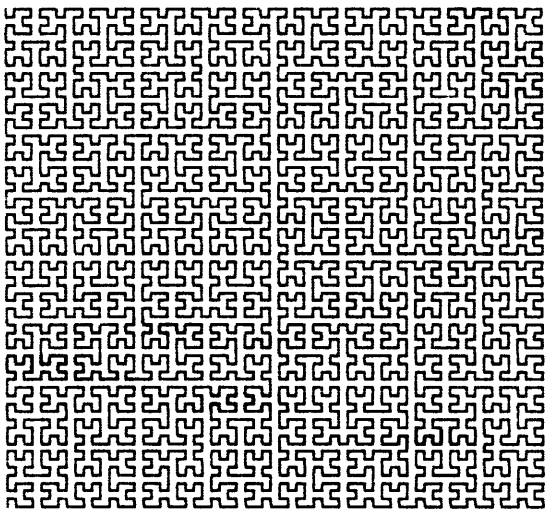
```
PROGRAM BANNER;
CONST
    H:ARRAY[0..3] OF -1..1=(1,-1,-1,1);
    V:ARRAY[0..3] OF -1..1=(1,1,-1,-1);
    W=16;
TYPE
    INTARRAY=ARRAY[0..23] OF INTEGER;
VAR
    A:BYTE;
    BAN,CODE:STRING[80];
    LETTER:ARRAY[0..10] OF BYTE;
    PIXEL:ARRAY[0..10,0..7] OF BOOLEAN;
    BIG,SMALL:INTARRAY;
    I,J,K,M,S,PAT:INTEGER;
    VIEW:BOOLEAN;
    C:CHAR;

{$I XGRAF}
```



```
PROCEDURE FIGURE(X,Y,PAT,N:INTEGER;VERT:INTARRAY);
VAR I:INTEGER;
    A:BYTE;
BEGIN
FOR I:=0 TO N-1 DO VERT[2*I]:=2*VERT[2*I];
FOR I:=0 TO N-1 DO VERT[2*I]:=VERT[2*I]+X;
FOR I:=0 TO N-1 DO VERT[2*I+1]:=VERT[2*I+1]+Y;
POLYGON(A,N,PAT,ADDR(VERT),0,0);
END;

BEGIN {BANNER}
S:=ROUND((SQRT(2.0)-1.0)*W);
BIG[ 0]:= W;BIG[ 1]:= S;BIG[ 4]:= S;BIG[ 5]:= W;
BIG[ 6]:=-S;BIG[ 7]:= W;BIG[10]:=-W;BIG[11]:= S;
BIG[12]:=-W;BIG[13]:=-S;BIG[16]:=-S;BIG[17]:=-W;
BIG[18]:= S;BIG[19]:=-W;BIG[22]:= W;BIG[23]:=-S;
WRITE(CHR(18));
VIEWPORT(FALSE);
WRITELN('Enter banner');
READLN(BAN);
FOR I:=1 TO 15 DO
  BEGIN
  RECTANGLE(A,4*I,10,(I-1)*40+20,30,I*40);
  STR(I:2,CODE);
  PCHAR(A,CODE[1],0,35,I*40-20,0);
  PCHAR(A,CODE[2],0,35,I*40-10,0);
  END;
WRITELN('Enter pattern');
READLN(PAT);
PAT:=4*PAT;
WRITELN('View only or print? (V/P) ');
READLN(C);
VIEW:=(C IN ['V','v']);
FOR M:=1 TO LENGTH(BAN) DO
BEGIN
CLEAR;
PCHAR(A,BAN[M],0,0,0,0);
BLOCK(4,1,10,ADDR(LETTER),0,0);
LETTER[10]:=0;
IF BAN[M]='N' THEN
  BEGIN
  LETTER[2]:=$22;
  LETTER[6]:=$22;
  END
ELSE IF BAN[M]='*' THEN
  LETTER[4]:=$3E;
CLEAR;
FOR I:=0 TO 10 DO
  FOR J:=0 TO 7 DO
    BEGIN
    PIXEL[I,J]:=ODD(LETTER[I]);
```

```
    LETTER[I]:=LETTER[I] DIV 2;
    END;
FOR J:=1 TO 6 DO
  FOR I:= 1 TO 9 DO
    IF PIXEL[I,J] THEN
      BEGIN
      FOR K:=0 TO 3 DO
        IF PIXEL[I-H[K],J] OR PIXEL[I,J-V[K]] OR
                              PIXEL[I-H[K],J-V[K]] THEN
          BEGIN
          BIG[6*K+2]:=H[K]*W;
          BIG[6*K+3]:=V[K]*W;
          END
        ELSE
          BEGIN
          BIG[6*K+2]:=BIG[6*K];
          BIG[6*K+3]:=BIG[6*K+1];
          END;
      FIGURE(4*(10-I)*W,2*(6-J)*W,PAT,12,BIG);
      END
    ELSE
      FOR K:=0 TO 3 DO
        IF PIXEL[I-H[K],J] AND PIXEL[I,J-V[K]] AND
                           NOT PIXEL[I-H[K],J-V[K]] THEN
          BEGIN
          SMALL[0]:=H[K]*W;
          SMALL[1]:=V[K]*W;
          SMALL[2]:=-H[K]*S;
          SMALL[3]:=V[K]*W;
          SMALL[4]:=H[K]*W;
          SMALL[5]:=-V[K]*S;
          FIGURE(4*(10-I)*W,2*(6-J)*W,PAT,3,SMALL);
          END;
IF VIEW THEN READLN
        ELSE SCRPRINT(1.5,0,0,80,196);
END;
WRITE(CHR(19));
END.
```





MagicBind Update

By Alan H. Nelson

I was mad with anticipation from the time I received an announcement of an update to the MagicBind printer-controller program, in particular an enhancement which would produce automatic multiple-column printing. In my original review of this program, in Compass, Vol. III, no. 4, pp. 12-14, I complained that I had "wasted hours and hours formatting the two-column text you see before you. It should be possible simply to give a "two-column" formatting command, with the necessary parameters for column width, space between columns, and so forth. ... If I sound exasperated, I am. This is all simply a matter of calculations, and calculations is what I bought my computer for!"

In spite of the theoretical ability of MagicBind to produce two-column printing, and in spite of the fact that your editor actually produced several issues with pages printed all at once on a Diablo 630 printer, in most recent issues we "cheated" by printing the material in long narrow columns, and then pasting up the results in side-by-side columns. In other words, cutting and pasting was faster than figuring out how to make the program print in double columns all by itself.

Well, I am delighted to report that MagicBind has now implemented a multiple-column command, and that it works at least as well as one might hope. The current issue of Compass has virtually all been printed in double columns automatically, except for program listings, which are still reduced on a xerox machine and then pasted up by hand.

The magic command for double-column printing is

..k

which prints out a column of material slightly less than half the designated page width to allow for separation between the columns, then backs the paper up to a spot opposite where the first column began, and prints the second column. The default is two columns with a space five characters in width, which is just fine with me; more columns with different gaps can also be specified.

By and large the command works exactly as I would wish it to: it manages the column structure very nicely, and at the last page divides the remaining material in half so as to leave all the blank space at the bottom (rather trying to print the first column to the bottom of the page, then leaving blank space at the end of the second column).

Are there any problems? Only slight ones. First, the program does not seem to handle widows and orphans as intelligently as with single-column printing, so it is sometimes necessary to adjust for single words carried over to the top of the next column. But this is an inherently difficult task which should perhaps involve human intervention in any case.

Second, the top line of the second column usually seems scrunched down a bit. The fault really is with the printer, not with the program. The Diablo 630 simply will not roll back to a blank line above the second column to ensure that the paper has always been going through the printer in a forward direction at the time the letters are struck. Since the paper inevitably slips ever so slightly on the pins of the bi-directional tractor feed, it is not positioned the same relative to the registering holes when the paper has been travelling backward as when it has been travelling forward.

When I did the formatting by hand, I forced the extra line with a dummy strike (see Compass, Vol. IV, no. 1, pp 7-8), a process which could conceivably have been implemented in MagicBind. Alternatively, the program might have printed the first line of the first column, then the first line of the second column, and so forth, rather than backing up; but this is asking quite a bit of a program designed to run on a microcomputer.

I have concluded that the problem is not worth the effort of solving. I apologize to members of INSUA for the flaw, but I exonerate Editype Systems of

the blame.

The multi-column format is an excellent enhancement, and the only improvement I have actually taken advantage of. Nevertheless, on the strength of the original program and of this improvement, I can recommend MagicBind, and its more sophisticated version MagicIndex with great enthusiasm.

Write or call:

Computer Editype Systems
509 Cathedral Parkway
New York, NY 10025
(212) 222-8148

# # #

Vendors Column

INSUA has been inundated with advertisements and newsletters concerning ZCPR3, the CP/M enhancement, from:

Echelon, Inc.
101 First Street
Los Altos, CA 94022

(415) 948-3820

Members who already know about ZCPR3 or whose curiosity or interest is piqued by the articles in this and forthcoming issued by Robert Bloom, may wish to write to Echelon to discover more|

$ $ $

For Sale

CompuPro S-100 system with 192 KB of memory, 8085/8088 dual processor board, system support board (with clock), Interfacer 4 (2 parallel + 1 serial port), Disk1 controller with two 8"-Qume drives (double-sided, double-density, 1.2 MB), set up to run all CompuPro standard operating systems (CP/M-80, CP/M-86, CP/M-816) as well as MS-DOS and PC-DOS 2.1, for sale, best offer.

Call (415) 944-2157 (day) or (415) 672-6783 (evening)
or write to:
G. E. Molau
30 Mt. Tamalpias Ct.
Clayton, CA 94517

Wanted

Wanted: One North Star D/Q disc controller board. Write to:
Dr. D. Yates
Botany Department
University of Queensland
St. Lucia, 4067
Queensland
Australia

Insua

International NorthStar Users Association
Publishers of The Compass Newsletter
PO Box 2910 • Fairfield, CA 94533

Bulk Rate
U.S. Postage
PAID
Walnut Creek
Permit No. 203

TATE 2761
JAMES TATE
23914 SPRING DAY LN.
SPRING, TX 77373

Postmaster: Address correction requested.
Please enter key number on form 3579.

The Compass

International NorthStar Users Association

Volume V No. 3

All Good Things . . .

NOTICE TO MEMBERS

Sept 17,1985

INSUA will be permanently closing down operations effective December 31, 1985. This decision was reached unanimously by the INSUA Board of Directors after months of consideration, and is reflective of the many changes that have occurred in the microcomputer industry over the 5 years of INSUA's existence.

We hope that our members and many supporters will, with us, look back at these years with a sense of pride and accomplishment. They have been filled with pioneering, invention, and discovery in which INSUA has played an important part.

For those of you who are current members (member numbers with an "I" suffix on your membership number), we have some news that will cheer you up right away. It is our way of saying thanks to you for your support and confidence.

First, your Board of Directors is going to work extra hard to assure an orderly transition, including providing for continuation of some of our disk library and back newsletter services.

Next, Compass newsletter will continue to bring you those great articles and tidbits of information through Volume V number 6.

Still more, we have negotiated an agreement for current members to receive several complimentary issues of Microsystems Journal, the new computer magazine by Sol Libes. We think you will really like this exceptional microcomputer journal. Naturally, Sol will want you to subscribe after the complimentary period expires, and we endorse and encourage this.

Is that all?? Not so! Each year, we have invited you to JOIN US for another year of activities. Each year brought those excellent Compass newsletters and disk offerings from the disk library. We never knew in advance whether finances would permit a free disk distribution, but each year allowed us to obtain and distribute useful software like new DOS releases or the INFOSOFT package.

So we should let you down this year? Not a chance! Watch for something incredible to be announced in the next newsletter issue.

Closing for now on behalf of the INSUA Board of Directors, we thank you for your support.

Robert Beaver
Chairman, INSUA

NOTICE TO CLUBS

We'd like our members to find good homes. If you think our members should hear about your club, please drop us a note to obtain complimentary advertising in Compass. Include fees and services provided. Inquiries concerning club-sponsored continuation of the INSUA disk library and newsletters are invited. Include contact name and phone number, please.

NOTICE TO AUTHORS

If you have ever had the thought of writing an article for INSUA, now is the time to do it! Time is running out! Because INSUA is closing down soon, we ask that you offer to supply copies of programs or routines directly to other members of INSUA. Please include your address and your nominal charge for supplying material on disk. --Ed.

The Compass

The Compass is published every two months by INSUA, the Interational North Star Users Association, P. O. Box 2910, Fairfield, CA 94533.

Subscription Information:

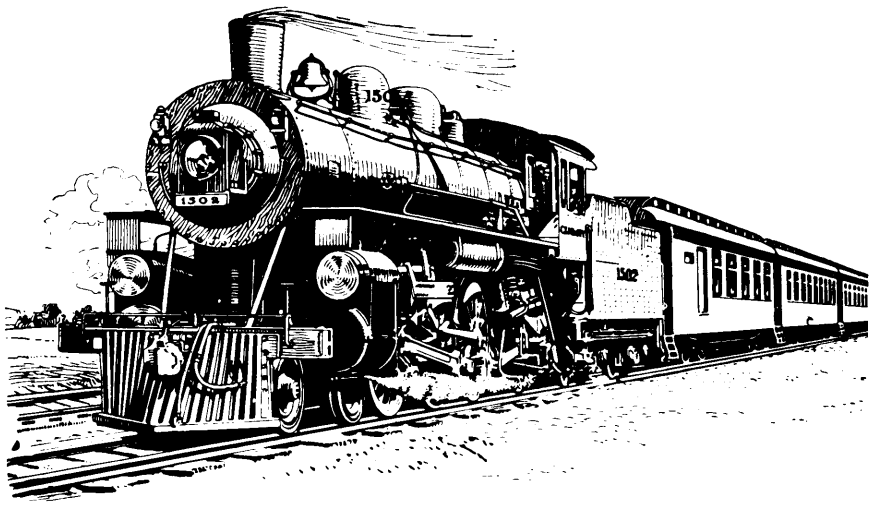
Subscription to The Compass is included in the $20.00 INSUA membership fee. Address subscription inquiries to INSUA, P. O. Box 2910, Fairfield, CA 94533.

DISCLAIMER

Programs printed in Compass and/or distributed through the INSUA disk library are offered to INSUA members in good faith. INSUA, however, is unable to guarantee the operation of any of these programs or to guarantee support. Users are advised to test the programs thoroughly for themselves in conditions under which they are to be used. Users who employ such programs in serious business or financial applications must do so at their own risk.

Facts or opinions published about manufacturers and dealers, and all opinions expressed in articles and letters, are the responsibility of the authors, and not of INSUA or the Editor of Compass. INSUA offers the right of reply to members and non-members alike.

Contents



Release 2.1.1 Software: Documentation

By Saul G. Levy
2555 E. Irvington Rd., Lot 47
Tucson, Arizona 85714
(602) 889-7753

North Star Computers, Inc. released a new version of its DOS and BASIC in 1983 to fix BASIC's phantom null error (see Compass, Vol. III, No. 2, p. 2). INSUA supplies this release on two single-sided diskettes (#1023 and #1024).

North Star's business has fallen off due to IBM's "theft" of the microcomputer market (half of North Star's staff was laid off at the beginning of September, 1984)! North Star has put a major effort into the IBM-(semi)compatible Dimension Computer which has prevented publication of the 2.1.1 documentation. The latest I've heard is that North Star has dumped support of its 8-bit machines onto two major distributors!

This article is an attempt to rectify the lack of documentation with a discussion of this newest (and maybe last) release of the excellent North Star DOS and BASIC. I may not cover every new feature in the software, but the major additions are obvious to anyone with some experience. I could use some help from North Star with a few of the new features, which are listed (roughly) by importance.

New DOS Features

1. A bigger DOS (again!). Now 14 blocks long; it loads at 100H and is relocatable with a new version of the MOVER program.

2. Much faster disk accessing (you can hear the difference!).

3. Full editing of the input commands (just like BASIC and the Monitors).

4. A long overdue file rename command (RN).

5. Output redirection from the DOS (same as OD in the Monitors).

6. A re-boot command from the DOS (IL, meaning initial load).

7. A configuration program (SYSGEN) for DOS and BASIC.

8. An easy modification can echo a Control-H when you hit your favorite rubout key (in DOS, BASIC, or any Monitor).

9. A minor attempt at a help command (HE).

10. A full-screen typing command (DO).

11. An equates file (EQUS).

12. Hooks into future hard disk versions.

New BASIC Features

1. A bigger BASIC. Now 56 blocks long; it loads at 1000H and is relocatable with the MOVER program.

2. 8-, 10-, 12-, and 14-digit BASICs are provided and relocatable.

3. Any untrapped, runtime error causes an automatic EDIT of the "bad" line. EDIT mode now displays the line so you can see what is already there.

4. A warning message if the MEMSET is incorrect for your current memory map.

5. A sign-on message which is also displayed when you reenter BASIC at its GO address (this SCRatches any program as usual).

6. An INFO() function which allows programs to access three of the system's software pointers.

DOS Discussion

1. A larger DOS means that much of your non-North Star, object code software is incompatible and must be used with older DOSes. By now we should be used to this, but every increase in size will create this problem (see Compatibility (or Lack Thereof) below). If you have two identical versions of any object code program which load at different addresses, you can use my ASSYZERO program to create a relocatable version to use with MOVER (see Vol. IV, No. 6, p. 19 for how to relocate the SECRETARY word processor).

2. WOW! North Star managed to increase disk accessing speed by about 25%. INitialize, DiskTest, LIst, etc. all run much faster. If your drives are marginal, 2.1.1 DOS will force you to get them repaired! One negative note is that not all file accessing programs will show a speed increase. I was quite surprised by this when using BASIC!

3. The previous command remains in the input buffer and can be edited with the same control codes as BASIC uses! Now you can type a command only once, and use or edit it over and over again.

4. Type "RN Old file name,drive New file name" to change any file's name. The drive number is optional and defaults to drive 1 as usual. RN won't allow duplication of an existing file name.

5. OD#x where x is 0-8 will change the output device number to the value of x (note that the corresponding input device number cannot be changed by the DOS). An x of 0 is the default, left serial port; 1 is the right serial port; and 2 is the parallel port. If an active device is not on the port you switched to, the DOS will hang as usual and prevent any further operation (reboot to clear a hung port).

Note that the device numbers repeat: 0-2, 3-5, and 6-8. Use device 3 to turn off the usual paging which asks you to HIT RETURN TO CONTINUE. This can be useful if you want to list the directory of a diskette when the user is not present to hit RETURN. It also works in BASIC so you can LIST#3 to quickly see the whole program to check that a just SAVEd file will reLOAD and LIST correctly (no missing sections or sectors). This doesn't happen very often, but three days before I wrote this, one of my North Star 16K memory boards decided to become a very useless ROM board while I was editing the SECRETARY relocation article! The delay line had quit. The board was not being refreshed and the end of my text was gone, too! I noticed this and saved what was left to a new file rather than just saving the truncated text to the original file which would have destroyed the unedited section. After switching memory boards, I appended the missing text and saved myself a lot of work! It pays to be careful!

6. IL will reboot the computer from the keyboard. I used to type "JP E800" a lot rather than disturb my cat by reaching for the reset switch! If you have a different boot PROM address, you will have to modify the jump at 140AH if you load the DOS at 1000H (50AH in the running DOS).

7. SYSGEN allows you to easily change the DOS to quad density (from the default double density), auto start any Type 1 program (GO program name), change the number of lines on your terminal, turn on interrupts, and list the maximum BASIC MEMSET. You can change BASIC to auto start any Type 2 program and delete various math functions. BASIC can't be personalized to echo a Control-H when you use your favorite rubout key (this is ridiculous)!

8. The character echoed for a rubout is at 192FH if you load the DOS at 1000H (B2FH in the running DOS). Change the 5FH to an 08H for a Control-H or to whatever you want. Likewise, if BASIC is loaded at 1000H change 1017H; if any Monitor is loaded at 1000H change 1012H.

9. Typing HE will display all of the two-letter commands. This won't tell you how to use them! The full list is:

CR DE **DO** GO

HE IL IN JP

LF LI **OD** RD

RN SF TY WR

The new commands are boldfaced.

10. Typing DO and RETURN will allow you to type any characters anywhere on your terminal's screen. The command parser is turned off until you type another RETURN. Burt Andrews wrote me that any character typed between DO and the RETURN (e.g. DO! RETURN) will then become the key to hit to end the DO command. If a third character is entered, you can use RETURN plus line feed and the cursor control keys to move around the screen (Control-J, -K, -H, and -L are the usual arrow keys). It is still not clear why North Star provided this command. I would like to find something useful to do with DO!

11. INSUA only gave us the first side of the 2.1.1 diskette (#1023). They correctly left out the relocation key (minus) files for the higher digit BASICs and the EQUS file which lists the DOS equates to the major DOS routines. If you're dying for these extra files, buy the second side of the system diskette (#1024). To look at the EQUS file, run the following program:

```
10 OPEN #0%0,"EQUS"
          (Type 0 file)
20 READ #0,&A
30 IF A=26 THEN END
          (26=1AH, the CP/M
          End-Of-File mark)
40 ! CHR$(A)
          (or "!#1,CHR$(A),"
          to print on the second
          serial port, or
          "!#2,CHR$(A)," to print
          on the parallel port)
50 GOTO 20
```

EQUS looks like it was written with a CP/M editor! Lots of interesting stuff here, but you're on your own!

12. David Young (Vol. IV, No. 4, p. 33) noticed three error messages (in BASIC) which hook into future hard disk versions of this DOS (if North Star ever releases another HDOS due to TurboDos taking over).

BASIC Discussion

1. BASIC's program load address is usually of no concern to the user. A BASIC program is a form of source code in any case. This eases switching to a bigger DOS and BASIC although I have already had to crunch (remove all REMs and spaces from) one program to allow it to run in 58K. This is one advantage of high-level languages.

2. North Star no longer sells higher-digit BASICs separately. It is nice to have 14-digits if you need them. Eight digits is enough for all of the work I have done in the past, but if you know any millionaires, more digits will be needed.

3. Any untrapped error in run mode will put the "bad" line into the edit buffer (which is also the direct mode's command input buffer), print the line (part of the EDIT command), and stop the program as usual. You can edit the line or ignore it and enter any other direct mode command. I appreciate this, although it causes an incompatibility problem with at least one non-North Star piece of software (see below).

4. An incorrect MEMSET message is actually useful if you swap memory boards or change MEMSETs often. Missing memory at or below the MEMSET address will make BASIC die very fast!

5. READY used to be enough, but now we also have a more professional sign-on message.

6. David Young (see DOS 12. above) also noted the INFO() function. If you type:

! INFO(1),INFO(2),INFO(3)

you will get three numbers printed. The first one is the GO address of BASIC (4096=1000H), the second is the load address of DOS (256=100H), and the third is where the sign-on message starts in

BASIC (4132=1024H). If you relocate DOS and BASIC, these addresses will be changed accordingly. Relocation may have been the reason for including this special function; otherwise I feel that this was a waste of programming effort. Maybe North Star had some other reason for the inclusion of INFO()! Mr. Young also noted SIZE in the token table. SIZE also appears in Release 5.2 BASIC, but seems not to be used by either BASIC.

Compatibility (or Lack Thereof)

I've already mentioned the problem that many of the object code (Type 1) programs we like to collect are useless with a larger DOS. But wait! If you try to use the ones with GO addresses of E00H, these will run! Part of the DOS' command jump table and all but one of these commands' routines are located in the last sector loaded, but this code isn't used by a Type 1 program. You will have to test any Type 1 programs used in this way. When you return to the DOS' warm start address (128H) the DOS can tell it has been partially destroyed and will display REBOOT. When ready, just hit RETURN. The DOS was obviously written this way on purpose and is another one of those little known features which makes North Star software so good.

Two of the programs I've tested are N*BUS and N*SORT from SZ Software Systems. N*BUS has no problem configuring itself to the new BASIC. N*SORT won't configure properly due to the changes in the new BASIC which allow an edit whenever an error is found. My thanks to Jim Lind (Vol. IV, No. 5, p. 18) for the fix to this problem! I tried numerous times without success.

Two other programs that won't configure properly to BASIC are SCAN and RENUMBER from ET Software Services. You can relocate 5.2 BASIC to 1000H with MOVER, then get SCAN (at least) to configure properly. Contact Jim Bailey for information about new versions of his excellent utilities.

North Star changed the HARD DISK ERROR message to DISK ACCESS ERROR, which is more informative.

Into the Future

The question here may be: "What future?" North Star Computers is still in business, but whether the Dimension Computer can keep North Star's head above water is yet to be determined. Fortune Systems' takeover bid didn't succeed although North Star did accept a loan when it needed one. This was reported in **InfoWorld** last September and, except for an occasional announcement or mailing, I haven't heard another word about the future of North Star Computers.

I'm not going to make any prognostications here about what the microcomputer market will be selling in 1990. I don't believe anyone can give you an accurate idea five years into the future. That IBM and Apple will be battling it out is fairly safe; as for North Star, who knows? I hope so!

This raises the question of what North Star programmers are going to do in the next five years. Sales of my software are severely limited and have been for several years. I hate to give up North Star BASIC, but must do something. Recently, I bought a copy of APCBASIC for use with Release 2.1.1 DOS. This BASIC is available for all of the major (business) operating systems and partly solves my dilemma (see APC's ad in the middle of my NOTES TO A SECRTARY, Vol. IV, No. 3, pp. 16-17).

APCBASIC is very fast, powerful, and compatible with North Star BASIC (see Bob Stek's review in Vol. III, No. 2, p. 13). I plan to write all of my future software with APCBASIC. The lowest-priced versions are $249.00 which is still a bit much to pay (it used to cost $400.00). APC gives a very generous discount to anyone who buys additional versions for other operating systems. The runtime versions are also powerful and can be sold by any registered owner for the very reasonable sum of $10.00 (no profit is made at this rate; income from runtime sales are sent quarterly to APC).

I'm currently rewriting my church contribution package for APCBASIC, and am planning the second rewrite of my disk catalog package.

If anyone can find other features of 2.1.1 system software which I missed, please let us know about them!

# # #

Midnight Express

By Peter Midnight

A few days ago I was a guest at a meeting of the INSUA board of directors. At that meeting, a small pile of letters from members was handed to me with the hope that I might be able to help out with some of the problems described in those letters or that they might help me come up with another article for the Compass based on solutions to some of those problems. I'd like to try to kill two birds with one stone by answering those letters in a general way that might be helpful both to the authors of those letters and to anyone who might be experiencing similar difficulties.

* * *

The first letter was from John Dent, of Australia. Along with some interesting economic and political comments, Mr. Dent described a recent experience with his Horizon which may sound familiar to a number of readers. It seems that after several years of "Unflagging service," his Horizon was modified to run at 6 Mhz instead of 4 Mhz. Within a week it died. "It was like a death in the family," says John. Of course, there is no real proof that the failure of Mr. Dent's Horizon was in any way attributable to the modifications it had just undergone. After all, it worked just fine for almost a week. However, as soon as he has the machine working again Mr. Dent plans to put it all back the way it was before and return to the "Trailing edge." Apparently he considers reliability more important than speed. Interestingly, Mr. Dent did not ask for any help in all of this. But his experience may be a help to others.

* * *

The second letter, a note from Lee Supowit of Vermont, puts a little bit of a new twist on an old question. People often ask about putting newer S-100 boards, designed around the IEEE S-100 standard, into their Horizons. Mr. Supowit, on the other hand, wants to put his North Star CPU and disk controller into a newer S-100 mainframe. Except for the need for new I/O ports and a new power supply for his disk drives, the problem is the same in either case, that is, how to make S-100 boards designed around different standards work together.

Back when the IEEE standards for S-100 systems were being proposed, the Horizon was already a well-established leader in the field. Therefore, there is very little in the standard that conflicts in any way with the Horizon. However, when the Horizon was designed, several of the lines on the bus had not yet been defined by any manufacturer of S-100 equipment. In general, these extra, undefined lines are grounded in the Horizon to improve the isolation and noise immunity of the other signals on the bus. Most compatibility problems with North Star S-100 boards are attributable to the grounding of these more recently defined signals. Fortunately, this problem was foreseen by the designers of the Horizon. And in most cases these extra ground connections are made with little jumper wires on the various boards. There has been some variation over the years, but for the most part these jumpers are labelled with a dotted line and the letter "G" on the board and are easily removed and replaced as needed.

Mr. Supowit also asks how to modify his DOS for this new machine. He is correct in assuming that he will need to prepare a new boot disk because it is very unlikely that the I/O facilities in the new box will exactly match the ports on the motherboard that was left behind. And he is smart to be thinking of this ahead of time. If his new machine has a front panel, a ROM monitor, or some other means of control, he could follow the instructions in the DOS manual for installing the DOS in a new system. But because he already has a working computer, it would be easier to use the old machine to prepare the boot disk for the new one, before he moves the boards from one to the other.

The first step is to write the new I/O routines on paper. There are six of them: COUT, CIN, TINIT, CONTC, ISTAT, and OSTAT. All six are defined in the DOS manual. And the source listings of the routines in the Horizon are provided. There is no way around the fact that these are machine language routines that have to be changed any time the I/O hardware changes. Just grit your teeth and do it. By doing it on paper, you can be a little vague in spots the first time around. But you have to get all of the details right before you can make your new I/O routines work.

Now that the hard part is done, make a new copy of your regular boot disk to be modified for the new machine. Don't ever hack up your only copy of your boot disk. Remember, no piece of software ever works right the first time. If it does, you're not testing it right.

All that remains to be done is to load a copy of the DOS into RAM, substitute the newly written I/O routines for the ones it contains, patch the six jumps for those routines in its jump table, and save the modified DOS back in its place on the new boot disk. This is the tricky part, especially in the case of a double density system. The place in memory where you find the jump table to be patched is the place where you put the copy of DOS you are changing, not at the normal load address of that DOS. The jump table at the normal load address is the one you are now using. If you patch that one you will go into hyperspace. The routines themselves are even more fun. They are written to run in a space starting 900h above the load address of the DOS. But if you have a double density system, the place you have to put them is not 900h above the jump table you have to patch. Instead it is 800h above that jump table. When you think you have found a good place to put your new routines, look carefully at what is already there. If you find a copy of your old I/O routines, you may well have guessed correctly.

Apparently, Mr. Supowit does not use any other operating systems besides the DOS in his Horizon. If he did he would need to modify each of them in a similar manner. However, they do not all need to be modified in his old computer before he takes it apart. Once he has the DOS running in his new machine, he can use it to modify the other systems. In fact, the I/O routines on the boot disks for CP/M and Pascal can be accessed as DOS files.

* * *

The next request is from Jerome Haggart, of New Jersey. He wants exact instructions for modifying all CP/M programs to run on the Horizon. The answer to this request is that almost all CP/M programs should run on the Horizon without any modification at all. On the other hand, almost all of those that don't run as-is require some specialized piece of hardware that would work fine in a Horizon but usually isn't there, such as a particular video board, modem, or pen plotter. Alternatively, some programs make calls to ports (for the monitor or printer, for example) that differ from computer to computer, and are thus again hardware-specific (see William Lawson's question below.) Since it is impossible to generalize about hardware, it is impossible to give a more comprehensive answer to this question.

* * *

When Maynard Fuller of Canada bought his Horizon five years ago, he did not do so for the purpose of learning anything. But given the opportunity, he has learned BASIC, FORTH, Pascal, C, and Prolog. At last he is finding that his horizons are starting to spread beyond his Horizon. He would like to explore the worlds of other operating systems besides DOS and CP/M. And he is concerned about some CP/M programs being available only on other media besides North Star diskettes. He is actively seeking ways to "Upgrade" his Horizon without sacrificing the advantages of a single user system.

I can't say which way Mr. Fuller should proceed. But I can suggest a couple of directions he might consider. One is to add a second entire computer to his system. Two different types of machines working together would be fertile ground for anyone with Mr. Fuller's obvious interest in learning new things. And the contrasts between the two computers would be fascinating. Currently fashionable computers tend to

be much more closed, dedicated machines than the Horizon, with only one operating system, one style of user interface, and only a handful at most of expansion slots. If the second machine were a portable, there would be even more contrast and new possibilities. Another direction Mr. Fuller might consider is a slave processor board, as he suggests in his letter.

To deal with Mr. Fuller's other concern if he does not add a second computer, auxiliary disk systems are available for both the Horizon and the Advantage to both read and write standard, 8 inch CP/M diskettes. The one for the Advantage comes with software to connect it into the existing CP/M system in that machine. Similar software is commonly used in Horizons with more than one floppy disk system, but is usually obtained separately from public domain sources. The basic principle on which these systems work is to extend the BIOS by hooking into its jump table and trapping the drive select calls from the BDOS. As each drive is selected, the extension software patches the BIOS jump table to use the appropriate set of routines for subsequent disk operations.

* * *

Willard Favre, Sr., of Louisiana asks a much more down to earth question: "How to program a simple menu to automatically come up on the CRT." Mr. Favre does not mention what language he would like to use to create this menu. But that's ok because the technique is basically the same in any language. The first step, of course, is to create the menu program and make it at least adequately functional. Then consider the steps you take after booting to invoke that menu. Ordinarily you will find that booting left you in the command mode of some operating system, that you first typed something to load a language or run-time package, and that you then typed something to load and run the menu program. In some cases, these two things you type will have been combined on a single command line or even into a single command. No matter how you have to type them, these are the two steps to automate.

Let's take the last step first. Every language provides a way to automatically load and run a program. In BASIC, for example, the technique is to save a copy of the BASIC with the program already loaded into it and with a flag byte set near the beginning of the interpreter. Some languages will not accept a whole program, but only the name of a program to be loaded from the disk. This makes it much easier to modify that starting program from time to time, such as to add options to the main menu. And it is common practice to also do it this way with languages, like BASIC, that are saved with a whole program, by making that program consist of only a single CHAIN statement. But for more spritely performance such languages could be saved with the menu program, itself. To find the precise details on how to set up any particular language for an automatic start you might actually have to read the instructions.

Similarly, every operating system provides a way to automatically load and run a machine language program, which is all a language or run-time package really is. In both DOS and CP/M there are command buffers which must be prefilled with a GO command or the name of a COM file (the first thing you would have typed after booting) and flag bytes that must be set to indicate that an automatic start is desired. And in both cases, that command buffer may be where the particular language you have chosen to use will look for the filename of the program you want it to run.

Again, directions for finding those buffers and bytes are in the dreaded manuals. But you might not really have to stoop so low as to read them. Both DOS and CP/M are provided with setup programs which will make all of these adjustments for you in all of the most common situations. For DOS, it is a BASIC program called SYSGEN. For CP/M, it is CPMGEN.COM. All you really have to do is produce that menu program, save it on a copy of your regular boot disk, and then run the appropriate setup program to make that new boot disk automatic.

* * *

The problem facing William Lawson of Washington does not appear to concern a

North Star product at all, but rather a public domain CP/M program recently distributed by INSUA, M712.COM. This is a general purpose modem program that does automatic dialing, file transfers inbound and outbound, data capturing, etc. Among this program's features is the ability to feed all incoming data to a printer, making a hard copy record of the exchange in real time, so that it can be referred to as needed during the phone call. Mr. Lawson's problem is that his printer is connected to the parallel port on his Horizon and he has found that it does not work with M712.COM. He has the source code of the overlay for the modem connection he is using. And he has been able to adjust the routines that access the modem to match the port to which he has it connected. But he did not find in this overlay any routines that he could adjust for access to his parallel printer. What he wants to know is how to modify M712.COM so that it will use the Horizon parallel port for the printer.

The big clue to the solution to this problem should be that M712.COM is a public domain program that was not written for the Horizon, in particular, but for any machine running CP/M. If the author of this program did not bury the modem access routines down inside the program where they could not be easily adjusted, he most likely did not do so with any other machine dependent code either. In the CP/M environment, any program that is written to be usable on various different machines will always use the facilities of CP/M whenever possible. The overlay system employed in M712.COM is necessary because CP/M was not designed to be used with modems and contains no facilities for accessing them. But it does provide both BDOS and BIOS calls for accessing a printer. It is, therefore, a good bet that the routines that need to be adjusted to allow access to Mr. Lawson's printer are not in M712.COM at all, but rather in the BIOS of his copy of CP/M.

Fortunately, North Star provides with its version of CP/M not only Digital Research's specifications for the BIOS, but also the source code of the portion of the BIOS that a user might wish to adjust, including the routines that access the printer. There are two such routines. The more commonly used of these simply sends one character to the printer. And, in fact, this routine is adjusted for you by CPMGEN.COM when it asks you if your printer is serial or parallel. However, if this routine is called when the printer or the port that drives it is not ready to accept another character, this routine will wait indefinitely for that situation to change. Therefore, this routine alone is not adequate for a modem program, which must also maintain communications between the terminal and the modem at the same time that it is sending data to the printer.

To allow the printer to be driven "In the background" while the processor is also doing other useful things, all but the oldest versions of CP/M also have a routine which can be called at any time to find out whether a call to the printer output routine would return right away or would cause the processor to wait. But, unfortunately, North Star does not provide this printer status routine in its BIOS. In its place North Star provides only a "Null implementation." This is a routine which does not check anything but always returns a not ready indication.

To avoid losing data from the modem or the terminal while waiting for the printer to accept a character, M712.COM will not call the printer output routine until the printer status routine says the printer is ready. And that's not going to happen until you replace that routine in your BIOS with something that really does test the status of your printer. DB 06 E6 01 2F 3C C9, for example, would test the PO flag on the Horizon motherboard and would probably make M712.COM suddenly start working with Mr. Lawson's printer. The same routine should also work for either of the serial ports on the motherboard or any of the ports on an HSIO-4 with the status register address for that port in place of the motherboard status register address of 06 in the second byte.

* * *

I have nothing but sympathy for Chuck Hardwick of Florida, and for North Star, as well, for that matter. Mr. Hardwick uses both NorthNet and Advantage 8/16s in his work. Both of these are among the most innovative and useful products North Star

ever developed. And the company has failed to see the wisdom of maintaining either one.

NorthNet, especially, rivals North Star's greatest achievements, although nothing can top being the first to make 5-inch floppies available for microcomputers. The number of bits that a computer uses and even the size of its memory are much less meaningful indications of its power than the number of processors it uses. Someday even the man on the street will know that, probably sometime after IBM figures it out. The network architecture is the most effective way yet developed to use multiple processors together cooperatively. This is the architecture that makes North Star's new computer, the Dimension, work so well.

But even the Dimension can't compete with a network of Advantages because it is all in one box. Whenever any of the shared components in that box fails, the whole network stops working. If an element of NorthNet fails, it is very unlikely to stop the network. And if it does, simply disconnecting it will allow the rest of the system to go on working without it. Even with no remaining file servers, when the fact that the network still works is not very useful, each of the remaining nodes is a complete and functional computer. In addition, the office wiring for a NorthNet system is nothing compared to that required for a Dimension.

The Advantage 8/16 was one of the first and is still one of the best multiprocessor personal computers. With both a Z-80 and an 8088 it runs both of the operating systems that have become accepted standards in the personal computer market, along with all of North Star's other single user software, as well. And when running MSDOS, it uses both of these processors together to outrun many other machines with real 16 bit processors, like the 8086. The best part is that it allows MSDOS programs to access CP/M files directly, as an alternative to the faster and more efficient file structure of MSDOS. This means that the whole problem of transferring files between these two operating systems simply does not exist for the user of an Advantage 8/16.

The only specific problem with either of these two systems that Mr. Hardwick mentions is "An inability to log to drive B" on the 8/16. I'm afraid I've never seen a problem like this. Almost all of the software development work on the Dimension project was done on Advantage 8/16s, both with and without hard disks. Some of those machines even had two half height floppies and a hard disk. To my knowledge, none of them ever exhibited any reluctance to log to B, or to any other drive.

* * *

Dr. Rux Jira of Thailand has a problem with the HD-5 he recently installed in his Horizon. He says he is unable to reset the system when it hangs. He also says the reset button is perfectly usable when booting the floppy disk system but does not cause a reset when running hard disk CP/M. I see two possible problems here that he may be talking about. The first is that maybe his system really does "Hang." The second may be that it does not boot from the hard disk.

We all know that computers aren't really supposed to ever hang. Of course, since we are not as perfect as they, sometimes we make mistakes that cause them to do that. But if a computer ever goes out into the weeds without being somehow forced to do so then something is wrong with that computer. If this is the case with Dr. Jira's Horizon, then it's not surprising that the reset button doesn't always work. If this is a new problem, it may well be attributable to the new hardware.

Dr. Jira did mention that he had had several hardware problems. It is quite possible, for example, that the new hard disk controller occasionally does something which incapacitates some part of the bus and thus the whole computer. If this happened, he would not be able to tell that the reset button had worked because the computer would still not function even enough to start up the drives. Turning the power off and on might make the problem go away for a while in a way that the reset button could not. This would prove that there was some sort of a hardware failure involved.

The other possibility that strikes me is that Dr. Jira might be expecting the reset

button to do something different when used with a hard disk than it does when used without one. He does not say the reset button works while running a floppy only system but rather while booting a floppy only system. He might be expecting the computer to perform a cold boot from the hard disk when using hard disk CP/M instead of booting from the floppy. It does not. There are several reasons for this. But most important is the requirement that the user have an opportunity to specify, each time the hardware is reset, just what software will be booted this time. Any given machine might be used at different times with several different operating systems and/or with several different automatically starting main menu programs. The most efficient and effective way for the user's desires to be specified is by his choice of which diskette to boot from. In fact, from the time you reset your computer until after that first diskette has been read, the machine doesn't even know it is a Horizon.

* * *

Finally, we have a letter from Chris Pesavento of California. He says he is using an Okidata 83A printer with "Okigraft I" and Image Maker with "Burigraph II". Mr. Pesavento has found that Image Maker does not work with his Okidata printer. And, further, he has found that his dealer thinks it should, even though it was written specifically for use with an Epson printer.

All of these popular little printers understand ASCII and will print the same text in response to the same signals from a computer. But there is no such standard way of representing a graphic image in those signals. Each printer manufacturer has to define the representation its printers will understand, and then has to publish it in the instruction books for those printers. Many manufacturers imitate each other's standards for the sake of compatibility. But each new printer has some new capabilities which must be represented in some new way. Often it is no more practical to make one printer act like another than to make it act like a pen plotter. An Okidata is not the same as an Epson.

This raises the question of why the dealer thought it would work. In the latter days of Image Maker, North Star provided free upgrades to users, much as they had for DOS and BASIC. However, instead of replacement diskettes at almost no cost, the upgrades for Image Maker took the form of lots of little CP/M files that just sort of floated around freely, as CP/M files are wont to do. Most of these files were new plotter drivers for connecting the existing Image Maker program to more and more new kinds of plotters. And among these were two plotter drivers that didn't really run plotters at all but instead ran Okidata printers. The dealer must have thought that Mr. Pesavento had the appropriate one of these files and knew how to use it.

To use one of these plotter drivers with Image Maker, you change its filename to DEFAULT.PLT. To print on an Okidata, you have to set up the Okidata driver file in this way and then tell Image Maker to plot, not to print. If the plotter driver is for a printer, Image Maker will then print, even though you told it to plot. If you tell it to print, it will send out its graphic data in the form understood only by an Epson or compatible printer, even if you don't have one. If, like Mr. Pesavento, you have both an Okidata printer and a plotter, then you'll have to rename a couple of files to switch back and forth between printing and plotting. (If you don't understand this paragraph, try reading it aloud real fast.)

Well, that just about does it for that little pile of letters. I'm sorry if yours was not among them. Maybe something in the discussion of one of these letters will be helpful to you now or sometime in the future. But if you have none of the problems covered here, that's better still.

# # #

Zap! Pop! Fizzle ...!

By Joe Maguire

The worst has happened. Finally old Zeke ... Herc ... Liberator ... or whatever you call your computer, has died. (You do have a name for your computer...don't you?) Now what do you do? Well, that's the purpose of this article.

There are a number of things you can try before giving up and calling the repair shop. (If you can find one that has ever heard of a North Star!) I'll stick to non-technical procedures which can be done right on your work table and without fancy test equipment.

Why do things go wrong?

Heat. That's the big bugaboo. Barring catastrophies like a lightning strike (I've had that happen!) or a power surge (far too common), excessive heat buildup is responsible for most computer equipment failures. Ninety-nine percent of the failures, from any cause, can be traced to bad ICs. The problem is to find them.

Preventative Maintenance

Before getting into specific trouble-shooting procedures, let's talk about a few things you can do before trouble appears. These can be good insurance.

If you have ever visited a large mainframe data processing center, you have already seen the signs: NO SMOKING, NO LIQUIDS, NO FOOD, ...practically no anything. These rules resulted from bitter experience. A smoke particle under a flying hard-disk-drive head can cause a catastrophic failure. A coffee spill can take down a million dollar piece of equipment for days. Modern technology has improved the tolerance of newer equipment to such abuses but the rules are still in effect. These should be the rules around your computer equipment also.

One insidious culprit is dust. Many of the personal computers being sold these days contain cooling fans but few offer dust filters. Over a period of several months of continuous use, the dust entering an unfiltered model gathers around the ICs forming a nice insulating blanket. Then the heat gremlin goes to work. Failure is only a short time away.

A cousin to the heat gremlin is the temperature cycling gremlin. If you use your computer like most of us, you turn it on when you start a work session and turn it off when you've finished. These temperature cycles can cause circuit boards to flex, inducing hairline cracks in circuit traces. But more often, they cause ICs to work up out of their sockets like rocks popping to the surface of your garden after the spring thaw.

Preventative Maintenance Rules

Rule #1. Periodically, about every six months or so, remove the cover from your computer and examine any socketed ICs for signs of looseness. You can press them back in with your finger. Be sure to provide support from behind before pushing on any S-100 plug-in board.

Rule #2. If you notice any dust buildup, get out your paint brush. That's right, a paint brush works wonders removing dust. Remove any plug-in boards and take the chassis to a ventilated area. Tip the chassis on its side and brush out all traces of dust and dirt. For tight areas I use a baby bottle brush. Don't forget to do the plug-in boards too.

If you notice a dust buildup, and your computer is a Horizon, I recommend that you install the fan filter kit as described in Compass, Vol. 4, No. 1, pp 9-12.

The Story of Norbert

This story is so incredible that I just have to share it with you.

Norbert was a Processor Tech Sol computer that belonged to a chain-smoking computer freak.

Norbert was in constant use for over five years in an atmosphere that must have resembled the inside of a chimney. Somehow he survived but his owner didn't. Norbert was given to me by the widow.

When I took his cover off, Norbert was so grungy inside that the ICs were

unrecognizable. The smoke film was so thick that it was impossible to make out the silk-screen markings on the motherboard. This was clearly a drastic case.

I tried the paint brush technique but that didn't begin to faze the grime. Then I had an idea.

Since Norbert had been given to me I figured I had nothing to lose. I disassembled the computer so as to separate the motherboard, power supply, etc., from the chassis. I then put everything that was waterproof into the dishwasher. That included just about everything but the power transformer and the fan. (There are no disk drives in a Sol.) After ten minutes on the wash and rinse cycles and about 20 on dry, I looked in to see the results. Norbert was as clean and shiny as a new penny! I reassembled all the parts and turned him on. He has been working faithfully here in my computer shack, alongside my Horizon, for the last four years.

I admit this was a drastic measure but in retrospect, I shouldn't have been surprised. ICs are waterproof as are the other components found on circuit boards. As long as everything is allowed to dry thoroughly before re-applying power, no shorts should occur. I'm not sure I would try it again on an expensive piece of equipment, but, the next basket case...??

Trouble Shooting

There are two general catagories of computer problems: partially malfunctioning and completely dead.

The dead case is the toughest to cure since the problem could be in one or more of several areas.

The Dead Duck

The first place to start with a dead one is the power supply. For this you should have a volt/ohmmeter, commonly called a multimeter, on hand. These are inexpensive and available at your local Radio Shack store. A multimeter is the single most useful piece of test equipment you can own. No computer shack should be without one.

AC power problems

To check if there is an AC power problem, see if the fan runs. If so, we go on to the DC power checks. If not, confirm that the AC wall outlet is live, that the power cord is good and that the fuse is OK. A blown fuse generally indicates further problems so if you find one, proceed with caution. Above all, do not replace it with a bigger fuse. If the correct fuse blows again, a short is indicated somewhere.

A defective ON/OFF switch is a possibility. Probing with the multimeter set on the ACV range should track this down.

DC power problems

Most power supply problems occur with the DC components. Filter capacitors can short, diodes and regulators can either short or open.

Check to see that the proper voltages are present by probing with the multimeter set on the DCV range. Refer to the schematic in your computer manual for the test points and the correct values.

If all voltages check OK, we move on to the the circuit board checks. If not, we proceed to isolate.

The isolation process starts at the power transformer and works toward the motherboard.

With a blown fuse, disconnect all output leads from the power transformer and, after replacing the fuse, reconnect one circuit at a time. When you connect the shorted circuit, the fuse will blow again. Now you proceed along this circuit path to isolate the short. The multimeter set on a low ohm scale can often detect a short without having to resort to the brute force method of fuse-blowing.

A missing DC voltage without a blown fuse indicates an open somewhere. Again, proceed from the power transformer, along the path of the circuit, while checking the voltage. When it disappears, you have found the open connection. Notorious for this problem are the terminal lugs where they are crimped to the wires. If you find a bad crimp, solder the lug to the wire instead.

Circuit Boards

A dead computer can be caused by a bad IC on the motherboard or on a plug-in S-100 board. Unfortunately, the Horizon is very vulnerable in this regard. If you have installed the PROM option on the CPU board, and are using your own startup program contained therein, you are in much better shape for trouble shooting.

In any event, whether the computer is dead or malfunctioning, once we determine the fault is with one of the circuit boards, the procedure is the same for either case.

Sick But Not Dead

The main trouble-shooting technique for an ailing circuit board is isolation. If you have some idea of where the trouble lies, you are halfway to solving the problem.

Where to Look

If your computer manual contains a function block diagram, it will help you determine where to look.

For example, in a Horizon, communication with the video terminal is handled by the USARTs on the motherboard. If the disk boots up but you get no signon message, nor can you command the computer to do anything, this might be one place to look.

Failure to bootup at all or repeated cycling of the drive heads may indicate trouble on the controller board.

Frequent program crashes often signal a bad memory board.

The real value of a users group becomes apparent when you have problems. Often, another member has had the same problem and can save you much time, frustration and money by telling you where to look. Don't hesitate to ask for help. That's what we are all here for.

About the only IC malfunction which can cause a completely dead computer would be the CPU. An actual CPU chip failure is rare but often one or more of the support ICs on the CPU board can go bad.

Before pulling out any ICs, make some voltage checks. Inherent in the S-100 bus design is that all plug-in boards contain their own voltage regulators. The regulator(s) drop the 8VDC from the power supply to the 5VDC needed by most of the ICs. A few ICs require other voltage values and if there are any of these on the board there will be additional regulators. The board schematic will identify which are which and what the voltages should be. In my experience, regulators fail about as often as ICs and, in fact, if an IC shorts, it will often take the associated regulator with it.

A very handy gadget for testing S-100 boards is an extender board. This allows the board under test to be raised above the other boards in the slots. This makes it much easier to get your multimeter probes to the desired test points. Extender boards cost less than $50 and are well worth it if you will be responsible for more than one computer.

If you find a bad regulator, replace it and then check its output voltage again. If the output is down by 10% or more, you probably have a bad IC elsewhere on the board and further checking will be required.

The Art of Fixing

Assuming the voltages on the board are correct, now we are down to testing the ICs. The easiest and fastest way to do this is by substitution. The easiest way to substitute is to have an identical board on hand. Just swap a few chips at a time from board to board until the trouble disappears. Don't forget to turn the computer off while removing or installing boards or ICs.

Where do you get the identical board? Ah, now you know the measure of a true friend. He is one who will lend you his computer for testing.

Before doing any IC swapping, swap boards instead. If this doesn't quickly clear up the problem, it might be on the motherboard or with the disk drives. Don't overlook your terminal either. I've had these go bad about as often as the computer.

If you're down to swapping ICs on the motherboard, be careful of the USARTs. These are the large 28-pin ICs. USARTs (and most other large ICs too) are static sensitive. Be sure to ground yourself

before touching any pins and try not to be working in an area of low humidity.

If you have swapped all the ICs on the board and the trouble is still present, I have one further suggestion. Many Horizons were built from kits. Occasionally I have found a weak solder joint that finally gave way. They are difficult to spot so I just go over the entire board with a soldering pencil and resolder each joint. There is no need to remove the ICs before doing this. You can confirm that your soldering pencil will not zap any ICs with stray voltage by checking with your multimeter from the tip to ground (before it gets too hot!)

Tools of the Trade

If you have fixed a problem or two yourself and are gaining confidence, you may wish to invest in some test equipment.

I have already mentioned the multimeter as the number one item.

An IC extractor and insertion tool is not expensive and will save many bent pins and punctured fingers.

If you want to get into signal tracing, you will need an oscilloscope or a logic probe. Oscilloscopes are expensive and are really not needed unless you want to service and align your disk drives. For general digital signal tracing, a logic probe will suffice.

Things like circuit analyzers and logic emulators are for professionals and are not needed for occasional trouble shooting.

* * *

The are a number of books available which will guide you through simple electronic servicing procedures. Recently, some servicing books have become available aimed specifically at the IBM PC. While not 100% applicable to the Horizon or Advantage, many of the techniques and procedures described will work with any computer.

Don't hesitate to get your feet wet, and good luck!

# # #

Coudal on WordStar

By Edgar F. Coudal

Users of WordStar! Is it a major pain to reformat a lengthy document to double-space print format, when you've created it as a single spaced document?

You know the problems. Manually inserting blank lines after every hard return. If justified, seeing the control QQ control B sequence end every time you run into a hyphen. And it's so slow, even with the 1 option to speed it up.

Here's the simple answer to end all those problems: At the first line of the program, put in the dot command .LH16. That makes the line height 3 to the inch, rather than six to the inch, which is single spacing and default setting. With .LH16 in place, the document will look like it's double spaced on the printout, but remain single-spaced on the terminal and thus remain easy to work on.

(Editor's note: The .LH command works for printers which can do small vertical increments like the Diablo 16xx and 6xx series, but not for printers which can do only full-line or half-line linefeeds like the Epson series.)

Another WordStar tip: Since I despise SpelStar as much as I hate WordStar, I use another spelling checker. Until I woke up, that meant leaving WS, entering the spelling checker, running it, then returning to WS to fix the errors. Finally, I realized that if I renamed the other spelling check SPELSTAR.OVR, WS, as stupid as it is, wouldn't know the difference and would indeed run the other program from the no-file menu just as if it were SPELSTAR. It does!

# # #

WordStar Patches

By Joel McClure, TECHWARE
474 Willamette Street, Suite 201
Eugene, OR 97401

(The enhancements described below work with both standard WordStar and CHARTECH. They may be reprinted without restriction other than crediting the author, his company, and his address. --Ed.)

Speeding up WordStar

When WordStar is invoked it clears the screen by sending a number of line feeds equal to the screen height (usually 24). WordStar 3.30 calls this routine twice. If your terminal has a command to clear the screen, you can speed up this operation. The code in WS.COM which sends the line feeds can be replaced by code which sends the "clear screen" command. The code segment in question differs with the version of WordStar. The code for 8-bit WordStars is given in the following table (all codes in hexadecimal):

WS 2.26 & 3.00		WS 3.30	
3DD0	LHLD 0248	3DE5	LHLD 0232
3DD3	CALL 1EBF	3DE8	CALL 1ED9
3DD6	DCR L	3DEB	DCR L
3DD7	JNZ 3DD3	3DEC	JNZ 3DE8
3DDA	RET	3DEF	RET

If the code for "clear screen" is one byte, change the segments to read as below, which is correct for the NorthStar Advantage. For most TeleVideo terminals change the 04 in the first line to 1A. (If you are using NorthStar's Enhanced WordStar, start the new code at 3DD3.) If you are not experienced in using the debug program (DDT or SID), read the section below called Patching.

WS 2.26 & 3.00		WS 3.30	
3DD0	MVI A,04	3DE5	MVI A,04
3DD2	CALL 106	3DE7	CALL 106
3DD5	RET	3DEA	RET

If the command is a two-byte sequence, follow the example below (which is correct for the Zenith Z29):

WS 2.26 & 3.00		WS 3.30	
3DD0	MVI A,1B	3DE5	MVI A,1B
3DD2	CALL 106	3DE7	CALL 106
3DD5	MVI A,45	3DEA	MVI A,45
3DD7	CALL 106	3DEC	CALL 106
3DDA	RET	3DEF	RET

Suppression of hyphenation

The authors of WordStar are doubtlessly proud of the hyphenation features. However, to many of us the symbol is a minus sign and it is annoying to have WordStar break an expression like a-b=c at the minus sign! We want WordStar to treat a "hard hyphen" as a regular letter, while treating the "soft hyphens" as before.

A simple one-line patch in the file WSOVLY1.OVR does the trick. For 8-bit WordStar the line should be changed from RZ to NOP. The addresses (in hexadecimal) are:

```
WS 2.26, 5902
WS 3.00, 63E7
WS 3.30, 656C.
```

Patching

Here is how to make the simple patches described above:

Put a disk containing DDT.COM in drive A and a disk containing the file to be patched in drive B. Log onto drive B. Type:

A:DDT filename

where filename stands for the full name of the file to be modified (WS.COM or WSOVLY1.OVR in the present examples) and ¼cr½ stands for "carriage return". When DDT comes up, note the number under "NEXT PC", call it ppdd, as it will be needed below when giving the SAVE command.

Begin actual patching by typing:

Ahhhh

where hhhh is the beginning hex address (3DD0 or 3D5E in the WS example). Be sure to type "zero" when needed and not "OH". The debugger will then show you the address and be ready to accept the first line of code. Type in the line given in the example, ending with "carriage return". The debugger then shows the address of the next line and is ready to accept it. After the last line has been accepted, type "period" (the symbol, not the word) followed by "carriage return". It is now wise to list the new code to check it. Type:

Lhhhh

and the debugger will show you the code. If you need to make corrections you can type:

Aiiii

where iiii is the address of the first correction. When you are sure that the code is correct type:

G0

where that is "zero", not "OH". When the monitor prompt returns, type:

SAVE nn filename

where filename is as before, and nn is a decimal number computed from the ppdd noted above. If dd is zero, reduce the hex number pp by 1. Then convert pp (or the reduced pp) to a decimal number to obtain nn. The following table contains all the numbers that should be found with normal WordStar versions.

ppdd:	3880	3900	3F00	4600	6C00	8400
nn:	56	56	62	69	107	131

# # #

ZCPR Continued

Robert W. Bloom
5 McCord Drive
Newark, Delaware 19713

(This is the third of four articles submitted by Bob Bloom. The first, on installing TEAC half-height drives, appeared in issue no. 1 of Vol. 5; the second, a general description of ZCPR, appeared in issue no. 2. --Ed.)

If you have gotten this far, you have installed the new drives in the horizon and have them working. This third article will tell you how to get the most out of them by patching the CP/M system. Software patches include reseting the 'Archive' directory bit (t3) on file modification, making files with the second user directory bit (f2) set 'public' or accessable from any user area, Plouffe's modifications to handle some non-standard but very useful disk formats (fully compatable with all standard formats), ZCPR2 buffer area initiation on cold boots, and Rich Conn's ZCPR2 system itself.

Originally this was going to addresses only Plouffe's mods, but the more the merrier. If one has trouble, just install one patch at a time and try them separately.

Also, this article talks only briefly about the reasoning behind the patch procedure - the previous one went into more depth why. I suggest that you read the second article if you are not sure of the basic patch procedure.

You will need some or all of following files. Names are from SIMTEL20 in ⟨CPM.*⟩ directories:

For Ploufe's alternate disk formats:

MICRO:⟨CPM.NSTAR⟩

FORMAT.DOC - if you want to know what it's all about
NEWCPY.16COM - replaces the NorthStar COPY.COM & handles the new formats
NEWFMT.14COM - replaces the NorthStar FORMAT.COM with new formats
NEWFRM.14ASM - the actual patches
GENSYS41.COM - NorthStar SYSGEN.COM replacement & handles the new formats

For ZCPR2 extended CCP:

MICRO:⟨CPM.ZCPR2⟩

ZCPR2.ASM - the actual CCP replacement (need not change)
ZCPRHDR.LIB - all the the equates and addresses to change

(That's all you **need**, but there's lots of utilities ⟨*.COM⟩, documentation ⟨*.WQ⟩ and help ⟨*.HLP⟩ files in this directory too.)

For the 'Archive' patch:

MICRO:⟨CPM.DSKUTL⟩

ARCHIVE.AQM - will backup all files with the archive bit (t3) reset and set it. It includes within the source code documentation and a bdos patch will cause the archive bit to be reset whenever a file is modified. File also included the BDOS patch.

For the 'Public' patch:

MICRO:⟨CPM.PUBPATCH⟩

PUBPATCH.LBR - a library consisting of the patch, a hex relocator, a setter/resetter, and SD and DISK7 patched to properly handle directory operations.

* * *

All of the following assumes a '64k' NorthStar CP/M system (actually 61k with a split BIOS) and that CPMGEN has been patched for the spurious $R/O BDOS error on the A: drive. (The patch is for CPMGEN.COM and changes ANI 02 and JNZ 7992 at 2F31H to MVI A,99 and STA 7FAB.)

1. If you want the special disk formats (very recommended), edit NEWFRM.ASM for the correct BIOS (F300) and CCP (D100) locations, set OCTAL TRUE and QUAD FALSE. (QUAD will not work if the USER code is changed with out further changes. And if you can do that, you really needn't be reading this anyway!)

```
;               NEWFORM  VERSION 1.4            *
;                     12/02/82                  *
;                R. L. Plouffe                  *

-- ETC.

OCTAL     EQU     TRUE          ;Want alternate octal format
                    ;for 2kb directory blocks?
QUAD      EQU     FALSE         ;Want alternate QUAD format
                    ;for 80 tracks?
;
BIOS      EQU     0F300H        ;BASE OF BIOS
CCP       EQU     0D100H

-- ETC. AS WRITTEN
```

2. If ZCPR2 is wanted (recommended), edit ZCPRHDR.LIB for the following addresses:

```
D100 CCP                          ZCPR2 Options:
D900 BDOS
E700 BIOS jump table              FB30 External Path
E800 Cold Boot PROM               EC00 Multi-Command Line
F300 "Real" BIOS                  EE00 External Stack
FA00 User Area                    EF00 External FCB
```

3. If ZCPR2 is desired, one also has to initialize certain buffers at cold boot time. Create a new USER.ASM, call it Z2USR, by adding the following code to the standard NorthStar-supplied USER area code to initialize the ZCPR2 system on a cold boot. (This Z80 code is from Frank Wancho and requires M80/L80 to assemble; convert to 8080 if necessary.)

```
aseg
.z80
.phase    0FA00H          ;ORIGIN FOR 64KQD SYSTEM

mcmdbuf     equ     0EC00h      ; ZCPR2 Multi Command-line (MLC) Buffer
```

```
--- normal USER area code (comment out serial printer code if parallel printer
--- and vice versa if you need more room.)

;WRAP IT UP BY SENDING A CR TO THE PRINTER & EXIT
;
     CALL     MULTI          ;ZCPR2 Initialization
     LD       C,0DH          ;A CARRIAGE RETURN
     JP       USERBAS-700H+15      ;GOTO PRINT JUMP IN BIOS VECTOR
;
;******* END OF STANDARD INITIALIZATION CODE *********

etc as written

;**** ZCPR2 INITS ***********
     ds       USERbas+130h-$      ; This fills to USERbas+130h
;
; External PATH is here:
;
pinit:     db       '$','$'           ; current disk/current user
     db       '$',0           ; current disk/user 0
     db       1,'$'           ; A:/current user
     db       1,0           ; A0:
     db       1,10           ; A10: for finding NAMES.DIR
     db       0           ; end of list
;
; Rest of space is available for PATH info up to USERbas+200h
; So we use this space below for the one-time init of the MLC
; only done at cold boot
;
multi:     ld       de,mcmdbuf
     ld       hl,clinit
     ld       bc,5
     ldir
     ret
clinit:    dw       mcmdbuf+4
     db       200
     db       0

     ds       USERbas+180h-$

     end
```

4. If you would like the BDOS 'archive patch' to reset the archive bit (t3) whenever a file is modified, extract the BDOS patch from the source code, remove the comments and patch in the correct addresses:

```
bdos$loc      equ      0D900H          ; base address of BDOS
ccp$base      equ      1500h          ; sysgen ccp base position
```

Alternately, get the pubpatch library and extract PUBPATCH.HXR and RELPUBLC.COM. Note that 'archive' and 'pubpatch' are mutually exclusive - one cannot have both.

5. Now assemble the parts. (In the command lines below, 'x' should be replaced with a drive letter indicating where the source is first x and where the HEX file second x should be placed.)

```
assemble the archive overlay to BDOS:
  ASM ARCPATCH.xxZ
or create a pubpatch.hex by running:
     RELPUBLC and giving a BIOS location of E700

assemble the USER area overlay with ZCPR2 inits
  M80 =ZUSR/M
  L80 ZUSR,ZUSR.OVR/N/E

Plouffe's mod's for 2k block size on 96-tpi drives
  ASM NEWFRM.xxZ

Finally assemble ZCPR2 proper
  MAC ZCPR $Ax Hx PZ SZ
```

6. Actual assembly and patch. The order of overlaying the various items went as follows to create NEWCPM.COM (a SYSGEN-like program containing the modified SYSGEN image):

```
NCPMGEN                 -- REMEMBER TO USE PATCHED VERSION!
ETC ...                    CREATE A VANILLA SYSTEM OF SIZE REQUIRED
                           SAVE VANILLA SYSTEM ON DRIVE 1
GENSYS                  -- TO READ SYSTEM INTO MEMORY (USE THE NEW
                           SYSGEN-LIKE PROGRAM!
1                       -- FROM DRIVE 1
^C                      -- EXIT WITHOUT WRITTING TO ANOTHER DISK
SAVE 51 SAVECPM.COM     -- SAVE THE SYSTEM INTO A NORMAL FILE
DDT SAVECPM.COM         -- PUT IT INTO DDT
IARCPATCH.HEX           -- ADD THE ARCHIVE BDOS PATCH
R                       -- NO OFFSET NEEDED, SYSGEN IMAGE ADDRESS
                           HARD CODED IN
                           (or alternately, use PUBPATCH)
IZUSR.OVR               -- AND THE USER AREA
R3100                   -- OFFSET NEEDED BECAUSE ZUSR.OVR IS NOT A
                           HEX FILE, SYSGEN IMAGE ADDRESS IS 3200H,
                           LOAD ADDRESS 0100H SO USE 3200-0100 = 3100
INEWFRM.HEX             -- SPECIAL MODS FOR QUAD/NINE DRIVES
R3800                   -- SYSGEN IMAGE @3200 - SYSTEM LOCATION @FA00
                           GIVES OFFSET OF 3800
F1500,1CFF,00           -- CLEAR THE OLD CCP OUT
IZCPR.HEX               -- ADD IN THE ZCPR2
R4400                   -- SYSGEN IMAGE @1500 - SYSTEM LOCATION @D100
                           GIVES OFFSET OF 4400
^C                      -- EXIT DDT
SAVE 51 NEWCPM.COM      -- AND SAVE THE PATCHED SYSGEN IMAGE (opt)
GENSYS                  -- READY TO  LOAD IT
CR                      -- SYSTEM IN MEMORY SO NO DISK READ
1                       -- WRITE IT BACK OUT TO DRIVE 1
CR                      -- COLD BOOT TO BRING IN NEW
```

(As I stated in the previous article, there is a way to skip the 'SAVE 51' lines if you care to.)

7. Now, with the new system in place, one can format disks with the 'O'ctal format of NEWFMT which gives a 2k block size. HOWEVER: beware using the octal format with any CP/M without the NEWFRM patch - sometimes it'll work, sometimes it won't, and you might not know the difference intil it's too late. Also, never try to copy 'O' format disks

with the old North Star COPY.COM, and never use the NorthStar SYSGEN.COM - use NEWCPY.COM and GENSYS.COM instead.

* * *

The above patched system is my everyday, normal working system. I have developed a special ZCPR3-based system for developmental purposes. The ZCPR3 system is of such a depth that one cannot quickly use all its facilities without considerable study. Installing the ZCPR3 system is the concern of the next of my articles.

# # #

Straight-Jacket: Rx for Disks

By Alan H. Nelson

Recently I have had troubles with two abused diskettes. The first had been placed under a heavy book. The jacket was so cruched down that the disk was pinched within and could not be made to spin, even by hand. The second disk had got bent by being caught in a pile of paper which was not lying flat. Again the disk would not spin within its jacket. My computer announced a disk-read error, that terrible message T 4 D 1 S 0000 (or some variant thereof).

In both cases my solution to the problem was the same. I carefully cut the jacket open along the back, then cut open the jacket of another disk which had been so flaky that I had meant to discard it. I slipped the damaged circle of magnetic medium out of the "good" jacket, and set it aside. Then I carefully slipped the good circular disk from the damaged jacket and slipped it into the undamaged jacket, making certain that the front of the medium kept facing the front of the jacket. Finally I placed the good-disk/good-jacket assembly into my disk drive, and found that I could read the disk perfectly.

The point is this: the circular magnetic disk is much more flexible than the jacket which protects it. Though the outer jacket might not recover from pinching or bending, the magnetic medium may very well be undamaged. (If the magnetic medium is folded or creased, you are probably beyond help!)

The design of the disk drive is helpful in executing this rescue technique. It is not necessary to do more than insert the magnetic disk loosely into the jacket to read the disk. The cut edge of the jacket does not need to be taped or fastened shut because the spindle in the disk drive will center the medium and hold it in proper position while reading the disk.

I found that scissors were much handier than a knife or razor blade for opening up the jacket - just cut off the smallest slice of the edge, and you should find that the disk can slip right out. It shoudn't even hurt if you nick the edge of the disk, but don't push your luck too far--take care!

The easiest procedure is to open the **outside edge** of the disk: if you think of the disk as a book when you insert it into a drive, you will be slicing off its cover along the spine. You could also cut off the top, but it would be a big mistake to cut off either the front edge or the bottom of the jacket as you might end up with the magnetic disk falling into the drive.

Of course this is not a permanent way to deal with a particular disk. Once you manage to read a disk, copy the whole thing permanently to a fresh disk, then throw away the original magnetic medium. (It might be a good idea to keep the good jacket around for future use.)

It isn't even necessary to have an old, flaky disk at hand, since opening up even a brand new disk for the sake of its jacket will probably seem like a good investment compared to the loss of programs or files on the pinched or warped disk.

# # #

Make Your Horizon an RCPM

By Glenn C. Steiner
440 Cesano Ct 310
Palo Alto, Ca 94306

There are complete code listing for your personal Remote CPM System at the end of this article. They total to almost 800 lines. If you do not wish to type the program into your system, you may order a copy of the software in North Star CPM format, DSDD. If you are interested in the program as well as the associated software you may order it at the address above with a check for $15. This offer is good for twelve months from the date of the article, i.e. until October 1986.

Many of us have used the Remote Bulletin Board Services that exist near or far from our homes. Probably at some time you have wished to be able to turn your computer into a Remote CPM (RCPM) for your personal use. This may have occurred when you wanted to fix that 'last bug' during your lunch hour at work or when you were at a friend's house and wanted to show off your latest program. The program which I have produced will allow you to turn your Horizon computer into such a system.

The personal RCPM program has several useful features that are most important for the environment in which it was designed to be used. First, the program has password protection to prevent unauthorized access. It also has automatic detection of carrier loss, followed by hanging up the phone, resetting of the computer and rebooting of the RCPM program. This will prevent your system from locking up your phone for hours, and will prevent unauthorized access to your computer which may have been left in an accessible state. It will permit the proper reboot of the system so that you may call back in a few seconds, rather than driving home to reset and reboot your computer.

Further, the program was designed to be used on a shared voice line. Thus, your friends who call will not be blasted out by the modem's carrier signal. The system is set up such that the computer will only answer the phone after the sequence of a single ring followed by a hangup and then another ring within 30 seconds. Of course you will have to restrain your wife and kids from answering the phone after only one ring.

Next, the RCPM program supports parallel operation from the console and the modem once a user is logged on. Consequently, you can watch or help a friend who is remotely using your system. Additionally, the program may be easily aborted when waiting for a call by simply typing the space bar on the console. Finally, automatic baud rate selection is supported by the system via the detected carrier signal.

The program does have a few drawbacks. Most of these problems result from the design philosophy of the program. The personal RCPM system was designed to be a single user system. And that user is YOU. Thus the program only supports one password and does not provide any means by which the users or usage of the system may be logged. (Of course you can add these features if you wish to modify the program.) The system also does not have any message facility. Finally, and most importantly, the personal RCPM system does not support any form of software protection. Thus, if a user decides to erase all of your files, he can. I understand that if you have ZCPR, you can build such protection in. However, as I said earlier, if the only user is you then you are not likely to damage your system and you will want full access to it.

Once installed the personal RCPM program is very easy to use. To start it you will run the program BYE from CPM. The program will quickly check your phone and modem connection by taking the phone line off-hook for a second. If you hear a dial tone from the modem speaker you will know that the system is

ready to go. The program will then notify you that it is waiting for a call. Then you can go over to a friend's place to try out your remote system.

Call home once, hang-up, and call again. Your computer and modem will then answer the phone with a carrier. Follow this by placing your friend's modem online. Your remote system will detect the carrier from the local modem and adjust its carrier and baud rate to the correct speed. Now that the connection is made, you will see your personal sign-on message as well as the request for a password. Type in your personal password and in a few seconds you will see your old familiar CPM prompt. Now you will be able to run your system as if you were at home.

To logout run the program BYE again. If you should forget to sign out or if you lose carrier while you are connected, your system at home will automatically hang-up, and run BYE for you. It will then be ready and waiting for your next call.

While you may experiment with other arrangements, the personal RCPM program was designed to operate with a particular configuration. The program expects to be run on a North Star Horizon with a Z80 board, CPM2.X, 56K of memory and a Hayes Smartmodem 1200. The program also requires 40 bytes of unused RAM in your system. This RAM may be any free space on your system.

Personal RCPM was written in Turbo Pascal. If you don't have a copy of Turbo Pascal, then I would like to strongly recommend your buying it. It is a really first rate programming system and allows generation of independent .COM files. Of course you can modify the program for another version of Pascal as well as different hardware.

* * *

Now, for the usual warnings. The personal RCPM program is very easy to use once it is properly configured. However, It does require a fair amount of setup to support all of its features. The setup is a combination of both hardware and software modifications to your system. If you are not skilled at both then you should probably just enjoy reading this article.

DO NOT ATTEMPT THESE HARDWARE AND SOFTWARE MODIFICATIONS OF YOUR COMPUTER IF YOU ARE NOT SKILLED AT BOTH HARDWARE AND SOFTWARE RECONFIGURATIONS. MODIFICATIONS MADE TO THE READER'S SYSTEM ARE DONE AT THE READER'S OWN RISK.

Now, in hopes of making the lawyers happy:

The author cannot accept responsibility for any damage caused by readers to their hardware or software in attempting to follow the configuration directions. The author also cannot accept responsibility for any damage caused to the reader's hardware or software due to inaccuracies in this article. Finally, the author cannot accept responsibility for the proper operation of the user's system or the program with or without the listed modifications.

I am sorry about the above paragraph and I do believe the patches listed below are correct. However, there have been some awfully frivolous lawsuits of late.

* * *

First, let's start with the hardware modifications:

1. Set the SMARTMODEM 1200 switches as follows:

1	up
2	up
3	down
4	up
5	down
6	up
7	up
8	down

2. Wire a male to male RS232 cable as follows:

Modem End		Computer End	
1	GND	1	GND
2	TxD	2	TxD
3	RxD	3	RxD
5	CTS	5	CTS

6	DSR	6	DSR
7	GND	7	GND
20	DTR	20	DTR
22	Ring Indicate	8	Carrier Detect
12	High Speed Flag	15	Right Special Clock R-DB

3. Install the cable between PORT-2 on your Horizon and the modem. Note that the modem and the computer ends of the cable must not be reversed when you plug in the cable.

4. Configure PORT-2 on your Horizon for 1200 baud:

Baud Rate Header (2D): Connect pin 13 to pins 5 and 6.

5. Clock interrupt on VI1 of 3.4 seconds for carrier loss detection:

A. Clock Rate Header (10A):

Connect pin 13 to pin 4.

B. Interrupt Header (1A):

Connect pin 16 to pin 2.

6. High Speed Detection (Right Serial Port (pin 15) to Status Port-In (bit 3):

A. Solder a wire on the clock header (2C) pin 1 and connect other end to Jumper J3 at the side of U3. (Make sure Jumper J3 is disconnected from ground. This requires the cutting of the PC board between the J3 jumper points.)

B. Solder a jumper wire on the clock header (2C) pin 2 to pin 12.

* * *

Now for the software configuration:

1. Modify your CPM I/O drivers so that both serial ports may be run in parallel via software control when the console is assigned to USR:. Listed at the end of this article are sample routines. This is probably the toughest part of the system configuration. The RCPM program also assumes that the console is assigned to TTY:.

2. Locate about 40 bytes of unused RAM in your system configuration. This is where the carrier watchdog routine will reside.

Change the INT_PROGRAM_ADDRESS constant in the program BYE to this address. (The interrupt service routine is self relocating.)

3. Change the PASSWORD constant in the program BYE to your personal password.

4. Modify your CPM autostart to execute the program AUTO upon a cold boot. (AUTO will stream the command file AUTO.SUB which will reset the system console and then run the program BYE).

That should do it. Compile the Pascal program to produce a .COM file on drive A:. On my system I call the program BYE.COM. Thus, when I call in and decide to logout I just type A:BYE to produce a soft logout. Of course you can also just hangup and the system will produce a hard logout.

* * *

Now a few random notes about the RCPM program. The constant section contains many of the parameters that are system dependent. You should check these values carefully to see that they match your hardware configuration. The program uses interrupt vector one. Make sure that nothing else uses this vector. The watchdog interrupt program requires 40 bytes of free memory. Since I have the Lifeboat CPM, I have the top 4K of my system free. Thus, I have the program load the watchdog routine at FC00h. You may or may not be able to do the same. The assembly code of the watchdog routine is listed as part of the BYE program.

The object code is duplicated as a character string which is loaded at the user specified address. The routine is self relocating so you will not have to re-assemble it if you decide to place it at a different address. Your CPM system must support the IOBYTE via the BDOS(7) call. If not you will have to add this feature.

This would be part of the Software Configuration step 1 listed above. The Procedure Timer is used to provide one tenth second ticks for the various time dependent routines in the program. It makes use of the Turbo Pascal software delay. Make sure that you properly configure Turbo Pascal for your system speed.

Procedure Writeout is designed to output characters to the modem only when a carrier is present. This should prevent the program from inadvertently programming the modem when a carrier signal is lost and the program is outputting a message to the remote user. Function String_Read also checks for carrier while an input string is read. This will prevent the program from hanging if carrier is lost when a remote user is attempting to log in.

Most of the other procedures in the program are for proper login sequencing and for error checking. They have been carefully checked for all sorts of strange phone call sequences and bad login sequences. The code is heavily commented so you should have little trouble modifying the code if needed. The code is also fairly modular.

Good Luck!!!

# # #

Console/Modem Input and Output Drivers:

I have all of my I/O drivers in ROM. They are exact maps of the standard CPM I/O calls. The routines listed below are for when the console is assigned to USR:. It assumes that your system has the Horizon Port-1 assigned to TTY:, your standard console and Port-2 assigned to CRT:, an alternate console. The routines also require that the CPM IOBYTE is properly set up for input and output definitions.

```
;       Special I/O Drivers
;
;       Modified for public domain March 1984
;       Glenn Steiner
;
;       Input and Output defined for North Star Horizon
;
;
        ORG    0F000H  ;ORIGIN FOR IO DRIVERS IN ROM
;
BASE    EQU     $
USER    EQU     BASE+700H
;
;
;CPM IOBYTE LOCATION
;
IOBYT   EQU     03H     ;CPM IOBYTE
;
;
HRZPROM EQU     0E800H  ;HORIZON BOOT PROM ADDRESS
RESHRZ  EQU     006H    ;HORIZON MOTHERBOARD RESET PORT
HRZSTAT EQU     006H    ;HORIZON STATUS INPUT PORT
PARALLEL EQU    000H    ;PARALLEL I/O PORT
;
;       I/O DEVICES
;
;-TELEPRINTER (CONSOLE)
;
TTI     EQU     2       ;DATA IN PORT
TTO     EQU     2       ;DATA OUT PORT
TTS     EQU     3       ;STATUS PORT - INPUT
TTYDA   EQU     2       ;DATA AVAILABLE MASK BIT
TTYBE   EQU     1       ;XMTR BUFFER EMPTY MASK
;
;-CRT SYSTEM
;
CRTI    EQU     4       ;DATA PORT - INPUT
CRTS    EQU     5       ;STATUS PORT - INPUT
CRTO    EQU     4       ;DATA PORT - OUTPUT
CRTDA   EQU     2       ;DATA AVAILABLE MASK
CRTBE   EQU     1       ;XMTR BUFFER EMPTY MASK
;
;       CONSTANTS
;
CR      EQU     0DH     ;ASCII CARRIAGE RETURN
```

```
LF      EQU     0AH     ;ASCII LINE FEED
;
;       I/O CONFIGURATION MASKS
;
CMSK    EQU     11111100B       ;CONSOLE DEVICE
RMSK    EQU     11110011B       ;STORAGE DEVICE - IN
PMSK    EQU     11001111B       ;STORAGE DEVICE - OUT
LMSK    EQU     00111111B       ;LIST DEVICE
;
;-CONSOLE CONFIGURATION
;
CTTY    EQU     0       ;TELEPRINTER
CCRT    EQU     1       ;CRT
BATCH   EQU     2       ;READER FOR INPUT, LIST FOR OUTPUT
CUSE    EQU     3       ;USER DEFINED
;
;-STORAGE INPUT CONFIGURATION
;
RTTY    EQU     0       ;TELEPRINTER READER
RCRT    EQU     8       ;CRT READER
RPTR    EQU     4       ;READER PORT
RUSER   EQU     0CH     ;USER DEFINED
;
;-STORAGE OUTPUT CONFIGURATION
;
PTTY    EQU     0       ;TELEPRINTER PUNCH
PCRT    EQU     20H     ;CRT PUNCH
PPTP    EQU     10H     ;PUNCH PORT
PUSER   EQU     30H     ;USER DEFINED
;
;-LIST DEVICE CONFIGURATION
;
LTTY    EQU     0       ;TELEPRINTER PRINTER
LCRT    EQU     40H     ;CRT SCREEN
LINE    EQU     80H     ;LINE PRINTER (EXTERNAL ROUTINE)
LUSER   EQU     0C0H    ;USER DEFINED

;
;       PROGRAM CODE BEGINS HERE
;
;
BEGIN   DEFS    3       ;GO AROUND VECTORS
;
;       VECTORS FOR CALLING PROGRAMS
;
;THESE VECTORS MAY BE USED BY USER WRITTEN
;PROGRAMS TO SIMPLIFY THE HANDLING OF I/O
;FORM SYSTEM TO SYSTEM.  WHATEVER THE CURRENT
;ASSIGNED DEVICE, THESE VECTORS WILL PERFORM
;THE REQUIRED I/O OPERATION, AND RETURN TO
;THE CALLING PROGRAM.  (RET)
;
;THE REGISTER CONVENTION USED FOLLOWS-
;
;ANY INPUT OR OUTPUT DEVICE-
```



```
;       CHARACTER TO BE OUTPUT IN 'C' REGISTER.
;       CHARACTER WILL BE IN 'A' REGISTER UPON
;       RETURN FROM AN INPUT OR OUTPUT.
;'CSTS'-
;       RETURNS TRUE (OFFH IN 'A' REG.) IF THERE
;       IS SOMETHING WAITING, AND FALSE (00) IF NOT.
;'IOCHK'-
;       RETURNS WITH THE CURRENT I/O CONFIGURATION
;       BYTE IN 'A' REGISTER.
;'IOSET'-
;       ALLOWS A PROGRAM TO DYNAMICALLY ALTER THE
;       CURRENT I/O CONFIGURATION, AND REQUIRES
;       THE NEW BYTE IN THE 'A' REGISTER.
;
CI      DEFS    3       ;CONSOLE INPUT
RI      DEFS    3       ;READER INPUT
CO      DEFS    3       ;CONSOLE OUTPUT
PU      DEFS    3       ;PUNCH OUTPUT
LO      DEFS    3       ;LIST OUTPUT
CSTS    DEFS    3       ;CONSOLE STATUS
IOCHK   DEFS    3       ;I/O CHECK
IOSET   DEFS    3       ;I/O SET
;
;
;       VECTORS FOR USER DEFINED ROUTINES
;
        ORG USER

XCILOC  JP      CIBOTH  ;USER CONSOLE INPUT--OUTPUT TO CRT AND TTY
XCOLOC  JP      COBOTH  ;USER CONSOLE OUTPUT--INPUT FROM EITHER CRT OR TT
XRPTPL  JP      CI      ;HIGH SPEED READER      (NOP - USE CONSOLE)
XRULOC  JP      CI      ;USER DEFINED STORAGE - INPUT
XPTPL   JP      LPRT    ;HIGH SPEED PUHNCH
XPULOC  JP      LBOTH   ;USER DEFINED STORAGE - OUTPUT
XLNLOC  JP      LPRT    ;LINE PRINTER
XLULOC  JP      LBOTH   ;USER DEFINED PRINTER
XCSLOC  JP      CSBOTH  ;CONSOLE INPUT STATUS ROUTINE
;
;
;DRIVER FOR CONSOLE OUTPUT TO BOTH TTY AND CRT
;
COBOTH  CALL    IOCHK   ;FETCH IO STATUS
        PUSH    AF      ;AND SAVE IT
        CALL    TTYCHK  ;SET UP FOR ADDRESSING TTY
        CALL    CO      ;OUTPUT THE BYTE
        POP     AF      ;RESTORE OLD IOBYTE
        PUSH    AF      ;AND SAVE IT AGAIN
        CALL    CRTCHK  ;SET UP FOR ADDRESSING CRT
        CALL    CO      ;OUTPUT THE BYTE
        POP     AF      ;RESTORE THE ORIGINAL IO BYTE
        CALL    IOSET
        LD      A,C     ;PUT THE BYTE WE OUTPUT IN ACC
        RET             ;AND RETURN
;
;DRIVER FOR INPUT FROM EITHER TTY OR CRT
```

```
;
CIBOTH  CALL    IOCHK   ;FETCH IO STATUS
        PUSH    AF      ;AND SAVE IT
        CALL    TTYCHK  ;CHECK TTY FOR DATA READY
        JR      NZ,DATAIN-$;INPUT DATA IF READY
        POP     AF      ;FETCH BACK ORIGINAL IO STATUS
        PUSH    AF      ;AND SAVE IT AGAIN
        CALL    CRTCHK  ;CHECK CRT FOR DATA READY
        JR      Z,CILOOP-$;IF NOT THEN LOOP BACK
DATAIN  CALL    CI      ;DATA IS WAITING--FETCH IT
IOEXIT  EX      (SP),HL ;GET OLD IOBYTE INTO HL
        PUSH    AF      ;SAVE INPUT BYTE
        LD      A,H     ;RESTORE THE OLD IOBYTE
        CALL    IOSET
        POP     AF      ;RESTORE THE BYTE READ
        POP     HL      ;RESTORE HL
        RET
CILOOP  POP     AF      ;NO DATA READY YET
        CALL    IOSET   ;RESTORE IOBYTE
        JR      CIBOTH-$;AND TRY AGAIN
;
;DRIVER FOR CONSOLE STATUS FROM EITHER TTY OR CRT
;
CSBOTH  CALL    IOCHK   ;FETCH IO STATUS
        PUSH    AF      ;AND SAVE IT
        CALL    TTYCHK  ;CHECK TTY FOR DATA READY
        JR      NZ,IOEXIT-$;IF DATA IS READY THEN EXIT
        POP     AF      ;FETCH BACK ORIGINAL IO STATUS
        PUSH    AF      ;AND SAVE IT AGAIN
        CALL    CRTCHK  ;CHECK CRT FOR DATA READY
        JR      IOEXIT-$;EXIT WITH STATUS
;
;TTY STATUS CHECK
;
TTYCHK  AND     CMSK    ;MASK OUT CONSOLE
        OR      CTTY    ;SET FOR TTY DEVICE
TTYC1   CALL    IOSET
        CALL    CSTS    ;SEE IF TTY HAS INPUT
        OR      A       ;TEST FOR DATA READY
        RET
;
;CRT STATUS CHECK
;
CRTCHK  AND     CMSK    ;MASK OUT THE CONSOLE
        OR      CCRT    ;SET FOR CRT DEVICE
        JR      TTYC1-$ ;REST OF CHECK IS AS ABOVE
;
        RET
;
THEEND  END     BASE
```


Auto Start Program:

This is a simple auto start program which must be assembled and placed on drive A as AUTO.COM. Your CPM should also be configured to run this program upon a cold boot. Follow your CPM configuration notes on how to configure it for an auto boot program. Even if you never use the RCPM system you may find this program to be very useful.

```
*********************************************************
*                                                       *
*              AUTO START PROGRAM                       *
*                                                       *
*********************************************************
;
;PROGRAM IS RUN AS THE CPM AUTO-START PROGRAM
;       THIS PROGRAM STREAMS THE FILE AUTO.SUB
;       AS "SUBMIT AUTO"
;
;DATA FOR AUTOSTART
;
BDOS    EQU     0005H   ;CPM BDOS ENTRY POINT
;
        ORG     0100H   ;PROGRAM ORIGIN AT 100H
;
;AUTOSTART COMMAND INSERTION
;
AUTO    LHLD    0001H   ;GET ADDRESS OF BIOS+3
        MVI     L,00H   ;ADJUST TO BIOS BEGIN
        MOV     A,H     ;SUBTRACT OFFSET TO GET THE BEGIN
        SUI     16H     ;OF THE CCP
        MOV     H,A     ;PUT IT BACK IN HL
        SHLD    CCP     ;SAVE CCP LOCATION FOR LATER USE
AUTO1   LXI     D,COMND ;POINT TO THE COMMAND LINE
        LXI     B,CMDLEN+2      ;SET TO LENGTH OF COMMAND + 1
        LHLD    CCP     ;GET CCP LOCATION
        MVI     L,07    ;LOCATION OF LENGTH IS AFTER CCP
        CALL    MOVE    ;MOVE COMMAND STRING
        LHLD    CCP     ;GET BACK CCP LOCATION
        MVI     L,88H   ;LOCATION OF LSP POINTER
        MVI     A,08H   ;GET DEFAULT LSP START
        MOV     M,A     ;PUT IT IN PLACE
        INX     H       ;LOCATION OF MSP POINTER
        MOV     A,H     ;GET DEFAULT MSB START
        MOV     M,A     ;PUT IT IN PLACE
        LHLD    CCP     ;GET BACK CCP LOCATION
        PCHL            ;AND EXIT
;
MOVE    LDAX    D       ;GET BYTE TO MOVE
        MOV     M,A     ;MOVE IT
        INX     H       ;INCREMENT DESTINATION
        INX     D       ;INCREMENT SOURCE
        DCX     B       ;DECREMENT COUNT
        MOV     A,B     ;GET COUNTER INTO A
        ORA     C       ;CHECK IF DONE
```

```
        JNZ     MOVE    ;LOOP BACK IF NOT DONE
        RET             ;EXIT IF DONE
;
;
;DATA FOR AUTOSTART
;
CCP     DS      2       ;ADDRESS OF CCP
CMDLEN  EQU     11      ;LENGTH OF COMMAND IN CHARACTERS
COMND   DB      CMDLEN  ;COMMAND LENGTH IN CHARACTERS
        DB      'SUBMIT AUTO'   ;THE COMMAND
        DB      00      ;TRAILING ZERO AT END OF COMMAND
END
```

Submit Program for Auto Start:

These two lines should be placed in a file named AUTO.SUB drive A. This short program will be submitted by AUTO.CO whenever a cold system boot is made. It will reset you console to TTY: (this should be where your console is) an then run the personal RCPM program, BYE. If you are at hom and do not want to automatically run the RCPM program the just edit the AUTO.SUB file and eliminate the BYE command.

```
STAT CON:=TTY:
BYE
```


Personal RCPM Program BYE:

This is the Pascal code for the personal RCPM program. It should be compiled using TURBO pascal into a COM file and then placed on drive A:

```
{$B+,C-,I+,R+,V+,U-,X+,A+,W2}           {C+ AND U+ ENABLE CONTROL C INTERRUPT}

                {Personal Remote CPM System}

{
                    COPYRIGHT NOTICE

This software and associated documents are copyrighted 1984 by Glenn C.
Steiner.  All rights reserved worldwide.  No part of this software or
documentation may be reproduced, transmitted, transcribed, stored in any
retrieval system, or translated into any language by any means without the
express written permisssion of:
 Glenn C. Steiner, 440 Cesano Ct. 310, Palo Alto, CA 94306   415-949-2468

                    SINGLE CPU LICENSE

The price paid for one copy of this program licenses you to use the product
on one CPU only.

                    DISCLAIMER

Glenn C. Steiner makes no warranties as to the contents of this program or
associated documents and specifically disclaims any implied warranties of
merchantability or fitness for any particular purpose.  Glenn C. Steiner
further reserves  the right to make changes to the specifications of the
program and documentation without obligation to notify any person or
organization of such changes.

Please -- This program required many days of work.  The copy charge is only
$15.  If you violate the copyright then please mail a check to the above
address to make us both feel better.
}

{PROGRAM TO HANDLE THE REMOTE LOGGING ON AND OFF OF A P.C. VIA A HAYES MODEM}
  {SYSTEM IS DESIGNED TO ANSWER A CALL WHERE THE CALLING SYSTEM CALLS ONCE}
  {HANGS UP AND THEN CALLS AGAIN WITHIN 30 SECONDS}

  {IF PROGRAM IS RUNNING THEN BY TYPING A SPACE FROM THE CONSOLE}
  {THE PROGRAM WILL TERMINATE}

PROGRAM GOODBYE;

  CONST
    PASSWORD='PASSWORD';          {PASSWORD REQUIRED}
    MOD_ESC_VAL=128;              {ASCII VALUE OF MODEM ESC CHAR (>128 = NO ESC}
    MODEM_DATA=$04;               {MODEM DATA PORT--UART}
    MODEM_STATUS=$05;             {MODEM STATUS PORT--UART}
```

```
    DSR_MASK=$80;                {DSR (CARRIER DETECT) MASK BIT}
    DTR_MASK=$02;                {DTR (MODEM RESET) MASK BIT}
    DTR_LOW=$25;                 {DTR LOW}
    DTR_HI=$27;                  {DTR HIGH}
    STATUS_PORT=$06;             {HORIZON STATUS PORT}
    RING_MASK=$10;               {MASK FOR RING DETECT BIT READ AT PARALLEL PORT}
    SPEED_MASK=$08;              {MASK FOR SPEED FLAG READ AT PARALLEL PORT}
    TTY=0;                       {CONSOLE IS TTY}
    USE=3;                       {CONSOLE IS USER}
    CONSOLE_MASK=$FC;            {CONSOLE MASK}
    ARM_CLOCK=$C0;               {VALUE TO ARM THE CLOCK INTERRUPT}
    DISARM_CLOCK=$40;            {VALUE TO DISARM THE CLOCK INTERRUPT}
    INTERRUPT_ADDRESS=$08;       {ADDRESS OF INTERRUP VECTOR}
    INT_PROGRAM_ADDRESS=$FC00;   {ADDRESS OF INTERRUPT PROGRAM TO BE LOADED}

  TYPE
    STRING_LINE= STRING[100];    {STRING TYPE USED FOR INPUT AND BUFFERS}

  VAR
    TIME:INTEGER;                {SECOND COUNTER}
    ABORT:BOOLEAN;               {FLAG INDICATING USER WISHES TO ABORT PROGRAM}


{FUNCTION TO RETURN TRUE IF HIGH SPPED CARRIER IS DETECTED}

FUNCTION HI_SPEED:BOOLEAN;

  BEGIN
    HI_SPEED:=NOT(PORT[STATUS_PORT] AND SPEED_MASK<>0);
    END;


{PROCEDURE TO SET LOW BAUD OR HIGH BAUD DEPENDING ON CARRIER SPEED DETECTED}

PROCEDURE BAUD_SET(HI_SPEED:BOOLEAN);
  BEGIN
    PORT[MODEM_STATUS]:=$A2;       {RESET UART FIRST WITH A DUMMY BYTE}
    PORT[MODEM_STATUS]:=$72;       {THEN RESET THE UART}
    IF HI_SPEED
      THEN PORT[MODEM_STATUS]:=$CE {IF HIGH SPEED THEN SET 16X CLOCK}
      ELSE PORT[MODEM_STATUS]:=$CF;{IF LOW  SPEED THEN SET 64X CLOCK}
    PORT[MODEM_STATUS]:=$27;       {RTS,REN,DTR,TEN}
    END;


{PROCEDURE TO STORE A HEX STRING AT A SPECIFIED ADDRESS}

PROCEDURE STORE_PROGRAM(ADDRESS:INTEGER;HEX_STRING:STRING_LINE);

  VAR
    HEX_VALUE:          STRING[3];{HEX STRING EXTRACTED FROM THE STRING}
    INT_VALUE:          INTEGER;  {INTEGER VALUE OF THE HEX VALUE}
    RESULT:             INTEGER;  {RESULT OF THE STRING CONVERSION}

  BEGIN
```



```
    WHILE LENGTH(HEX_STRING)>=2
    DO BEGIN
      {EXTRACT THE HEX CHARACTER PAIR FROM THE STRING}
      HEX_VALUE:='$'+COPY(HEX_STRING,1,2);
      {REMOVE THE FIRST TWO CHARACTERS FROM THE STRING}
      DELETE(HEX_STRING,1,2);
      {CONVERT THE HEX STRING TO AN INTEGER}
      VAL(HEX_VALUE,INT_VALUE,RESULT);
      {STORE THE HEX VALUE AND INCREMENT THE ADDRESS POINTER}
      MEM[ADDRESS]:=INT_VALUE;
      ADDRESS:=ADDRESS+1;
      END;
    END;{STORE_PROGRAM}

{PROCEDURE TO LOAD THE CLOCK INTERRUPT VECTOR AND PROGRAM}

PROCEDURE LOAD_INT_PROGRAM;
{ASSEMBLY LISTING OF THE INTERRUPT PROGRAM
;
;       CLOCK INTERRUPT ROUTINE TO CONFIRM THAT CARRIER
;          IS PRESENT WHEN REMOTE MODEM IS OPERATING
;
CLKRES  EQU     050H    ;RESET CLOCK FLAG CONTROL VALUE
CLKDIS  EQU     040H    ;DISABLE CLOCK INTERRUPT VALUE
OUTCTL  EQU     006H    ;HORIZON CONTROL PORT
UARTCTL EQU     005H    ;HORIZON UART CONTROL PORT #2
CARRIER EQU     080H    ;CARRIER MASK BIT (REALLY DSR)
BOOTROM EQU     0E800H  ;ADDRESS OF DISC BOOT ROM
;
INTPROG EQU     0FC00H  ;INTERRUPT VECTOR ADDRESS FOR THIS PROGRAM
;
        ORG      INTPROG
CLKINT  PUSH    AF              ;SAVE REGISTERS
        PUSH    BC
        LD      A,CLKRES        ;RESET CLOCK INTERRUPT FLAG
        OUT     (OUTCTL),A
        LD      BC,0            ;SET COUNTERS TO ZERO
CLKINT1 IN      A,(UARTCTL)     ;CHECK FOR CARRIER
        AND     CARRIER
        JR      Z,CLKINT2-$     ;IF CARRIER THEN RESTORE REGISTERS
        POP     BC              ;AND EXIT
        POP     AF
        EI                      ;AFTER ENABLING INTERRUPTS
        RET
CLKINT2 EX      (SP),HL         ;NO CARRIER -- WASTE TIME
        EX      (SP),HL         ;  FOR DELAY OF ABOUT 1 SECOND
        DEC     C               ;LOOP ON INNER COUNTER
        JR      NZ,CLKINT1-$
        DJNZ    CLKINT1-$       ;LOOP ON OUTER COUNTER
        LD      A,CLKDIS        ;TIME IS UP -- DISABLE CLOCK INTERRUPT
        OUT     (OUTCTL),A
        POP     BC              ;RESTORE REGISTERS
        POP     AF              ;AND REBOOT SYSTEM
        JP      BOOTROM
THEEND  END     CLKINT
```

```
}

  VAR
    INTERRUPT_PROGRAM:  STRING_LINE;
    INTERRUPT_VECTOR:   STRING[10];

  BEGIN
    INTERRUPT_PROGRAM:=
    'F5C53E50D306010000DB05E6802804C1F1FBC9E3E30D20F110EF3E40D306C1F1C300E8';
    INTERRUPT_VECTOR:='C300FC';
    STORE_PROGRAM(INT_PROGRAM_ADDRESS,INTERRUPT_PROGRAM);
    STORE_PROGRAM(INTERRUPT_ADDRESS,INTERRUPT_VECTOR);
    END;


{PROCEDURE TO LOAD THE INTERRUPT ROUTINE AND
  START THE CLOCK INTERRUPT CHECKING FOR CARRIER}

PROCEDURE CLOCK_START;

  BEGIN
    LOAD_INT_PROGRAM;                  {LOAD THE INTERRUPT PROGRAM}
    PORT[STATUS_PORT]:=ARM_CLOCK;      {ARM THE CLOCK INTERRUPT}
    INLINE ($ED/$46/$FB);              {SET INTERRUPT MODE 0 AND ENABLE INTERRUPTS}
    END;


{FUNCTION TO FETCH THE IOBYTE}

FUNCTION GET_IOBYTE:INTEGER;

  BEGIN
    GET_IOBYTE:=BDOS(7);
    END;


{PROCEDURE TO SET THE IOBYTE}

PROCEDURE SET_IOBYTE(VALUE:INTEGER);

  BEGIN
    BDOS(8,VALUE);
    END;


{PROCEDURE TO COUNT TIME IN ONE TENTH SECOND INCREMENTS}

PROCEDURE TIMER;

  BEGIN
    DELAY(100);
    TIME:=TIME+1;
    IF TIME>30000 THEN TIME:=30000;
    END;
```



```
{PROCEDURE TO RESET THE MODEM BY TAKING IT OFFLINE}

PROCEDURE MODEM_OFFLINE;

  BEGIN
    {SET DTR LOW FOR ONE SECOND}
    PORT[MODEM_STATUS]:=DTR_LOW;
    DELAY(1000);
    PORT[MODEM_STATUS]:=DTR_HI;
    WRITELN('No Carrier -- Modem is Offline');
    END;


{PROCEDURE TO CHECK FOR CARRIER--WAITS UP FOR ONE SECOND IF WAIT FLAG IS TRUE}

FUNCTION CARRIER(WAIT:BOOLEAN):BOOLEAN;

  VAR
    STATUS_IN:BOOLEAN;
    I:INTEGER;

  BEGIN
    {FETCH CARRIER STATUS}
    STATUS_IN:=(PORT[MODEM_STATUS] AND DSR_MASK)<>0;
    {IF NO CARRIER THEN TRY AGAIN AFTER .1 SECOND}
    IF (NOT STATUS_IN) AND WAIT
    THEN FOR I:=1 TO 10
    DO BEGIN
      TIMER;
      STATUS_IN:=(PORT[MODEM_STATUS] AND DSR_MASK)<>0;
      END;
    CARRIER:=STATUS_IN;
    END; {CARRIER}


{FUNCTION TO RETURN TRUE IF RING IS DETECTED}

FUNCTION RING:BOOLEAN;

  BEGIN
    RING:=(PORT[STATUS_PORT] AND RING_MASK)=0;
    END;


{FUNCTION TO COUNT THE NUMBER OF RINGS READ BY MODEM}

FUNCTION RING_COUNT:INTEGER;

  VAR
    RINGS:INTEGER;

  BEGIN
    WRITELN('Waiting for Phone to Ring -- Press Any Key on Console to Return to CPM');
```

```
    TIME:=0; RINGS:=0;
    {WAIT FOR AND COUNT NUMBER OF RINGS}
    WHILE ((RINGS=0) OR (TIME<50)) AND (NOT ABORT)
    DO BEGIN
      {SEE IF PHONE HAS RUNG}
      IF RING
      THEN BEGIN
        TIME:=0;
        {WAIT UNTIL THE RING STOPS}
        WHILE RING AND (TIME<40) DO TIMER;
        IF NOT RING
        THEN BEGIN
          RINGS:=RINGS+1;
          WRITE('Ring ');
          TIME:=0;
          END;
        END;
      {DELAY FOR A SHORT TIME AND INCREMENT TIME COUNT}
      TIMER;
      {ABORT IF KEY IS PRESSED}
      IF KEYPRESSED THEN ABORT:=TRUE;
    END;
    IF ABORT
      THEN WRITELN('Wait For Call is Aborted')
      ELSE WRITELN(' -- Rings: ',RINGS);
    RING_COUNT:=RINGS;
    END;


{PROCEDURE TO WRITE A SPECIFIED MESSAGE OUT CHARACTER BY CHARACTER}
{ -- ONLY IF CARRIER IS PRESENT}

PROCEDURE WRITEOUT(MESSAGE:STRING_LINE);

  BEGIN
    WHILE CARRIER(FALSE) AND (LENGTH(MESSAGE)>0)
    DO BEGIN
      WRITE(MESSAGE[1]);
      DELETE(MESSAGE,1,1);
      END;
    END;


{PROCEDURE TO WRITE A SPECIFIED MESSAGE OUT CHARACTER BY CHARACTER}
{ FOLLOWED BY A CR LF -- ONLY IF CARRIER IS PRESENT}

PROCEDURE WRITEOUTLN(MESSAGE:STRING_LINE);

  BEGIN
    WRITEOUT(MESSAGE);
    IF CARRIER(FALSE) THEN WRITELN;
    END;


{PROCEDURE TO READ A LINE UNTIL A CARRIAGE RETURN IS FOUND}
```



```
{LOSS OF CARRIER IS CONSTANTLY CHECKED}
{LINE MUST BE ENTERED WITHIN 30 SECONDS}

FUNCTION STRING_READ:STRING_LINE;

  VAR
    CHAR_IN:CHAR;
    STRING_IN: STRING_LINE;
    DONE:BOOLEAN;

  BEGIN
    STRING_IN:='';
    DONE:=FALSE; TIME:=0;
    WHILE (NOT DONE) AND CARRIER(FALSE) AND (TIME<300)
    DO BEGIN
      IF KEYPRESSED
        THEN BEGIN
          READ(KBD,CHAR_IN);
          IF EOLN(KBD)
            THEN DONE:=TRUE
            ELSE IF LENGTH(STRING_IN)<80
              THEN STRING_IN:=STRING_IN+(CHAR(INTEGER(CHAR_IN) AND $7F));
          END
        ELSE TIMER
      END;
    STRING_READ:=STRING_IN;
    END;  {STRING_READ}


{PROCEDURE TO HANG UP THE MODEM}

PROCEDURE HANG_UP;

  VAR
    I:INTEGER;
    DUMMY:STRING[10];
    CHAR_IN:CHAR;

  BEGIN
     {RESET THE CONSOLE TO THE TTY ONLY}
     SET_IOBYTE((GET_IOBYTE AND CONSOLE_MASK) OR TTY);
     {DISABLE THE CLOCK INTERRUPT}
     INLINE ($F3);                          {DISABLE INTERRUPTS}
     PORT[STATUS_PORT]:=DISARM_CLOCK;       {DISARM THE CLOCK INTERRUPT}
     {TAKE THE MODEM OFFLINE}
     MODEM_OFFLINE;
     {RESET SYSTEM TO HIGH BAUD RATE}
     BAUD_SET(TRUE);
     {RESET THE MODEM - DISABLE LOCAL ECHO - RESULT CODES AS DIGITS}
     {DO NOT ANSWER WHEN CALLED - SET NEW ESCAPE CODE - MODEM OFF HOOK}
     {TAKE PHONE OFF HOOK FOR A FEW SECONDS TO TEST ALL CONNECTIONS}
     WRITELN(AUX,'ATE0V0H1S0=0S2=',MOD_ESC_VAL);
     DELAY(3000);
     WRITELN(AUX,'ATH0');
     {FLUSH THE INPUT BUFFER OF ANY DATA SENT}
```

```
    DELAY(500);
    DUMMY:=CHAR(PORT[MODEM_DATA]);
    END; {HANGUP}


{PROCEDURE TO WAIT FOR 1 RING, HANGUP, THEN ANOTHER RING WITHIN 15 SECONDS}

PROCEDURE CALL_IN_WAIT;

  VAR
    CALL_OK:BOOLEAN;              {FLAG INDICATING PROPER CALL-IN SEQUENCE}
    RESULT_CODE:INTEGER;
    CHAR_IN:CHAR;

  BEGIN
    CALL_OK:=FALSE;
    WHILE (NOT CALL_OK) AND (NOT ABORT)
    DO BEGIN
      {WAIT FOR FIRST CALL BEING ONLY ONE RING IF NOT IN TEST MODE}
      REPEAT UNTIL (RING_COUNT=1) OR ABORT;
      IF NOT ABORT
      THEN BEGIN
        WRITELN('Single Ring Detected -- Modem Ready to Answer Next Call');
        {SEE IF WE GET A RING WITHIN THE NEXT 45 SECONDS}
        TIME:=0;
        WHILE(TIME<=450) AND (NOT RING) DO TIMER;
        {IF RING THEN SET MODEM FOR ANSWER}
        IF RING
        THEN BEGIN
          WRITELN('Second Call Detected -- Modem Placed Online');
          WRITELN(AUX,'ATA');
          {FLUSH THE INPUT BUFFER OF ANY RESULT}
          DELAY(500);
          CHAR_IN:=CHAR(PORT[MODEM_DATA]);
          {SEE IF WE GET CARRIER WITHIN THE NEXT 30 SECONDS}
          TIME:=0;
          WHILE(TIME<=300) AND (NOT CARRIER(TRUE)) DO;
          {IF NO CARRIER/ANSWER THEN HANG UP}
          IF CARRIER(TRUE)
          THEN BEGIN
            CALL_OK:=TRUE;
            WRITE('Carrier Detected -- ');
            {SET BAUD RATE FOR DETECTED CARRIER SPEED}
            BAUD_SET(HI_SPEED);
            IF HI_SPEED THEN WRITE('1200') ELSE WRITE('300');
            WRITELN(' Baud -- Console Assigned to Modem');
            {SET THE CONSOLE TO BOTH THE MODEM AND THE TTY}
            SET_IOBYTE((GET_IOBYTE AND CONSOLE_MASK) OR USE);
            END
          ELSE HANG_UP;
          END;
        END;
      END;
    {IF GARGAGE CHARACTER THEN FLUSH IT}
    IF KEYPRESSED THEN READ(KBD,CHAR_IN);

    END; {CALL_IN_WAIT}
```



```
{TOP LEVEL LOGGING ON / OFF PROCEDURE}

PROCEDURE BYE;

  VAR
    PASS_SENT:STRING_LINE;        {LINE READ FROM THE USER}
    I:INTEGER;
    BOOT_FILE: FILE;

  BEGIN
    PASS_SENT:='';
    WHILE (PASS_SENT<>PASSWORD) AND NOT ABORT
    DO BEGIN
      WRITEOUTLN(''); WRITEOUTLN('');
      WRITEOUTLN('Thanks for Calling Today');
      WRITEOUTLN(''); WRITEOUTLN('');
      {TURN OFF CARRIER AND HANGUP}
      HANG_UP;
      {SYSTEM WILL ANSWER WHEN THERE IS ONE RING, USER HANGS UP}
      {AND THEN CALLS BACK WITH CARRIER WITHIN 30 SECONDS}
      CALL_IN_WAIT;
      IF NOT ABORT
      THEN BEGIN
        {CLEAR THE SCREEN}
        FOR I:=1 TO 10 DO WRITEOUTLN('');
        WRITEOUTLN('          Welcome to the REMOTE CPM Computer System');
        WRITEOUTLN(''); WRITEOUTLN('');
        WRITEOUTLN('                    SYSOP is Your Name');
        FOR I:=1 TO 7 DO WRITEOUTLN('');
        END;
      I:=1; PASS_SENT:='';
      {CHECK FOR CORRECT PASSWORD -- UP TO 3 TIMES}
      WHILE (I<=3) AND CARRIER(TRUE) AND (PASSWORD<>PASS_SENT) AND NOT ABORT
      DO BEGIN
        WRITEOUT('Enter Password: ');
        PASS_SENT:=STRING_READ;
        IF (PASS_SENT<>PASSWORD)
          THEN WRITEOUTLN('--Invalid')
          ELSE WRITEOUTLN('--OK');
        I:=I+1;
        END;
      END;
    IF NOT ABORT
    THEN BEGIN
      {VALID PASSWORD READ -- SIGN ON TO CPM}
      WRITEOUTLN(''); WRITEOUTLN('');
      WRITEOUTLN('Your Wish is My Command');
      WRITEOUTLN('');
      WRITEOUTLN('Remember to Type A:BYE When You Are Done');
      {START INTERRUPT CLOCK CHECKING FOR CARRIER}
      CLOCK_START;
      END;
    END; {BYE}


{THE MAIN PROGRAM}

  BEGIN  {MAIN PROGRAM}
    ABORT:=FALSE;                 {SET ABORT FLAG TO FALSE}
    BYE;                          {WAIT FOR A CALL}
    IF ABORT THEN HANG_UP;        {ABORT BY HANGING UP IF ABORT FLAG IS SET}
    END. {PROGRAM -- EXIT TO CPM DUE TO SIGNON OR ABORT}
```

Insua

International NorthStar Users Association
Publishers of The Compass Newsletter
PO Box 2910 • Fairfield, CA 94533

Bulk Rate
U.S. Postage
PAID
Walnut Creek
Permit No. 203

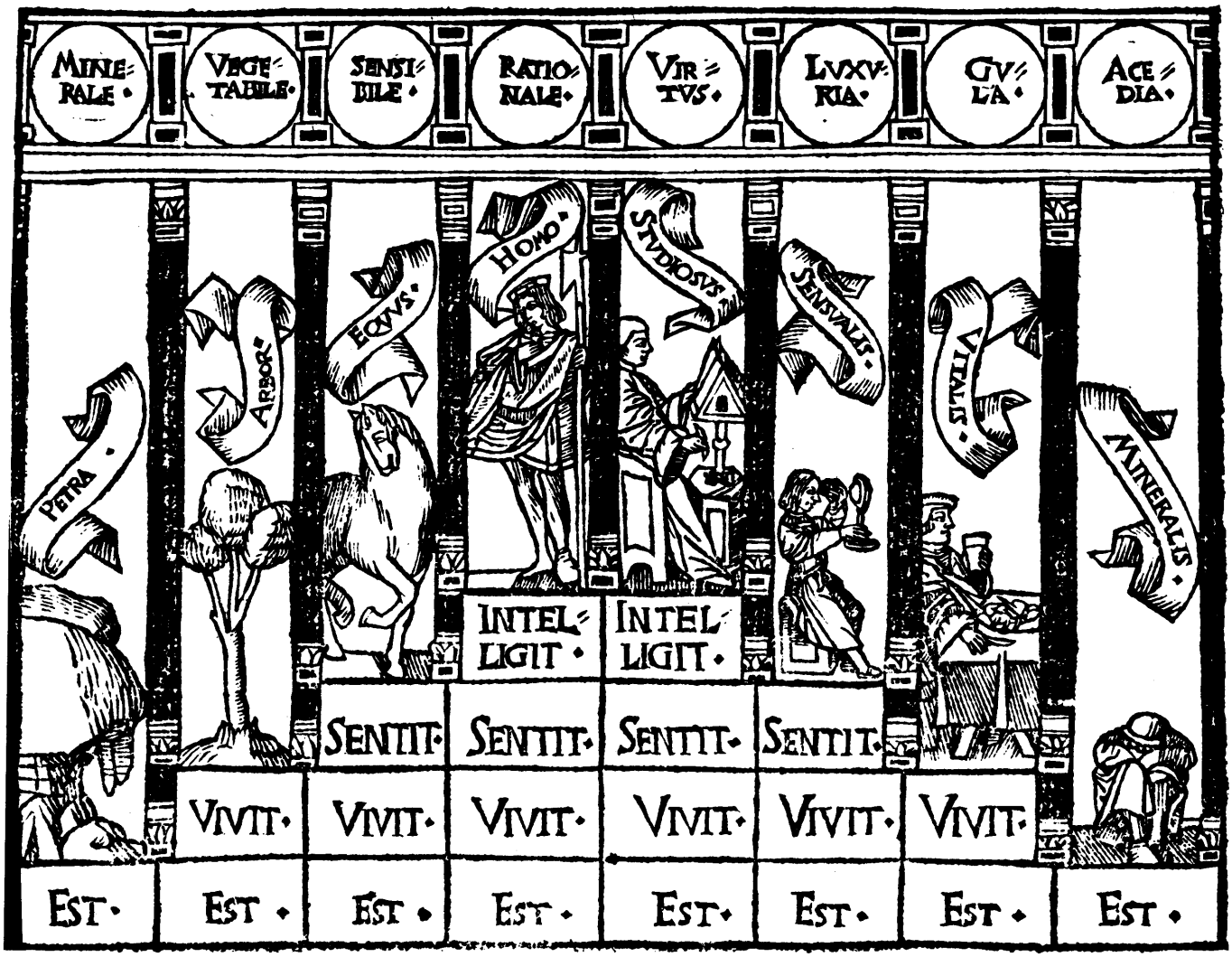
TATE 2761
JAMES TATE
23914 SPRING DAY LN.
SPRING, TX 77373

Postmaster: Address correction requested.
Please enter key number on form 3579.

The Compass

International NorthStar Users Association

Volume V No. 4



FREE POWER! TO CONTROL YOUR CP/M® MICRO

to all INSUA members

A Remarkable Program For CP/M Users.

Of course, CP/M is a wonderful operating system. That's why so much serious business software has been created for it.

BUT, CP/M is not easy to work with. That's why you need to take the **POWER!** trip.

POWER! is a super-power-packed, user-friendly program that lets you take immediate and complete control of CP/M. And at a cost of only $3. per command, it's the software buy of the year.

Over 55 Housekeeping Utilities.

POWER! is over 55 prompted, user-friendly CP/M utility programs all rolled into one 15k package. It takes care of all of these frustrations and more:

—BDOS errors? POWER! ends BDOS errors and gives you a way out.

—Accidentally erased a file? If you accidentally erase a program or disk file, **POWER!** restores the erased files.

—Can't remember file names? POWER! assigns a number to each file on your disk. So, to copy files from disk to disk, you don't have to fiddle with PIP anymore. You just pick the file from a numbered menu and **POWER!** copies it for you. No more typing errors! **POWER!** also marks original files and their copies for you; and you can compare files to find identical copies regardless of name.

—Lose data on a glitched disk? If a glitched disk makes it impossible to call up a long word processing text, **POWER!** can fix the glitch. This means you may have to retype only a couple of sentences instead of losing 20 pages of text.

—Trouble with "bargain" disks? POWER!'s disk testing function gathers any bad sectors of the disk into a special file so that CP/M thinks those parts of the disk are already used and never attempts to write to them. The rest of the disk is then safe to use.

—CP/M scrolls too fast through text files? POWER! spools through files page by page, file by file, or line by line with instant halt by touching the space bar.

—Need to reorganize files? POWER! sorts and formats the directory in 4 different ways. And you can easily copy or move files from user area to user area. **POWER!** creates 32 user areas instead of CP/M's 16.

—Need to change memory? POWER! searches, displays and lets you change memory wherever you want. You can even automatically run software anywhere in memory. And you can inter-mix your search with as many wild card jokers as you need to find, for instance, all occurrences of "Sam Jones" and "Sid James" just by typing "S??J??". And **POWER!** also lets you read or write to any sector or track very simply.

—Changing disks? You can forget the ubiquitous Control C to change disks. **POWER!** can do it for you automatically. And **POWER!** doesn't require a system disk in any drive, so Drive A is open for use, when **POWER!** is in control of CP/M.

—Afraid of HEX numbers? POWER! automatically converts Hex to Decimal, Binary or ASCII.

Special Password Protection, Too.

POWER! now includes a special program that lets you lock sensitive files, so that only you can access them. Without the secret PASSWORD which you can create and change at will, no prying eyes will ever know your secret file even exists. A great way to protect financial or scientific data from unauthorized eyes. Just this single program alone would be worth the price of **POWER!,** but there are over 55 more just as valuable programs in this power-packed-package.

At $169., It's A Bargain.

Space doesn't permit describing all the wonderful ways **POWER!** can put you in complete control of your CP/M micro. But see for yourself. There's a Money Back Guarantee. At the low price of $169., each powerful command costs you less than $3. A true bargain!

InfoWorld
Software Report Card
Power

	Poor	Fair	Good	Excellent
Performance	☐	☐	☐	☑
Documentation	☐	☐	☑	☐
Ease of Use	☐	☐	☐	☑
Error Handling	☐	☐	☐	☑

POWER! Is Better Than Ever!

Eventhough "InforWorld", "Microsystems" and "Interface Age" call **POWER!** great, we have improved **POWER!**—including a completely rewritten 120-page easy-to-read documentation. (Previous purchasers of **POWER!** may exchange their original disk for an updated version with the new commands and a brand new manual—for only $35.)

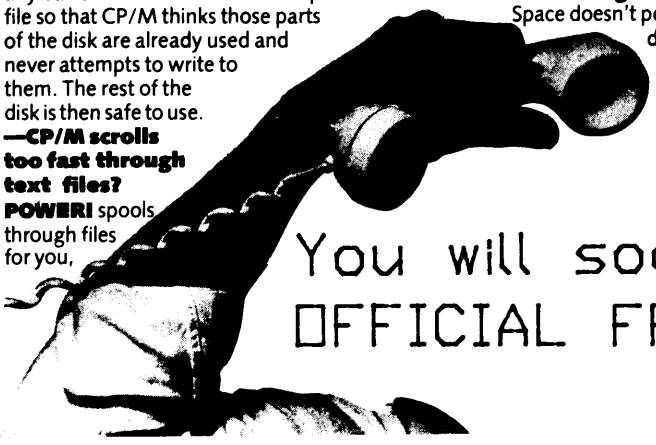
Take The POWER! Trip Today!

POWER! will operate in any standard CP/M or MP/M system, including CP/M-86, IBM PC, Apple (Z80 card), Osborne, Kaypro, HP, TeleVideo, TRS-80 conversions. S100's including NorthStar, Vector, Morrow, CompuPro, etc. Up and running immediately, no configuration necessary—for hard disks and floppies.

At only $3. per command, you can afford to **Take the POWER! Trip.** Call or send in your order today.

NOW AVAILABLE FOR MS-DOS, TOO.

ONLY $169. Money Back Guarantee.
Charge & COD Orders Welcome.
TOLL FREE (800) 428-7825 Ext. 96
IN CA: (800) 428-7824 Ext. 96
DEALERS AND OEM'S (415) 567-1634 Ext. 96AK



COMPUTING!
2519 Greenwich San Francisco, CA 94123

~~ONLY $169~~

You will soon receive your OFFICIAL FREE program and manual

POWER! is NOT public domain software. This is a special purchase for INSUA members use only. POWER! is a registered trademark.

The Compass

The Compass is published every two months by INSUA, the International North Star Users Association, P. O. Box 2910, Fairfield, CA 94533.

Subscription Information:

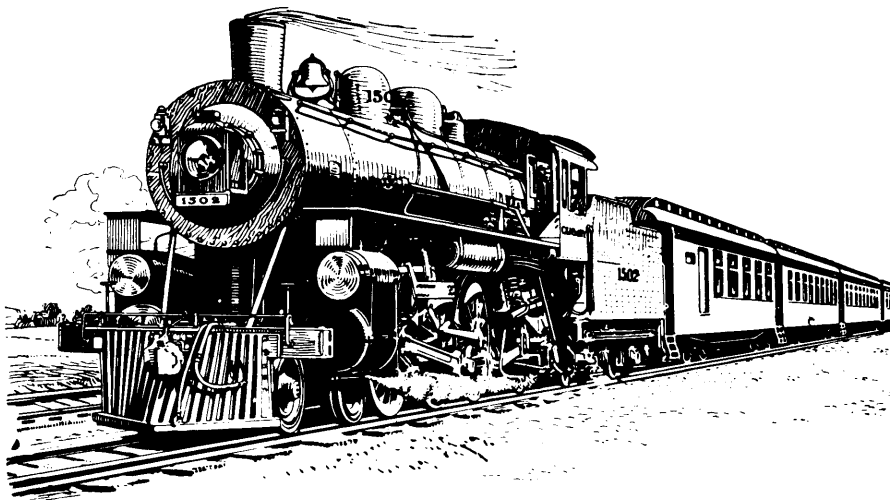
Subscription to The Compass is included in the $20.00 INSUA membership fee. Address subscription inquiries to INSUA, P. O. Box 2910, Fairfield, CA 94533.

DISCLAIMER

Programs printed in Compass and/or distributed through the INSUA disk library are offered to INSUA members in good faith. INSUA, however, is unable to guarantee the operation of any of these programs or to guarantee support. Users are advised to test the programs thoroughly for themselves in conditions under which they are to be used. Users who employ such programs in serious business or financial applications must do so at their own risk.

Facts or opinions published about manufacturers and dealers, and all opinions expressed in articles and letters, are the responsibility of the authors, and not of INSUA or the Editor of Compass. INSUA offers the right of reply to members and non-members alike.

Contents



Recovering SECRETARY Files

By Saul G. Levy
2555 E. Irvington Rd., Lot 47
Tucson, Arizona 85714
(602) 889-7753

Why? I'm Glad You Asked!

Have you ever gotten upset over losing a text file after lavishing a lot of time on it? The SECRETARY word processor has a RECOVER command, but it won't work if only the first byte of a file has been clobbered! I have written a description of this limitation (see NOTES 10-11), and much more, in my Notes to a SECRETARY (Diskette)" in Compass, Vol. IV, no. 3, p. 8.

My NOTES also describe a major error in SECRETARY (see Nasty Error 2) where it will (more often than I thought!) save one sector too many. This will clobber the first sector in any file which immediately follows the one you are saving! The just-saved file is O.K.; the following file can't be RECOVERed by SECRETARY. I now understand why this occurs (see Appendix A for the fix).

This article describes Program SECRECOV which will recover any SECRETARY text file. The final version (see listing) evolved over two short periods of time into something more useful than I originally contemplated. Most of this article is a detailed look at how to recover clobbered text files and especially, how I wrote this program. You can recover BASIC program files too (see Appendix B)!

Think First

Before I start to write a major program I think a lot about how to accomplish my desires. This way I tend to discover the major problems before I have written code which won't work. (I can still paint myself into a corner!) There are also occasions when the file structure has to be changed, or when a routine is just too slow or clumsy and has to be rewritten.

My method is to start at a low-middle level. This refers to the elitist's top-down, structured programming method which I don't find very useful. The top level is a very general look at a program's structure. The gory details, which overwhelm beginning programmers, aren't found at this level. As you proceed to the bottom level, you add details until you reach the bottom where you have a completed, running program which has all of the bugs fixed. Structured programmers give me the impression that their programs are perfect as first written! At least the honest ones won't claim quite that much!

I tend to start at a middle level with lots of things on my mind about what I'm trying to do. At this point I'm not concerned with the exact form of the code or which variable names I'll use. Later, when I start to write the program itself, I jump right into the bottom level and write the program from first line to last, debugging as I go. Finally, I continue testing the program until I'm satisfied with its operation.

I can't claim my method is structured at all. Middle-level details grow in my head until I'm ready to start writing code at the bottom level. I admit to being spoiled by an interpreted language (BASIC) which allows me to run a program 30-60 times per hour (if needed) to make debugging and exact formatting an easier chore (permit me to ignore Forth here). The elitists will gasp at this statement, but I'm not asking anyone to use my method; just describing what works for me.

SECRETARY's Text File Structure

The first thing we need to know is the line structure in a SECRETARY file. NOTE 12 gives the details which are repeated here:

Byte #	
1	Line length in hex bytes including itself and a carriage return (0DH)
2-5	Line number in four ASCII hex bytes (0000-9999)
6-...	Text in ASCII hex bytes (132 maximum)
Last	Hex carriage return (0DH)

Note that the line-length byte can be used to find the last byte (the carriage return). The last line will be followed by a 01H End-Of-File (EOF) mark instead of another line-length byte. This format started with Processor Tech and is called SCS (Self-Contained System) format. If you load a file which isn't in this exact format (BLOAD works differently), you won't be able to RECOVER all, or any, of that file. Note that BASIC will convert each hex byte into its decimal equivalent and that we have to READ and WRITE each file one byte at a time (no numbers or strings). The latter requirement doesn't drastically slow the program down because all we are doing is reading, writing, and printing the text (one line at a time) with no other processing.

Recovering Text Files

The last thing we need to know is how to recover a clobbered file. If SECRETARY wrote one sector too many, the whole first sector has been lost, but the rest of the file is O.K. We can't just ignore the first sector, so we should start at the beginning of the file and look for the matching carriage-return byte at the end of the first good "line."

If the first byte in a file starts a complete line, then the file has not been clobbered and the program should stop without writing any output. If the first line is bad, READ the next byte and see if it points to a carriage return at the end of that "line." This continues until we find a good "line." Note that a clobbered section of text **could** contain a lot of carriage returns and fool the program.

This recovery method had a problem. It kept skipping over the first good line in my test file! The method should work, but the bad data I used to clobber the file may have caused this problem. I changed the method to look for the **next** carriage return instead, and then checked the following line. This worked, but it bothered me that the original method didn't work. Later on, I rewrote the program to use the original method and got it to work properly. This shows that I had a better idea of what I was doing by then (using the same test file).

That is all we need to know to start writing a program to recover any SCS-format file!

What is a Good Line?

Note my use of "line" above. This denotes a line which **may** be bad. A clobbered sector can contain garbage, but still give good lines because the program will accept **any** line with a proper line-length byte and ending carriage return. A good line should contain a properly formatted line number and whatever data is acceptable to the program which wrote the file. SECRETARY files should contain text, while XEK or PDS' ASMB files should contain assembly source code (all are SCS-format files).

The easiest way to handle this problem is to print the first questionable "line" from a clobbered sector and let the user decide if it is good. This way the program doesn't have to be more intelligent than needed and can run faster. We could also test that the line number bytes are ASCII numbers, but this takes too much time and adds a complication which really isn't needed. SECRETARY will accept bad line numbers (you can RENUMBER them and DELETE any bad lines), but garbage within the text itself can cause your terminal to clear the screen and go berserk when running SECRETARY. SECRECOV will suppress any nonprintable characters although everything will be written to the output file.

A Few Changes

While writing and testing SECRECOV I thought it was dumb to stop the program when an unclobbered file was run. Instead I tell you if the beginning of the file is O.K. (unclobbered). This way the program will process a damaged or an undamaged file.

Another dumb idea was to prevent

recovery of files which had been clobbered in **any** sector. The change to the program to do this was trivial! A damaged sector anywhere in a file won't prevent the good lines that follow from being recovered. Why didn't I think of that sooner!

I already thought of writing an EOF mark after the last good line to properly terminate the output file. At first I wanted to write the EOF at its normal place if any error occurred while reading the input or writing the output files. This worked fine until an error occurred! The EOF mark was written into the middle of a line of text which is not a valid location as far as SECRETARY is concerned! SECRETARY ignored that EOF when I loaded the output file because the rest of the good lines were still in the file from earlier runs I'd made (I used it over and over again). I deleted the error handing so when an error occurs, the output file won't have an EOF mark in a bad location. If no errors occur, a valid EOF mark is written to the output file and you are told the program has finished.

Program Outline

It's handy to have an outline of what each section of SECRECOV does:

Line #'s	
10-160	Description of program, DIMension, title line
170-400	Get file names, check for valid names, create output file if needed, open files
410-1120	Main routine, skip clobbered lines, ask user about first good "line" found, if good; write it to output file, do next line
1130-1320	Closing routines, write EOF mark, CLOSE files, END
1330-1510	Subroutines used by main routine

Program Details

This section covers only the important or interesting details (be sure to read the REMs as we go along):

40...SECRECOV is copyrighted and may not be sold in any way or form by anyone including user's groups which charge any fee for programs or diskettes (INSUA is excepted). I do allow hobbyists to give a free copy to any other hobbyist, but be sure to let them read my comments about lines 190-230 below! All commercial distribution is prohibited without my permission.

110...If you try to enter a line longer than 138 bytes (line-length byte, four-digit line number, 132 bytes of text, and a carriage return), you will eventually clobber SECRETARY itself! For neatness, a line should end before the extra carriage return at the right end of a full line on your terminal.

140...Clear screen for ADM-3A. Change to suit your terminal. If you use a Soroc IQ-120, for example, add two spaces before the word 'PROGRAM'.

190-230...'CR' means enter just a carriage return to quit, or to use the same file for **both** input and output. If you are recovering the only file copy you have, it is dangerous to WRITE over your only copy! If you entered the code for SECRECOV yourself, you may have made a mistake or have bad memory! The program won't WRITE an invalid EOF mark and shouldn't clobber files when RUN. The above is included because I won't be responsible if you destroy valuable files! YOU HAVE BEEN WARNED!

The code in lines 210-230 was written this way so the screen format remains the same no matter what you entered in line 210. I don't like having two similar IF statements (220-230), but this code uses the shortest number of lines I can think of with only one statement per line.

490-520...The D-array will hold each good "line" from the input file. P is the pointer that keeps track of which byte in the D-array we are reading and writing (it doesn't need to be initialized, but the REM makes clear what it is for). P1 is the same type of pointer for the entire input file. It is needed to allow us to check for the ending carriage return. P2 is the maximum number of bytes in the input file. Since the first byte in a file is numbered byte 0, P2 points to the first byte **after** the physical end of the file. If P1 becomes equal to P2, we are trying to

READ too far! This occurs if the normal EOF mark is missing. These three pointers are the heart of this (or any) program. Keeping our pointers up-to-date at all times and using them correctly are very important in making any program work properly.

F is a flag I had to add to the program to allow simplifying of the original code which was a mess. I had sections of the code you see repeated in two, or even three, places. This is due to the method I used to write this program (jump right in and do it). When it came time to write this article I really couldn't excuse my sloppy style. I knew I could write this code only once, but that meant I had to make the program more complicated and use a flag to keep track of which "mode" I was in. The mode flag F is set to -1 for the initial mode (no good lines have been found as yet), or to 0 when a questionable "line" is found (it may be good or bad; if the format is O.K., let the user decide), or to 1 for a good line.

550...Point to the first byte in the input file (byte 0). This code will be used to increment P1 **before** every READ we make. Note that it is **not** used before READing a carriage return byte!

580...If the EOF mark is missing, we have to READ every byte in the input file to insure that all of the good lines will be written to the output file. This code tests that we aren't trying to READ beyond the physical end of the file. A greater than symbol is not needed because we READ every byte.

610...READ the first byte which should be the length of the first "line" (if it is a good line, we don't know this yet).

640...Test for an EOF mark **only** when we have found a good line. A clobbered file may contain many 01H bytes which will fool the program. We assume that the input file contains some text, otherwise, why are you running it?

670...Check that the length byte is not too small or too large (the shortest line has one byte of text plus six bytes overhead, the longest 132). If out of bounds, the beginning of the input file has been clobbered.

700-710...Whenever an error is found in a "line's" length byte or an ending carriage return is missing, this code will reset F to 0 for a bad line and go READ the next byte until an ending carriage return is found.

740-760...The length byte is reasonable, so check for an ending carriage-return byte. First we have to check that we aren't READing beyond the physical end of the file. Note that we don't just end the program like in line 580. We want to be sure to READ every byte in a file which has a missing EOF mark. A missing carriage return means the current line was clobbered no matter whether we are on the first line in the file or any other line. All of the causes of a bad line will skip over bad lines and **not** WRITE them to the output file. Note that the two pointers are basically the same; the file pointer includes the current value of P1. The -1 is used to correct for the length byte's always pointing to the **first byte** in the **next line** when we want to check the **last byte** in the **current line.** Our count is off by one because we started counting from byte 2 of the current line rather than byte 1.

820...We may have a good line because the beginning of the file is O.K. Tell the user about this **only** if the first line is O.K.

850-890...Read the rest of this "line." D(0) is where we put the length byte. The FOR...TO loop READs from the second byte to the last byte of text. The ending carriage return was read in line 750 so why READ it again? Note that line 750 also saved the carriage return in the correct place in the D-array! The same increment, test, and READ logic is used here as was used in lines 550, 580, and 610.

920...Print only the printable ASCII characters so the user can see what is going on (too many programs ignore the user!).

970...Ask the user if the first good "line" found is really a good line (we don't know as yet). After a bad line is found (F was

reset to 0), we will ask the user again for help.

1010...If asked, the user has decided whether this is a good line or not. A bad line will have the program READ the next byte until we find another line which has an ending carriage return. A group of good lines will ignore this code (F=1).

1040-1060...The line is good, WRITE it to the output file. No file pointer is needed because this file is written in sequential order. Don't WRITE an EOF mark after each byte (although it makes no real difference here). No EOF is used at the end of each line, but we will put one at the end of the last line later on.

1090...Note that we didn't READ the carriage return a second time in line 890 so we have to skip over its location in the input file. When I rewrote the method I used to READ the file, I forgot the need for this line and did a lot of head scratching before figuring out why I was being asked if **every** line was a good line!

1120...We have finished a good line, jump back and do the next one.

1190-1270...If one of the length bytes was an EOF mark (01H), we jump here and tell the user that fact, WRITE the very important EOF mark to the output file, CLOSE all files, and END the program. This is the normal end.

1300-1320...If we tried to READ beyond the physical end of the input file, we jump here instead. If no EOF mark was READ, tell the user, WRITE that very important EOF mark to the output file, CLOSE all files, and END the program. This is an abnormal end (if the end of the input file was clobbered).

1390-1420...If the beginning of the input file is O.K., tell the user, but do so **only** if the first line of the file is in SCS format (the contents could be garbage). Note that F is reset so this code can't be printed again!

1450-1500...Ask the user for help with the contents after the first "line" is read, and after any bad "lines" are found (F=0). If the user says the "line" is good, reset F to 1. If the line is bad, reset F to 0 to clear the previous value (-1 or 1). A clobbered sector may ask this question many times! You should look at each line carefully to determine if it is a good line. Any line which gets this far is in the proper format to be LOADed into SECRETARY (and presumably other programs) without an error occurring. Only you can tell if a questionable line should be written to the output file! Once a good line is found, the following "lines" are assumed to also be good lines and are written to the output file. If another clobbered sector is found, a carriage return will be missing and the bad sector will be skipped over until a good "line" is found.

Note: The short cross-reference listings follow the program listing.

Speeding Up SECRECOV

Try running SECRECOV on any SCS-format file and you will see that it runs at a reasonable rate. This doesn't mean that we wouldn't be happier to have it run faster!

I have been using Allen Ashley's COMSTAR compiler on a few of my programs (see Vol. I, No. 2, p. 1 for a review by Bob Stek). COMSTAR will increase the speed of a program by a factor of only 2-3, but this is still helpful. The BASIC version of SECRECOV takes up 13 blocks; the compiled version 50 blocks. If anyone would like a compiled version of SECRECOV, please let me know which DOS you will use with it. The compiled version can be loaded anywhere you have 50 blocks (12.5K) of free memory, but usually it is loaded just above the DOS (0000H for the Advantage Computer). Note that **only** double-density DOSes can be used! Please send me a diskette and $5.00 for postage and handling. It may take a week or two to compile your copy since I have to visit a friend to use his copy of COMSTAR!

# # #

Appendix A: How to Fix Nasty Error 2

After much effort I finally tracked down the cause of this problem. Gary Young made an editing error! He moved parts of his old code around to create the current version and left an important line out! The routine is (2D00H version):

```
Addr.   Obj. code   Instruc.    Comment

3B59    3A 42 54    LDA 5442    Load number of blocks in file to
                                be saved
3B5C    0F          RRC         Divide by 2 to get sectors for
                                DCOM call later on
----------
3B5D    3C          INR A       Increment (add 1 to) number of
                                sectors
3B5E    E6 7F       ANI 7F      AND to zero high bit
3B60    32 42 54    STA 5442    Store number of sectors in same
                                location
```

This code is only used for double-density file SAVEs (not NSAVEs). Note that no test occurs before the number of sectors is always incremented! This means that roughly half of the time the program was saving a sector too many! The missing line of code (at the dashed line above) should have been (with the rest of this routine moved down three bytes in memory):

```
Addr.   Obj. code   Instruc.    Comment

3B5D    D2 61 3B    JNC 3B61    If no carry, jump around increment
3B60    3C          INR A       If carry, add 1 to number of
                                sectors (we had an extra block
                                to save)
3B61    E6 7F       ANI 7F
3B63    32 42 54    STA 5442
```

Of course, we can't just add the missing line here because we would have to move the rest of the program down in memory by three bytes (a major task)! We need nine bytes of free memory somewhere within the program (so our changes will be loaded too). Well, Gary apparently didn't leave us any space! I noticed that his copyright notice just sits there at the beginning of SECRETARY. Since this program is now in the public domain, we will put our corrected code over the word "copyright" (I don't know if I can be sued for this)! I hereby absolve myself if anyone else gets sued for making the following changes!

Type at the DOS' prompt:

```
LF SECRTARY 2D00    Load SECRETARY at its normal load address
                    (do not run it!)
GO Mxxxx            Run any Monitor below the DOS or above
                    SECRETARY's code in memory (above 60FFH)
DA 3B50,20          Display hex and ASCII around the location
                    of the original routine.  Check that the
                    bytes match the original code!
DS 3B5D             Display and substitute a jump to our
                    corrected routine in low memory (at the
                    word "COPYRIGHT").  This code is:
                    JMP 2D0F

  3C=C3  Space bar for next one (on same line)
```

```
   E6=0F  Space bar
   7F=2D  RETURN to end

DA 3B50,20              Check your work!
DA 2D00,20              Display around word "COPYRIGHT"
DS 2D0F                 Substitute corrected code for whatever is
                        there as before.  This code is:

                        JNC 2D13    Missing line
                        INR A       Old line
                        ANI 7F      Old line
                        JMP 3B60    Finish at the last
                                      instruction of original
                                      code

D2  13  2D  3C  E6  7F  C3  60  3B  <===== This is the string


DA 2D00,20              Check your work!
OS                      Jump to DOS
SF SECRTARY 2D00        Save your changes
```

That's it! If you have an E00H version, reduce all addresses by 1F00H. I tested my changes on single- and double-density files with no other errors occurring (and Nasty Error 2 is gone)! Now, if only I can figure out what is causing Nasty Error 1!

Appendix B: Recovering BASIC Program Files

You may have noticed that SCS format is similar to the format used by BASIC for BASIC programs. The only differences are that the four-digit line number is a two-digit, assembly-style line number (low byte first) and that the key words (INPUT, PRINT, READ, etc.) are tokenized to just one byte.

This suggests that it wouldn't be very difficult to convert SECRECOV to recover BASIC program files! I recently could have used such a program, but had to recover the file manually (only part of a sector had been clobbered! I don't know why.). The major program change would be a routine to convert the tokenized key bytes back into words so we could print each line of code.

SECRECOV can be used right now on BASIC program files! Change the DIM in line 110 from 138 to 165 to handle the longer line length allowed by BASIC. Also change line 670 from 6 to 4 and from 139 to 166. This will properly test the line lengths.

It will be hard to tell if any good "lines" from a clobbered sector contain garbage because some lines won't print anything! You can't just ignore the question in line 1470 and always answer with a CR! Add the following line to print the line numbers:

```
911 IF P=2 then ! D(1)+256*D(2),
```

The output file may give a TOO LARGE OR NO PROGRAM error during a LOAD. RUN Leonard Morgenstern's excellent Program RESTTYP2 (Vol. III, no. 4, p. 37) to reset the byte in the diskette's directory which keeps track of which sector actually contains the EOF mark (only used for BASIC program files).

I haven't had enough clobbered BASIC program files to bother, but if anyone wants a complete recover program for BASIC programs, please let me know and I'll put one together for you!

# # #

```
10 REM PROGRAM SECRECOV, SECRETARY RECOVER
20 REM WRITTEN BY SAUL G. LEVY, TUCSON, ARIZONA, JULY 23-4, 1984
30 REM LAST CHANGED SEPTEMBER 4, 1984
40 REM COPYRIGHT (C) SAUL G. LEVY 1984
50 REM
60 REM UNLIKE SECRETARY'S RECOVER COMMAND, THIS PROGRAM WILL RECOVER FILES
70 REM WHICH HAVE BEEN CLOBBERED BY SECRETARY ITSELF (IT SAVES ONE SECTOR TOO
80 REM MANY AT TIMES)!  I NOW KNOW WHY!
90 REM
100 REM DIMENSION ARRAY FOR MAXIMUM LENGTH LINE FROM INPUT FILE
110 DIM D(138)
120 REM
130 REM PROGRAM NAME
140 ! CHR$(26),"PROGRAM SECRECOV, SECRETARY RECOVER FOR CLOBBERED FILES"
150 !
160 !
170 REM
180 REM GET FILE NAMES FROM USER
190 INPUT " INPUT FILE NAME (,DRIVE)  CR=QUIT: ",F1$
200 IF F1$="" THEN END
210 INPUT1 "OUTPUT FILE NAME (,DRIVE)  CR=INPUT FILE NAME: ",F2$
220 IF F2$="" THEN ! F1$ ELSE !
230 IF F2$="" THEN F2$=F1$
240 !
250 REM
260 REM DOES INPUT FILE EXIST?, T IS THE FILE TYPE
270 T=FILE(F1$)
280 IF T<>-1 THEN 330
290 ! "INPUT FILE ",F1$, " DOES NOT EXIST!"
300 GOTO 150
310 REM
320 REM OPEN INPUT FILE TO GET ITS FILE SIZE IN BLOCKS (S)
330 OPEN #1%T,F1$,S
340 REM
350 REM DOES OUTPUT FILE EXIST?, CREATE IT IF NEEDED TO SAME SIZE AND TYPE AS
360 REM INPUT FILE
370 IF FILE(F2$)=-1 THEN CREATE F2$,S,T
380 REM
390 REM OPEN OUTPUT FILE
400 OPEN #2%T,F2$
410 REM
420 REM
430 REM MAIN ROUTINE (SKIP OVER CLOBBERED LINES)
440 REM
450 REM
460 REM P IS D-ARRAY POINTER, P1 IS INPUT FILE POINTER, P2 IS NUMBER OF BYTES
470 REM IN INPUT FILE, F IS FLAG FOR MODE (-1 = NO GOOD LINES YET, 0 = BAD LINE
480 REM OR CONTENTS NOT CHECKED BY USER YET, 1 = GOOD LINE)
490 P=0
500 P1=-1
510 P2=S*256
520 F=-1
530 REM
540 REM POINT TO NEXT BYTE IN INPUT FILE
550 P1=P1+1
560 REM
570 REM ARE WE BEYOND THE END OF THE INPUT FILE?
580 IF P1=P2 THEN 1300
590 REM
600 REM READ LENGTH OF "LINE"
610 READ #1%P1,&L
620 REM
630 REM END-OF-FILE (EOF) MARK? (TEST ON GOOD LINES ONLY)
640 IF F=1 AND L=1 THEN 1190
650 REM
660 REM IS LENGTH IN BOUNDS?
670 IF L>6 AND L<139 THEN 740
680 REM
690 REM BAD LINE, READ NEXT BYTE
700 F=0
710 GOTO 550
720 REM
730 REM GOOD LENGTH, TEST FOR CARRIAGE RETURN AT END OF "LINE"
740 IF P1+L-1>=P2 THEN 700
750 READ #1%P1+L-1,&D(L-1)
760 IF D(L-1)=13 THEN 820
770 REM
780 REM BAD LINE, READ NEXT BYTE
790 GOTO 700
800 REM
810 REM BEGINNING OF FILE IS O.K. (TELL USER THE FIRST TIME ONLY)
820 IF F=-1 THEN GOSUB 1390
830 REM
840 REM READ THE REST OF THE "LINE"
850 D(0)=L
860 FOR P= 1 TO L-2
870 P1=P1+1
880 IF P1=P2 THEN EXIT 1300
890 READ #1%P1,&D(P)
900 REM
910 REM SHOW "LINE" TO USER (PRINTABLE CHARACTERS ONLY)
920 IF D(P)>31 AND D(P)<127 THEN ! CHR$(D(P)),
930 NEXT P
940 REM
950 REM HAVE THE USER DECIDE IF CONTENTS ARE BAD (FOR FIRST LINE AND AFTER
960 REM BAD LINES)
970 IF F=0 THEN GOSUB 1450
980 !
990 REM
1000 REM GOOD LINE?
1010 IF F=0 THEN 550
1020 REM
1030 REM GOOD LINE, WRITE IT TO OUTPUT FILE
1040 FOR P= 0 TO L-1
1050 WRITE #2,&D(P),NOENDMARK
1060 NEXT P
1070 REM
1080 REM SKIP OVER CARRIAGE RETURN AT END OF GOOD LINE
1090 P1=P1+1
1100 REM
1110 REM READ THE NEXT LINE
1120 GOTO 550
1130 REM
1140 REM
1150 REM CLOSING ROUTINES
1160 REM
1170 REM
1180 REM END-OF-FILE MARK FOUND IN INPUT FILE (01H), WRITE IT AND CLOSE
1190 !
1200 ! "NORMAL END-OF-FILE MARK FOUND"
1210 WRITE #2,&1,NOENDMARK
1220 CLOSE #1
1230 CLOSE #2
1240 !
1250 ! "FINISHED"
1260 !
1270 END
1280 REM
1290 REM BEYOND THE END OF INPUT FILE (NO END-OF-FILE MARK FOUND)
```

```
VARIABLE  CROSS REFERENCE FOR SECRECOV
D()       110     750     760     850     890     920     1050
F         520     640     700     820     970     1010    1410    1490
F1$       190     200     220     230     270     290     330
F2$       210     220     230     370     400
L         610     640     670     740     750     760     850     860     1040
P         490     860     890     920     930     1040    1050    1060
P1        500     550     580     610     740     750     870     880     890
          1090
P2        510     580     740     880
S         330     370     510
T         270     280     330     370     400
Z$        1480    1490
```

```
LINE NUMBER  CROSS REFERENCE FOR SECRECOV
150 -  300
330 -  280
550 -  710     1010    1120
700 -  740     790
740 -  670
820 -  760
1190 -  640
1210 -  1320
1300 -  580     880
1390 -  820
1450 -  970
```

Shake Hands with IBM?

By Daniel D. Stuhlman
6247 N. Francisco Ave.
Chicago, IL 60659

I regularly transfer files between my North Star Horizon and my MS-DOS IBM-compatible Eagle. The process took a while to figure out, but it is much easier than Warren Lambert's description in Compass, vol. V, no. 1, pp. 8-12.

First make a custom cable. Pins 2, 3, & 7 are wired straight through. Pins 5, 6, 8, 12, 21, & 22 are wired together to fool the IBM-compatible computer into thinking a modem is attached. You can use a Scooter gender changer to make the pin connection and then a 3-wire connection (2, 3, 7) between computers. Make sure baud rates match and both are using 8 bit no parity 1 stop bit.

To transfer files between North Star CPM and MS-DOS I use a modem program on each computer with XMODEM checking. To transfer North Star DOS files I use the ASCII capture feature of PC-TALK (i.e. type pg dn without any =) then type LIST#1 on the Horizon. The Horizon sends out to the second serial port. With this method there is no error checking. This does not work with QMODEM. Transfer time depends on the baud rate. Both methods require the typing of file names for each file transfered.

(Editor's note: With MODEM7 on each computer, it is usually possible to send files in batch mode, eliminating the need to type filenames on the receiving computer. MODEM7 also recognizes wildcards at the sending end.)

# # #

More on Release 2.1.1

By Peter Midnight

While reviewing my article in Compass, Vol. V, no. 3, I could not help noticing another article in the same issue, Saul Levy's documentation of Release 2.1.1 of North Star DOS and BASIC. And I couldn't help feeling compelled to respond to that article. First of all, I would like to thank Mr. Levy for presenting this information and for his kind words about the overall quality of North Star's system software. I would also like to answer some of the questions he raised regarding the new features of this release.

Size

Mr. Levy's first point is that, once again, this DOS is bigger than the one it replaces. But, as he explains later in his article, this does not make it incompatible with older programs, even though it does have to overlay the latter portion of itself to load and run them. One of the basic design principles at North Star was always to make each new release of the software completely replace all previous releases, so that both the users and the company would have only one release at a time to worry about supporting. When the DOS first started growing, it was given the ability to overlay any part of itself that extends beyond the user's I/O routines, which are always 900h to 9FFh above the base of the DOS.

For example, let's suppose you have a program for trimming your sails that you wrote to run under Release 1 DOS back in 1976. In those days, the DOS always loaded at 2000h. And your program would be written to run at 2A00h, immediately following the DOS. To run this program today you would need a copy of the latest DOS which had been relocated by MOVER to 2000h. When you enter GO SAILTRIM, everything in the DOS from 2A00h through 2EFFh is wiped out to be replaced by your program. That area contained most of the new goodies. But it did not contain any of the code which performs any of the library routines of the previous releases of the DOS and which your program might use. Therefore, your program works perfectly, just the way it always did.

When your sails are set and you have achieved maximum boat speed, your program jumps to 2028h to return control to the DOS. But before the DOS passed control to your program, it made a note of its lobotomy. Much of the DOS is no longer there. And it no longer knows how to do any of your bidding. But it knows that it does not know. All it can do is offer to reboot your computer and give you a chance to stick in a boot disk in place of that jib data disk.

The more technically curious readers will have noticed the similarity between this process and the operation of CP/M. The first 2.5 Kbytes of the DOS are analogous to the BDOS and the BIOS and must always remain in memory. The remainder of the DOS is analogous to the CCP and may or may not be wiped out by a program. The difference is that the DOS ordinarily does not require a "System disk" from which to reload itself. And when it does need to be reloaded, it gives you a chance to mount an appropriate boot disk before it will try to use it. And it will then perform a complete cold boot. Therefore, it never matters what kind of boot disk you chose to use next. It doesn't even have to be a DOS disk.

Speed

The next improvement Mr. Levy noted was a dramatic increase in the speed of the disks. Actually, you may not have noticed this. If you have "Fast stepping" drives, they have always stepped at 5 millisecond intervals. And they still do. But if you have "Slow stepping" drives, they were designed to step at 25 millisecond intervals. Originally, the 20 millisecond sector times, as detected by the disk controller, were used to generate this interval. The best they could do without being too fast for the drives was 40 milliseconds.

The new DOS uses the same timing loops it has built in for the faster drives and can step the others at 25 milliseconds. This makes the "Slow stepping" drives step almost twice as fast as they did before, and as they still do

with other operating systems. Of course, stepping speed alone has almost no effect on the overall throughput of your computer. But the new sounds that the drives make are really very exciting.

A more subtle increase in speed comes from some obscure changes in the coding of DCOM. It has always been possible to read or write an entire track in a single revolution with one call to DCOM. But now this can also be done with ten calls, a separate call for each sector. The importance of that is not easy to explain to those among you who may not already have been wishing for such an improvement. But consider this. When an entire track is written and verified, only a single sector instead of an entire revolution is lost between the write and the verify. This improvement is accomplished by verifying sectors 1 through 9 first and sector 0 last. And this makes a difference you can hear when using CD to copy disks.

Line editor

Next Mr. Levy noted that the full line editor that you have learned to use in BASIC and the MONITOR is now in the DOS, as well. What he hadn't noticed about it was the presence of a new editing character, which is not yet in the BASIC but has proven itself to be quite handy in GDOS and HDOS. The new character is the line feed, or control J. It is equivalent to a control G followed by a carriage return. And it allows you to repeat a whole command with a single keystroke. This might be useful any time you find yourself doing the same thing repeatedly to each of a large number of diskettes.

Another new editing feature is the ability to slip in a command without it replacing the editor's "Old line." Let's suppose you have just created a new machine language program. So you type GO SAVE THE WORLD. But it doesn't because you forgot to make SAVE type 1. If you begin a command line by repeating the prompt character, that line will not be saved as the next old line. So you type +TY SAVE 1 1000 to correct your mistake. The DOS sees that plus sign as a repeat of its own prompt character and skips its usual practice of saving this command line for your use in creating the next. As a result, when you get the next prompt from the DOS, your "Old line" for editing is still the GO command that you entered before the TY command. At this point you can just hit the line feed key to repeat the GO command and make yourself a hero.

REname

Now I take exception to Mr. Levy's remark that the ReName command was long overdue. I have never heard the lack of this command described as a shortcoming except by those who learned to use CP/M before they learned to use the DOS and are not accustomed to walking without crutches. The only reason I put this command into the DOS was that I got tired of hearing people say it couldn't be done. Just like people who really feel a need for dynamic file allocation, people who really feel a need to rename files have been doing so for years. When they do it by hand, rather than inside a program, they use DE and CR. All I have done is reduce this to a single command with a little error checking. I don't think this was overdue. This was just a way to fill up a little space and make the new DOS end at a page boundary.

OD

Some of what Mr. Levy had to say about the OD command is not quite correct. The range of legal I/O device numbers is still 0 through 7, not 0 through 8. And the business about the device numbers repeating is totally dependent on the individual customization of your I/O routines and does not happen with the set of routines that North Star supplies for the Horizon. In an unmodified Horizon DOS, what he says about device 3 coming out on device 0 would apply to devices 3 through 7, because they are not implemented.

It should be noted that the OD command in the DOS does not work quite the same as the MONITOR command of the same name. In the DOS, output is not redirected but duplicated on the specified device. This change is intended to make it easier to continue interacting with the

system even if your printer is not right in front of you or does not print at all until it has an entire line. Whatever you type still comes out on device 0, as well as on the specified additional output device. In fact, if the device specified to the OD command is not implemented, you can expect to see each character come out twice on device 0. Another difference is that this duplication of console output on a second output device will continue to apply to all console output from any program running under the DOS until it is turned off. By using the OD command, it is now relatively easy to make a hard copy record of any sequence of interactions with any program.

IL

Like the OD command, the IL command was first seen in the MONITOR and has since been duplicated in all the various flavors of the DOS. This command is there because some people really do have nonstandard PROM origins in their computers. If you either didn't know or just forgot about this, you might tell one of those modified machines to JP E800. It would then do so and blow up in your face. Try to get into the habit of saying IL instead when that is what you mean. Besides, no one should really have to remember any hex numbers just to run a computer.

However, DO NOT patch your DOS as Mr. Levy suggests. If you have a nonstandard PROM set, you had to use MOVER to make the new DOS work in your machine at all. When you did, it made ALL of the changes required for running with the disk controller at a nonstandard address. It did not leave anything untouched that you need to patch. If you have made any patches, they will come back to haunt you some time after you have forgotten what you did.

Control-H

Next, Mr. Levy expresses a very common misunderstanding about control H. Because this is a backspace, most people would expect it to be a good thing to use for correcting typing errors. This misconception is further supported by the fact that in many cases it appears to work correctly. However, the control H does not serve this purpose correctly, because it is not a destructive backspace. According to the ASCII standard as implemented in almost all ASCII terminals and printers, the character that preceded the control H is still there. If the control H was echoed in response to an editing key which removed that character from the input buffer, then the contents of that buffer will be displayed incorrectly on the output device, at least until some other character is entered in its place. What is really needed is a destructive backspace, which would remove the preceding character from the display. However, the control H cannot be redefined to mean a destructive backspace because most printers are simply incapable of performing this function.

This problem is compounded by the fact that any of the I/O devices might be any sort of hard copy or CRT terminals. Or they might not be terminals at all, but some entirely different sort of device that might not take kindly to any crude attempts at editing.

The solution to this problem that North Star offers in this and the other versions of DOS is based on the fact that only the human user knows where and how his typing is being displayed. If the device is capable of a destructive backspace, this can be achieved with the three character sequence: control H, blank, control H. On printers which cannot erase, the deletion of a character must be indicated by another printing character. The user indicates his choice of display action by his choice of editing keys. A control H causes the destructive backspace sequence described above, while a DELETE causes an underscore to be echoed. Either key has the same effect on the input buffer. (The only exception to this rule is in GDOS. The Advantage does not have a backspace key. Therefore, GDOS responds to a DELETE as though it had been a control H. To get the underscore response out of GDOS, you have to use the underscore key.)

HElp

The HElp command is, as Mr. Levy describes it, a minor attempt. All it does

is list the available commands. In HDOS and GHDOS, where space is no problem, this command gives other useful information, as well. In the floppy only versions, it is really just displaying the characters in the table that the DOS uses to interpret commands. The real purpose of this command in these versions is to arouse your curiosity about the commands you are not familiar with and make you look them up in the book. This is unfortunate for users of 2.1.1 in the Horizon, for which there is no book.

DO

The DO command is for Device Output. In the example that Mr. Levy described, he had let the device number default to 0 because it had turned out to be a secret that you can specify an output device number to this command, just as you do to LI. What this command does is send everything you type directly, without any interpretation or modification, to the device you specified until you press the magic key. If you don't specify the magic key, it defaults to the return key. When Mr. Levy entered DO! he started typing right back at his own terminal. From his description of its response to various control keys one might be able to figure out just what kind of terminal Mr. Levy has. The DOS ignored everything he typed until the next exclamation point. Now that you know a little more about how the DO command works, I'll give you a little more time to guess what it's for. Maybe I'll tell you a little later on.

EQUS

Meanwhile, I'd like to tell you about the EQUS file. There's a good reason why it looks so much like a CP/M file inside, especially that control Z at the end. That's a dead giveaway if I ever saw one. Back when the DOS was first written, by Dr. Grant and Dr. Greenberg, there were no disk-based operating systems on microcomputers at all. So they used a minicomputer. Later versions of the DOS were written on Horizons using the DOS, itself.

The assembler that was used for this latest version just happens to run under CP/M. And if I had gotten a chance to do it again I might have used a Dimension. The EQUS file is, indeed, a copy of a CP/M file, which was included in the assembly of the latest versions of DOS, HDOS, GDOS, and GHDOS. It defines a bunch of symbols which represent such things as the significant locations in the DOS jump table. It was included in the release to facilitate any development you might wish to do of software to run under the DOS.

INFO()

The only other new feature Mr. Levy pointed out on which I feel the need to shed some light is the INFO() function in BASIC. As Mr. Levy described, the information it gives you is the actual address of the BASIC, the DOS, and the text that describes the revision level of the BASIC. This is powerful stuff that ordinarily should not be messed with. But when the situation warrants such action, you can do some very interesting things with EXAM, FILL, and CALL.

To get at the facilities you might want to access with these tools, you have to know where they are. Originally they were in known locations, 2000h for the DOS and 2A00h for the BASIC. But soon things started moving around. Now that MOVER is available, there's no telling where these things might be found. There have been BASIC programs written that actually asked the poor, innocent user where in memory the BASIC interpreter was. How ugly! The INFO() function plugs one of the few remaining holes in an otherwise very thorough programming system.

The DOS Jump Table

All of the remaining new features of Release 2.1.1 are in the jump table of the DOS and will be of interest primarily to assembly language programmers. If you are not an assembly language programmer, please do not be offended by the sight of information you are not prepared to use. However, one of these changes will affect any one who changes the I/O routines in his DOS. That change is the structure that allows the OD command to work as described above. The jump at base+0Dh, that always used to go to the user's

output routine, now goes instead to a routine inside the DOS which performs the duplication of output, as specified by the user through the OD command. That routine calls the user's actual output routine through a new jump at base+3Eh. Right below that jump, at base+3Dh, is a byte called ADEV which contains the device number to which output is to be duplicated or zero if none. All the OD command really does is change this byte. And it may be desirable for other programs, as well, to change this byte at the user's whim.

If a user is unaware of this new arrangement and patches in a new set of I/O routines in the old way, putting the jump to his new output routine at base+0Dh, then his system will work as he expects it to work. He has done no harm other than to disable the feature he was not aware of. The jump at base+3Eh, which points to where the old output routine used to be, will never be called because there is no longer any jump to the routine that called it. But if the user is aware and wants the OD command to work with his new output routine, he will leave the jump at base+0Dh alone and instead point the one at base+3Eh to his new output routine.

Immediately following this jump, at base+41h, are the jumps to ISTAT and OSTAT that were added in Release 5.2. These still are not used by any software supplied by North Star. But they still are highly recommended and, in fact, practically indispensable to people who use modems and/or print spoolers. If either or both of these routines is not implemented in a user's new set of I/O routines, the corresponding jumps should be replaced by AF 3D C9. This works just like the "Null implementation" of the printer status routine that North Star supplies in its CP/M BIOS.

Next up, at base+47h, is the entry point to the same line input editor that you use in the command mode of the DOS. What this means is that any new machine language program can call on the same editing features that the user already knows to maintain a consistent and convenient user interface. Be aware, however, that this new feature does disappear when the tail end of the DOS is overlaid, as described earlier in this article.

The calling conventions for the line input editor routine are reproduced here as they appear in the GDOS and HDOS manuals. Notice that the calling program still has to provide its own input buffer along with a pointer to some text to be used as the "Old line." Typically, this is a second buffer to which each command line is copied if it does not begin with a repeat of the prompt character. It could, however, be the line the user has chosen to edit. Or it could be some sort of default response or template. Notice also that this old line text will not be affected by the editor.

```
THIS IS THE NORTH STAR LINE EDITOR
ON ENTRY:
   A= I/O DEVICE NUMBER
   C= LENGTH OF INPUT BUFFER
   DE= ADDR OF OLD LINE TERMINATED
       WITH CR
   HL= ADDR OF INPUT BUFFER
ON EXIT:
   HL, DE, AND B RESTORED
   C= SPACE UNUSED IN INPUT BUFFER
   A= RESULT CODE:
      0: RETURN ENTERED
      1: CONTROL C ENTERED
      2: @ OR CONTROL N ENTERED
      3: TOO MANY CHARACTERS ENTERED
   OLD LINE IS NOT CHANGED
   CRLF IS NOT ECHOED
   NEW LINE ENDS WITH A CR
```

The last two positions in the jump table represent the entry points to the File Manager and the HDCOM routines of HDOS, at base+4Ah and base+4Dh respectively. Of course, there are no such routines in this DOS. And there are no jumps at these positions in the table, either. What is there instead is more like what takes the place of a status test routine that is not implemented. These are little, tiny routines that just return error codes. The significance of having them there is that now any machine language program running under any flavor of the DOS can always safely call the hard disk File Manager, even if only to find out whether or not it is there. In previous releases, a call to either of these addresses would have branched right off the edge of the world.

The HMEM Byte

Another feature carried over from the other flavors of DOS is the HMEM byte at base+3Ch. This byte is to contain the high order byte of the address of the upper limit of memory. The DOS is supplied with a zero in this byte, which indicates that the upper limit of memory is not known. But if you put the truth in this byte, the BASIC will actually use it.

There are two reasons for putting this information in the jump table of the DOS instead of in the BASIC, where it used to be. The first is to make it available to other programs. Without this byte to guide them, other programs have often been written to charge off blindly through all of the memory above them and grope around for the actual, physical end. Such programs can be very messy around video boards. And in the Advantage, where there is no physical end of memory, they often end up wiping out the whole operating system. A program with better toilet training will respect the limit, if any, set by the HMEM byte.

The second reason is to facilitate the dynamic allocation of memory for coresident goodies, such as special device drivers, spoolers, and interfaces. Getting such things in and out is a trickier operation than I will attempt to describe here. But having an upper limit for general usage specified in a fixed location is the key to making such things possible.

DCOM changes

There is a new command in DCOM in addition to read, write, and verify. If the command byte in B is 3, DCOM will compute the size in sectors of the drive specified in C. In this case, none of the arguments in the other registers has to be within legal limits because they will not be used. The twos complement of the total number of sectors on the specified drive will be returned in HL. That may seem like a lot of hassle just to get one of two possible numbers, -350 or -700. But it really does simplify the algorithms required for creating a file or computing the space remaining on a disk. In addition, any program that uses this data whenever it needs the size of a disk will be compatible with any system that uses some other medium, like a RAM disk or a hard disk file to simulate a floppy and ends up with some size other than 350 or 700 sectors.

Last but not least, the recoding of DCOM has made some improvements in the OFTEN call, as well. Most importantly, that pesky worst case of 40 milliseconds between calls has finally been eliminated. The worst case is now 32 milliseconds. And that, my friend, is often enough to service a 300 baud serial line. That means that new modem programs under DOS will never again have to halt communications while they read or write their disk buffers unless a higher baud rate is being used. This is especially important in circumstances where communications cannot or must not be halted.

All the DOS's

As you should have noticed by now, this new DOS for the Horizon makes the four flavors of DOS, HDOS, GDOS, and GHDOS much more alike, both in their jump tables and in their user interfaces. In fact, they now share more of the same code than they ever did before. (In spite of this sharing of source code, the floppy only Horizon version is still composed entirely of 8080 code outside of the user's I/O routines and should be used to replace the Release 5.2 DOS in your Altair and your SOL.) It was hoped that this unifying effort would facilitate North Star's future support of this family of operating systems. Even if there is no more support from North Star or even from INSUA, it should now be easier for Horizon and Advantage users to continue supporting each other, both with software and with knowledge.

Oh, the DO command? That's for setting up all those magical little options in your printer or even the ones in your terminal. Things like compressed print or smooth scrolling are frequently commanded by control characters that would not be echoed by the editor and might even make it do something unintended. But with the DO command, all those things you can do with !CHR$(this) and !CHR$(that) can now be done directly from the keyboard too. Isn't that nice?

# # #

Be Precise - Change Precision

By Joseph A. Foster

Have you ever had a precision problem in a North Star file? The program has been running fine for years, and then the totals go over $999,999.99 and the files are in 8-digit precision BASIC! Now the only solution seems to be to re-boot in 10-digit prec-ision BASIC and key in all those records!.

Another solution is to read the files in 8-digit precision, change the numeric fields into alpha fields (N$ = STR$(N)), and save on a temporary file. Re-boot in 10-digit precision, read the temporary file, change the alpha fields to numeric (N = VAL(N$)), and write to a third file... Tedious and time-consuming, even if no mistakes are made in selecting the width of the temporary string.

The solution presented in this paper is based on the internal file structure of the North Star type 3 file. Strings are stored as literal characters in ASCII with a length of the string plus 2 bytes (3 bytes for strings greater than 256). Their representation is the same in all precisions of N* BASIC.

Numbers, however, are different. They are stored as Binary Coded Decimal (BCD) plus an exponent, and look like this on disk:

	8-DIGIT				
N	BCD				EXP
10	16	0	0	0	66
100	16	0	0	0	67
1000	16	0	0	0	68
10000	16	0	0	0	69

	10-DIGIT					
N	BCD					EXP
10	16	0	0	0	0	66
100	16	0	0	0	0	67
1000	16	0	0	0	0	68
10000	16	0	0	0	0	69

Each byte in BCD is capable of storing 00 to 99, so that the 8-digit precision can store a maximum of 99999999 in four bytes, plus the exponential byte.

Since each number is stored left-justified, an increase of precision from 8- to 10-digit requires the insertion of another byte just before the exponent. This makes the programming easier to implement. The existing precision is entered (line 60) as variable P1, and the new precision is entered (line 80) as variable P2. The length of the numeric variable is calculated as P1/2+1, which means that the first P1/2 bytes can be written to the new file. The last byte in both files must be the exponent, and the intervening bytes (if any) are zero-filled.

The program (see p. 31) has been generalized to convert any precision. While in theory precision could be any precision, the program is confined to 8, 10, 12, and 14-digit precisions. This is accomplished by the variable P3, which is set to the lesser of P1/2 or P2/2. The first P3 bytes are written to the new file. If P2 is less than P1 (e.g. going from 10- to 8-digit), the loop at line 250 does not execute, since K evaluates from P3+1 (5) to P2/2 (4). Line 260 writes the last (exponent) byte.

Another advantage of this method is that the precision of the BASIC resident in RAM does not affect file reading or writing. The TYP error occurs only on reading a wrong precision numeric variable. In this case, the numeric variable is read as a series of bytes. Of course, the original precision of the numeric variable must be known.

Line 210 is in the program to prevent file reading on an unknown type (which shouldn't occur normally). The program which follows is submitted as a small contribution to the public domain.

See p. 31 for the BASIC program.

# # #

ZCPR with Bells & Whistles

By Robert W. Bloom
5 McCord Drive
Newark, Delaware 19713

(This is the fourth of four articles submitted by Bob Bloom, and the third of three on ZCPR. --Ed.)

I covered ZCPR2 in the last article, now on to ZCPR3! I do not intend to repeat any of the notes, instructions, etc., that are already in Conn's manuals. What this article will have is how to adapt those particular instructions to the Horizon hardware. All of the ZCPR3-related files may be found in SIMTEL20, in the directory MICRO:<CPM.ZCPR3>. I was able to fit all of the files onto 4 'octal' drives with no problems.

ZCPR3 can be installed many different ways. I chose to install it with all possible options turned on, or what Rich Conn calls the 'maximum configuration.' The most important file in all of ZCPR3, as far as installation is concerned, is Z3BASE.LIB. As the documentation indicates, this file contains all of the system addresses in the system and is used by all of the major components of Z3CPR. What follows is the header for that file with all the addresses for the ZCPR3 system. Interestingly, all of the buffers that ZCPR3 uses fit exactly above the disk controller. If one is not going to split the BIOS, one might as well create a max system. But even with the split BIOS '64k' system there is 1.75k wasted space directly above the disk controller that can be used for ZCPR3 buffers. A 'minimum' ZCPR3 system could easily be created with that much working space and have a tpa as large as the plain vanilla system!

But I'll save that discussion for another time. (Or leave it as an exercise for the reader - hint: work with the memory map of the second article and figure where you want to put everything.)

Memory organization for the Horizon with ZCPR3 (from Z3BASE.LIB):

```
; Z3BASE - Maximum Configuration
; Offset to use in DDT:  5000h (C500h + 5000h = 1500h)
;*****************************************************************
;*                                                               *
;*  Z3BASE.LIB -- Base Addresses for ZCPR3 System                *
;*      by Richard Conn                                          *
;*                                                               *
;* Memory Map of System: (North Star 58k CP/M)                    *
;*                                                               *
;*      Address Range     Size  Function                         *
;*      -------------   ------- --------                       *
;*          0 -   FF    256 b   Standard CP/M Buffers except     *
;*         40 -   4A     11 b    for ZCPR3 External Path         *
;*         4B             1 b   Wheel Byte (unused)             *
;*        100 - C4FF      49  K TPA                             *
;*       C500 - CCFF       2  K ZCPR3 Command Processor         *
;*       CD00 - DAFF       3.5K BDOSZ                           *
;*       DB00 - E7FF       3.5K CBIOSZ with Buffers             *
;*       E800 - E8FF    256 b   Cold Boot PROM                  *
;*       E900 - EBFF     .75  K Disk Controller RAM             *
;*       EC00 - F3FF       2  K Resident Command Package        *
;*       F400 - F9FF       1.5K Redirectable I/O Driver Package *
;*       FA00 - FBFF       0.5K Flow Command Package            *
;*       FC00 - FCFF    256 b   Environment Descriptors         *
```

```
;*                                  Bytes 00H-7FH:  Z3 Parameters   *
;*                                  Bytes 80H-FFH:  Z3 Terminal Cap *
;*       FD00 - FD7F    128 b   ZCPR3 Shell Stack                    *
;*       FD80 - FDCF     80 b   ZCPR3 Message Buffers                *
;*       FDD0 - FDFF     48 b   ZCPR3 External FCB                   *
;*       FE00 - FEFF    256 b   Memory-Based Named Directory         *
;*       FF00 - FFCF    208 b   Multiple Command Line Buffer         *
;*       FFD0 - FFFF     48 b   ZCPR3 External Stack                 *
;*                                                                   *
;*********************************************************************
```

Following is a directory listing of my ZCPR3 system generation disk. I found it easier to put all of the needed files on one disk, and now that I have the TEAC drives it's much easier to do just this. Doing this is a two edged sword, however - make sure you keep a backup of the generation disk. Whenever you are doing significant things and changing the system it is very easy to trash a disk directory with a disk error. (It happened to me three times during the research for this article, so beware!)

Also, it is better to use the standard 4k block 'N' format on the system generation disk instead of the 2k block 'O' format. This is because you might find it necessary to generate a system without the special modifications for the 'O' format. All sorts of badness happens (and you don't always get error messages when things do go bad) if you try to read or write a disk in 'O' format with the standard CP/M system.

```
Files:      ZS - ZCPR3 system file        PT - patch file
            ZU - ZCPR3 utility            GU - general utility
            AS - assembler

ALIAS    .COM  4k (ZU) - creates a synonym command line
ARCPATCH.ASM  2k (PT) -   patch to zero archive bit if file is modified
ARCPATCH.HEX  2k (PT) - /
ASM      .COM  8k (AS) - standard DRI assembler
CLEANDIR.COM  2k (ZU) - sort and pack directory
CPM58    .COM 14k      - vanilla 58k CP/M system in a GENSYS file
CPMZ3    .COM 14k      - complete ZCPR3 system in a GENSYS file
CRC      .COM  6k (ZU) - generates crck's
DDT      .COM  6k (GU) - standard DRI debugger
DIFF     .COM  4k (ZU) - difference between two files, byte by byte
DIR      .COM  4k (ZU) - transient directory program
DPROG    .COM  4k (ZU) - program for remote devices (printer, crt, etc)
DU3      .COM 12k (ZU) - disk utility, rewritten for zcpr3
ERROR1   .COM  2k (ZU) -
ERROR2   .COM  4k (ZU) -
ERROR3   .COM  2k (ZU) -   > error handlers, what to do if command line
ERROR4   .COM  2k (ZU) -  /     won't parse correctly
ERRORX   .COM  2k (ZU) - /
GENSYS   .COM  2k (GU) - modified SYSGEN, will handle special disk formats
IF       .COM  4k (ZU) -   commands executed under conditions
IFSTAT   .COM  2k (ZU) - /
L80      .COM 10k (AS) - LINK-80 linker
LDR      .COM  4k (ZU) - ZCPR3 loader - buts system segments into postion
M80      .COM 20k (AS) - MACRO-80 assembler (like Z80 mnemonics!)
MAC      .COM 12k (AS) - DRI standard macro assembler
MLOAD    .COM  4k (AS) - Ron Fowler's multiple loader, hex to com
MU3      .COM  4k (ZU) - memory utility for ZCPR3
NEWCPGEN.COM 14k (GU) - patched CPMGEN
NEWFMT   .COM  4k (GU) - modified FORMAT, wil handle special disk formats
NEWFRM   .ASM 10k (PT) -   patches for special disk formats
```

```
NEWFRM  .HEX  2k (PT) - /
NSWP    .COM 12k (GU) - file maintence utility
PAGE    .COM  6k (ZU) - view files
PATH    .COM  2k (ZU) - view/change path
PROTECT .COM  4k (ZU) - view/change file attributes
PUBLIC  .ASM 10k (GU) -   sets/resets 'public' attribute
PUBLIC  .COM  2k (GU) - /
PUBPATCH.ASM 10k (PT) -
PUBPATCH.HEX  2k (PT) -  ) patches to handle public attribute correctly
PUBPATCH.HXR  2k (PT) - /
REG     .COM  2k (ZU) - display ZCPR3 registers
RELPUBLC.COM 14k (GU) - relocates pubpatch.hex for different size system
SETFILE .COM  2k (ZU) - sets up the 4 ZCPR3 'system' files
SH      .COM  4k (ZU) -
SHCTRL  .COM  2k (ZU) -  )ZCPR3 shell processing
SHDEFINE.COM  4k (ZU) -  /
SHFILE  .COM  2k (ZU) - /
SHOW    .COM  8k (ZU) - show all the information about ZCPR3 system
SHSET   .COM  2k (ZU) -  ZCPR3 shell processing
SHVAR   .COM  4k (ZU) - /
STARTUP .COM  2k      - alias for ZCPR3 system to execute on cold boot
SUB     .COM  4k (ZU) - submit
SYS     .ENV  2k (ZS) - system environment segment
SYS     .FCP  2k (ZS) - system flow control segment
SYS     .INS  2k (ZS) - installation list of files
SYS     .IOP  2k (ZS) - system i/o package segment
SYS     .NDR  2k (ZS) - system named directories segment
SYS     .RCP  2k (ZS) - system resident command package segment
SYSENV  .ASM  2k (ZS) -
SYSENV  .LIB  4k (ZS) -
SYSFCP  .ASM 18k (ZS) -
SYSFCP  .LIB  6k (ZS) -
SYSIOP  .ASM 32k (ZS) -    ) source for system segments
SYSNDR  .ASM  2k (ZS) -   /
SYSNDR  .LIB  2k (ZS) -  /
SYSRCP  .ASM 44k (ZS) -  /
SYSRCP  .LIB 12k (ZS) - /
TCMAKE  .COM  6k (ZU) - make a 'z3tcap' - terminal capabilities
UNERASE .COM  2k (ZU) - what is does
VFILER  .COM 12k (ZU) - screen oriented file maintenance
WS      .COM 18k (GU) -  text editor
WSOVLY1 .OVR 34k (GU) - /
XD      .COM  4k (ZU) - )directory utilities
XDIR    .COM  8k (ZU) - /
Z3BASE  .LIB 10k (ZS) - all addresses in system
Z3HDR   .LIB 20k (ZS) - what you want in the ZCPR
Z3INS   .COM  2k (ZU) - utility installer
Z3LOC   .COM  2k (ZU) - where is you system located?
Z3USR   .ASM 12k (PT) -  user area patches
Z3USR   .HEX  2k (PT) - /
ZCPR3   .ASM 66k (ZS) -  )ZCPR3 proper
ZCPR3   .HEX  6k (ZS) - /
ZCPR3   .INS  4k      - instructions to myself on how to install
ZEX     .COM  6k (ZU) - in-memory submit
```

There are many more files than just these to the ZCPR3 system. However, this set is all you really need to generate a starting point. And indeed, not all of these are absolutely

needed - for example, the 7 shell-related files, 2 flow control files, and 5 error handling files are not really needed. Extensive use of the MACLIB facility requires MAC to assemble the ZCPR3 system segments and the ZCPR3 itself. I also used it to create the USER area initializations. It's neat to have all your equates in one file and just to call it in. If you decide to re-assemble any of the ZCPR3 utilities (to change defaults for example) you will also have to have M80/L80 as Conn has used lots of external routines contained in the .REL files SYSLIB, VLIB, and Z3LIB

The actual steps to generate a 58k system with ZCPR3 are these:

1. Define System Addresses as in the Z3BASE.LIB excerpt above. Modify Z3BASE.LIB so that the correct addresses are contained within the file. In the case of the system segments where more than one is available (i.e. SYSRCP1.LIB, SYSRCP2.LIB, etc.) take the #1 version and rename to drop the number.

2. Assemble the archive or pubpatch patch to BDOS if you want it:

ASM ARCPATCH.xxZ
ASM PUBPATCH.xxZ

Alternately, use the RELPUBLC program to generate a PUBPATCH.HEX from PUBPATCH.HXR for the right size system.

3. Assemble Plouffe's mod's for compatible 40-trk option for quad drives and 2k blocks for octal drives. I suggest setting OCTAL true and QUAD false. Setting quad true requires one to patch in different addresses within NEWFRM as described previously in article 2.

ASM NEWFRM.xxZ

4. Finally assemble the parts of ZCPR3 (after modifying various LIB's to your choices):

MAC ⟨Z⟩ $PZ SZ

where ⟨Z⟩ is SYSENV, SYSNDR, SYSIOP, SYSRCP, SYSFCP, and ZCPR3.
Offset needed for patching of ZCPR3: address of CCP in sysgen image (1500) - address of CCP in system (c500) = 5000. (same as offset for all modules because of non-split BIOS!) Each assembly called its associated .LIB file and Z3BASE.LIB.

5. Assemble the user area overlay with ZCPR3 inits. (Take USER.ASM from the North Star CP/M disk, insert/append initiation code from Z3NEWS.003, make code to be conditionally assembled as much as possible.)

MAC Z3USR $SZ PZ

I rewrote much of the original user code so that much of the code is conditionally assembled based on equates found in the first part of the file. In this way the USER area code can be assembled as small as possible, which is important as I also initiate the environmental descriptor and the Z3TCAP on warm boot. The initiation of the environment takes almost 512 bytes in itself. Much of the the added-on code parallels that in the CBIOS.LIB file of Z3NEWS.003 and was put in the beginning of Z3USR.ASM. SYSENV.LIB and Z3BASE.LIB must be on-line because they are called in by a MACLIB statement. One has to do this (INITENV = true) if you expect to have a file autoexecuted on cold boot. Note that one equate is used from Z3BASE that is not normally there: NSUSER EQU 0E200H. This equate can be either put in Z3BASE.LIB or in with the equates at the top of Z3USR.ASM.

6. Actual assembly and patch:

```
NCPMGEN                 -- REMEMBER TO USE PATCHED VERSION!
    ETC ...                CREATE A VANILLA SYSTEM OF SIZE REQUIRED
                           SAVE VANILLA SYSTEM SOMEWHERE OTHER THAN 1!
GENSYS                  -- TO READ SYSTEM INTO MEMORY
1                       -- FROM DRIVE 1
^C                      -- EXIT WITHOUT WRITTING TO ANOTHER DISK
DDT                     -- CALL IN DDT
IARCPATCH.HEX           -- ADD THE ARCHIVE BDOS PATCH (OR PUBPATCH)
R                       -- NO OFFSET NEEDED, SYSGEN IMAGE ADDRESS
HARD CODED IN
IZ3USR.HEX              -- AND THE USER AREA
R5000                   -- SYSGEN IMAGE @3200 - SYSTEM LOCATION @e200
                           GIVES OFFSET OF 5000
INEWFRM.HEX             -- SPECIAL MODS FOR QUAD/NINE DRIVES
R5000                   -- SAME OFFSET FOR ALL PATCHES
F1500,1CFF,00           -- CLEAR THE OLD CCP OUT
IZCPR3.HEX              -- ADD IN THE ZCPR2
R5000                   -- SYSGEN IMAGE @1500 - SYSTEM LOCATION @c500
                           GIVES OFFSET OF 5000
^C                      -- EXIT DDT
GENSYS                  -- RUN IT
 CR                     -- SYSTEM IN MEMORY SO NO DISK READ
1                       -- WRITE IT BACK OUT TO DRIVE 1
 CR                     -- COLD BOOT TO BRING IN NEW
```

The USER area can be assembled to much smaller code if one sets 'INITENV' to false in Z3USR.LIB. This causes the environment area to be simply zeroed and any cold boot alias WILL NOT be executed. (I haven't tried a direct command so can't say if that would work or not.) Setting INITENV true will cause the cold boot routine to load the SYSENV.LIB file in with the USER area and block move it to its correct location.

Also, if INITENV is true, one should define 'ENDENV' in the SYSENV.LIB file at the physical end of the Z3TCAP so one need only move as many characters as have been defined:

From SYSENV.LIB:

```
; Terminal Capabilities Data
;
 etc.

 DB 0    ;TI String
 DB 0    ;TE String
endenv: db 0 ;end marker
 ds 80H-($-envorg2)  ; make exactly 80H bytes long
;
;  End of Environment Descriptor
;
 endm
```

As the full environmental descriptor (512 bytes) will most likely overflow the available 200h bytes, ENDENV is also used to see if only unimportant fill zeros have overflowed:

```
;test for too big
  IF (ENDENV GT NSUSER+0200H)
 SIZERR EQU NOVALUE ;USER area exceeds 200h bytes
  ENDIF
```

Well, without actually doing the the work for you, that should be it. I wish you luck.

# # #

Questions and Answers

By Robert W. Bloom

During the course of writting my four articles, I was able to figure out and find the answer to some questions that just didn't fit into the above discussions. They are tacked on to the end here for the lack of putting them anywhere else. Most of the answers are from Frank Wancho, Al Plehn, or Bob Plouffe.

Questions/Answers

Q: Can the old SSDD SHUGART drives use the 40-track modification, although being restricted to one side of the disk? Or are they mechanically limited to only 35 tracks?

A: There is a mechanical stop that stops the head at 35 tracks. It may or may not be movable.

Q: What is the difference between the two SSDD formats of NEWFMT.COM? One is labeled 'old'. Any advantage of one over the other?

A: The one labelled "old" is a preliminary Lifeboat CP/M 1.4 format. Use the newer one.

Q: Should or should not the 'PM' jumper be in place on the TEAC drives? I have HS, IU, and SM on, along with the appropriate DS0/1/2/3. Should any others be activated?

A: My reading of the manual seems to say that this jump is a 'auto-on' for the disk motor keyed to the status of the disk locking lever. Open=off, Closed=on. As I don't need this, I first had it off. Later I tried it on and I heard a difference as the drives did not seem the chatter as much with it on. So I'd leave it in the default condition (on).

Q: What changes are needed to install ZCPR2/ARCPATCH/NEWFRM on a hard disk CP/M vs. a floppy CP/M?

A: The same changes should work, modulo the relocation of addresses. Note the user area is smaller by 80h bytes

Q: Can TSS/C use the 96-tpi drives? - I note the FORMAT.COM program that came with it includes **ON THE MENU** an option for 'N'!

A: The N option **may** be an oversight on the part of the TSS/C author. With a great deal of gyration, it **is** possible to patch the TSS/C BIOS by disassembling the code to find the correct locations.

Q: How can I change the skew factor for a disk? The defined location for the address of the skew factor table in the DPB is '0000' - meaning no skew, and I know that ain't right!

A: It **is** right. Unfortunately the skewing is hardcoded with the following code:

```
BIOS+67BH       MOV     A,D     ;GET SECTOR #
;PERFORM SKEW, 0-0, 1- 5, 2- 1, 3- 6, 4- 2, ...
```

```
BIOS+67CH       RAR
BIOS+67DH       JNC     BIOS+682H
BIOS+680H       ADI     5
BIOS+682H       MOV     D,A     ;SKEWED SECTOR TO D
```

Write a small program to NOP this section, after verifying the locations and contents to turn skew off. Beware that once you've done this, you'll no longer be able to read **any** of your disks!

Q: Has anyone looked at 'eset' in regards to vanilla North Star CP/M? It's in simtel20. (ESET allows a change an any of the disk-based parameters but was written for an AMPRO.) It is touted as 'reads any disk if you know the disk parameters'. Can it even be done on North Star hardware? One should be able to at least read the North Star TurboDOS disks. (1 reserved track vs. 2, and zero skew vs. 5)

A: The North Star controller can read only hard-sectored disks. Changing the disk parameters doesn't buy you anything, except the remote possibility of being able to read N* TurboDOS disks, but, see above.

Q: Is there any way to 'tune' a disk drive for best performance by adjusting skew factors?

A: That's what FAST and FAST2 were all about (besides directory and track buffering). There is a timing test in that collection that is supposed to bypass the CP/M skewing to try your own. However, that is defeated by the hardcoded skew in the low-level driver.

Q: My 'old' North Star DOS 48-tpi disks still boot, but disk errs if one tries to read a program. How can one format a 96-tpi under North Star DOS?

A: A fellow in NY (George Ford? ACB.TYM@OFFICE-2?) has done this, check with him. You have to patch DOS in a manner similar to what was done for CP/M. It is **not** clear that anybody has bothered.

Q: Some of my old disks will not format to 96-tpi. (This is not altogether surprising as some are single sided.) Can I force the format and then lock out the bad sectors with FINDBAD? How? Even if a disk is 50% bad, at 96-tpi it's still more than double the capacity of a SSDD, and can be used on the 96-tpi drives. The 96-tpi drives will not write on a 48-tpi formatted disk.

A: (I'm still waiting for an answer on this one.)

Q: How can one learn more about how the Horizon is set-up and other methods of modifying the system. There seem to be no good books on hacking the BIOS.

A: "The hard way." Disassembling a BIOS reuires a heck of a lost of work and a self taught knowledge of the skeletal BIOS in the DRI manual on CP/M. It took me (Plouffe) a looong time to figure it out. You also need to know how the BDOS works and the manuals plus a disassembly is necessary. "Good luck."

* * *

Well, again I'm pushing over 20k characters without even including the source to Z3USR.ASM. But that about wraps it up. Now to get back to the real fun and cut out this 'unnecessary' documentation.

# # #

SUPER SORT: Machine Sort Called from BASIC

By Joseph A. Foster

A generalized assembly-language Shell-Metzner sort program by Albert J Marino appeared in the April 1981 issue of Microcomputing. Since I was interested in a fast sort, I decided to try out the program, and adapt it for the North Star.

The results were great! 1000 items sorted in approximately 4 seconds! Timing was accomplished by ringing the 'bell' before and after the CALL (CHR$(7)). In fact, for lists of less than 400 items (numeric or alpha), the two 'bells' could not be distinquished.

There were three steps involved in the process: (1) type in the Marino program, compile, and test, (2) print the compiled version and convert the hex code to ASCII, and (3) write the BASIC program, TESTSORT, to limit memory, generate random sort data, and test the results.

Step 1 was only necessary to produce the hex code and to define the call subroutine ORG address, which has to be absolute. I decided arbitrarily to limit memory to CFFFH (53247D) and ORG at D000H in a 56K RAM system. This is accomplished in APCBASIC in the CONFIGuration program, and by MEMSET 53247 in North Star BASIC. (However, the sort string address in RAM is not as easily found in North Star BASIC). The ORG can be changed to suit individual needs.

The conversion from hex to ASCII, step 2, was done by a rather simplified BASIC program which read data statements of the hex code strings and converted them to the ASCII equivalent. While there must be better and easier ways of conversion, this seemed to be the quickest as a 'one-shot' project.

The third step, the BASIC program, was quite straightforward. The first 26 lines sets up the number of items to sort (N9), the record length (K9) of the substrings, tests high memory, and allows for an alpha or numeric sort. The data statements on lines 150 - 260 are the ASCII representation of the Marino program hex statements (2A06D0 = 042 006 208, etc.). Lines 280 - 320 set up random substrings for the alpha sort, or random 'numbers' for the numeric sort, are are stored in I1$.

Note that numbers are stored in a Binary Coded Decimal format in North Star and must be converted to a string (I1$ = STR$(S,"6I")) in order to sort correctly. The "6I" refers to the right justification of a numerical string in APCBASIC, and is equivalent to the North Star format, %6I. The alpha strings are left-justified.

The CALL to the machine-language subroutine is set up as a function, FN S0(A,B,C), where the variables A, B, and C are passed to the function. Variable A is the number of records to sort, variable B is the record length, and variable C is the address of the first character of the sort string, I1$. The values of these variables are placed in memory at D000H (53248D), and the program is called at D010H (53264D). The bell is sounded before and after the call (lines 370 and 380) only to provide a timing reference.

Following the call, the sorted string is printed (line 400). Note that this a demo program. Normally, a sort string would be created on lines 280 - 320 by reading a key field from a file on disk and appending a position 'number' of 4 to 5 bytes depending on the expected number records on file. The combined substring is sorted and the file would be accessed by the appended address, thus:

```
FOR K = 0 TO N9-1
  REM extract the record address, A9
      from the last 4 bytes
  A9 = VAL(I1$(K*B+1-4,K*B+B))
  READ#0%A9*R9,.....
  REM where R9 is the file record
length
```

```
   ...
   ...
   ...
NEXT K
```

Finally, was it worth it? Admittedly, the effort was somewhat time-consuming, but where just a few hundred records are involved, the decrease in time from several minutes to 1 or 2 seconds is highly gratifying. Having set the routine up once, and incorporated it into an auto-boot menu driven system, there is no more to worry about. The call will be the same each time, only the variables will change - and IT'S SUPER FAST!

# # #

```
10 REM ** TESTSORT ** TEST M-L SORT SUBROUTINE, APCBAS10
20 !CHR$(27)+"*";!"M-L SORT ROUTINE TEST";!
30 N9=0400;REM N9 NUMBER OF RANDOM SUBSTRINGS (CHANGE AS DESIRED)
40 L9=0006;REM RECORD LENGTH - NUMERIC, 10-D PREC.
50 K9=0020;REM RECORD LENGTH - STRING (CHANGE AS DESIRED)
60 DIMI$(K9),I1$(N9*K9),B$(K9)
70 M=PARAM(8);!"HIGH MEMORY:",M;REM D000H
80 IFM<53249THEN90;!"HIGH MEMORY MUST BE SET AT D000H (53248)!",CHR$(7);END
90 !"DO YOU WANT AN (A)LPHA TEST OR A (N)UMERIC TEST? (A/N): ",;Z$=INCHR$(0)
100 !Z$;A9=0;IFZ$="A"THEN130;A9=1;IFZ$="N"THEN120
110 !"INVALID RESPONSE - MUST BE AN 'A' OR AN 'N'!",CHR$(7);!;GOTO90
120 !"THIS PROGRAM WILL GENERATE",N9," NUMBERS.";GOTO140
130 !"THIS PROGRAM WILL GENERATE",N9," SUBSTRINGS OF LENGTH",K9
140 FORN=53264TO53450;READX;FILL(N),X;NEXT;REM D000H + 16
150 DATA 042,006,208,229,042,004,208,229,175,042,002,208,124,031,103,125
160 DATA 031,111,034,002,208,180,194,044,208,193,209,201,235,042,000,208
170 DATA 125,147,111,124,154,103,034,004,208,033,001,000,034,006,208,034
180 DATA 008,208,045,193,197,025,011,120,177,194,069,208,034,010,208,235
190 DATA 193,225,229,197,034,012,208,034,014,208,235,025,235,193,197,026
200 DATA 150,194,111,208,035,019,011,120,177,194,095,208,195,168,208,210
210 DATA 168,208,197,070,026,119,120,018,035,019,193,011,120,177,194,114
220 DATA 208,042,002,208,124,047,087,125,047,095,042,008,208,025,210,168
230 DATA 208,035,034,008,208,042,014,208,235,042,010,208,123,149,111,122
240 DATA 156,103,034,014,208,195,093,208,042,006,208,035,034,006,208,034
250 DATA 008,208,235,042,004,208,125,147,124,154,218,024,208,042,012,208
260 DATA 209,213,025,235,042,010,208,235,195,084,208
270 REM SET UP RANDOM STRING, I1$
280 FORK=0TON9-1;IFA9=0THEN300;S=INT(10000*K9*RND(-1)+1)
290 I1$(K*L9+1,K*L9+L9)=STR$(S,"6I");!K+1," ",I1$(K*L9+1,K*L9+L9);GOTO320
300 S=INT(K9*RND(-1)+1);FORJ=1TOS;R=INT(26*RND(-1)+65);I$(J,J)=CHR$(R);NEXTJ
310 I1$(K*K9+1,K*K9+K9)=I$(1,S)+B$;!K+1," ",I1$(K*K9+1,K*K9+K9)
320 NEXTK
330 REM SET UP SORT PARAMS
340 A=N9;REM NUMBER OF RECORDS
350 IFA9=0THENB=K9ELSEB=L9;REM RECORD LENGTH STRING OR NUMERIC
360 C=[I1$(1)];REM ADDRESS OF SORT VARIABLE
370 !CHR$(7),"SORTING...."
380 S=FNSO(A,B,C);REM CALL SORT ROUTINE
390 !CHR$(7),"END OF SORT..."
400 FORK=0TON9-1;!K+1," ",I1$(K*B+1,K*B+B);NEXTK;END
410 DEFFNSO(N1,K1,J1);REM CALL M-L SORT ROUTINE
420 REM N1: NUMBER OF RECORDS TO SORT, M1=N1
430 REM K1: RECORD LENGTH
440 REM J1: SORT STRING ADDRESS
450 FILL(53248),@N1,@N1,@K1,@J1;REM PASS SORT PARAMS
460 A=CALL(53264);REM CALL M-L SORT ROUTINE
470 RETURN1;FNEND
```

```
                        ;General Shell-Metzner machine language subroutine
                        ;
                        ;adapted from a program by A.J.Marino published in
                        ;Microsystems, April, 1981, page 164
                        ;
                        ;
D000                            orig    org     0d000h  ;change as reqd
D000 =                          n1      equ     orig    ;number of records to sort
D002 =                          m1      equ     n1+2    ;range of records being sorted
D004 =                          k1      equ     m1+2    ;record length
D006 =                          j1      equ     k1+2    ;starting addr
D008 =                          i1      equ     j1+2    ;i pointer
D00A =                          mi1     equ     i1+2    ;address offset
D00C =                          dj1     equ     mi1+2   ;record d(j) address
D00E =                          di1     equ     dj1+2   ;record d(i) address
                        ;
D000                            ds      16      ;16 byte scratch area
                        ;
D010 2A06D0             div:    lhld    j1              ;get st.addr
D013 E5                         push    h               ;and save
D014 2A04D0                     lhld    k1              ;get length
D017 E5                         push    h               ;and save
D018 AF                         xra     a               ;mi = mi/2
D019 2A02D0                     lhld    m1
D01C 7C                         mov     a,h
D01D 1F                         rar
D01E 67                         mov     h,a
D01F 7D                         mov     a,l
D020 1F                         rar
D021 6F                         mov     l,a
D022 2202D0                     shld    m1              ;save new m1 = m1/2
                        ;
D025 B4                         ora     h               ;if m1=0 then done
D026 C22CD0                     jnz     ndon            ;jump if not zero to NDON
D029 C1                         pop     b               ;remove from stack
D02A D1                         pop     d               ;remove from stack
D02B C9                         ret                     ;return to calling program
                        ;
                                ;set k1=n1-m1, m1 is in hl
                        ;
D02C EB                 ndon    xchg                    ;move m1 to hl to de
D02D 2A00D0                     lhld    n1              ;get n1
D030 7D                         mov     a,l             ;compute ne k1=n1-m1
D031 93                         sub     e
D032 6F                         mov     l,a
D033 7C                         mov     a,h
D034 9A                         sbb     d
D035 67                         mov     h,a
D036 2204D0                     shld    k1              ;save k1
                        ;
D039 210100                     lxi     h,1             ;set j=i=1
D03C 2206D0                     shld    j1              ;save j1=1
D03F 2208D0                     shld    i1              ;save i1=1
```

```
                ;
                        ;calc and save addr.offset = m1+i1
D042 2D                 dcr     l
D043 C1                 pop     b
D044 C5                 push    b
D045 19         lp1     dad     d
D046 0B                 dcx     b
D047 78                 mov     a,b
D048 B1                 ora     c
D049 C245D0             jnz     lp1             ;jump if not zero to lp1
D04C 220AD0             shld    mi1             ;save new mi1
                ;
D04F EB                 xchg                    ;calc andd save d(j),d(i),d(i+m)
D050 C1                 pop     b
D051 E1                 pop     h
D052 E5                 push    h
D053 C5                 push    b
D054 220CD0     lp2     shld    dj1
D057 220ED0             shld    di1
D05A EB                 xchg
D05B 19                 dad     d
D05C EB                 xchg                    ;hl has d(i), de has d(i+m)
                ;
                        ;compare strings and switch if necessary
D05D C1         cp1     pop     b
D05E C5                 push    b
D05F 1A         lp3     ldax    d
D060 96                 sub     m
D061 C26FD0             jnz     neg
D064 23                 inx     h
D065 13                 inx     d
D066 0B                 dcx     b
D067 78                 mov     a,b
D068 B1                 ora     c
D069 C25FD0             jnz     lp3             ;jump is not zero
D06C C3A8D0             jmp     nsw             ;jump if done, don't sw
                ;
D06F D2A8D0     neg     jnc     nsw             ;if d(i) < d(i+m), don't switch
                                                ;change to jc for descending
                ;
D072 C5         sw      push    b               ;switch bytes not equal
D073 46                 mov     b,m
D074 1A                 ldax    d
D075 77                 mov     m,a
D076 78                 mov     a,b
D077 12                 stax    d
D078 23                 inx     h
D079 13                 inx     d
D07A C1                 pop     b
D07B 0B                 dcx     b
D07C 78                 mov     a,b
D07D B1                 ora     c
D07E C272D0             jnz     sw              ;repeat for all bytes
                ;
                        ;string switched, ck if i1-m1 <1
```

```
D081 2A02D0             lhld    m1
D084 7C                 mov     a,h
D085 2F                 cma
D086 57                 mov     d,a
D087 7D                 mov     a,l
D088 2F                 cma
D089 5F                 mov     e,a
D08A 2A08D0             lhld    i1
D08D 19                 dad     d               ;if i1-m1<1then jump to same
D08E D2A8D0             jnc     nsw             ;as no sw
                ;
                                ;calc new d(i), d(i+m)
D091 23                 inx     h
D092 2208D0             shld    i1
D095 2A0ED0             lhld    di1
D098 EB                 xchg
D099 2A0AD0             lhld    mi1
D09C 7B                 mov     a,e
D09D 95                 sub     l
D09E 6F                 mov     l,a
D09F 7A                 mov     a,d
D0A0 9C                 sbb     h
D0A1 67                 mov     h,a
D0A2 220ED0             shld    di1
D0A5 C35DD0             jmp     cp1             ;goto compare string
                ;
                                ;check for j > k
D0A8 2A06D0     nsw     lhld    j1
D0AB 23                 inx     h
D0AC 2206D0             shld    j1
D0AF 2208D0             shld    i1
D0B2 EB                 xchg
D0B3 2A04D0             lhld    k1
D0B6 7D                 mov     a,l
D0B7 93                 sub     e
D0B8 7C                 mov     a,h
D0B9 9A                 sbb     d
D0BA DA18D0             jc      div             ;if j>k goto div (m=m/1)
D0BD 2A0CD0             lhld    dj1             ;calc new d(j), d(i)
D0C0 D1                 pop     d
D0C1 D5                 push    d
D0C2 19                 dad     d               ;ne d(i) = old d(j+1)
D0C3 EB                 xchg
D0C4 2A0AD0             lhld    mi1             ;addr.offset
D0C7 EB                 xchg
D0C8 C354D0             jmp     lp2
```

Random Update

Joe Maguire
2321 Foxhall Drive
Anchorage, AK 99504

(Joe Maguire has submitted the following update to his "North Star BASIC Random Number Generator," Compass, vol V, no. 1, pp. 27-29. --Ed.)

My friend finally came through with some of the random number test data which I had requested. He put out a request on CompuServe and boy did he get some action! (He explained in his message that it was a test program to see how good the random number generators of various Basics were.) Now he tells me that everyone on the net is feverishly testing his Basic to see if it's any good! Fortunately, most seem to be but he did uncover one dud. It's SBasic. See results below.

The results he obtained surprised me somewhat. I expected APC Basic to give a better showing than it did. Also, the general result is that most Basic's RN generators are good. This is a much better showing than when I ran a similar series of tests about seven years ago.

Here's the results he obtained for me:

Basic Version		Chi-square statistic	Pi-test
APCBASIC	(1)	11.417	3.1327
Xitan	(2)	7.809	3.1409
S-Basic	(3)	119.233	3.1648
Cromemco	(4)	10.998	3.1482

(1) Version 3.1.15 running on an Advantage.
(2) Version 1.11 also known as CDL Business Basic.
(3) This Basic is supplied with the Kaypro computer. I understand it was written in house. It is a compiled Basic and appears to be based on an early version of C-Basic, a public domain Basic. This result is typical of the early Basics. You can see how much improvment has been made since.
(4) Version 03.65 Structured Business Basic.

# # #

Proportional Puzzle

From the Editor

One of the frustrations of printing the Compass with MagicBind (or any other similar program) is the difficulty of finding a proportional metal print wheel with all the regular ascii characters. Proportional wheels tend to have daggers, paragraphs, raised TM's, quarter or half signs, or other business-oriented symbols in place of such old regulars as square, pointed, or squiggly brackets. (Some wheels I've used reverses the ! and the ¡.) Does anyone know of a metal wheel for Diablo printers which has both proportional characters (e.g. skinny l's and fat m's) plus standard ascii characters for all punctuation?

```
10 REM ** PRECISE ** CONVERT N* PRECISION OF UNKNOWN FILE CONTENTS
20 REM WRITTEN BY JOSEPH A. FOSTER, 7/20/85 FOR PUBLIC DOMAIN
30 DIM A$(60),D$(200),H$(200),F1$(10),F2$(10)
40 C9$=CHR$(26)+CHR$(27)+"*";!C9$;REM CLEAR SCREEN
50 !TAB(20),"CONVERT NORTHSTAR FILES FROM ONE PRECISION TO ANOTHER";!;!
60 INPUT"WHAT PRECISION IS USED IN THE PRESENT FILE (8,10,12,14): ",P1
70 IFP1=8 OR P1=10 OR P1=12 OR P1=14THEN80;GOSUB280;GOTO60
80 INPUT"WHAT PRECISION IS USED IN THE NEW FILE (8,10,12,14): ",P2
90 IFP2=8 OR P2=10 OR P2=12 OR P2=14THEN110;GOSUB280;GOTO80
100 IFP1<>P2THEN110;!"PRECISIONS MUST BE DIFFERENT!",CHR$(7);GOTO60
110 IFP1<P2THENP3=P1/2ELSEP3=P2/2;REM P3 IS # BYTES IN SMALLER PRECISION
120 INPUT"ENTER OLD FILE NAME, DRIVE NUMBER: ",F1$;IFFILE(F1$)=3THEN140
130 !F1$," FILE DOES NOT EXIST - TRY AGAIN!",CHR$(7);GOTO120
140 INPUT"ENTER NEW FILE NAME, DRIVE NUMBER: ",F2$;IFFILE(F2$)=-1THEN170
150 !F2$," FILE ALREADY EXISTS - TRY AGAIN!",CHR$(7);GOTO140
160 !"CANNOT HAVE FILES WITH THE SAME NAME AND DRIVE NUMBER!",CHR$(7);GOTO120
170 IFF1$=F2$THEN160;OPEN#0,F1$,L;CREATEF2$,L*1.2,3;OPEN#1,F2$
180 IFTYP(0)=0THEN270;REM END OF FILE
190 IFTYP(0)=1THEN220;REM NEXT DATA TYPE IS ALPHA
200 IFTYP(0)=2THEN230;REM NEXT DATA TYPE IS NUMERIC
210 !"INVALID DATA TYPE IN FILE!",CHR$(7);GOTO270
220 READ#0,H$;WRITE#0,H$;GOTO180
230 FORJ=1TOP1/2+1;READ#0,&H(J);NEXTJ;REM ORGINAL 'NUMBER' AS BYTES
240 FORK=1TOP3;WRITE#1,&H(K);NEXT;REM WRITE BYTES LESS ONE
250 FORK=P3+1TOP2/2;WRITE#1,&0;NEXTK;REM INSERT ZERO BYTES FOR INCR.PREC.
260 WRITE#1,&H(P1/2+1);GOTO270;REM WRITE EXPONIENT BYTE LAST
270 END
280 !"PRECISION MUST BE EITHER 8, 10, 12, OR 14 - TRY AGAIN!",CHR$(7);RETURN
290 !A$,"? (Y/N): ";Z$=INCHR$(0);!Z$;IFZ$="Y"THENRETURN;IFZ$="N"THENRETURN
300 !"INVALID RESPONSE - TRY AGAIN!",CHR$(7);!;GOTO290
```

Listing 1.

BASIC program for changing precisions. **See p. 17 for explanation.**

ATTENTION: NORTH STAR USERS!

Round out your North Star and CP/M libraries. Expand your knowledge of the workings of your North Star.

** JOIN NSCS **

The North Star Computer Society is an information forum for Northstar computer users. Our broad spectrum of members includes accountants, doctors, engineers, teachers, businessmen, and students.

The Society publishes Polaris, a monthly newsletter, with technical articles, NorthStar gossip, advertisements, and software reviews. In addition to the newsletter members enjoy up-to-date Public Domain software, NorthStar compatible software information, an exchange of technical information, and discounted costs of advertising. The cost of membership is $18 per year in the U.S., $30; overseas.

PUBLIC DOMAIN LIBRARY OF 44 DISKETTES AVAILABLE TO MEMBERS
(partial list follows)

 # 5 Utilities and games, a few in FORTRAN
#10 CP/M utilities - SQUEEZE, UNERA, QUIKKEY, SWEEP, calculator.
#11 ZCPR1 - CP/M CCP replacement.
#13 MODEM795 - ready to run versions for Horizon and Advantage.
#14 PC-FILE - easy-to-use freeware file manager.
#15 Small C programming language.
#17 CP/M Utilities - SETBAUD, SORT, XREF, LIST, DISKCAT, FMAP.
#21 Forth 83 programming language.
#24 JRT Pascal compiler.
#26 BDS-C programs.
#28 CP/M Utilities - DESPOOL, ERAQ, POW, STATUS, IO-CAP, SE.
#38 - 43 CP/M Word-processing utilities including:
#35 TOUR Outlining freeware program; a real gem of a program.
#47 Turbo-Pascal tutorial lessons and programs; 20 lessons.
#48 MEX modem program complete with auto-log script files.

New diskettes are being added at an average rate of one a month.
Complete Catalog Available With Membership

JOIN NOW!!!

Send coupon with check or check with name and address to:

To:North Star Computer Society
P.O.Box 311
Seattle,WA 98111

Date_____

Name_____

Address_____

City_____State_____Zip_____

NORTHSTAR USERS

MAKE YOUR OWN CIRCUITS

Prototyping Board for Advantage
— make your own RAM disk, clock board, A-D converter, etc (plans not included)
— space for DB-25 or 37
— power and ground extend around board

Extender Board for Advantage
— raise any board above bus so it can be debugged easily
— has small prototyping area

Both boards:
— same size as Serial I/O board
— gold fingers, industrial quality
— only $40 each

COMMUNICATIONS PACKAGE

— works on Horizon, Advantage, S-100 Z-80, or BigBoard I
— easy to use, menu-driven INSTALL program to change terminal codes, default parameters
— dynamic changing of port, communication parameters, and file transfer protocols
— automatic dial with Hayes (or any compatible) modems
— compatible with MODEM7 and XMODEM
— includes both NorthStar DOS and CP/M versions
— supports batch mode file transfer (except under DOS)
— includes 40 page reference manual with index
— requires 80×24 or 64×16 screen with cursor addressing
— only $75, with 3 free updates (plus $5 postage/handling for each update)
— specify DOS version (if req'd) and single or double density. BigBoard version will be 8 inch SSSD.

Canadians please pay in Canadian funds. Others please pay in US funds. Visa and MasterCard accepted. No COD's, PO's, or personal cheques. Phone (613) 722-0690 for more info, or to order, or write to:

Anderson Techno-Products

947 Richmond Road, Ottawa, Ontario
Canada K2B 6R1

Insua

International NorthStar Users Association
Publishers of The Compass Newsletter
PO Box 2910 • Fairfield, CA 94533

Bulk Rate
U.S. Postage
PAID
Walnut Creek
Permit No. 203

TATE 2761
JAMES TATE
23914 SPRING DAY LN.
SPRING, TX 77373

Postmaster: Address correction requested.
Please enter key number on form 3579.

The Compass

International NorthStar Users Association

Volume V No. 5

Notes from the Editor

As INSUA approaches its final wind-down, and as the penultimate issue of Compass appears, reaction from most INSUA members has been understanding and supportive:

** Vance Holden of Santa Monica, CA: "Sorry to hear about the demise of INSUA. I've gotten more out of that publication than any I've subscribed to."

** Nora M. Taylor of Bethesda, MD: "Sorry to hear INSUA is closing down, but I think you are smart to do it in a good way as planned in Compass, Vol. V, no. 3, rather than just to peter out. I have received two issues of Microsystems and didn't know why until your announcement. Thanks."

* * *

In line with its policy of closing down INSUA in an orderly fashion, the INSUA Board of Directors makes the following announcements concerning the INSUA Disk Library and Compass:

The INSUA Disk Library consists of many programs which are in public domain and many which are not in public domain. Since not all programs are in the public domain, it is not possible for INSUA to pass its library on intact to another group. Therefore the INSUA Disk Library will cease to exist as such with the termination of INSUA.

Disk Library programs in public domain may by definition be distributed by anyone, though credit should always be given to the authors and developers. All Software not in the public domain has been distributed by INSUA by special agreement with the authors or owners. Ownership rights to programs not in the public domain belong to the author-owners, and in no case have been acquired by INSUA. Persons or organizations who wish to distribute such software must therefore obtain permission anew from the author-owners.

Software in the INSUA library but not in the public domain includes but is not necessarily limited to the following programs: North Star DOS and BASIC, Lance Rose's Assembler, Leonard Garcia's Telestar, Info-Soft's North Star BASIC under CP/M, Ted Warschawer's Microcount II, and POWER!, distributed by INSUA through special arrangement with COMPUTING! Of these programs, several, including the InfoSoft programs and POWER!, are strictly commercial programs distributed by INSUA with the understanding that copyright will be respected absolutely.

* * *

In the interest of continuing to make back issues of Compass available to North Star users, INSUA has made arrangements with Randy Fischer of Fischer Computer Systems, who has agreed to maintain and distribute back issues. For further information, contact Fischer Computer Systems, 445 Bay Street, Angwin, CA 94508 - (707) 965-2414.

All material published in Compass has been under INSUA copyright. The Board of Directors of INSUA announces its intention to permit this still-copyrighted material to be reproduced and disseminated freely by anyone as long as credit is given to the original author and to INSUA/Compass, and the publication remains substantially intact. Where articles are accompanied by announcements of restrictions, these restrictions must be honored. Regardless of copyright considerations, it is always appropriate to secure the understanding of the author.

* * *

POWER! was mailed on 21 January 1986. Please allow four to six weeks for postal delivery.

The Compass

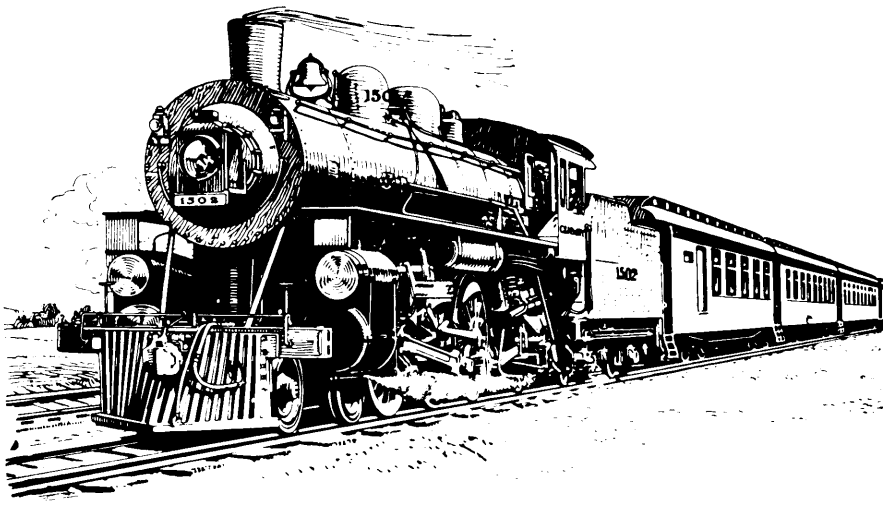
The Compass is published every two months by INSUA, the Interational North Star Users Association, P. O. Box 2910, Fairfield, CA 94533.

DISCLAIMER

Programs printed in Compass and/or distributed through the INSUA disk library are offered to INSUA members in good faith. INSUA, however, is unable to guarantee the operation of any of these programs or to guarantee support. Users are advised to test the programs thoroughly for themselves in conditions under which they are to be used. Users who employ such programs in serious business or financial applications must do so at their own risk.

Facts or opinions published about manufacturers and dealers, and all opinions expressed in articles and letters, are the responsibility of the authors, and not of INSUA or the Editor of Compass. INSUA offers the right of reply to members and non-members alike.

Contents



Repairing Software

By Joe Maguire

In a previous article, I suggested some techniques for repairing your computer hardware (see Compass, Vol. 5, no. 3, pp. 12-15.) This article will address the software problem.

That #@!$%!# program!

While hardware failures do occur, more often it's the software that gets cussed or discussed, either the result of bugs or something the program doesn't do according to your wishes. Take heart! There are tools available for fixing software and you don't even have to get your hands dirty!

If you have written the software yourself, you can no doubt fix it; but more often it has been written by someone else--and you don't have the source code. In such cases the software tools available for help include:

Source Code Disassemblers
Trace Disassemblers
Debuggers
Simulators
Emulators

The Source Code Disassembler

This is probably the most popular software tool of all for probing into the innards of an unfamiliar program. What this disassembler does is reconstruct a semblance of the original source code. There are some important differences which I will describe below but the end result is often a listing which can be used to pinpoint the part of the program which must be changed.

One of the best source disassemblers ever written is RESOURCE by Ward Christensen. It is available on INSUA disk #1014. I will use RESOURCE to illustrate how a disassembler works.

A Real Example

For this example I will use a small segment of object code for which we have the original source code. This will serve to illustrate how the output of the disassembler differs from the original.

The machine code segment I have selected is the RAM PARITY ERROR routine located in the very first portion of the I/O block of DOS 2.1.1. North Star has provided us with the source code for the I/O block in Rev. 2.1 of their software manual. This original code is given in Listing 1.

Let's imagine we don't have this information and we are going to use RESOURCE to try and reconstruct it.

```
;
;     THE FOLLOWING ROUTINE IS
;     CALLED IF A PARITY ERROR
;     OCCURS.  PRESS CR TO CONT
;     OR OTHER KEY TO REBOOT.
;
PERR        PUSH PSW
            PUSH B
            PUSH H
            LXI  H,ERTXT
;
PERR1       MOV  B,M
            INX  H
            XRA  A
            CALL COUT
            ORA  A
            JRNZ PERR1
            CALL CIN
            CPI  13
            MVI  A,40H
            OUT  0C0H
            JNZ  BADDR
            POP  H
            POP  B
            POP  PSW
            EI
            RET
;
ERTXT       DB   13,10
            DB   'RAM PARITY ERROR '
            DB   7,0
```

Listing 1

Original source code from North Star software manual Rev. 2.1.

The RESOURCE Disassembler

RESOURCE is a CP/M program, so in preparation we have saved a copy of DOS 2.1.1 on a CP/M disk as DOS211.COM.

Next we run RESOURCE and follow the directions in the DOC file (also on INSUA disk #1014) for calling in the target program.

```
0A00 PUSH PSW
0A01 PUSH B
0A02 PUSH H
0A03 LXI H,0A20H
0A06 MOV B,M
0A07 INX H
0A08 XRA A
0A09 CALL 010DH
0A0C ORA A
0A0D DB ' ' ;20H
0A0E RST 6
0A0F CALL 0110H
0A12 CPI 0DH
0A14 MVI A,'@' ;40H
0A16 OUT 0C0H
0A18 JNZ 0E800H
0A1B POP H
0A1C POP B
0A1D POP PSW
0A1E EI
0A1F RET
0A20 DCR C
0A21 LDAX B
0A22 MOV D,D
0A23 MOV B,C
0A24 MOV C,L
0A25 DB ' ' ;20H
0A26 MOV D,B
0A27 MOV B,C
0A28 MOV D,D
0A29 MOV C,C
0A2A MOV D,H
0A2B MOV E,C
0A2C DB ' ' ;20H
0A2D MOV B,L
0A2E MOV D,D
0A2F MOV D,D
0A30 MOV C,A
0A31 MOV D,D
0A32 DB ' ' ;20H
0A33 RLC
0A34 NOP
 END
```

Listing 2

Output from "pass one" of RESOURCE disassembler.

Now we do a "first pass" disassembly of the address area we are interested in. From articles in Compass, or from other sources, we think that the I/O block starts at A00 in DOS 2.1.1. If you don't have such information, you can always start at the beginning of the program.

We do a first pass to try and spot the beginning and end of the routine we want, and to determine if there are any data areas. These determinations come with a little practice. Most routines end with either a RET or a JMP, which also marks the beginning of the next routine. Data areas can often be spotted by the meaningless instructions they generate. Listing 2 is the output of the first pass.

It is a happy accident perhaps that the hex code for Ascii characters is the same as that for most of the MOV r,r instructions of the 8080 and Z80 CPUs. Therefore, if we see a bunch of MOV r,r instructions all grouped together, it must be a data area as no sensible program would be coded that way.

RESOURCE is also smart enough to figure that out and for "pass two" we invoke some of the powerful commands contained in a good disassembler.

These commands direct the disassembler to indicate the data areas and to affix "labels" to addresses. RESOURCE automatically assigns a data area if it finds eight or more MOV instructions grouped together. It attaches a label to any address referenced by another instruction. The process is not infallible and we might have to make some corrections but let's first look at the output of pass two which is Listing 3.

Notice that it spotted the RAM PARITY ERROR message correctly and also affixed a label at address 0A20 because that address was referenced by the LXI instruction at address 0A03. However, RESOURCE missed the Z80 relative jump instruction at 0A0D because it was written as an 8080 disassembler. In such cases, RESOURCE will designate such unfamiliar instructions as data bytes. That really doesn't matter because if Listing 3 were run back through an assembler, it would generate the proper code.

Now we do some editing. Other commands within the disassembler allow us to substitute names for labels, correct improper instructions (i.e., change an instruction to a data byte or vice versa),

or even insert comments.

```
0A00 PUSH PSW
0A01 PUSH B
0A02 PUSH H
0A03 LXI H,L0A20 ;0A20H
0A06 MOV B,M
0A07 INX H
0A08 XRA A
0A09 CALL L010D ;010DH
0A0C ORA A
0A0D DB ' ' ;20H
0A0E RST 6
0A0F CALL L0110 ;0110H
0A12 CPI 0DH
0A14 MVI A,'@' ;40H
0A16 OUT 0C0H
0A18 JNZ LE800 ;0E800H
0A1B POP H
0A1C POP B
0A1D POP PSW
0A1E EI
0A1F RET
L0A20:  ;0A20
0A20 DB 0DH,0AH
0A22    DB      'RAM PARITY ERROR '
0A33 RLC
0A34 NOP
 END
```

Listing 3

Output from "pass two" of RESOURCE disassembler. "A" and "B" commands used to assign labels and mark data areas.

We know from North Star's software manual that the addresses 010D and 0110 are COUT and CIN respectively, and that E800 is the address of the disk prom, so we change those labels accordingly. We make believe we don't know how North Star labeled the error message so we just call it MSG.

This latter illustrates the most important difference between the output of a disassembler and the original source code. We have no way of knowing what the original programmer called his routines so we must make up our own names. At times it may be difficult to determine just what the routine does. In that case we just leave the label attached that the disassembler assigned to it.

After we have done as much editing as we can do with the disassembler, we save the resulting assembly code in a text file on our disk. This final output is given in Listing 4.

Once the file is on the disk we can have a go at it again with a more powerful editor such as WordStar (using the **N** rather than the **D** editing option).

Using WordStar we can change the DB and RST 6 instructions into the correct Z80 JRNZ instruction. At this time we also put in any patches or make any other corrections we want.

When finished, we run the output through a Z80 assembler and we have our new object code module.

```
;RAM PARITY ERROR ROUTINE
 PUSH PSW
 PUSH B
 PUSH H
 LXI H,MSG ;0A20H
 MOV B,M
 INX H
 XRA A
 CALL COUT ;010DH
 ORA A
 DB ' ' ;20H
 RST 6
 CALL CIN ;0110H
 CPI 0DH
 MVI A,'@' ;40H
 OUT 0C0H
 JNZ DPROM ;0E800H
 POP H
 POP B
 POP PSW
 EI
 RET
MSG:  ;0A20
 DB 0DH,0AH
       DB      'RAM PARITY ERROR '
 RLC
 NOP
 END
```

Listing 4

Output from final pass of RESOURCE disassembler. Names assigned to labels and comment included. This pass is written to an .ASM file on disk.

If your target program contains a large amount of Z80 code, you can use ZESOURCE, also on INSUA disk #1014, which is a Z80 disassembler similar to RESOURCE.

The Trace Disassembler

A source disassembler disassembles the object code exactly as it finds it in memory. But often we wish to see the instructions as the CPU executes them. In other words, when the CPU comes to a jump, we want to follow it to the new address. A trace disassembler can do this. It can also follow CALLs and RETurns from calls. It is actually more of a debugging tool than a disassembler, and in fact is often contained within a debugger.

The Debugger

The debugger is probably the second most useful software tool.

Unless you have had a good deal of assembly language programming experience, you probably will have some difficulty trying to determine the purpose of the various routines disclosed by the disassembler. A debugger can follow the CPU, step by step, through the routine under examination. The various registers of the CPU can be examined and the results of conditional instructions (i.e., JZ, RNZ, etc.) can be determined.

A good debugger will contain a trace disassembler as well as routines for displaying memory, altering memory, setting breakpoints, and sometimes, also, an assembler.

Actually, there are as many ways to write a debugger as there are to write a book, and as a result there are dozens available to choose from. As good a debugger to start with as any is DDT, which is furnished with CP/M.

The Simulator

Most debuggers are not very smart. If the CPU meets an instruction which causes it to jump to non-existent memory, or if a stack error causes the stack to grow into an instruction area, the debugger will probably allow the disaster to occur. It will follow the CPU right into the twilight zone and no doubt allow itself to be destroyed in the process!

A simulator is a type of debugger which follows the CPU in a closely supervised manner. A list of parameters specified beforehand sets the memory bounds for the CPU, the amount of stack space, and a host of other restrictions. If a buggy program should cause the CPU to go wild, the simulator halts the action before a disaster can occur. The programmer can then use various commands to dump memory, examine registers, etc., in order to try and find the problem.

If the control enforced by the simulator is so superior to that of the debugger, why aren't all debuggers simulators? Speed is the answer. The simulator holds the CPU's hand each step of the way. This close supervision causes the speed of execution to slow down drastically. Some operations, such as disk access or interrupt handling, require that the CPU run at full speed. A debugger allows the CPU to run in "real time" until a breakpoint is encountered, which then stops execution.

The Emulator

An emulator allows a program targeted for a different type of computer to run on the host computer. This could be because of different operating systems or even different CPUs. Emulators are often used during the development of new hardware. The Z80 card for the Apple is one example of a popular hardware/software emulator.

Compiler Output

In the early days of software development practically all commercial software was written in assembly language. This was so because efficient, high-level language compilers were not yet available and memory was limited. The result was clean, tight object code (if the programmer was good) which could easily be disassembled to reveal how the program worked.

But alas, along came various high-level software development languages such as Pascal, "C", Forth, etc., and with them scads of available memory. Nowadays, most commercial software is written in a high-level language and run through a compiler to produce the object code.

Depending upon the efficiency of the compiler, a disassembly of the resulting object code can either be mildly confusing or a horrendous mess. The compiler has not yet been written which can code as clearly or as efficiently as a human programmer. I point this out lest you expect the disassembly of any software package to be a serendipity experience!

The North Star Serial Port

By Peter Midnight

This article has evolved out of an attempt to explain how to install a generic program in a North Star computer. If the new microwave oven you got for Christmas has an RS-232 connector on the back and comes with all the software to produce a variety of seven-course meals, there is, unfortunately, a good chance that you won't find North Star on the list of computers that it is already set up to run on. However, there may well be a CP/M version of that software included. And then there is no reason you should not be able to use it on your North Star Horizon or Advantage.

How you get that software transferred onto a North Star diskette is just a matter of ingenuity. But when the software is transferred, you will find some sort of installation procedure that you must perform to configure the software to the computer on which you plan to use it. To complete this process, you will probably have to answer a bunch of questions about your machine. You might have to edit a few files that somehow represent the information the program needs in the form of text. And you might even be called upon to write a few simple assembly language subroutines.

My goal for this article was to somehow provide you with all the information you would need in order to get through this installation procedure.

But the more I thought about the process, the more I became convinced that the only thing a program is likely to need that CP/M does not provide is adequate access to the I/O devices and that those devices are almost always connected through serial ports. In addition, it became clear to me that the installation of programs that use special peripherals and the installation of those peripherals are all part of the same problem for the person who wants to use them. Therefore, I need to describe the serial ports from more than one point of view to show how to use them to make the logical and electrical connections between a program and the devices it controls.

A History Lesson

First, I think a little historical perspective would be helpful. In days of old when knights were bold and computers were bigger than people, the normal way to talk to one was with a terminal and a modem. The serial port conventions and terminology that we use today grew out of the connection between those two devices.

The terminal is called Data Terminal Equipment, or DTE. And the modem is called Data Communication Equipment, or DCE. The cable between the two has one wire allocated for data from the DTE to the DCE and another wire allocated for data flowing the other way. Therefore, every serial cable connector actually has two orthogonal genders. The connector, itself, is physically either male or female. And the direction that data flows on each signal wire to or from the connector determines whether that connector represents DTE or DCE. A DTE connector only mates with a DCE connector and vice versa, although either of the two could be the male connector.

When microcomputers were invented they did not have built-in screens and keyboards, like the Advantage. Instead, they had serial ports, like the Horizon. These ports had DCE connectors so that they could connect to terminals. Conveniently, printers generally had DTE connectors so that they could connect to modems in place of terminals. These printers also connected easily to the new microcomputers. But modems, with their DCE connectors, were a different story.

When the Horizon was invented, all of the signal wires from its serial connectors went through configuration plugs. By changing the wiring of the plug, the user could change the connector on the rear panel from DCE to DTE and make it compatible with a modem. This method can still be used because all North Star serial ports in both the Horizon and the Advantage still have configuration plugs that serve this purpose. But this method

requires the user to painstakingly construct the new plug. And it does not allow the port to be conveniently switched back and forth between DTE and DCE unless the user wired his new plug to a multiple pole switch on the rear panel of his computer.

The "Mixed-Up" Cable

The invention that made all this easier to deal with was the mixed-up cable. This is a cable in which each of the signal wires is crossed with an analogous signal wire of the opposite direction. Such a cable allows the DCE connector on the back of a computer to be connected to the DCE connector on a modem. Once you understand the difference between DTE and DCE, and as long as none of your equipment is IBM-compatible, it should always be clear when to use a straight cable and when to use a mixed-up cable.

(If you wish to make your own cables, you will need to refer to the circuit diagram of the serial port you are using in order to determine which connector pin actually carries each circuit. To make a mixed up cable, refer to the configuration plug diagrams to determine which circuits to exchange with which.)

The Serial Port

Now let's look at a serial port from the inside, as seen by a program running in your computer. What the program sees are two I/O addresses that it can read from or write to, using the Z-80's input and output instructions. In North Star serial ports these are an even-numbered port for data and the next higher odd-numbered port for control and status.

Except for the ports on the Horizon motherboard, each port also has an address for baud rate control and an address for interrupt control. (The use of interrupts is beyond the scope of this article. So I'll just tell you to leave the interrupt control port alone.) The actual port addresses depend on the wiring of a plug or the positions of switches in the Horizon or the slot number in the Advantage. The numbers to use for each port in your computer should be determined by reading the instructions.

In the middle, between the two I/O addresses and all those wires on the configuration plug, is a magic black box called an 8251 USART. (USART means Universal Synchronous Asynchronous Receiver Transmitter.) This is a powerful little box that can communicate in more ways than we know how to use. So you have to tell it what you want before you can use it at all. But once it has been set up, its normal operation is pretty simple.

You can read from the status address whenever and as often as you like. Bit 0 of the status byte becomes a one whenever the USART is able to accept another byte to be sent. And bit 1 becomes a one whenever a new byte has been received. Reading or writing the data port automatically clears the corresponding status bit. This is why a normal input or output routine works by reading one bit of the status port continuously in a loop until it becomes a one and then either reading or writing the data port.

Instructing the USART

There are four steps to setting up a serial port. But the heart of the process is the issuance of new instructions to the USART. These instructions are encoded into two bytes which are written in sequence to the control port. Unlike the data port, the control port does not have a bit in the status byte to tell you when it is ready for you to write the next byte into it. But the Z-80 is not fast enough to write these two bytes so quickly that the USART could fail to get them both correctly.

The first of these two bytes is called the mode byte. The normal value is 4E in hex, or 1032 in base 4. The first digit (bits 6 and 7) controls the number of stop bits, or the minimum time that the transmitter will pause after each byte before sending the start bit for the next byte. The second digit (bits 4 and 5) controls the parity bit that may be transmitted with each outgoing byte and expected with each incoming byte. The third digit (bits 2 and 3) controls the number of bits of each data byte to be transmitted or received. And the fourth digit (bits 0 and 1) controls how the actual baud rate is derived from the baud rate clock signal that the USART receives

from its associated hardware.

The normal value for stop bits is 1, which means one stop bit. Other legal values are 2, for one and one half stop bits, and 3, for two stop bits. You may have noticed that North Star's software often sets up the serial ports for two stop bits. That is because the ports may be used at 110 baud and some equipment that operates at that baud rate requires a two-bit delay between bytes. In all other known cases, one stop bit is enough.

The normal value for parity is 0, which means no parity bit is sent or expected. Other legal values are 1, for odd parity, and 3, for even parity. The USART does not generate or test for zero parity or one parity. But this can be easily done in software if it is ever needed. Seven-bit data with zero parity is really the same thing as eight-bit data with a constant zero in bit 7.

The normal value for byte length is 3, for eight bit data. Other legal values are 2, for seven bit data, 1, for six bit data, and 0, for five bit data. When using only a specific code, such as ASCII or BAUDOT, one might wish to send less than eight bits.

The normal value for the baud rate factor is 2, for 16x clock. This causes the actual baud rate to be one sixteenth of the frequency applied to the USART's clock input. Other legal values are 1, for 1x clock, and 3, for 64x clock. 1x clock is not useful when the range of available frequencies is as broad as it is in North Star serial ports. However, a good use has been found for 64x clock. If a port is set to operate at 1200 baud, but 64x clock is used instead of 16x clock, then the actual baud rate will be 300 instead of 1200. On the Horizon motherboard, where the baud rates are not software controlled, this trick allows software selection between the two baud rates commonly used by currently fashionable modems.

The second of the two bytes that are written to the control port is the command byte. The normal value of this byte is 37 in hex, or 00110111 in binary. If any more bytes are written to the control port, even after normal operation has commenced, they will be interpretted by the USART as new command bytes.

Bit 7 of the command byte is not used in normal, asynchronous operation.

A one in bit 6 of the command byte causes the USART to reset itself. After a command byte is written with a one in bit 6, the next byte written to the control port will be interpretted by the USART as a new mode byte instead of as another command byte.

Signal Wires

A more complete serial cable has two signal wires in each direction, besides the actual data wires. The signals on these two wires come directly from bits 5 and 1 of the command byte at the originating end of the cable. These two signals are called RTS, DTR, CTS, and DSR. And their actual meaning has become clouded in the mists of time. However, what works is to put ones in both of these bits.

A deeper look reveals that if two North Star serial ports are connected together (using a mixed up cable because they are both DCE) then the value written to bit 5 of the command byte of each USART will appear in bit 7 of the status byte of the other and a zero written to bit 1 of the command byte of either USART will cause the other to stop transmitting until a one is written to that bit. Either of these behaviors may be useful for handshaking purposes. And I'll try to explain that a little later on.

Error Detection

Certain communication errors can be detected by the USART. These are reported to the program through bits 3, 4, and 5 of the status byte and are traditionally ignored. A one written to bit 4 of the command byte would be used to clear these bits if the software were paying attention. But since it never is, the normal value for this bit is zero.

The Break

Bit 3 of the command byte can be used to send a break. The procedure is to wait until bit 2 of the status byte becomes a one, write a one to bit 3 of the command byte, wait a long time, and then write a zero to bit 3 of the command byte. The one in bit 2 of the status byte means that the transmitter has finished sending all of the bytes it has been given to send. The one in bit 3 of the command byte then

pulls the transmit data wire away from its resting state, as though to send the start bit of another byte.

The delay should then be long enough to make it obvious to all concerned that something funny is happening. No one knows quite how long that should be. But half a second seems to work ok. The zero in bit 3 of the command byte then releases the data wire for normal use. If a break were sent into your serial port from the device to which it was connected, it would look like a bunch of nulls, possibly followed by a garbage character, and a bunch of the errors you are ignoring.

Enabling Commands

The ones in bits 0 and 2 of the command byte simply enable the transmitter and the receiver, respectively. Writing those two bytes to the control port is step three of the four-step setup process. Step one is to reset the USART, so that you can tell when it is expecting a mode byte and when it is expecting a command byte. One way to do this is to send the reset command to the Horizon motherboard. But you have to do that four times in a row because just once doesn't seem to work very well. A better way to reset the USART - a way that works on all North Star serial ports - is to write a 3, another 3, and a 64 (40h) to the control port. The two 3s ensure that the USART will expect a command byte next. (So says the manufacturer of the USART.) The 64 then commands the USART to reset itself.

Step two is to set the baud rate. This is done by writing the appropriate number to the appropriate port address. Those numbers are not the same for both the HSIO-4 and the Advantage SIO boards. So, again, you will have to read the instructions to get the actual numbers.

On the Horizon motherboard, the baud rates are set in hardware rather than in software. But step two actually serves a second purpose of giving the USART enough time to finish resetting. The act of setting the baud rate takes plenty of time. But if you're not doing that, then do two XTHLs instead.

The fourth and final step is to read from the data port twice. This is done to ensure that the input status bit starts out cleared. Otherwise, any extra bits that may have been lying around inside the USART might be mistakenly presented to your program as newly received data.

Is It Working?

Ok, let's say you have everything hooked up correctly. You have all the wires crossed or not crossed as needed. Further, you have installed all the correct routines or information into the program for initializing the serial port, testing the input and output status bits in the USART's status byte, and sending and receiving bytes of data through the serial port. But still it doesn't fry. Somehow, the command to flail the meat just isn't getting through to the oven. What else could be wrong?

Handshaking

Receiving data over a serial line is like taking dictation. If you get too much data too fast, faster than you can write it down, you will forget some of it. If the sender can talk faster than the receiver can write, then the rate at which data is sent must be controlled by the receiver. When a secretary is taking dictation, he might just look up when he is ready to receive more. Or when taking down an address over the phone, you might say "Uh huh" when you are ready for the next line. But in either case, the sender and receiver are cooperating in order to transfer the data at the maximum successful rate. In Computerese this type of cooperation is called "Handshaking."

The extra signal wires in a serial cable, besides the actual data wires, can be used like the body language of the secretary taking dictation. One of those bits, as described above, is visible at the other end. This one is useful when the other end goes to the trouble of paying attention to it. The other bit is like the secretary who puts his left hand over the boss's mouth while he finishes writing with his right hand. This might be considered a little crude in human circles. But it is actually easier for the boss to cooperate in this way, without dividing her attention between the data and the secretary. And this is the method most commonly used by serial printers to control the output of their bosses.

However, when you're talking on the phone, you can look up 'til the cows come home and you won't get any more data until you say "Uh huh." In this case, those body language wires just mean that your modem is working. For this reason, equipment designers who fear their products might someday be used remotely, over the phone, or just through El Cheapo, 3-wire serial cables, often designate certain control characters to mean "Uh huh" and "Just a moment, there's somebody at the door."

If this is how your oven works, the software that came with it should take care of this and you should not have any problems until you try to roll your own. But if handshaking by some other means is required, you might have to do something else right before you can get your data flowing without overflowing. For example, you might have to provide a byte output routine that waits for two bits to come true in the USART's status byte, showing that both the USART and the machine at the other end are ready for more data.

The Break-out Box

The only tool I can recommend, to help you see what is happening when the right thing isn't, is a thing called a break-out box. These are available for only a few times what they're worth from any hip electronics store or from Radio Shack. You can probably get along quite nicely without one. But when all else fails, a break-out box can show you which wires really do have signals on them and what those signals are really doing. In addition, it provides a relatively easy way to temporarily rearrange some of the wires to see what, if any, effect this might have. This information can often help you identify the problem.

Over to the User!

I can't begin to tell you all the specific things you might have to do to make a successful, serial connection to your oven. Like any other fully standardized interface in the computer industry, no two serial connectors are 100% compatible with each other. But I hope I have provided enough insight to enable you to work out the details of your particular problem for yourself.

So now you've read all the way through my article on serial ports and you still don't have step-by-step instructions for hooking up your public domain spreadsheet program to the RS-232C connector on the back of your new microwave oven and making it count calories for you. Well, life is tough and then you die. But in the meantime, if you want to find out what you can do with that new oven, I hope you will benefit from the knowledge that the letters DTE, stamped next to the connector on the back, do not mean "Don't Touch Ever."

# # #

```
ENDMLOC:  dw   DISKBUF+200h   ;end of used memory
          db   01h
          db   25h
          dw   0              ;softsectored controller info
trackA:   db   80             ;physical number of tracks drive A
trackB:   db   80             ;physical number of tracks drive B
trackC:   db   35             ;physical number of tracks drive C
trackD:   db   35             ;physical number of tracks drive D
trckcur:  db   0              ;physical number of tracks current drive
DBUFLOC:  dw   DISKBUF        ;location of the diskbuffer
          db   0,0,0          ;free for future expansion
OFTEN:    RET
          dw   0              ;for jump to own OFTEN routine
drvtrk:   db   0              ;special use
TYPERR:   db   1              ;type of diskerror
PROMLOC:  dw   PROMADR/100h   ;high byte of promaddress
NRTRK:    db   35             ;number of tracks on current floppy
HORIZON:  db   'H'
CONFIG:   db   11000011b      ;configuration byte: side & stepspeed
NRDRIV:   db   4              ;number of drives online
DRIVPAR1: dw   0              ;Format byte per drive
DRIVPAR2: dw   0
horiob:   db   11010100b      ;initial IO-byte
MODE1:    db   10h            ;enable interrupt
```

(Upper case: Original entries; lower case: added entries)

Listing for "80 Tracks", p 14.

80 Track Drives in Lifeboat CP/M 2.2

By Joep Bär
P.O. Box 24
1920 AA Akersloot
The Netherlands

Bob Bloom has published two articles about installing TEAC half-height 80 track drives, and making North Star CP/M work with them (The Compass, vol. V, nos. 1 and 3).

I also installed these TEAC drives in my North Star Horizon, but went the hard way in changing the Lifeboat BIOS myself. (In The Netherlands almost all hobbyists use the Lifeboat implementation.)

The hardware story.

Since I bought my 64Kb North Star Horizon in 1981 with two Shugart SA400 35 track SS/DD drives, my need for data-storage increased. At first I added two identical second-hand Shugart drives. But after a while four drives with 165 Kb storage each proved not to be the right solution. Sometimes I ran out of space with printfiles of 200 Kb or even more. So I had to move to larger drives and I decided to buy 2 TEAC FD55F halfheight 80 track DS/DD drives. They have the advantage of a formatted storage of 800 Kb (system tracks included) and you can make them switchable from 80 to 40 tracks (remove R16 and install a 10 ohm resistor in R14, or install a (double) switch between both the resistors and the board for easy going back and forth between 40 and 80 tracks).

I needed the switching capability because I had to modify the BIOS myself in 40 track mode, and I still can use the drives in 40 track mode for my unmodified North Star Dos.

For installation of the drives see Bob Bloom's first article.

The Software Story

In our Dutch / Belgium Users Group we needed a relocated CP/M to use the RAM above the diskcontroller board location. One member started the project by disassembling Lifeboat CP/M and modifying the loader and warm-boot routines. A second member added special routines and a lot of documentation. Since 1982 I have customized this version for different needs for legal users.

So for me it was easy to start the 80 track project. It took several steps each in which new features were added but in a period of nine months I managed to get the project ready, including a new formatter, a copy- and a setupprogram.

Now I have these possibilities:

1. I can configure the BIOS per drive as 35, 40 or 80 tracks drive, single or double sided, single or double density.

2. The following floppy-type's can be used:

Format-byte:	Description:
10	35 track SS/SD CP/M 1
B0	35 track SS/DD CP/M 1
90	35 track SS/DD CP/M 2
F0	35 track DS/DD CP/M 2 (standard CP/M formats)
94	40 track SS/DD
F4	40 track DS/DD
98	80 track SS/DD
F8	80 track DS/DD (own version 2 Kb blocks)
F2	80 track DS/DD (N* Nine floppy: 4 Kb blocks)

New types can also be added. (I made my own version for 80 track DS/DD because at that time I didn't know of the North Star implementation.)

3. CP/M switches from 35 to 40 and to 80 track floppys as easy as from single sided to double sided. All are recognized by the format byte on the floppy and all internal parameters are set accordingly.

4. A mix of 35, 40, and 80 track drives both single and double sided is possible. This is a configuration option.

5. I can read and write 35 and 40 track floppy's on a 80 track drive. A BDOS-Select error is given when I try to use a 80 track floppy on a 35 or 40 track drive or a 40 track floppy on a 35 track drive.

6. The sequence of CP/M on the system tracks is modified (explained later in this article). At warm boot time a warning is given when a wrong CP/M version is present in drive A. The systems waits until you have loaded a correct CP/M version.

7. I use this CP/M version also on a second unmodified North Star. It is 100% compatible.

More Improvements

I made some other improvements in this CP/M version as option:

1. I have 4 memory banks (3 at 59 Kb and 1 at 64 Kb on a modified 16 Kb board). I load the CCP and BDOS from memory at warm boot instead of from drive A, and go directly to the current logged-in disk (it saves wear and tear of drive A). The rest of the extra memory is used as a RAM-disk (drive M): it greatly improves working with WordStar and Multiplan.

2. I improved the ^S function in the BDOS (see Microsystems, vol. 3, no. 4, p. 67).

3. Delete key acts like the Back-space key (see Microsystems, vol. 3, no. 4, p. 65).

4. The BDOS-err Bad Sector gives the real error type, e.g. BDOS-err Floppy write protected.

5. The CCP-prompt can give the user-number e.g. A5 (See The CP/M programmers Handbook, p. 289).

6. Files in user 0 are public (see Lifelines, vol. 1, no. 1, p. 14).

7. Files with the high bit of byte 2 set are public (see Dr. Dobb's Journal, 97 (November, 1984), pp. 48-72).

8. Files with the high bit of byte 11 set are archived (see Dr. Dobb's Journal, 99, (January, 1985), pp. 36-60 for Back-up program, and Dr. Dobb's Journal, 104 (June, 1985), p. 15, for CP/M patch).

9. Diskcontroller proms at E800 and FC00 are supported.

10. I always forget to give a ^C after disk change. So I trapped the BDOS-R/O error. It asks you if you would like a disk-reset and will continue with the program.

Your Move

Would you like to customize the BIOS yourself? If you would like it, here are some hints (I assume you know enough about the CP/M internals and the manuals):

1. Disassemble the BIOS and the Loader in Z80 code and change as much code as possible into shorter Z80 code, e.g. jumps into relative jumps and move-loops into LDIR. You need the extra storage because the BIOS has to fit into 2.25 Kb (including USER area).
But ... take care with relative jumps in the physical disk routines because some timing loops may not be changed.

2. Define the new disk-parameter blocks:

a. 40 track drives have the 35 track parameters except for the total storage. The storage increase with 25 Kb for SS and 50 Kb for DS floppy's.

b. 80 track SS floppy's have the same parameters as 40 track DS.

c. 80 track DS floppy's have 128 directory entry's and 2 Kb or 4 Kb blocks (4 Kb for North Star Nine floppy's). They have a total storage of 790 Kb (excluding system tracks).

3. The drive select logic calls a routine

that initializes the DPB at first select. Change this routine (use the North Star Horizon HRZ-D-Doc manual):

a. It loads the Format byte from the floppy and checks it with a table with format-dependent information. Enlarge this table with all new Format bytes and extra data. Per table entry you find: the Format-byte, next the blocksize in 128 byte units (8 or 16), followed by the address of the corresponding DPB and last the address of the sector translate table. For storage reasons I changed this table to read: the Format-byte, next the number of tracks per side, and last the DPB-address. Calculate the blocksize out of the DPB: at DPB-address + 3 you can find this blocksize - 1.

The address of the sector translate-table is always the double density table address unless the Format-byte equals 10h. Then it will be the single density table address.

b. When you have the Format-byte checked against the new table you must check whether this floppy can be handled in this drive: 35 track floppy's only in 35 track drives, 40 track floppy's in 35 and 40 track drives and in 80 track drives all of them. At error, return the select error code (register A = 1, Z-flag = 0). (The number of tracks per drive is located in the USER - 20h area.)

c. The drive select routine now must set at every call both the floppy's and drive's number of tracks per side in the USER - 20h area.

3. Change the low level read-write routines (DCOM):

a. If the Format-byte = F8h, then read/write the floppy at as follows:

side 0 at track# 0, 1, 2 and even tracknumbers from 6 up to 158

side 1 at track# 3, 4 and all odd tracknumbers from 5 up to 159

This is especially done for accessing the directory in a single sided 80 tracks drive.

b. If the Format-byte = F2h (N* Nine) then read/write all even tracks at side 0 and all odd tracks from side 1.

c. If the drive has 80 tracks and the floppy 35 or 40 then first make the side select and then double the tracknumber.

d. Normally the write-precompensation bit will be set at track 20. For 80 track drives this has to be set at track 40.

e. I changed the logical search for track zero in a search for track 59h. (Track 59h means: drive not at a known track.) This will always give a search for physical track zero. This was done as precaution: you can fiddle with the 40/80 track switch or use the DCOM-routines directly from a Format- or Copyprogram without setting all internal parameters.

5. Check if your version has a jump in the jumptable at BIOS-start to a BIOS-disk-reset routine. If you have a Lifeboat version 2.23a or later, it is already implemented. Change the routine used at cold- and warmboot. The BIOS-disk-reset must only: set the current tracknumber of every drive at 59h (i.e. track # not known), set the drive-parameters of every drive to zero (i.e. not selected) and clear the host-active flag, the unallocated-record-count flag and the host-written flag. (Use appendix G of the CP/M manual: blocked disk I/O as guide).

6. I like to boot from a 80 tracks A-drive both from 35 and 80 track floppy's: change the order of the CP/M parts at the system tracks. The old order is: Loader - CCP - BDOS - BIOS. The new order is: Loader - BIOS - CCP - BDOS. The Loader loads now only the BIOS and then jumps to the cold-boot code. Change this code: first perform all initialization and then go along with the warm-boot routine. The warm-boot loads the CCP and BDOS using DCOM with the correct track-stepping factor.

7. If you have more than one memory-bank you can load the CCP and BDOS at cold-boot time in bank 1 (bank 0 is the default bank). At warm boot load then from bank 1 to bank 0. The CCP and BDOS take up together 1600h bytes = 5.5 Kb. You can, after warm boot, go directly to the current drive (as per address 4 hex). But take care:

a. clear out the user number, otherwise you will select a nonexistent drive.

b. Change the CCP-code. This always selects drive A first.

c. Change both the CCP and BDOS to use an extra parameter (stored in the BIOS) because if you not login to drive A a SUBMIT-file will not be recognized because drive A is not searched for it.

8. The USER area is preceded by a block of 32 bytes (20hex) with some parameters. This USER - 20h area now has the layout described in Fig. 1 (p. 10).

To modify the BIOS is certainly not a beginners task. I hope I have provided you with some help to do the modification.

My Move

If you have the Lifeboat CP/M version 2.2 and a North Star Horizon with a Z80 CPU, you can join the 80 track world. However, if you can't make the modifications yourself I can customize this CP/M version with your options if you have made no modifications or only small (in code-size) modifications in the USER area. Send $25.00 (U.S., or the equivalent in your currency), a proof of purchase of your Lifeboat version 2.2, and your serial number. You will get a customization form to fill out with all current options. Next you will receive a SS/DD N* disk with SYSGEN80, FORMAT80, COPY80, and SETUP80, with the necessary documentation.

# # #

Serial and Parallel: Mixes and Matches

By Alan H. Nelson

While installing a North Star Dimension computer with ten workstations attached, I discovered a need to interface parallel printers to serial connectors both at the central module and at several work stations. I tried out two different interface devices, a port converter by Tigertronics, and the Passport by The Printer Works. This is my report on these two devices.

Quirks of the Dimension

A problem with the Dimension results from the fact that the individual workstations are provided with serial ports only. The reason for this is clear: workstations are placed at the end of a cable which may be from 25 to 1,000 feet away from the central module. A parallel connection would require at least eight wires for the data alone, whereas a serial connection requires only two wires for communication to the port. The Dimension cable is already fairly thick: nobody would wish it thicker! Also, the user may wish to use a modem at the workstation - again, a serial port is needed.

North Star in fact assumes that you are more likely to use a modem at the workstation than a printer, for it has wired the workstation port for a modem. A null-modem cable or connector must be installed to serve a serial printer. If I had been consulted, I would have told North Star to wire the port for a printer. However, they did not think of asking my advice.

If you wish to install any printer at a Dimension workstation, then you will first need to purchase or make a null modem. These cost about $15.00 or may be wired up from parts in about twenty minutes. You may find that you also need to buy or build a gender changer, since the port at the workstation module is a male, and your printer cable may also end in a male. (Here again North Star has followed IBM, which, as far as I am concerned, went against both microcomputer convention and logic in supplying its computer with a male serial printer port.)

So far, no really serious problems. In fact, a handy solution to many problems, since the baud rate and other

characteristics of the workstation port can be set automatically in software with batch file commands, including autostart commands.

The problem comes if what you already own or want to use is a parallel printer, since the Dimension has no provision for parallel interfacing at the workstation port. Even on the central module, there is only one parallel port for shared peripherals, against two serial ports. Clearly, North Star loves serial!

Tigertronics Converter

The easiest way to interface a parallel printer to any serial port on the Dimension, including the workstation port, is to use a Tigertronics 700-series Converter, a small "black box" which has a serial connector on one side and a short parallel cable protruding from the other. The Tigertronics device I purchased worked perfectly the first time I installed it, and has been working perfectly since. The only thing I had to do besides plugging the module in-line between the workstation module and the printer was to open up the box by removing four screws and move a jumper to set the baud rate as desired. (I chose 2400.)

The Tigertronics comes serial-parallel or parallel-serial, and with or without an external power supply. Although I ordered a unit with an internal power supply (power supplied by a pin on the printer-side), the unit I actually received had an external power supply (a standard battery-eliminator). Since I had an outlet handy, I used the unit as it came rather than exchanging it for the one I had ordered.

The Tigertronics is listed at $89.95, but requires a connector for the printer at $10.00. The external power supply option costs $15.00 more. All in all, a lot cheaper than a new printer if you already have a parallel printer!

The Passport

A much more sophisticated option, though a more expensive one, is the Passport, manufactured and sold by The Printer Works of Hayward, California. The Passport is a "black box" with four ports and four switches. Two of the ports are parallel, one for parallel-out, and one for parallel-in. The other two ports are serial, and either can be used for serial-out or serial-in, though by preference the first is used for input, the second for output. In addition, the Passport contains a 64K buffer (optional 256K), along with switches to clear the buffer or to hold/copy (stop printing or send text held in the buffer once again, or any number of times).

I first installed a Passport as a serial-to-parallel converter with a 64K buffer. This required setting a few dip switches for baud rate selection, a task made simple by the clear printed instructions. The Passport worked perfectly the first time I installed it, and has never given the least bit of trouble.

The Passport can be configured almost any way you can think of hooking up as many computers and printers as you can connect to four ports. Thus one computer can be connected to two printers, and the Passport will send text to which ever printer happens to be turned on at the time. If both printers are turned on, both will receive text. In the current release, printing will proceed at the rate of the **slower** printer, but an upgrade just around the corner will allow text to be sent to the printers at the natural printing speed for each printer, so long as free space is available in the buffer to handle the differential. (The Passport communicates intelligently with the printers, using standard protocols.)

The Passport can also be configured with two computers feeding into one printer. It will send the text stream from either computer to the printer. In the current release the Passport will merge print streams if both computers try to print at the same time, but in the upgrade this problem also will be fixed by more intelligent, dynamic allocation of the buffer.

The final frenzy of interfacing might connect two computers through a Passport to two printers, for a first-class integration of computers and peripherals suitable for a small-office environment.

A 64K buffer is large enough for many office applications, such as letters and short reports, and the 256K version will be even more helpful. The Passport uses a circular buffer configuration, so that as it spews characters out to the printer(s), it can accept more characters from the

computer.

Using a print buffer is almost as good as having a second computer, since, except for very lengthy documents, it tends to free up the computer for uses other than printing.

The Passport is understandably more expensive than the Tigertronics Converter since it is much more versatile, has more ports, and includes a buffer. The basic price of $245.00 covers a 64K print buffer plus a choice of any two cables. The 265K version with the same two cables is $295.00. Additional serial cables are $19.00, and additional parallel cables are $29.00. (Cables are required for all connections, since the ports on the Passport are non-standard, a sacrifice made in the interest of miniaturization.

The Switchless Switchbox

The Printer Works also manufactures the Switchless Switchbox, a unit with three **parallel** ports, including one input port and two output ports. It allows two parallel printers to be interfaced to a single parallel computer output port. It will send a print stream to whichever printer is turned on at the time, and will send a print stream to both printers (at the rate of the **slower** printer) if both printers are switched on. The Switchless Switchbox has no buffer. Its price is in line with its more humble function: $119.95.

Which One is for You?

If all you need is simple conversion from serial to parallel (or vice versa), the Tigertronics Converter is the more reasonable purchase. It does one job, and does it well. If, on the other hand, you can foresee a more complex interfacing task in the near or distant future, or if you desire a print buffer, the Passport is the right buy. I would also recommend the Passport as a simple print buffer for serial-to-serial or parallel-to-parallel interfacing. The unit is well constructed and cheaper than many other buffers of comparable size, and can be used later with different (e.g., upgraded) equipment, whether computers or printers, using virtually any mixture of parallel and serial.

Potpourri

All of the units I have just described can be mixed and matched, and any or all may prove to be a lifesaver when you are trying to interface devices which were not originally created for one another.

The Tigertronics Converter and the Passport are particularly useful to the personal microcomputer user who creates or supplements a system by mixing units designed by different manufacturers, or makes a habit of frequent upgrading. The Passport will enhance the performance of a system by the provision of a buffer. Either the Passport or the Switchless Switchbox will seem like a godsend to the user who spends half his day or evening moving cables around from printer to printer or computer to computer. All units are highly recommended.

Buffers

Although I personally could scarcely survive without printer buffers, I have found that some users, particularly beginners, dislike the "spongy" effect a buffer creates between the computer and the printer. These users like to be able to tell the computer to tell the printer to stop, and to trust that the printer will in fact stop on a dime. With a buffer between the computer and the printer, the computer can be told to stop printing, but by this time the buffer may hold 32K or 64K of text which it will continue to send to the printer unless the buffer is cleared or the printer is turned off.

My advice is to let the novice learn on a system without a buffer, and then to add a buffer when the novice has become a confident user.

Details

Tigertronics
2734-C Johnson Dr.
P.O. Box 3717
Ventura, CA 93006
(805) 658-7466
(805) 658-7467

The Printer Works
1961 Alpine Way
Hayward, CA 94545
(415) 887-6116

DOS's INitialize Command

By Saul G. Levy
2555 E. Irvington Rd., Lot 47
Tucson, Arizona 85714
(602) 889-7753

For several years I've wanted to dig into the IN command's code and make a minor change. I want to be able to initialize only the back sides of my diskettes! Why? Because I've a large number of single-sided and flippy diskettes (where both sides are used independently). I knew that eventually I would switch over to two double-sided drives (Shugart's, not Tandon's!), making my Horizon a fully-configured, quad-density system (the next eventuality is a hard disk!).

When I converted my diskettes from single to double density on a one-drive system it demanded a lot of file copying. Going from double to quad is just as demanding, even on a two-drive system! My old diskettes are in good shape and I want to continue using them without having to replace the labels I've written on (manufacturers' labels are a bear to remove). By initializing only the back side, I save having to copy the front side onto a temporary diskette. Flippy diskettes are a bit messier because the back side has to be copied BEFORE it is initialized.

How Does IN Work?

Release 2.1.1 DOS was used for what follows (earlier DOSes don't use the same code!). First, I had to find the initialize routine by using the Monitor. Typing 'SM 100-FFF "I","N"' gives two addresses: 71AH and DF7H. The first one is in the PRESS RETURN TO CONTINUE message; the second is in the DOS' command jump table:

```
49 4E 63 0F   The initialize routine starts at F63H (the low byte
 I  N  c        always comes first)
```

Second, I had to disassemble this routine which ends at F1AH. Using the PDS assembler on the source code gives the listing titled INITIALIZE ROUTINE. This routine is called by the DOS' command input and parsing routines which I won't discuss here. I also skip lightly over the two subroutines called by INITIALIZE (which in turn call two other subroutines). Only the main routine is important to what I want to do.

Third, I had to figure out how this code works. The IN command has several variations:

```
IN       On default drive 1, default density is double
IN 2     On drive 2, default density is double
IN 1 D   On drive 1, double density
IN 2 S   On drive 2, single density
```

If you give the density, you must include the drive number. The CALL 078BH in line 3 determines if a drive number was entered, otherwise, default drive 1 is assumed. The call in line 6 looks for a density specifier (D or S). These calls are to addresses out of BOUNDS to the routine which called them (F63H-FA1H here). My source code labeling program (SLABEL) flags these along with any other 16-bit values which are outside of the INITIALIZE routine. A programmer has to check each of them to determine what is being referred to (it may not be obvious!).

Lines 7-14 use the drive number and density to set the B and C registers for the call to DCOM in line 30. Register C is the drive number (the high bit is set for double density). B is the DCOM mode command to be used:

-2 is initialize write in double density
-1 is initialize write in single density
0 is write a sector
1 is read a sector
2 is verify a sector

Lines 16-23 initialize two pages (one sector) of memory (512 bytes) with 20H's (a space). 800H is the starting RAM address of the DOS' disk buffer which holds one sector of 512 bytes. Register D is set to 0 and decremented 256 times (FFH to 00H) while register pair HL is incremented TWICE and a space is stored into each memory address pointed to by HL. When register D is 00H again the zero flag is finally set to a 1 and the loop from lines 18-23 ends.

Lines 25-37 use DCOM to initialize one track (10 sectors) with each pass through the loop from lines 26-37. Line 25 sets register pair HL to the starting sector address (0000H) for a DCOM call. Line 26 sets the accumulator to 10 sectors (one track) for the DCOM call (the SOFT-DOC Manual says that this is in blocks which is incorrect). It appears that the negative DCOM mode byte will make DCOM write 10 copies of the sector buffer rather than starting with the sector buffer and writing 10 contiguous sectors of memory. It also appears that a negative mode byte preinitializes each sector, otherwise, a T2 error occurs (CRC compare). I haven't figured out all of DCOM yet!

Line 29 sets register pair DE to the starting RAM address of the initialized disk buffer at 800H. DCOM uses registers B, C, H, and L so these have to be saved on the stack by lines 27-28 (the accumulator is also used, but is reloaded in line 26 each pass). DCOM is called in line 30 and the disk address is restored to HL in line 31. Lines 33-34 add 10 sectors to HL which points to the first sector in the next track (HL is set to 0, 10, 20, 30, etc. so DCOM will initialize sectors 0-9, 10-19, 20-29, 30-39, etc.). The DCOM mode is restored in line 36. Line 37 continues running the loop until the carry flag is set.

Carry is set by DCOM whenever an error is made in setting DCOM's parameters. The usual error is trying to initialize sector 350 (15EH) on a single-sided drive or sector 700 (2BCH) on a double-sided drive. The carry flag has to be saved on the stack by line 32 (PSW stands for program status word, i.e. the accumulator and flags registers) because line 34 will always reset the carry flag to a 0 (HL will never be larger than 15EH or 2BCH; nowhere over FFFFH which would set the carry flag to 1). The flags are restored in line 35.

Lines 39-42 end the routine. Lines 39-40 check for a DOS error. If the accumulator is not an 0AH, the DOSERR routine is jumped to (this error is usually found earlier than line 40 if the IN command contains an error). Otherwise, a 0 is stored in 135H and the routine returns to wherever it was called from. There are three STA 0135H's in the DOS, but no LDA 0135H's to match. My guess is that the byte at 135H is for use by the people who wrote the DOS and is not for use by anyone else.

INITIALIZE is a straightforward routine so long as you understand the IN command and how DCOM works. Lines 7-14 are understandable only if you also understand the four subroutines I haven't discussed! You don't need to understand very much of the above: modifying two bytes will do what I need!

Modifying the DOS to Initialize Only the Back Side

We can change line 25 (F81H-F83H) to LXI H,015EH (low byte first, i.e. 21 5E 01) to make IN initialize only the back side. An address table for the three latest DOSes is:

DOS Release	Object code to search for	Address in running DOS	Address if DOS loaded at 1000H	Monitor search command
2.1.1	21 00 00 3E	F81H	1D81H	SM 100-FFF 21,0,0,3E
5.2	21 00 00 F5	DC4H	1BC4H	SM 100-DFF 21,0,0,F5
5.1	21 00 00 F5	2CBCH	1BBCH	SM 2000-2CFF 21,0,0,F5

There is only one occurrence of these four bytes in each DOS. Change only the two zero bytes (DS F82 or DS 1D82, etc.)! Check your work! Note that 015EH can be changed to the first sector of any track (evenly divisible by 0AH) if you want to start at some other track.

Another Way to Do It (Program INBACK)

If you change the DOS, you will have to save this special version and be careful to load it only when needed. This is a pain to keep track of so I incorporated the initialize routine into a separate program titled PROGRAM INBACK which I will reassemble for use in high memory. I will use BASIC and the CF utility to copy files and don't want to have to reload anything (see Copying Files below).

INBACK differs in a number of ways from the DOS' initialize code. I don't bother parsing the DOS' command input buffer (the density will always be double). I have to output a message to tell you what this program does and allow you to enter one character for the drive number (1-4 are valid, otherwise, the program returns to the DOS without writing on the diskette, this also occurs if you use one-sided drives). I also allow you to hit the RETURN key or the space bar if you want default drive 1. INBACK can be reentered at the GO address and run over and over again (many programs aren't reentrant and must be reloaded each time). INBACK is fully commented. A few additional comments follow:

6-8...Change the program's origin and DOS' origin to suit your needs. Note that I didn't need to use the DOSERR routine (line 40 in INITIALIZE). If the drive door is open or no diskette is loaded, the DOS will give a T4 error (no index pulse); opening the door during initializing will give a T5 error (density mismatch, a strange error indeed!).

15...Always use an internal stack! Otherwise, it's possible to destroy the DOS if the program stack moves down into the DOS' object code.

23...Output the entered byte (character) so the user can see what was typed.

29...DRINO (drive number) is a shortened version of the subroutine called from line 3 in INITIALIZE.

32...I use an internal sector buffer (SBUF) rather than the DOS' sector buffer.

43...Note that the starting sector is 350 (15EH).

44...The DCOM mode is -2 which is an FEH.

45...This is a Z80-only instruction which sets the density to double. If you have an 8080 CPU, replace this instruction and add two more:

```
79      MOV A,C   Move drive # to A
F6 80   ORI 80H   Set high bit
4F      MOV C,A   Move back to C
```

I have not tested this modification, but it should work (using a Z80 saves two bytes).

60...Replaces lines 39-42 in INITIALIZE.

73...DRINO calls SPACE (line 88) which calls CR (line 94). These are two of the subroutines which I haven't discussed. SPACE originally skipped over spaces in the IN command; CR checked for the ending carriage return in the IN command. INBACK shortens these subroutines to check the entered byte for a CR or space and use either as the default drive number 1. The strange calling logic is due to the original code in the DOS. I used it rather than change it.

76...The entered byte is in ASCII and must be converted to a number by subtracting 30H. I store the byte and drive number in memory locations BYTE and DNO as a debugging aid (I can use the Monitor to check that these values are handled correctly). Note that BYTE's storage is two bytes long (a 16-bit word). I did this so that BYTE is always followed by a 0 byte; this forces the OUT subroutine to output only the entered byte.

104...Note that the last byte in the program is at 10AEH so only one block of disk space is needed to store INBACK. The sector buffer (SBUF) and stack areas don't need to be saved to the diskette and can contain anything before the program starts.

107...The stack pushes down into these 20H bytes which is more than is needed by the program.

Copying Files

After initializing the back side of each diskette I will have to copy the extra files to the now doubled file space. CF (Copy File) is used for this, but having to type 100's of file names (twice!) is a pain. This is why I will use BASIC and my Smart Copy File (SCF) program to quickly copy files using CF without typing any file names (see Compass, Vol. IV, no. 4, p. 21).

Another File Offer

Morris Miller's ZDIS disassembler and my SLABEL labeling program create PDS files for use by Program ASMB and are available for FREE (ZDIS is also available in a version for the TDL macro assembler). SCF is $20.00. Please add $3.00 for postage and handling of each order (not each program) and add $5.00 for one of my diskettes (it pays to send me one of yours).

I hope that INSUA members will find the discussion of INITIALIZE helpful and Program INBACK useful (even if for tasks I haven't thought of)!

# # #

```
1000                    0001 ; PROGRAM INBACK
1000                    0002 ;
1000                    0003 ; INITIALIZES BACK SIDE OF DISKETTE ONLY!
1000                    0004 ; 'IN' CODE FROM 2.1.1 DOS
1000                    0005 ;
1000                    0006        ORG   01000H        ORIGIN, CHANGE TO SUIT
1000                    0007 ;
1000                    0008 DOS    EQU   0100H         DOS' ORIGIN
1000                    0009 COUT   EQU   DOS+0DH        CHARACTER OUTPUT
1000                    0010 CIN    EQU   DOS+10H        CHARACTER INPUT
1000                    0011 DCOM   EQU   DOS+22H        DISK COMMUNICATIONS
1000                    0012 WARM   EQU   DOS+28H        DOS' WARM ENTRY ADDR
1000                    0013 ;
1000                    0014 ;
1000 31 CF 12           0015 START  LXI   SP,STACK       SET STACK POINTER
1003                    0016 ;
1003 11 7D 10           0017        LXI   D,MSG          OUTPUT SIGN-ON MESSAGE
1006 CD 4C 10           0018        CALL  OUT
1009                    0019 ;
1009 CD 10 01           0020 INPUT  CALL  CIN            INPUT DRIVE #
100C 32 AD 10           0021        STA   BYTE           STORE ENTERED BYTE
100F                    0022 ;
100F 11 AD 10           0023        LXI   D,BYTE         OUTPUT ENTERED BYTE
1012 CD 4C 10           0024        CALL  OUT
1015                    0025 ;
1015 11 A9 10           0026        LXI   D,MSG2         OUTPUT CRLF
1018 CD 4C 10           0027        CALL  OUT
101B                    0028 ;
101B CD 59 10           0029        CALL  DRINO          WAS DRIVE # ENTERED?
101E                    0030 ;
```

```
101E                            0031 ; INITIALIZE SECTOR BUFFER WITH SPACES
101E 21 AF 10                   0032         LXI    H,SBUF          POINT TO SECTOR BUFFER
1021 16 00                      0033         MVI    D,00            LOOP INDEX
1023                            0034 ;
1023 36 20                      0035 INBUF   MVI    M,20H           LOAD SPACE
1025 23                         0036         INX    H               INCREMENT ADDR
1026 36 20                      0037         MVI    M,20H           LOAD 2ND SPACE
1028 23                         0038         INX    H               INCREMENT ADDR
1029 15                         0039         DCR    D               LOOP 256 TIMES
102A C2 23 10                   0040         JNZ    INBUF           LOOP AGAIN
102D                            0041 ;
102D                            0042 ; SET UP REGISTERS FOR DCOM CALL
102D 21 5E 01                   0043         LXI    H,015EH         START AT SECTOR 350
1030 06 FE                      0044         MVI    B,0FEH          INIT WRITE COMMAND
1032 CB F9                      0045         STB    7,C             SET HIGH BIT OF DRIVE #
1034                            0046 ;
1034                            0047 ; INITIALIZE BACK SIDE OF DISKETTE
1034 3E 0A                      0048 INDSK   MVI    A,0AH           10 SECTORS (1 TRACK)
1036 C5                         0049         PUSH   B               SAVE REGISTERS
1037 E5                         0050         PUSH   H
1038 11 AF 10                   0051         LXI    D,SBUF          POINT TO SECTOR BUFFER
103B CD 22 01                   0052         CALL   DCOM            WRITE ONE TRACK
103E E1                         0053         POP    H               RESTORE DISK ADDR
103F F5                         0054         PUSH   PSW             SAVE FLAGS
1040 01 0A 00                   0055         LXI    B,000AH         POINT TO NEXT TRACK
1043 09                         0056         DAD    B               NEXT TRACK'S 1ST SECTOR
1044 F1                         0057         POP    PSW             RESTORE FLAGS
1045 C1                         0058         POP    B               RESTORE WRITE COMMAND
1046 D2 34 10                   0059         JNC    INDSK           LOOP AGAIN
1049 C3 28 01                   0060         JMP    WARM            FINISHED
104C                            0061 ;
104C                            0062 ; OUTPUT ASCII MESSAGE
104C 1A                         0063 OUT     LDAX   D               LOAD CHAR
104D FE 00                      0064         CPI    0               ZERO BYTE?
104F C8                         0065         RZ                     YES, END OF MESSAGE
1050 47                         0066         MOV    B,A             MOVE TO B FOR COUT
1051 AF                         0067         XRA    A               ZERO ACC FOR PORT #
1052 CD 0D 01                   0068         CALL   COUT            OUTPUT 1 CHAR
1055 13                         0069         INX    D               DO NEXT CHAR
1056 C3 4C 10                   0070         JMP    OUT             LOOP AGAIN
1059                            0071 ;
1059                            0072 ; CHECK FOR DRIVE # ENTERED OR USE DEFAULT
1059 0E 01                      0073 DRINO   MVI    C,01            DEFAULT DRIVE #
105B CD 70 10                   0074         CALL   SPACE           SPACE OR CR IN BYTE?
105E CA 6B 10                   0075         JZ     DRIN2           YES, USE DEFAULT
1061 D6 30                      0076         SUI    30H             CONVERT ASCII TO #
1063 3D                         0077         DCR    A               DECREMENT DRIVE #
1064 FE 04                      0078         CPI    04              DRIVE # IN RANGE?
1066 D2 28 01                   0079         JNC    WARM            NO, GREATER THAN 4
1069 3C                         0080         INR    A               INCREMENT DRIVE #
106A 4F                         0081         MOV    C,A             SAVE IT
106B                            0082 ;
106B 79                         0083 DRIN2   MOV    A,C             LOAD DRIVE #
106C 32 AC 10                   0084         STA    DNO             STORE IT
106F C9                         0085         RET
1070                            0086 ;
1070                            0087 ; CHECK FOR SPACE ENTERED
1070 CD 77 10                   0088 SPACE   CALL   CR              CHECK FOR CR
1073 C8                         0089         RZ                     CR WAS FOUND
1074 FE 20                      0090         CPI    20H             CHECK FOR SPACE
1076 C9                         0091         RET
1077                            0092 ;
1077                            0093 ; CHECK FOR CR ENTERED
1077 3A AD 10                   0094 CR      LDA    BYTE            LOAD BYTE
107A FE 0D                      0095         CPI    0DH             CR?
107C C9                         0096         RET
107D                            0097 ;
107D 49 4E 49 54 49             0098 MSG     DB     'INITIALIZE BACK SIDE ON '
     41 4C 49 5A 45
     20 42 41 43 4B
     20 53 49 44 45
     20 4F 4E 20
1095 57 48 49 43 48             0099         DB     'WHICH DRIVE (1-4): '
     20 44 52 49 56
     45 20 28 31 2D
     34 29 3A 20
```

```
10A8 00                 0100          DB    0               END OF MSG
10A9 0D 0A 00           0101 MSG2     DB    0DH,0AH,0       CRLF
10AC                    0102 ;
10AC 00                 0103 DNO      DB    0               DRIVE # STORAGE
10AD 00 00              0104 BYTE     DW    0               ENTERED BYTE STORAGE
10AF                    0105 ;
10AF                    0106 SBUF     DS    200H            512 BYTE SECTOR BUFFER
12AF                    0107          DS    20H             ROOM FOR STACK
12CF                    0108 STACK    EQU   $               START STACK HERE
12CF                    0109 ;
12CF                    0110 ; WRITTEN BY SAUL G. LEVY, TUCSON, ARIZONA,
12CF                    0111 ; MAY 15-6, 1985
12CF                    0112 ;
12CF                    0113          END
```

SYMBOL TABLE

```
BYTE   10AD    CIN    0110    COUT   010D    CR     1077    DCOM   0122
DNO    10AC    DOS    0100    DRIN2  106B    DRINO  1059    INBUF  1023
INDSK  1034    INPUT  1009    MSG    107D    MSG2   10A9    OUT    104C
SBUF   10AF    SPACE  1070    STACK  12CF    START  1000    WARM   0128
```

```
0F63                    0001 ; INITIALIZE ROUTINE
0F63                    0002 ;
0F63 CD 8B 07           0003          CALL  078BH           BOUNDS
0F66                    0004 ;
0F66 F5                 0005          PUSH  PSW
0F67 CD FB 0C           0006          CALL  0CFBH           BOUNDS
0F6A 78                 0007          MOV   A,B
0F6B 0F                 0008          RRC
0F6C E1                 0009          POP   H
0F6D B4                 0010          ORA   H
0F6E 4F                 0011          MOV   C,A
0F6F 78                 0012          MOV   A,B
0F70 2F                 0013          CMA
0F71 47                 0014          MOV   B,A
0F72                    0015 ;
0F72 21 00 08           0016          LXI   H,0800H         BOUNDS
0F75 16 00              0017          MVI   D,00H
0F77 36 20              0018 L0F77    MVI   M,20H
0F79 23                 0019          INX   H
0F7A 36 20              0020          MVI   M,20H
0F7C 23                 0021          INX   H
0F7D 15                 0022          DCR   D
0F7E C2 77 0F           0023          JNZ   L0F77
0F81                    0024 ;
0F81 21 00 00           0025          LXI   H,0000H         BOUNDS
0F84 3E 0A              0026 L0F84    MVI   A,0AH
0F86 C5                 0027          PUSH  B
0F87 E5                 0028          PUSH  H
0F88 11 00 08           0029          LXI   D,0800H         BOUNDS
0F8B CD 22 01           0030          CALL  0122H           BOUNDS
0F8E E1                 0031          POP   H
0F8F F5                 0032          PUSH  PSW
0F90 01 0A 00           0033          LXI   B,000AH         BOUNDS
0F93 09                 0034          DAD   B
0F94 F1                 0035          POP   PSW
0F95 C1                 0036          POP   B
0F96 D2 84 0F           0037          JNC   L0F84
0F99                    0038 ;
0F99 EE 0A              0039          XRI   0AH
0F9B C2 2C 01           0040          JNZ   012CH           BOUNDS
0F9E 32 35 01           0041          STA   0135H           BOUNDS
0FA1 C9                 0042          RET
0FA2                    0043 ;
0FA2                    0044          END
```

SYMBOL TABLE

```
L0F77 0F77   L0F84 0F84
```

Databases from Online Sessions

By Alan H. Nelson

A few issues back I described Bank of America's HOMEBANK facility: see "Banking on your Computer," Compass, Vol. IV, no. 6, pp. 27-30. In concluding that article, I offered a challenge to anyone who used a homebanking program to write program which would save the results of a homebanking session for further systematic analysis. I thought it must be possible to turn the report of recent bank transactions into a useful database which could be analyzed automatically. Since nobody took up the challenge, I did the job myself, using dBASE II.

As it turns out, solving this problem is similar to solving the problem of turning any material organized in one fashion into a database of material organized in a different fashion. Thus the enterprise has implications beyond my immediate purposes.

Briefly, the following routine is used to save a HOMEBANK session to a file:

1. Using standard modem hardware, invoke MODEM7, T(erminal) option, with a file designation (I call my file "SESSION"). Phone up the HOMEBANK number.

2. When the connection has been established, hit Ctrl-Y (to save to memory everything that appears on the screen). Review all recent transactions with the ELECTRONIC STATEMENT option.

3. At the end of the session, hit Ctrl-E to return to the MODEM7 menu, then issue a WRT command to save the session in memory to a disk file. Return to CP/M.

Examining the Captured Session

The HOMEBANK session as saved on disk should be a pure ascii file. If you want to inspect it with a program like WordStar that's o.k., but use the **N** rather than the **D** option, since the **D** option can turn the ascii file into a text file with the high bits set. High bits give dBASE conniption fits.

When I examined a typical "SESSION" file (see Listing 1), I noted that the whole page printed to the screen by HOMEBANK is 39 characters wide, including two blank columns at the left margin. Although the session does not have the rigorous column and row organization of a fully structured data base file, various items are organized more or less regularly and therefore predictably. Dates, identification or transaction numbers, and dollar amounts tend to fall into columns, at least for the same kinds of information. Descriptions are usually contained either in the same line as the dollar amount, or in the next line.

As I thought about turning this partly structured information into a fully structured database, I broke the procedure down into two major steps.

A Hybrid Database

Step 1 is to construct a database structure which will accept the semi-structured source material as it comes, and at the same time provide the fully structured fields required by the target database. Since the source material is not all organized into columns or into any other clearly recognizable structure, I decided to copy each line of the source file into one long, undifferentiated field in the working file. The only problem here is to find an appropriate width for this field. Since no line goes past the thirty-ninth column on my screen, I set the field at 39 characters wide.

I then created a number of differentiated fields into which material extracted from the first field could be copied. These differentiated fields include one field for the dollar amount, one field for a verbal description, and a logical field for recording whether the dollar amount is a credit or a debit. Other fields include date of transaction, bank number of transaction, check number, etc., etc.

My database structure is shown in Listing 2.

```
** MAIN MENU **
...
5 ELECTRONIC STATEMENT
...
ENTER: 5

CHECKS
ENTER SECTION NBR OR P FOR NEXT PAGE
   CHECK    POSTED        AMOUNT
      2496  06/10          $215.00
      2497  06/07           $74.72

OTHER DEBITS
ENTER SECTION NBR OR P FOR NEXT PAGE
  TRANSACTION    DATE          AMOUNT
876543210 000000 05/31           $8.00
HOME BANKING SERVICE CHARGE FOR THE
 MONTH OF MAY---- FOR 876543210
876543210 151028 05/31         $543.21
PAYMENT TO BA VISA
  4321-1234-5678-9098
812345678 008816 06/10          $40.00
CASH WITHDRAWAL

876543210 165034 06/14         $233.50
PAYMENT TO AMERICAN
 EXPRESS 345678909876543

OTHER DEBITS

SERV CHARG       06/25           $0.00
SERVICE CHARGE (BASED ON AVERAGE
 DAILY BALANCE OF 1,234.56)

DEPOSITS
ENTER SECTION NBR OR P FOR NEXT PAGE
EFFECTIVE  POSTED          AMOUNT
           06/05           $298.50
           06/25           $558.08

OTHER CREDITS
ENTER SECTION NBR OR P FOR NEXT PAGE
  TRANSACTION    DATE          AMOUNT
MAY SALARY       05/31       $1,345.67

** MAIN MENU **
0 LOGOFF
...
ENTER: 0
...
YOUR HOMEBANKING SESSION
HAS BEEN CONCLUDED.

remote: (2021/4): call cleared (c  0,d  0): dte originated
```

Listing 1

Sample HOMEBANK session (simulated)

The Hunt for Distinctive Identifiers

Step 2 is to write a program (see Listing 4) which will examine the session one line at a time, extract information from the undifferentiated field, and copy selected items into the differentiated fields. An overall **while loop** instructs the program to move from one line to the next. This loop checks for END OF FILE, and if EOF is not found, skips to the next line.

As the program pauses at each line, it must first identify the nature of the information in the first field. The hard part of writing the program is to discover a fool-proof system for distinguishing one kind of information from another.

Distinctions can be made only by inspecting the source information carefully. Curiously, the identification may be easier by using indirect rather than direct evidence. I noted, for example, that dates associated with significant transactions are indented different amounts from the left margin. The placement of the date thus served as my first key to establishing the nature of the transaction.

Although the date field is always five characters wide, it is not necessary to check for the full field, since the slash (/) by itself establishes the identity of the transaction, as follows:

Slash in Column	Category
16	Deposits except salary checks (credits)
17	Checks (debits)
22	All other significant transactions (both credits and debits)

Armed with this discovery, I knew I had the chief information I needed to write the program.

Even when the transaction can be placed in the correct category, a full description of the transaction may take some effort to establish. Companies paid by a HOMEBANK transaction rather than by check, for example, are noted by name - the Power company, VISA, etc. But the name is on the line **following** the line containing the dollar amount. In this case the program must skip a line, store the name to memory, skip back to the line containing the dollar amount, and then copy the name from memory into the description field.

Skipping down and back is no trouble, except that it would be an error to skip past the very last line in the source file. Since I determined that my source file always ends with a signoff message which cannot conceivably contain a dollar amount or the name of a payee, I assume that the program will never try to jump past last line at this stage. I do not bother to check for an EOF marker while skipping ahead when I know I can safely skip right back again.

As the program pauses at each line, a series of case statements test for the various distinctive characteristics which have been noted for each kind of transaction. When any case proves true, extracted information is stored to the proper differentiated, fields, and logical fields are set appropriately. Thus the deposit flag is set .T. for all money which goes into the account (credits), and is set (or left) .F. for all other cases (debits). (Another program, run subsequently on the completed database file, will draw the balance by first summing the .T.'s, then and summing the .F.'s, and then taking the difference.)

Non-Salary Deposits and Checks

Within the controlling while loop, then, the program first checks for a slash in either the sixteenth or seventeenth column. These make up two of the three possible identifying marks for significant entries. The dollar amounts are extracted and copied into the **amount** field. (See below for more on this procedure.) The program then distinguishes between deposits and checks as follows:

** If the slash occurs in the seventeenth column, the transaction is a check drawn on the account. Store the date and checknumber, replace description with an underline (as a reminder that this information must later be filled in by hand), and set the deposit flag to false (this a debit).

```
.
. disp stru
Structure for file: A:SESSION .DBF
Number of records: 00060
Date of last update: 00/00/00
Primary use database
Fld     Name          Type Width  Dec
001     RECORD        C    039
002     DATE          C    005
003     CHECKNO       C    004
004     RECNO         C    006
005     DESCRIPT      C    020
006     AMOUNT        N    009    002
007     DEPOSIT       L    001
008     TICKED        L    001
009     DEDUCT        L    001
** Total **              00087
.
.
```

Listing 2

```
06/10 2496          -----check-------      215.00 .F.  .F.  .F.
06/07 2497          -----check-------       74.72 .F.  .F.  .F.
05/31               Monthly charge          8.00 .F.  .F.  .F.
05/31       151028  VISA                  543.21 .F.  .F.  .F.
06/10       008816  Cash withdrawal        40.00 .F.  .F.  .F.
06/14       165034  American Express      233.50 .F.  .F.  .F.
06/25               Service charge          0.00 .F.  .F.  .F.
06/05               DEPOSIT               298.50 .T.  .F.  .F.
06/25               DEPOSIT               558.08 .T.  .F.  .F.
05/31               MAY SALARY           1345.67 .T.  .F.  .F.
```

Listing 3

Processed Database

** Otherwise the slash occurs in the sixteenth column, and the item is a deposit. Store the date, write "DEPOSIT" in the description column, and set deposit flag to true (this is a credit).

Other Transactions

All other significant transactions are flagged by the slash in the twenty-second column. Since salary deposits are easiest to recognize, I deal with these first. They can be recognized by the fact that the number "8" **does not** occur in the first column of text (the third column of the field), nor does the letter "S" (which identifies a SERVICE CHARGE).

** Check for the non-occurrence of the designators "8" and "S" using the "unequals" operator (#), and store all the information, including the description supplied by HOMEBANK, which gives the month of the salary deposit.

This leaves four categories still to be dealt with: the regular monthly charge, service charges, Versateller cash withdrawals, and HOMEBANK cash transfer payments. Since all four are debits (even if the service charge is zero), the deposit flag is set to false, and the dates and dollar amounts are stored properly. Then the descriptions are taken care of, as follows:

** "000000" identifies the transaction as the regular monthly charge: store "Monthly charge" to description.

** "SERV" identifies the transaction as a service charge: store "Service charge" to description.

All other descriptors occur in the **second** line of the entry; first store the transaction number, then skip to the next line, and do the following before skipping back:

** Known descriptors given in all upper case are printed in the description field in mixed upper and lower case.

** Unknown descriptors are ignored except that "----unknown----" is printed in the description field.

Dollar Amounts

I noted that all HOMEBANK dollar amounts begin with a dollar sign, all contain a decimal point, and all contain a comma when the amount is over $999.99. Since the number cannot be stored directly with dollar signs and commas included in the number, the program breaks the number into two elements: first, thousands if any (a one, two, or three digit number); second, units (also a one, two, or three digit number) together with hundredths (always a two-digit number). This second element always contains a decimal point, but that's o.k. since dBASE understands decimal points. (Anyone who is likely to go over $999,999.99 will need a third category, but since I know I never will, I have not reckoned with this possibility.)

Case statements check for the dollar sign in a particular column. According to the column where the sign is discovered, thousands (if they occur), and units together with hundredths are each stored into a separate variable; then they are added together (using the proper multipliers) and the sum is stored into the dollar amount field.

I decided to repeat this routine, with various column options, for each of several situations where dollar amounts are dealt with. The program would have been more elegant if this had been made a separate routine called from various locations within the program, with the beginning column number passed as a parameter. However, this would have made the program more complicated, and I just wanted to get the job done.

Junk

The final step is to delete junk, which I define as any line that does not fit into a recognizable category. This is where the very last **otherwise** command comes into play. If none of the cases is true, then delete. The program ends by packing to eliminate the deleted lines. What's left should be pure gold (see Listing 3).

Other Categories?

Significant transactions other than the ones already described should eventually be anticipated and accommodated in the

program: possible overdraft fines, transfers from other accounts, monthly interest payments, and so forth. The need for these additional routines may be discovered when the eventual computer balance does not accord with the bank's balance.

Running the Program

Once the hybrid database structure has been created (see Listing 2), **use** the file, and then type the following command, assuming that the session has been saved into a file called "session":

append from session. sdf

It is important to add the period (or dot) after the file name - otherwise dBASE will think you are appending from a standard database file (with a .DBF extension). I used to think that dBASE only allowed an **sdf** append from a file with a TXT extension, but this is not so.

A HOMEBANK session may run to many hundreds of lines, including a lot of junk (for example, menus). Do not be surprised, therefore, if you end up with a very long database file with many records after the append command has been issued. This is a temporary condition only.

Examine the file with the dBASE edit command. Items from the session file should be in the first field only, while the rest of the fields should be blank. If the appended material has contaminated other fields, modify the structure to **lengthen** the first field and try again.

Now run the program using the **do** command. Check the results carefully for omissions or incorrect information in the various fields. (A little extra caution at this point can save a lot of grief later on.) The end result should be a nicely formatted database with all significant information intact and properly interpreted. A copy of the database can now be made without the undifferentiated first field, which by now will be pretty meaningless anyway.

Eliminating Duplicates

During my on-line sessions I often ask for the same information more than once (whether deliberately or inadvertently); also, some of the information gained in one session may already have been extracted in a previous session. Hence there is always a chance that the same information will be recorded more than once. The result might be that your June salary check is listed and eventually credited twice, or a check may be deducted twice.

Check for duplicates is fairly simple. Index on any given field or fields to bring similar items together. Check the indexed database either by hand (e.g., using **browse**) or with a program. Look for identical information in adjacent records. For example, sort on checknumber. Any two records with identical checknumbers (excluding blank checknumbers!) should by definition be identical! Delete duplicated records and pack.

The Final Steps

After the program has been run and duplicates eliminated, you may wish to go through the material to fill in fields such as descriptions for checks. The rest of the fields should have been filled in automatically and effortlessly.

A final step might be to tick the checks in the database by changing .F. to .T. in the "TICK" field, much as you tick off the checks in your checkbook when you get ready to balance your account. A simple program which sums the credits and sums the debits, and then takes the difference between them, should give you your overall balance. And I hope that after all this work, you are in the black!

# # #

```
*************************************
*  This program organizes a Homebank Session
*  into a database.  After the
*  program has run its course, processed database
*  should be checked for duplicate entries.
*************************************
goto top
do while .not. eof
*************************************
*  First, look for checks, and for
*  deposits other than salary checks
*************************************
  do case
    case $(record,16,1) = "/" .or. $(record,17,1) = "/"
      store "0" to nxamount
      store "0" to namount
      do case
        case $(record,27,1) = "$"
          store $(record,28,2) to nxamount
          store $(record,31,6) to namount
        case $(record,28,1) = "$"
          store $(record,29,1) to nxamount
          store $(record,31,6) to namount
        case $(record,30,1) = "$"
          store $(record,31,6) to namount
        case $(record,31,1) = "$"
          store $(record,32,5) to namount
        case $(record,32,1) = "$"
          store $(record,33,4) to namount
      endcase
      store (&nxamount * 1000) + &namount to mamount
      replace amount with mamount
*************************************
*  Now distinguish between (1) checks and (2) deposits
*************************************
     if $(record,17,1) = "/"
       store $(record,15,5) to mdate
       replace date with mdate
       store $(record,9,4) to mcheckno
       replace checkno with mcheckno
       replace descript with "-----check-------"
       replace deposit with F
     else
       store $(record,14,5) to mdate
       replace date with mdate
       replace descript with "DEPOSIT"
       replace deposit with T
     endif
*************************************
*  Second, look for transactions other than checks
*************************************
    case $(record,22,1) = "/"
      do case
```

```
************************************
*  a) Take care of salary deposits
************************************
        case $(record,3,1) # "8" .and. $(record,3,1) # "S"
          store $(record,3,15) to mdescript
          replace descript with mdescript
          replace deposit with T
          store $(record,20,5) to mdate
          replace date with mdate
          store "0" to nxamount
          store "0" to namount
          do case
            case $(record,29,1) = "$"
              store $(record,30,2) to nxamount
              store $(record,33,6) to namount
            case $(record,30,1) = "$"
              store $(record,31,1) to nxamount
              store $(record,33,6) to namount
            case $(record,32,1) = "$"
              store $(record,33,6) to namount
            case $(record,33,1) = "$"
              store $(record,34,5) to namount
            case $(record,34,1) = "$"
              store $(record,35,4) to namount
          endcase
          store (&nxamount * 1000) + &namount to mamount
          replace amount with mamount
************************************
*  b) Take care of debits other than checks
************************************
        otherwise
          replace deposit with F
          store $(record,20,5) to mdate
          replace date with mdate
          store "0" to nxamount
          store "0" to namount
            do case
              case $(record,29,1) = "$"
                store $(record,30,2) to nxamount
                store $(record,33,6) to namount
              case $(record,30,1) = "$"
                store $(record,31,1) to nxamount
                store $(record,33,6) to namount
              case $(record,32,1) = "$"
                store $(record,33,6) to namount
              case $(record,33,1) = "$"
                store $(record,34,5) to namount
              case $(record,34,1) = "$"
                store $(record,35,4) to namount
            endcase
            store (&nxamount * 1000) + &namount to mamount
            replace amount with mamount
            do case
              case $(record,13,6) = "000000"
                replace descript with "Monthly charge"
              case $(record,3,4) = "SERV"
                replace descript with "Service charge"
              otherwise
```

```
                    store $(record,13,6) to mrecno
                    replace recno with mrecno
                    skip
                    do case
                      case $(record,3,4) = "CASH"
                        store "Cash withdrawal" to mdescript
                      case $(record,14,5) = "SHELL"
                        store "Shell" to mdescript
                      case $(record,14,3) = "P G" .and. $(record,35,1) = "1"
                        store "PG&E 2600" to mdescript
                      case $(record,14,3) = "P G" .and. $(record,35,1) = "3"
                        store "PG&E 2592" to mdescript
                      case $(record,14,5) = "PACIF"
                        store "Pacific Bell" to mdescript
                      case $(record,14,5) = "AMERI"
                        store "American Express" to mdescript
                      case $(record,14,5) = "BA VI"
                        store "VISA" to mdescript
                      otherwise
                        store "---unknown---" to mdescript
                    endcase
                    skip -1
                    replace descript with mdescript
                endcase
        endcase
*************************************
*  Finally, delete inconsequential records
*************************************
      otherwise
        delete
    endcase
    skip
*************************************
*  End of loop; repeat or conclude
*************************************
enddo
pack
return
*************************************
*  End of program
*************************************
```

Letters to the Editor

INSUA:

Since moving to the Northwest from southern California, I have joined the NS Computer Society of Seattle, Washington, and as of this time have been unable to attend any meetings due to my remote location. My only connection has been through correspondence, which has been answered promptly and to the point. In can highly recommend it as a club that will further the aims of North Star Users.

The NS Computer Society does the following:

1. Publishes a monthly Newsletter (Tech articles, Software reviews, etc.

2. Monthly meetings in Seattle (Business/Technical).

3. Software Library.

All inquiries should be directed to:

NS Computer Society
P.O. Box 311
Seattle, WA 98111-0311

Sincerely,
A. F. Horner

INSUA:

The members of the NS Users Group of Metropolitan Washington (D.C.) were very sorry to read of the end of Compass. Many of us are subscribers, and we have frequently discussed work work that appeared in Compass. We all express our admiration for the work you have done in the past, and are sorry that it is coming to an end.

In reference to your "Notice to Clubs," we would like to offer a notice. We welcome all persons in the Washington, Maryland, and Virginia area who maintain an interest in the NS computer and related operating systems. We meet monthly on the second Tuesday of every month. Our dues are $5.00 per year and we offer a program of interest each month.

Those wishing to attend a meeting or to join are welcome to write or call:

Alan J. Warshawer, President
NS Users Group
of Metropolitan Washington, D.C.
2407 Brentwood Place
Alexandria, VA 22306
(703) 768-4804

In view of the changing nature of the micro computer world, those of us who remain in the NS 8-bit world are increasingly left out of the mainstream - a fact of which you are all too well aware. Therefore, those of us who still remain feel a certain necessity to help out all we can.

Sincerely,
Alan J. Warshawer, President

INSUA:

I am sorry to hear that INSUA is going out of business. I recently became a dealer for NS. We are specializing in the programming and installation of the Dimension. However, I still own a Horizon.

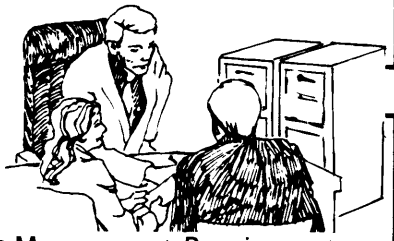
I operate a BBS using FidoNet software. One section is devoted to NS messages. Anyone is welcome to call and exchange ideas, ask questions, get answers, or just look. Any hardware or software topic dealing with North Stars is welcome in this area. I have files for downloading, but none are NS specific. If someone gives me a file or program, I will let others use it.

For now the board is part-time, from 11 p.m. to 9 a.m. daily plus Fridays 5 p.m. to Saturday 6 p.m., and Saturday 11 p.m. to Sunday 1:30 p.m. The number to call is (312) 262-8959.

Sincerely yours,
Daniel Stuhlman
Chicago, IL

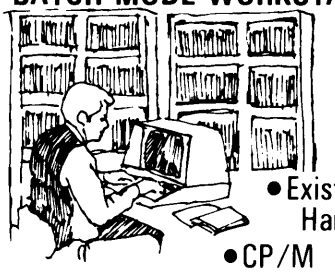
Business Opportunities IN THE X/ACT Environment

END USER



- Management Requirements

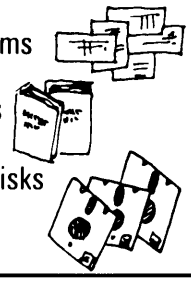
BATCH MODE WORKSTATION



- Existing Hardware
- CP/M

OUTPUT

- Cut Forms
- Reports
- Data Disks



WORKSTATION SYSTEM ENGINEER

- End User Employee
  -or-
- Consultant
  -or-
- Software Cottage

- Operating Specifications
- Code/Debug Programs
- Design Data Load Sheet
- Train Operators
- Software Maintenance

Q BOS^

- Operating System
- Site License

Q BIC

- Basic Dialect
- Interpreted by Q BOS^

$50.00

X/PROTOCOL Write for Free Copy

- X/ACT Workstation Specification
- Q BOS^ User Manual — $20.00
- X/WSESPAK Start-Up Package = Q BOS^ + X/PROTOPAK — $100.00
- X/PROTOPAK Applications Library — None over $50.00
- X/NET Newsletter = Annual Q BOS^ Update Disk — $20.00/yr
- X/DIRECTORY X/CONSULTANT (Names and Expertise) — $20.00/yr

FREE LIST OF CONSULTANTS IN YOUR AREA (Specify Phone Area Code)

ComMarket West

MICROCOMPUTER SYSTEM ENGINEERING SOFTWARE

P.O. BOX 1144, REDLANDS CA 92373

Insua

International NorthStar Users Association
Publishers of The Compass Newsletter
PO Box 2910 • Fairfield, CA 94533

Bulk Rate
U.S. Postage
PAID
Walnut Creek
Permit No. 203

TATE 2761
JAMES TATE
23914 SPRING DAY LN.
SPRING, TX 77373

Postmaster: Address correction requested.
Please enter key number on form 3579.

The Compass

International NorthStar Users Association

Volume V No. 6

Letter from the Chair

ALAS, POOR YORIC....

MAY 22, 1986

As most members are aware, INSUA has closed operations effective December 31, 1985, but has continued some things into 1986. We hope you have enjoyed these last issues of the Compass, the complimentary issues of Microsystems Journal, and that free POWER!* software package.

Alas, with this issue of Compass, INSUA must make it's final farewell. But with a farewell, comes a reminder to again say thank you to the many helpers and volunteers who helped make it happen for these six years.

The editor suggested that we should name names to assure that the people responsible for INSUA could eventually brought to their just rewards. This I decline to do, on humanitarian grounds as well as with regard for space limitations. Suffice it to page through those back issues of the Compass and appreciate the efforts of contributing authors, and to flip through the disk library while reflecting on the contributions of time and talent needed to bring these to you.

For those many of you who helped make INSUA a success, be you director, coordinator, author, library helper, letter stuffer, printer, zip code sorter, disk copier, question answerer, or member...THANK YOU on behalf of all!

Fare thee well...until we meet again.

Robert Beaver
Chairman, INSUA

LOOKING FOR A USER GROUP?

For those of you looking for a good user group now that INSUA is gone, we recommend the North Star Computer Society, P.O.Box 311, Seattle , 98111.

LOOKING FOR BACK ISSUES OF THE COMPASS?

In the interest of continuing to make back issues of the Compass available, INSUA has made arrangements with Randy Fischer of Fischer Computer Systems, who has agreed to maintain and distribute these. For further information, contact Fischer Computer Systems, 445 Bay Street, Angwin, CA 94508. Phone (707) 965-2414.

* POWER! is a trademark of COMPUTING!

The Compass

The Compass is published every two months by INSUA, the Interational North Star Users Association, P. O. Box 2910, Fairfield, CA 94533.

DISCLAIMER

Programs printed in Compass and/or distributed through the INSUA disk library are offered to INSUA members in good faith. INSUA, however, is unable to guarantee the operation of any of these programs or to guarantee support. Users are advised to test the programs thoroughly for themselves in conditions under which they are to be used. Users who employ such programs in serious business or financial applications must do so at their own risk.

Facts or opinions published about manufacturers and dealers, and all opinions expressed in articles and letters, are the responsibility of the authors, and not of INSUA or the Editor of Compass. INSUA offers the right of reply to members and non-members alike.

Contents

Mind-Machine Relationships

By Peter Midnight

In my years of experience with computers, I have learned a great many details about several different kinds of machines. But I have also begun to see some very basic, underlying principles that invariably determine how we relate to these machines, how we use them, and how we react to them. These are the principles that guide my thinking on such subjects as operating systems, storage media, procedures, and all of the other topics I have touched on in my recent articles. I want to take this opportunity, in the final issue of the Compass, to show you my point of view on this more humanistic side of the computer phenomenon.

Before I go any further, I want to clarify my use of the term "Relationship." What I am talking about is the overall aspect of the interaction between a person and a machine. Any relationship between two people can be discussed in the same sense. But I have found that for many people, the word "Relationship" refers more to the connection between two thinking and feeling beings. I do not want to imply that I think computers have actual thoughts or feelings. But I do see computers reacting to every keystroke and people reacting to everything their computers do. That is the interaction I wish to discuss.

What I find unsettling about this relationship is the personality or attitude that personal computers tend to display towards their users. I'm sure we have all seen computers behave in a manner that is authoritarian, condescending, and most of all stubborn towards people. But I think it is important for us to remember that these dominant attitudes that we see expressed are not those of the machine itself, but rather those of the people who set up the machine to perform in this manner. It is the programmers, managers, and marketing executives, who define how these machines will behave, that I wish to criticize. And my criticism of these people is not that they are authoritarian, condescending, and stubborn. I will criticize them for that elsewhere. My point here is that they see their products too much as their children and fail to see the importance of portraying in them a much different personality than their own.

Subservience

The relationship that I believe is appropriate between a person and a machine is that of a master and an ideal slave. The slave may be treated with respect and is certainly worth caring for. But there is never any question of whose whims are to be satisfied, whose needs are necessarily to be met, or who is to be the judge of what actions are appropriate or desirable.

Only the master may impose arbitrary restrictions. The slave actually imposes many restrictions and will do only what it can do. But these restrictions result purely from the limitations imposed by the very nature of the slave and not from any decisions made for any reason by any one other than the master. For example, the difference between an arbitrary decision and a natural limitation is the difference between 55 miles per hour and 186,284 miles per second.

In my experience it seems that almost every machine we use in our daily lives is completely subservient with the glaring exception of our computers. Our cars will go as fast as they can when we tell them to, often faster than is legal or safe. Our telephones will connect us with almost any other phone in the world whether we can afford it or not, even to areas where no one speaks our language. And our microwave ovens will apply any amount of the power they have for almost any amount of time to whatever we put in them, dead or alive. But if I wanted the computer I'm writing on to add the two numbers in the preceding paragraph, I would have to create a whole new program in order for the machine to do it. That's like having to lengthen the gas pedal on the car, add a long distance dial to the phone, or install a back door on the microwave oven.

User Friendliness

Personal computers have not always been as inflexible as they are today. In the early days, each computer had a monitor program. This program gave the user complete control over every cell of memory and every I/O port in the machine. And programs, whether loaded from paper tape, audio cassettes, or North Star's new Micro Disk System, usually came with a source code listing. If you wanted them to run a little differently you could change them. You were the boss.

Unfortunately, this did not make computers easy to use. It was like having to give your car the proper firing order and timing each time you wanted to start it. Everyone recognized the need for simpler and easier procedures for making commonly useful things happen. Commands were added to the monitors for more and more complex functions. Automatic startup facilities were added. And pretty soon it was possible to turn on your machine and have a menu appear on the screen. A single keystroke would then make it do any one of the half a dozen things it knew how to do.

But when people started setting up computer systems to be easy for other people to use, something very important was often overlooked. The computer had been transformed into a machine that could do only half a dozen things. The market accepted these crippled machines because so many people were afraid of computers and would only have them to solve a few business problems. But the self driving car we've all dreamed about in science fiction cannot replace the old fashioned cars of the '80s if it has only a handful of destinations it can ever go to. Even the new and improved model with a hundred or a thousand destinations is worthless without manual override controls. The slave that requires the master to completely specify the job at the time of purchase is not adequately subservient.

The logical extreme of user-friendliness has yet to be reached. Let us hope that the MacIntosh is as far as it will ever go. When that machine was first delivered, it contained one of the most powerful processors that had ever been put into a personal computer. And yet it was not capable of doing anything at all that was not on a pop-up menu. To make even the slightest change in any program in the MacIntosh, you had to buy the even more powerful and expensive Lisa.

Imagine how hard it would be to order a pizza if your telephone were as user friendly as a MacIntosh. It would have a system of pop-up menus so vast that with only a sequence of simple mouse movements you could pick out the most desirable one of the millions and millions of little icons representing each of the other telephones in homes, schools, factories, offices, radio stations, hospitals, airports, yuppiemobiles, stores, theaters, amusement parks, hotels, gas stations, military bases - and yes, even pizza parlors.

Simplicity

Fortunately, your telephone still works the old fashioned way that was developed before computers were available. And you can be talking to the pizza man in seven keystrokes or less. In fact, it may well have been the absence of computers that allowed the telephone to become so utterly subservient. It was not practical, when the phone system was developed, for it to try to help you find which other phone in the world you should be connected to. Instead you have to make that decision for yourself every time you pick up the phone to make a call.

Ordinarily, when I decide to order a pizza I have a pretty good idea of what I want. And sometimes I know where to get it. But often I have to consider the options available to me and the taste of the person who might help me pay for the pizza. Eventually I reach a decision and am prepared to make the call. That is about as much work as I should have to do to select a pizza.

But if my phone were more sophisticated in the ways that my computer keeps trying to be, there is a second mental exercise that I would have to perform. Having decided who I want to call, I would then have to make up a way of describing a preference in pizzas which would lead my phone to the same conclusion. Often this description would have nothing to do with the reality of my desires but rather with my understanding of the information available to the phone and the algorithms by which it was trying

to relieve me of the effort of making a decision. It might even take several tries to get the phone to match my choice, if it could ever be done at all. That is certainly more work than I should ever have to do to select a pizza.

Every time you make a call on one of today's unsophisticated phones you must specify very succinctly which other phone you have decided to call. Unless you use a slave to dial your phone, this may not be as simple an act as pressing a single button marked "Pizza." But it is conceptually much simpler than having the optimum pizza parlor selected for you in some mysterious manner that may not quite keep up with your changing taste. And it is this conceptual simplicity that makes your telephone so much easier to use than your computer even though the phone system is actually vastly more complex.

Your car is another excellent example of simplicity. There are many subtleties to the balance and geometry of a car that few of us understand. But we all understand what the steering wheel does. To the extent that you turn it, the path your car tries to follow becomes a curve. It does not matter where you are, which way you are going, or whether or not the car could possibly follow that path at its present speed. The circumstances and the consequences are irrelevant to the steering wheel. You are the boss. The conceptual simplicity of the steering wheel is the reason that you can steer your car to any place in the world that is accessible by road from where you are. And it's so easy that you barely know you're doing it.

Spectrum of Detail

Has all of our progress in personal computers served only to make them worse? My first personal computer was conceptually simple in the extreme. It just had a great many lights and switches on the front. With the addition of a monitor program it was still very simple. And with an assembler it became relatively easy for me to write programs that would flash those lights in any sequence I wanted. That computer was flexible and easy to use. But to make it do any part of what this one is doing as I write was a painstaking process indeed.

It often seems that the only way to make a computer completely subservient is to program it down to the smallest detail in assembly language. But is that really subservience? Your ideal slave would be willing to back the car out of the garage at a single command, even though the actual procedure might involve hundreds of tiny, little steps. You would not even have to tell the slave to open the garage door first. Besides flexibility, we want our machines to have the power to expand a simple command into all the required steps to perform a very complex procedure.

On the other hand, you might wish to tell the slave whether or not to leave the motor running, whether or not to close the door, or where to leave the car. Your command to the ideal slave may be very simple or may be very complicated, depending on the circumstances. Complete subservience must include the willingness to accept commands in whatever level of detail best expresses your desires. (The best example of this type of subservience that I have seen in a computer is FORTH.)

Programming

The act of telling a computer what to do is called programming. And if any computer has ever responded to anything you did, then you are a programmer. Even if all you ever do is fill in the blanks in forms that are presented on your screen, you are programming. By the way you position the cursor, you are telling the machine what to do with each item of data. And with the return and backspace keys you tell it how to interpret your other keystrokes. Any sequence of these commands that you enter in an appropriate order to get the desired result is a program.

It does not make any difference what language you use to express yourself to the machine, whether it is COBOL, a sequence of menu selections, or a good swift kick to the disk drives. All of these languages have their strengths and weaknesses. But they all have the same purpose: they give you some measure of control over the behavior of a machine.

There is a common misconception that programming is necessarily difficult and that it would be valuable to be able

to use a computer without having to be a programmer. Most of the people who design and sell computers seem to believe this. And they spend a great deal of time and money in an effort to make their products appear to run themselves without depending on the user for guidance. But as a user you know that you always do have to provide some guidance. And as a programmer you know that most of what you do is really very easy and that the situations that are the easiest to deal with are those in which you have the most control.

You also know that if any computer really could do your job by itself or run your business by itself, you wouldn't have a job or a business. But you are not seriously threatened. It is only you that makes your work happen, not any of your tools. And it is only you that can tell your computer how best to help you do it. You will never need to know all of the details of all of the languages that are running in your computer. But if your computer is to be of any use to you, only you can be its ultimate programmer.

Responsibility

What we are looking for is power and authority over our own machines. And we must remember that with power and authority in any aspect of life comes responsibility. When you dial a wrong number on your telephone, it is your responsibility to determine that something is amiss, to find and correct the mistake, and to pay for the call or arrange for a refund. Every mistake you make with your computer should be as minor an inconvenience.

To the extent that your computer is subservient to you, it will do what you say. If you would rather have it do what you meant, then you would rather have it be subservient to some other person who may be good at guessing what you mean by what you say. But don't forget how profoundly stupid a computer is. Even if you could buy your programs from someone who really did know more than you about your own needs, those programs still could not implement judgement that could hold a candle to your own. The only way you can get your computer to serve more of your own needs than of the needs of the people you bought it from is to accept the ultimate responsibility for the operation of your own machine.

And what if you have an employee, a subservient person who operates your computer for you? To whom should your computer be subservient in that event? Now you are setting up a computer to be used by someone else. And you may be poised to make the same mistakes as all of us who came before you. It is so tempting to program a computer to force a person through a procedure like a rat through a maze. But to use a person as a peripheral to a computer is a tragic waste. No person is so stupid that they cannot serve you better using your computer as a tool. I have come to firmly believe that when responsibility for any process is laid on a computer instead of on a person it is always misplaced. Only a person can be responsible for the operation of a machine.

Layers of Abuse

Unfortunately, there is one situation in which a machine really must refuse to be subservient to the person who is using it. When a slave must serve more than one master, it really cannot be totally subservient to any of them. It can act as though each user is the boss as long as the users are all very careful not to give it conflicting orders, such as to sell the last remaining widget in stock to each of two customers. But to avoid any unpleasant surprises, the slave with multiple users must acknowledge that in fact no more than one of them is its master and that all of the others are potentially hostile intruders. For this reason computers that serve multiple users, either one at a time or by time-sharing, are almost universally set up primarily to restrict the users to those operations that the system operator considers harmless, rather than to serve the users in any way they may desire.

This type of guarded, limited cooperation may be appropriate for a machine that sits unattended in a public place selling soft drinks. It is appropriate for a vending machine to have as many users as it can attract. And, being built like a vault, it makes no pretense that any of them is its real master. But a computer that behaves in this manner is impersonal and cannot be as valuable a

tool for the user as would a personal computer in the same position. This is why networks of computers are so superior to time sharing systems. It isn't just their greatly increased processing power. It is that only the network server computers must be limited in their capability. Each user still has the full potential of a personal computer for his or her own use.

Any machine or program that is dedicated to restricting the user, either to protect other users or in an egotistical attempt to protect the user from his or her own mistakes, is obviously inefficient. Before complying with each of the user's orders, such a system takes time out to test and measure that order to ensure that it falls within the limits prescribed by the person that is its real master.

In typical applications of personal computers, this inefficiency is compounded several times. The assembler restricts the authors of the operating system and the development language. The system and the language restrict the author of the application program. And the application program restricts the end loser. This is how a machine that performs literally millions of operations per second can sit around for several seconds just scratching its virtual head before it finally has the gall to ask you for the time of day.

Dinosaurs

Many of us grew up in the days of the dinosaurs, room-filling computers called "IBM machines." Those monstrosities had to have multiple users because no single person had enough money to own one. And many of us grew up with the idea that a multiple user environment was a reasonable way for a computer to relate to its users, because in those days it was the only way. But, fortunately, times have changed. Computers are now smaller, cheaper, and less important than you are. And there is no longer any excuse for them not to act like it.

When I first became seriously involved with computers, the personal computers were easily distinguishable from the dinosaurs by their size. And I was advised by those in the know at the time never to trust a computer that I could not lift. But the distinction has now become much less clear. Most readily available computers these days are no more than miniaturized dinosaurs. They look like personal computers. But they act like IBM machines.

In fact, some of them literally are IBM machines. It seems to me very ironic that the first company to use the term "Personal computer" as its product name has also been a major driving force behind the effort to make computers impersonal. But it should not be a surprise. A properly subservient computer makes a dinosaur almost completely obsolete. What else could that company do but make a series of desk top lizards that everyone would buy for the name on the front, make the public believe that these were personal computers, and show the public that these machines would serve them no better than any of the company's other computers?

Advice

There is still hope, however. When you shop for computers and especially for software, it is still possible to find products which are designed more to give you control over your own tools than to give you what the designers presumed to know you should want. When a salesman shows you how something works, always ask him or her how to make it work a little differently. Or ask how you would transport all the data you had entered into that system into a newer and better system, when it becomes available. The answers to those questions will always show you something you need to know about either the product or the salesman. Be patient and wait for a good answer.

My advice to you is not to trust every computer you can lift. Don't ever let a mere computer usurp your authority to decide what you want. Always find out whose rights are protected and whose interests are served by every machine and program. And never trust a computer that is not entirely dedicated to serving you.

# # #

Dear Joe:

By Joe Maguire

Here are some more answers to questions sent to me by INSUA members.

Dear Joe: I am trying to use an additional RAM board to increase available memory in my computer but I'm having problems. In order to disable the RAM at the address occupied by the disk controller I pulled out the memory chips for that area. The board still doesn't work. Why not? E. B. Cardiff, CA

Dear E. B.: The reason is that almost all memory boards are "buffered." That means that the data bus does not actually see the contents of the memory cell but the state of the latching flip-flop in the isolation IC.

The isolation IC (a typical example is the 74LS373) has three states: high, low, and off. The high or low state is set by the contents of the memory cell but the on/off state is set by the address decoder circuitry on the board. If you just pull out a memory IC from the board, that only means that the buffer IC will assume some unpredictable value of high or low. When that area of the memory board is addressed, that value will be passed to the data bus just as if it were a valid memory bit.

The only way to disable a particular memory area is to decode the address and use that information to turn off the buffer IC. That usually means adding additional circuitry to the board or using a board that has "windowing" capability. With most newer boards you can select windows. And now the cost of memory is becoming so low that it just isn't economical to jury-rig old boards any more.

Dear Joe: In a previous article you said you were hoping someone would come up with a way to increase a North Star memory board to 256K. Did anyone ever send you that information? J. H. Santa Rosa, CA

Dear J. H.: No, but in the September, '85 issue of Byte p. 247, there was an article about modifying the Atari 800XL for 256K that looks like it could be adapted to a North Star board. Two different schemes were given that would seem to apply to either North Star's older boards or to the newer HRAM.

Dear Joe: I finally got around to using your Phantom article (Compass Vol 4, No. 1, p. 5) to increase my memory board to 64K but now I have another problem. I reconfigured my CP/M for 64K and I am getting all sorts of strange results. Some programs work and other don't. What's wrong? J.W. Ft. Ashby, WV

Dear J.W.: They gotcha! What some vendors of CP/M fail to make clear is that their implementation requires a continuous block of RAM. No PROM or missing memory can exist in the area which will be used by the program. When you increased your memory board to 64K, you in reality have only 63K. One K is still occupied by the disk controller. That 1K is 5K below the top of your memory and when you configured your CP/M for 64K, that put it smack in the middle of your CP/M's CCP module! Some programs use the CCP (Console Command Processor) and some don't. That's why you got mixed results.

What you must do is configure your CP/M as a 58K, or less, system or obtain a version which can "jump over" the disk PROM. North Star's version of CP/M can do this but others, such as Lifeboat's, must be modified by the user, not an easy task. Ask your local North Star dealer for more info.

Dear Joe: I have been a North Star owner for many years and we use several of them in our business. I have been under increasing pressure by some of our employees to "upgrade" to newer equipment, i.e. IBM. In your opinion, what is the future of North Star and its products?

F. S Honolulu, HI

Dear F. S.: Wow! You don't want me to stick my neck out much, do you! However, considering that INSUA is disbanding, this question deserves an answer.

In my opinion, North Star's products have been, and still are, of the highest quality and among the best performers in the industry. I say this from ten years of experience with North Star's products and as a former part owner of a computer store.

In our store, we saw thousands of different pieces of equipment from dozens of manufacturers come through. North Star's products, and the Horizon in particular, were always our favorites. I have had personal experience with well over 100 Horizons and I consider it the finest S-100 machine ever designed. (A close second is the Processor Tech Sol but the Horizon outperforms it.)

If this is true, why then has the Horizon, the Advantage, and North Star been bypassed by the customer?

Advertising. North Star was not nearly aggressive enough in their advertising. They had the jump on Apple in the education market but failed to exploit it. Ask any educator what the system of preference was at the university level in the late '70s and he will tell you--the Horizon.

A number of years ago, Interface Age ran an evaluation of some fifty different computer business systems. The evaluation lasted over a period of several years and was extensive. Each system was evaluated under conditions in which it would actually be used. That meant the computer with all the required additional equipment: terminal, printer, etc. and multiple business software programs using realistic data.

The Horizon came out as the number one system in performance and second only to the Apple in lowest price.

As configured by Interface Age, the Horizon system was priced around $4,000. It beat out other systems costing up to $30,000.

As I recall, North Star used the results of these tests in their advertising only a few times.

In fairness to North Star, they had a difficult problem. The Apple was cheap to produce but was selling at a premium price. The Horizon was more expensive to manufacture but North Star's management tried to keep the price competitive with the Apple. The difference had to be made up somewhere and they chose the advertising budget. It was a costly mistake.

But to get to the heart of your question, what do you do with your North Star equipment now that everyone is jumping on the IBM bandwagon?

The answer depends on your computing requirements.

If you are using your North Star as an accounting system in your business, it's doubtful you will find anything to outperform it.

If you are doing mostly word processing, stay with your North Star.

Look at what Malcolm Rubel had to say about WordStar 2000 running on an IBM PC in the September, '85 issue of Byte.

"In all comparative tests (with WS 3.3), I found a substantial reduction in performance. The screen-rewrite time is so slow that when you hold the down-arrow key to scroll through the file, the display goes blank as you move off the current screen."

So much for high performance with an IBM PC! Show that review to your IBM-user friend the next time he scoffs at your North Star.

The major problem area is not with hardware but with the software. The newer, exotic software is just not being written for use with CP/M, or the Z80, anymore.

Graphics is the one area in which the Horizon, and to a lesser extent the Advantage, is falling behind the newer generation of personal computers.

If you must have pie charts and bar graphs in sixteen colors, if your spreadsheet must flash at you like a neon sign, or if you are into computer aided design, then perhaps it's time to retire your North Star.

At least one reader came up with a somewhat better solution. He wrote to tell me that after he bought a Macintosh, he now uses his Horizon as a huge print buffer.

There are other alternatives. As an S-100 machine, the Horizon can be made into practically any computing system you desire simply by changing boards. Several manufacturers (among them, Viasyn and Lomas Data Products) are offering sets of S-100 boards which will turn your Horizon into an IBM PC lookalike. (Lomas claims a 3 to 1 increase in performance over the IBM PC.)

As to the future of North Star and the guarantee of support for your present equipment:

That seems to depend on the success of the Dimension. The Dimension is IBM PC compatible and truly state-of-the-art in design but that is not enough in today's market.

It matters not what the equipment is, but what it is perceived to be by the customer. Unfortunately, North Star seems to be making the same mistake with the Dimension as it did with its earlier products.

I hardly receive a magazine in the mail these days that doesn't contain an ad for IBM, Apple, Atari, Commodore, or a dozen others, but I haven't seen an ad for North Star in more than two years!

If North Star doesn't join in with the rest of them, it's only a matter of time, in my opinion, before they slip quietly out of sight. One of the first things to go will be support for older equipment since that is only marginally profitable at best.

My advice is to get the most out of your present equipment that you can. Be alert that the advertising hype for some of the newer systems is just that--hype.

Eventually, change will be required, just to gain software or disk compatibility; but when that time comes, look carefully--so that you don't lose performance instead of gaining it.

If you want my opinion as to a company to watch, look at Tandy. They have come a long way since their "Trash"-80. Their newer equipment offerings are consistently on the edge of technology. If I finally decide to go IBM compatible, a Tandy would most likely be my choice.

North Star has had a fine reputation in the personal computing market and someday their products will take their place in history alongside the Altair and the Sol.

* * *

Author's Note:

Since INSUA is closing down The Compass, this will be my last article. I wish to thank all those readers who wrote, called, or otherwise got word to me that they enjoyed reading my articles--and even used some of the information!

It is the audience response which sustains an actor--or an author--and you have been terrific! Thank you all very much.

# # #

Programming in N* DOS

By Peter Midnight

This one started out as "How to Write a Program for a North Star Computer." But it doesn't take long to figure out that the ins and outs of programming for CP/M, PASCAL, and FORTH systems are already covered in great detail in all the popular literature on personal computers. This leaves only North Star DOS to be the private realm of the very hip. And it doesn't take much longer to figure out that the DOS is also the way to realize the fullest potential performance and versatility from either a Horizon or an Advantage. Therefore, although some of the following information may be useful in other systems, the DOS will be the primary focus of this article.

The first question that is likely to come to mind is what system you will use to develop your new program. This should really be one of the last questions you ask yourself. But it might also be one of the

first to get answered. So let's consider it here. The answer is that it really does not matter. Use whatever you are comfortable with. I like to program in assembly language and have a Z-80 assembler of which I am particularly fond. That assembler runs under CP/M. But moving the code it produces into a DOS file is no problem, as you know. You may have your own favorite assembler. Or you may prefer some sort of compiler, such as a PASCAL or C system. And, of course, for BASIC lovers, North Star BASIC would be the obvious choice. Almost any system you choose to use will provide you with the means to produce a program that will run under North Star DOS.

Portability

Now before we go any further, let's consider the question of portability. Those of you who are already accomplished programmers will please bear with me while I explain the importance of this. Even when you write a program that is strictly for your own use, it is wise not to build into it any more specific hardware requirements than are really necessary.

For example, let's suppose you have written a hang-gliding simulation game. You might play this game by sitting on your mouse and wiggling your tail, as you would in a real hang glider. This program would need to input data from the mouse continuously while at the same time moving mountains of data, or, at least, data of mountains. Let us further suppose that in your quest for speed you have allowed this program to suck directly on the serial port to which your mouse is connected. What would happen if your new packet radio controller had a hard-wired port address that required you to readdress your serial ports before you could install it? You could jump off Half Dome into Yosemite Valley and suddenly find that your tail doesn't work any more and you have only a couple of minutes to debug the program before you hit the rocks! You could try screaming at the author. But when the author is you, even screaming won't make you feel much better.

On the other hand, if you always input your mouse data through the character input routine in your DOS, you have a better chance of retaining control. When you changed the address of your serial ports, the only code you would have had to adjust would be the I/O routines in your DOS. The same corrections you made to keep your word processor and spreadsheets working would result in the hang glider also continuing to function properly. In addition, you could more easily share your program with other North Star flyers, even if their mice were not connected through serial ports, at all.

A Little Background

Now you were probably already familiar with the importance of doing as much as possible of your I/O through the DOS. This was, in fact, the original reason for microcomputers having operating systems, at all. The need for standardized I/O routines preceded the availability of disk systems. The Disk Operating System is an operating system that includes access to disks, not just a system for operating disks.

Originally, each microcomputer had just one terminal. If hard copy was needed, that terminal would be a teletype. (That is probably why the output statement in BASIC is called "PRINT.") And if storage was needed, the teletype included a paper tape punch and reader. Fortunately, times have changed. First printers and disk systems were added. Then modems became very popular. And more recently it has become fashionable to have a mouse, track ball, touch pad, or some other sort of graphic input device.

I/O Device Numbers

Some operating systems have a specific set of routines for each of these devices. But the more modern or adaptable systems allow for the fact that the times continue to change and some room for expansion is necessary. CP/M loses on this point. MSDOS allows for additional device drivers to be linked into the system at boot time. The provision for expansion in North Star DOS is the use of I/O device numbers. A device number in the range of zero to seven is passed as a required argument to each of the four, basic, character I/O routines, CIN, COUT, ISTAT, and OSTAT.

The key to understanding device

numbers is the fact that they are completely arbitrary. They are not related in any way to any I/O port addresses which may be used by the routines for each device. In fact, most devices are accessed through a pair or more of port addresses. And some devices, like video boards, use no I/O port addresses, at all.

Most of the eight possible device numbers still have no meaning in most systems. But zero always means the main terminal. (This is sometimes called the "Console", for historical reasons. Computers used to look like nuclear power plants. And they usually had a big control console, where the main terminal was located.) And 1 normally means the main printer. Beyond this there are no universal standards. Even the printer is often 2 instead of 1. But even so, simply specifying the I/O device numbers that represent the printer and the modem in your system would have been much easier than what you had to go through to install MDM712.COM!

Relocation

All of you users of Horizons and other S-100 systems are familiar with MOVER. For the Advantage users, let me explain.

Horizons have widely varying amounts of memory, from as few as 16 up to several hundred kilobytes or more. In addition, the disk controller, which is not memory, also appears in the memory map of these machines. Because of the flexibility of these machines, the arrangement of the DOS and other programs varies from one machine to another. Often the arrangement even varies from time to time in the same machine, depending on how it is being used.

Originally North Star offered special versions of all of its software to run at different addresses according to a customer's specifications. But that was expensive. And it was often difficult to explain to a customer what was possible and to determine what he or she wanted. As the quantities of software and of customers grew, that approach rapidly became impractical.

Eventually North Star released a program, called MOVER, which would allow any customer to produce any possible relocated version of all of the software on any factory master, DOS diskette. This procedure cost the user nothing but time. And it allowed the users to determine for themselves what would work, what would not work, and what would best suit their needs.

Finding the Elusive DOS

The device numbers are just the first step in the direction of portability. They allow one program to run on several computers or on one computer that gets changed and improved from time to time. But to use them, the program must still know where in memory the DOS is located. And with MOVER, the location of the DOS is among the aspects of the computer that the user can change. If the program is maintained in such a form that MOVER will move it, along with BASIC and the rest of the utilities, it is reasonable for a program to assume a specific location for the DOS. This assumption will be adjusted by MOVER whenever the program is relocated.

This works just fine, as long as you keep each copy of the program associated with the corresponding copy of your DOS. But you might also want a version of your program that could figure out where the DOS was each time it was loaded. Such a program could be passed around among your friends in a form which would run on both Horizons and Advantages, without the help of MOVER.

Some programs have been written which actually scan through memory in search of something that looks like the DOS. I shudder at the thought. A MOVER enthusiast could easily have several old copies of his DOS lying around in memory at any given time. He might not even know which one such a program had found to call on for its I/O. It may well be an old one in the memory he will now be using for data. After a good eight hours of typing, he may overwrite his own input routine, causing the machine to take a vow of silence at the most inconvenient of times.

You probably know about the command line pointer that programs like CD and CF use to find the arguments on the command line that invoked them. The GO and JP commands in the DOS and the

JP command in the MONITOR always pass a pointer into the command line buffer in the HL registers of the processor. And you may also know that these same commands always leave a valid return address on the processor stack. This allows any program to terminate gracefully by restoring the stack pointer to its initial value and executing a RET instruction. You might assume that your program could determine the location of the DOS from either the return address on the stack or the address of the stack itself.

That was a very good guess. And, in fact, this technique is mostly successful. However, there is an easier and more universal solution. Both the DOS and the MONITOR also pass the high byte of the base address of the DOS in the accumulator. Any program can use this value to adjust its own, internal jump table and any other code which is dependent on the DOS origin.

Writing for MOVER

Being able to relocate any DOS software that came from North Star is all well and good. But if North Star were the only source of software, there certainly would not be much available to run under the DOS. Therefore, one of the primary purposes of MOVER was to allow anyone who chose to do so to produce software in a form that MOVER could relocate.

Most software development systems produce some sort of intermediate, relocatable object format, from which the final code can be generated to run at any specific address. But there are so many different sets of standards for these formats that there is, in fact, no standard. MOVER could not have been written to decypher all of the different relocatable formats that would ever be developed. Instead, it gleans all of the data it needs from something that any development system can produce, two complete copies of any program that are identical in every respect except for their origins. And the first of these two copies will always be ready to run for any one who does not wish to relocate it at all.

By making a comparison between the two copies, MOVER classifies each byte as either absolute or relative to the location of the program, the location of the DOS, or the location of the disk controller. Therefore, the second file that you provide as relocation information for MOVER must be identical with the first except that all three of those locations are assumed to be zero. If you are comfortable with the use of whatever linking loader your particular development system may include, you may be able to make it generate both of the files for you. Or if all else fails, you can at least define whichever of the three key locations your program may refer to as symbols at the beginning of your source code. You can then produce the second file by changing those three or fewer definitions to zero and running the entire compilation or assembly again. But in either case you must ensure that any undefined bytes in your program come out the same both times to avoid confusing MOVER.

All of this cleverness would have accomplished nothing if the requirements of MOVER had not been made clear to independent software developers. Therefore, MOVER was written to reveal its requirements on three different levels. First, when you run it, it tells you about the unprotected copy it needs of the diskette it is to process and leads you through the process of specifying the origins you desire. Second, you can LIST it because it was written in BASIC. When you do you will find REM statements that go into greater detail. Finally, you can read the code that performs the actual relocation to clarify your understanding of exactly what it does.

I would urge any one who writes for the DOS to take a moment to generate the one extra copy of their work that is needed by MOVER. Even if the program is only for your own use, the extra file will allow you to rearrange your system quickly and easily if you ever choose to do so. And when you install specialized I/O routines in your DOS, if you also install the relocation copy of those routines in the relocation copy of the DOS, then MOVER will produce a relocated DOS for you that is already personalized with your own special routines.

I/O Block Expansion

So far, I have mentioned several things you might add to the I/O routines

in your DOS. And with a little imagination, you can think of many more. But, as you know, the space available for these routines is only 256 bytes. You may be wondering how to make more space available, just in case you get carried away.

The key to making more space for your I/O routines is the TINIT routine. And several good techniques have evolved around its use for this purpose. Because TINIT is called once, when the DOS is first loaded and before it does anything else, it is a perfect opportunity to call DCOM and get some more stuff loaded into RAM from your boot diskette. This stuff can include the other I/O routines, and even some more of TINIT itself.

If you are just a little bit short of space, TINIT might copy the other routines from their temporary location outside the DOS into the I/O block inside the DOS, overlaying the first part of TINIT. A single-density sector could even be read directly into the I/O block. But this may be undesirable in a double density system, for reasons that we will get to shortly.

If you need even more space, you'll just have to find a place to put it. Any RAM you may have above your disk controller might be a good place and is usually an excellent place to put the whole DOS. The upper and lower limits of memory are o.k., but you need to be careful there to avoid conflicts. Finally, you might use MOVER to slide the whole system up a bit and tuck in the extra stuff where your DOS origin used to be. If your I/O routines are truly enormous, you might get into bank switching. But that is beyond the scope of this article.

Storing Extra I/O Routines

Once you have found space for the extra routines in RAM, you must also find space for them on your boot disk. A good programmer's first inclination might be to put them in a file, just like anything else. But this brings up an important secret to making this whole thing work. You must not call DLOOK from TINIT.

Because you can't use DLOOK, your extra routines must be stored at a fixed location on the disk, just as the DOS itself must be in order to boot. You could make a point of always putting the file of extra I/O immediately after the DOS on every boot disk. But because it is so intimately linked with the DOS, it might as well be a part of the same file.

If you save your DOS in a slightly larger file, you will effectively reserve the disk space immediately following the DOS without affecting the way the DOS boots itself up. By saving your extra I/O routines in that reserved space you can ensure that they will never be accidentally separated from the version of your DOS that uses them.

The advisability of saving the DOS and the extra I/O routines together in a single file is the reason you would not want to use a single density sector with a double density DOS. Mixed densities are not allowed within files.

Renaming Files

Now here's a peeve I've mentioned before. People have often disparaged the DOS in the belief that it has no facility for renaming files. But DLOOK reads a portion of the directory into RAM and returns a pointer into it. And DWRIT writes that portion of the directory back to the disk. This facility allows you to change anything at all about the file, including its name. In fact, that is how you delete a file, by changing its name to blanks.

What the DOS does not do is protect you from your own creativity. To make everything work in the normal manner, you have to put each file name in the directory left-justified in a field of ASCII blanks. If you give several files on a disk the same name or give one a name you cannot type, you may only be making trouble for yourself. On the other hand, you may be making a new and better use of your computer that no other operating system would allow.

Creating Files

The procedure for creating files is just a bit more complicated than the one for deleting them. This is because you must find both a place on the disk for the file and a place in the directory for its directory entry. To find both you must call DLOOK twice.

On the first call, you pass it the name of the file you are about to create.

If a file by this name is found, you need to do something to avoid ending up with two files of the same name. Even if what you choose to do is to rename the old file, you still need to call DLOOK again with the new name and have it not be found. Only when DLOOK fails to find the filename you pass it will it find and return the disk address of the space that is available on that diskette for your file.

On the second call, you pass DLOOK a pointer to an ASCII blank, instead of a filename. This causes it to find the first directory entry in which the name is blank. In other words, this is how you find an unused directory entry to use for your new file.

The two calls to DLOOK must be made in this order because creation is completed with a call to DWRIT. And there must be no other disk activity between the call to DLOOK that gets you the directory entry to use and the call to DWRIT. What does happen between these two calls is that you fill in all sixteen bytes of the directory entry for your new file. And to do this you must have already obtained the starting disk address for the file from the other call to DLOOK.

Let's take a moment here to review the structure of the directory entry. The first eight bytes contain the filename, as described above. The next two bytes contain the starting disk address as a sixteen bit, binary number. And the next two bytes contain the number of sectors allocated to this file. (This is the amount of physical space reserved on the disk. The file size or the amount of data that the file can hold is dependent on both the physical size and the density.) The next byte contains both the file type number in bits zero through six and the density flag in bit seven. The remaining three bytes may be used in any manner you like for whatever file types you may define. For type one files, the first two of these bytes contain the GO address and the last one is undefined.

Again, you are not protected from your own creativity. What is described here is only the normal way of doing things. But it is not unusual to create a file at a predetermined disk address for a special purpose. For example, the ¼*½ file (often pronounced "Splat") contains the directory itself, starting at disk address zero. When creating special purpose or overlapping files, the first call to DLOOK may serve only to determine whether or not the new file has already been created, or that call may not be required, at all.

Finding Available Disk Space

One additional check is normally required during the process of creating a file to ensure that the file will fit inside the remaining space on the diskette. To perform this test, your program must know the size of the diskette on which the file is to be created. However, it is not practical, nor necessarily desirable, to determine the size of a DOS format diskette.

All other operating systems that run on Horizons and Advantages have different diskette formats for single density, double density, and double sided diskettes. And each diskette that is formatted for use with one of these other systems bears a symbol, usually somewhere in sector zero, that tells the corresponding operating system which of its diskette formats to use on that diskette. But the DOS is unique in that it has only one format. The format of the first side of a double sided DOS diskette is identical to that of a single sided DOS diskette. The second side is just more space available for files.

Whether or not any diskette is double sided is much less important than whether or not the disk drive it is mounted in is double sided. Every diskette actually has a second side, although it may not be initialized or guaranteed flawless. But single sided drives really have only one head. If you try to access the second side of a diskette in a single sided drive, it will read from or write to the first side of that diskette without giving any indication that it did not do what you told it to do. That is why it is so important for the DOS to be configured correctly when used with any single sided drives. DCOM uses this configuration data to determine which range of disk addresses are legal and safe to use. If it knows a drive is single sided, it will refuse any call for access to the second side.

Because every diskette actually has a second side but some drives don't, the capacity that matters when you are creating a file is that of the drive, rather than that of the diskette. Prior to the

release of Version 2.1.0 of the DOS, the capacity was found by interpreting the configuration byte in the jump table of the DOS. This information revealed the capacity to be either 350 or 700 sectors. But under the new DOS, you can get the capacity of a drive by calling DCOM with a command byte of 3 in the B register. This call to DCOM is simpler and easier. And if you have installed something that replaces DCOM with another routine which simulates diskettes using some other type of storage, the new routine can give you a correct capacity indication even if it does not happen to be either 350 or 700 sectors.

The starting disk address you got from the first call to DLOOK was one more than the address of the last sector that was found to have been allocated to any existing file. The capacity you get from DCOM is a sixteen bit, two's complement, negative number (-350 or -700), because that is the form in which it has been found to be the most useful when creating a file. Simply add these two numbers to find the negative of the number of sectors available for the new file. If the result is zero then the disk is full. If any of the second side has been allocated but the diskette is now in a single sided drive, then the result will be more than zero, showing that the disk is more than full. In either case, the addition will have produced a carry, which you can take to mean that there is no room for your new file.

Now decrement by one the result of the previous addition and add to that the number of sectors you wish to allocate to your new file. If this addition produces a carry then the result was not negative and you can take that to mean that there is not enough room for your new file. Otherwise, you can proceed with the creation of the file knowing that you will be able to access all of it.

Dynamic Allocation of Disk Space

One practice that is often referred to as dynamic allocation is that of reusing the disk space that is released when a file is deleted. Under other operating systems this is often required by programs which maintain two copies of their data files. For example, many text editors operate in this manner. As each new version of the data is being developed by such a program, the previous version of that data remains intact in a separate file. Because of the difficulty of reusing an existing file under other systems, such programs usually delete the older of the two files and create a new one each time they are used. Fortunately, under the DOS there is no need for any deletions or creations nor for any changes in the allocation of disk space. Simply swapping the names of the two files is sufficient.

There are times, of course, when it would be desirable to be able to reuse the space formerly occupied by files that have been deleted. But the lack of a provision in the DOS for doing this has never been shown to be a serious problem. Even the CO utility, which bubbles up any unallocated space to the end of the disk, is rarely if ever used by experienced DOS users. Certainly a routine could be written to create files by analyzing an entire directory, finding the first unused space that was large enough, and allocating the required amount of that space to each new file. But, to my knowledge, no one has ever needed such a routine enough to bother to write it.

Allocation as an Act of Faith

One of the greatest advantages of the DOS over other operating systems is that it allows you to specify in advance how much disk space will be required by each file. The required amount of space is allocated when the file is created. If the required amount of space for any given file does not remain on any given diskette, you find out that you need to use a different diskette before the file is even created. As long as you can predict the amount of space you will need, you can't get caught in the middle of generating a large amount of data with no place to put it.

Under other systems, disk space is not allocated until you actually write into it. Each file is allocated more and more space, in the smallest possible increments, as the data is being generated, in the hope that you will run out of data before you run out of space.

As foolhardy as this practice might seem, there are times when it is useful. For example, if your modem program has the ability to save all incoming data in a

file or if it implements a file transfer protocol which does not convey the size of a file before it conveys the contents, then you will need to start writing the file to disk before you can know just how big it is going to get.

As any CP/M user can tell you, even if something is written to the disk, it does not necessarily become a part of the file into which it is written until after the file is closed and the directory is updated. What may be less apparent is that under the DOS, it is not even necessary for the file to have been created until that time.

When you want to save an unknown quantity of data under a known name, you begin to create the file by making the first of the two calls to DLOOK with that name. As described above, that first call gives you the disk address from which all the rest of the space on the disk is unallocated. You can then start writing data from that address onward.

You don't even need to test the capacity of the drive because DCOM will make this test for you each time you tell it to write another buffer to the disk. After the last of the data has been written with no error indications, or after you have run out of disk space, the size of the file becomes known. You can then make your second call to DLOOK and complete the process of creating the file, as described above.

Taking Advantage of Hard Sectoring

Please note that you can create or extend a file onto the second side of a diskette that was purely single sided before you did this. This is one of the advantages of North Star's unique, hard sectored disk system. No part of a soft sectored diskette can be either written or read until it has been formatted. But there is no formatting process for hard sectored diskettes. It is only necessary that each sector be written first before it can be read.

The IN command in the DOS may seem like a formatting command. But all it really does is write blanks over all of the diskette that the drive it is in can reach. This is useful when the read after write check is enabled because it does a quick test of the entire surface of the diskette at the best possible time to discover any damage, before you start depending on it. Writing first is also necessary in the directory because you can't use DLOOK on a diskette until all four of those sectors are legible. And BASIC cannot write into any sector of a data file without first reading it. But there is no reason why files cannot be written onto the second side of a single sided diskette just as soon as it is mounted in a double sided drive.

There is a danger here for any one who has any flippy floppy diskettes. These are diskettes that have extra holes in their jackets such that they can be mounted in a drive either side up. North Star never considered the use of such diskettes because they are potentially harmful to disk drives. If you ever have to use one of these diskettes in such a way that you might create or extend any files on it, you should reconfigure your DOS to consider the drive it is in as single sided. This will ensure that nothing on the opposite side of the diskette will be overwritten.

Efficient Use of DCOM

The efficiency of the DOS is perhaps most dramatically demonstrated when you wish to copy an entire diskette. All operating systems seem to come with a utility program for copying diskettes. But most systems are so painfully slow that their own diskette copying programs don't even use them. Instead these programs often bypass most of the system and either manipulate the hardware directly or call the actual subroutines somewhere in the bowels of the system that do so. On the other hand, the CD utility that comes with the DOS simply calls DCOM, the same routine that is used for all other access to the diskettes. And yet it can complete a copy in less time than any other system.

The capability that is required in order to copy an entire diskette in any reasonable period of time is the ability to read or write an entire track in a single revolution of the diskette. DCOM has this capability. And it uses this capability to save time whenever it is called upon to access more than one sector on the same track. How to make the best possible use of this feature turns out to be something of an art. And the ultimate solution, if it were worth the effort, would often involve

calculating where in a given file the track boundaries fall. But a good rule of thumb is that the more sectors you call for in each call to DCOM the more time you are likely to save.

Recognizing the Advantage

Everything in this article up to this point has applied equally to the Advantage and to S-100 systems, like the Horizon. But the Advantage has some unique features which are accessed through its unique version of the DOS. If you are writing a program which may be run on both types of machines, you may want it to determine whether or not it is running in an Advantage before it decides to use any of that machine's unique features.

One of the bytes in the jump table of the DOS contains the high order byte of a page address. A jump to that page address causes the boot PROM to be executed. In an S-100 machine, that page address is the base address of the disk controller board and, therefore, of the boot PROM itself. In the Advantage, there is no memory address for the disk controller. And even the boot PROM does not reside in the memory map during normal operation. But there is still a page address that has the same effect if you jump to it. And that is the address indicated in the jump table of the DOS in the Advantage.

As you know, the controller board address can only be a multiple of 1024. Therefore, bits zero and one of that byte in the jump table are always zeros in an S-100 machine. In the Advantage, the same two bits are ones because the reboot address is the base of the secondary jump table at F700h. Even though the base address of the DOS has always been F800h in the Advantage and can never be F800h in an S-100 machine, this byte is still the only official indication of which type of machine your program is running in or of the location of the secondary jump table when that machine is an Advantage.

Keyboard Proceccor Tricks

Whenever your program finds itself in an Advantage, it can take advantage of the special features of the keyboard on that machine in ways that are not possible on ordinary computer terminals. This is done through calls to the KMODE routine in the secondary jump table.

Among the options that are controlled by this routine are the ALL CAPS mode and the CURSOR LOCK mode. Ordinarily, a typist will expect these modes to be strictly under his or her own control, like the SHIFT and CONTROL keys. And it might be somewhat rude for a program to change them unexpectedly. But there may be times when it is clear which state will be needed, such as when requesting the user to move the cursor or enter numbers. In such cases, a program can make these selections as a convenience to the user if it has found that it is running in an Advantage.

The more exotic option available through KMODE is the alternate coding mode. In this mode, characters are not produced only by character keys and only when they are pressed. Instead, every key on the board, even ALL CAPS, produces one unique code when it is pressed and a second unique code when it is released. This allows your program to interpret the use of the keyboard in very special ways. For example, any key can be interpreted as a shift type modifier of the meaning of any other key. Or the two normal SHIFT keys could be endowed with separate and independent significances. In fact, the user could literally play chords on the keyboard if you chose to program a situation in which they would have some meaning.

Floppy Drive Motor Control

We all know how frustrating it can be when a program accesses a diskette just slowly enough that the motors consistently time out and stop just before they are needed again. Even if you calculate that this is only costing you a few seconds per day, it can still be very aggravating. Some Horizon users face this same frustration when entering a sequence of commands to the DOS. After each command they have only a specific amount of time to type in the next one before the motors stop. And the feeling is even worse if this time pressure causes you to make typing errors.

Advantage users don't face this pressure while entering commands because of the TIKLE routine. Once for each call to the keyboard input routine, the user's

I/O routines provided with the Advantage version of the DOS also call TIKLE. This is an entry point in the secondary jump table which resets the motor timer but does not start the motors. As long as the user is typing, the motors do not stop. If the user has to stop and think, then the motors take a rest, too. And they do not start again until the next time they are needed.

The TIKLE routine can also be called from any program that turns out to be using a disk file at just the wrong rate, as described above. But be careful not to use it in such a way that it keeps the motors running all the time, even when they are not needed. Remember that the TIKLE routine was intended to help you to reduce the strain on your hardware, not to increase it.

If your program is not strictly restricted to running in the Advantage, then you may need to write a routine which determines which machine it is in, as described above, and then does the best it can do under the circumstances to keep the motors from stopping. In an Advantage, of course, it would just call TIKLE. But in any S-100 machine, the problem gets a little trickier.

How to TIKLE a Horizon

Unfortunately, there is no TIKLE routine in the Horizon version of the DOS. But here is a routine that might work for you: 3A 10 00 E6 10 C8 3A 90 00 E6 10 C0 3A 18 00 C9. Note that the disk controller is influenced only by memory reads. Therefore, this routine will have no effect unless the three zeros are changed to the base address of your disk controller plus three. In fact, this routine would be harmless if accidentally executed in an Advantage. Any program which contains this routine can correct the three zeros when it is executed by adding three to the value it finds in the jump table of the DOS and storing the result in this routine. Or these bytes will be adjusted by MOVER if they contain EB in the executable copy and 03 in the relocation copy.

Unfortunately, while this routine works just fine in single density systems, it also has no effect on an unmodified, double density disk controller. However, it is possible to modify your double density disk controller board to make this routine have the desired effect. In fact, with a modified, double density controller, only the first three bytes of the routine are necessary (plus, of course, a C9 if it is still to be called as a subroutine).

I made this modification to my own controller board many years ago. I did it by cutting the trace on the front of the board from the feed through between 7A and 8A and putting a jumper on the back from that feed through to pin 15 of 10A. This change causes the motor timer to be reset whenever the controller status is read instead of being reset when the motors are turned on.

Unfortunately, this modification has one unpleasant side effect in that it causes the boot PROM to fail to start the motors about half of the time. Ordinarily, I would consider that side effect intolerable. But it can be eliminated by replacing the boot PROM with the newer version. North Star never put the new version into production because they had so many of the old ones in stock. But the newer PROM is available from Fischer Computer Systems (see ad on back cover).

Farewell

For the time being, at least, this is the end of my opportunity to provide assistance to users of North Star equipment. Let me just leave you with the following observation. North Star DOS is a uniquely simple and powerful operating system, the like of which we are not likely to see again for some time to come. But even if North Star someday stops building and selling new Horizons, your computer will probably keep running just as long as you want it to. More power to you!

# # #

A Curious Checksum Problem

Saul G. Levy
2555 E. Irvington Rd., Lot 47
Tucson, Arizona 85714
(602) 889-7753

I've used the checksum (CK) utility in the past on double- and quad-density drives and formed a nagging suspicion that something was wrong. Recently, I disassembled CK to determine how it worked and decide whether to convert it to checksum single files. I've decided not to do this because such a program isn't really needed and I'm not sure I could do the conversion. I can use more assembly language experience, but have many other programming tasks right now (and assembly language is much like pulling teeth!). The problem with CK is that it won't checksum the back side of quad diskettes! In case you don't know, the back side is the side with the label.

The Curious Cause of this Problem

Release 5.2 and 2.1.1 system software included identical copies of CK (the Advantage version is shorter, but very similar). I'm not going to describe the entire program (the source code takes up five pages). Here is a summary of what takes place up to the routine which has the "error."

The sign-on message is printed first, then the DOS' command input buffer is checked for a drive number, or you are asked to enter one. Next the drive number is tested (1-4 only) and, if valid, the high bit is set thus assuming double density for the first DCOM read. The original hard disk error (HDERR) jump table entry address at 11AH-11BH in the DOS is saved on CK's stack and replaced with CK's DCOM error routine. This routine will reset the high bit of the drive number if DCOM sets the carry flag due to an error in the parameters used for the first read. An error causes CK to try the second and following reads in single density (the latter reads create the checksum).

The first DCOM call reads the first sector on the diskette (at the beginning of the directory). A successful read puts that sector into CK's disk buffer which starts at CK's stack address (the stack address decrements BEFORE writing each word DOWNWARD in memory; the disk buffer increments AFTER writing each byte UPWARD in memory; there is no overlap). Next comes the routine with the "error:"

```
104E  E1          0001  L104E  POP H          RESTORE HDERR ADDR
104F  22 1A 01    0002         SHLD 011AH     SAVE IT IN DOS
1052  3A D1 12    0003         LDA 12D1H      STACK ADDR + 0FH
1055  FE 4F       0004         CPI 4FH        'O'?
1057  CA 7E 11    0005         JZ L117E       OCTAL DISKETTE! ILLEGAL
105A  FE 51       0006         CPI 51H        'Q'?
105C  21 BC 02    0007         LXI H,02BCH    QUAD DISKETTE, STARTING
                                                DISK ADDR IS 700
105F  CA 65 10    0008         JZ L1065       RUN AS QUAD DISKETTE
1062  21 5E 01    0009  L1062  LXI H,015EH    DOUBLE OR SINGLE
                                                DENSITY DISKETTE,
                                                STARTING DISK ADDR
                                                IS 350
1065  11 00 00    0010  L1065  LXI D,0000H    ZERO CHECKSUM
1068  01 F6 FF    0011  L1068  LXI B,0FFF6H   -10 SECTORS (1 TRACK)
106B  09          0012         DAD B          SUBTRACT 10 SECTORS
                                                FROM STARTING DISK
                                                ADDR (700-10 OR
                                                350-10)
```

The comments cover most of what these lines do. It took a while for the meaning of Line 3 to dawn on me! The stack and disk buffer start at 12C2H. 12D1H is the 16th byte in the sector just read (the difference + 1). This byte must be an ASCII 'Q' to checksum a double-sided diskette! Each directory entry takes up 16 bytes, but only the first 15 bytes are used. None of North Star's floppy-based software changes the space (20H) which is written into the 16th byte by the initialization (IN) routine in the DOS! (There are a few directory utilities which do so.)

OCTAL diskettes? It was rumored in the late 1970s that North Star was working on octal-density hardware. This rumor was correct! I believe that North Star tried to write 1K (1024) bytes per sector on a 40-track per inch drive and found that it was unreliable.

Today, higher capacity diskettes are usually on 96-track per inch drives where 80, or more, tracks can be written per side, thus doubling the capacity. There are a number of 1K per sector disk controllers and drives available now, but this technology suffers from poor alignment of the diskette in the drive, which is very critical (this is one reason why hard cases first appeared on micro diskettes). Besides the reliability problem, North Star probably felt that if anyone wanted higher capacities, they could plunk down hard cash for a hard disk!

A few more comments about the source code shown above: Line 5 jumps to a small routine which outputs a message stating that this version of CK can't check octal diskettes! This is due to extra code being needed to checksum octal density's 10K bytes per track. Lines 7 and 9 determine how many sectors will be checksummed. Note that assembly programmers find it easier to start just above the highest sector, decrement it to zero, and test that sector 0 has been read (this is not shown above). Lines 11-12 add -10 to the current, starting sector number. DCOM can only access UPWARD from a starting sector number. The second and following DCOM calls (not shown) will read 10 sectors (1 track) per pass at sector 690, 680, 670,..., 0 (single density reads at 340, 330, 320,..., 0). CK was obviously written in the impending octal density days and wasn't modified to match the quad-density DOSes that North Star released!

Two Fixes

If you use SINGLE-SIDED drives, DO NOT MAKE ANY CHANGES! Quad users should make ONLY ONE of the following two changes by loading CK at its GO address (the Advantage version has the same bytes in the same relative locations!):

105FH change CAH to C3H (at 005FH in the Advantage version)
or
105BH change 51H to 20H (at 005BH in the Advantage version)

I prefer the first change, but both give the same result with one limitation in usage. Either change will checksum BOTH sides of quad- AND double-density diskettes. If you try the latter, you will get a T5 error because the back side is not initialized! Either change works fine with single-density diskettes on a Horizon (one-sided only).

If you use ONLY quad diskettes, save the new version to the original CK file. If you checksum double AND quad diskettes, you'll need separate versions for each format! Either use the original CK file for single-sided only and make a new CKQ file for the modified quad version, or rename the original CK file to CKS for single sided and save the modified version in a new CK file for quad.

There are 10 bytes available starting at 1052H which could be used to check the CONFIG byte in the DOS for a quad drive (running double-density diskettes in a quad drive would still cause an error). Can anyone figure out how to rewrite this code? I can't see a way to do it in 10 bytes. You don't have to write an article, just send me the changes or a hex dump and I'll write it up and make you the first author!

If anyone wants a copy of the source code for CK, please send a diskette and $3.00 for postage and handling (or add $5.00 for one of my diskettes). The source code can be read by PDS' ASMB program. This time the source code will be commented (once I figure out the few remaining unknowns in the program)! I may have to ask a real assembly programmer for some help!

DRAW!

Reviewed by George E. Hughes

(This article is reprinted from Polaris, Newsletter of the North Star Computer Society of Seattle, October 1985. --Ed.)

The North Star Advantage was designed with an emphasis on graphics capability. There has been some software written which makes use of this feature such as IMAGEMAKER and busigraph by North Star, and some others, such as those written by some of our members for Public Domain, but for the most part the capability has not been exploited. The following article reviews a new piece of software that allows the user to bring out the full graphics capability of the Advantage and sells for a price that is substantially lower than the previously commercially available software.

The software product is DRAW! It is produced by AJW Associates (address given at end of article). Price: Version 1.01 (North Star DOS) $75.00, Version 1.02 (CP/M) $80.00.

This is a review of the CP/M version. It is also available for North Star DOS. It comes with documentation, and this review will follow the document sections.

Since most of my previous experience with graphics for the Advantage has been with IMAGEMAKER from North Star, my comparisons will of necessity be made against that software. It should be pointed out that this is not entirely fair when one considers the considerable price-differential of the two pieces of software. Present list price on IMAGEMAKER is about three to four times that of DRAW! DRAW! will do most things done with IMAGEMAKER, and has at least one capability not readily available with IMAGEMAKER. However, DRAW! does not always do these things with the same ease as IMAGEMAKER.

Getting Started with DRAW!

Instructions are very clear and no trouble was encountered in booting up DRAW!

Introduction to DRAW!

This section of the documentation describes the capabilities of DRAW! saying it "allows the user to draw and sketch on the screen. Circles, rectangles, ellipses, arcs, polygons, chord slices can be created, filled with pattern, held in memory, and redrawn in new locations with a single stroke." I was able to do all these things as I progressed through the documentation. This is the largest Section of the document and contains all the necessary instructions to create a picture and transfer it to disk or paper.

The program consists of a Main Menu and three Sub Menus. The Main Menu allows the user eight potential operations:

**To Review Pictures on File (on either drive)

**To Start DRAW! (Picture not saved to disk)

**To Start a New Picture (and save it to disk)

**To Retrieve an Old, Saved Picture

**To Rename an Old, Saved Picture

**To Copy an Old, Saved Picture to Another Disk

**To Destroy an Old, Saved Picture

**To End DRAW! Session and Return to System

All of these options are activated by single key strokes which lead the user into further menu-driven instructions by which to complete that particular activity. For the most part they all perform well. Like most first approaches to unfamiliar software it takes a few trials to master it. But I would say the documentation is written with only a

minimum of fuzzy parts. It does take some digging to maintain continuity in some of the instructions.

Sub Menus

Three Sub Menus are also available: Figure Menu, Letter Menu, Brush Menu. These will all be discussed in more detail later.

There is a page of "Important Notes" that contains ten do's and don'ts and general instructions. It tells you how to dump your picture from screen to printer, for example. For the CP/M version this is accomplished with a Control T. I use an Epson MX-80 printer and found no trouble in accomplishing the dump. Other instructions involved clearing the screen, error recovery, a warning not to press Control W (it turns off the printer driver), and a few more. All are important and careful study is recommended.

A page follows giving instructions on "Disk Operations." It describes actions to be taken following single key strokes from the Main Menu such as Save to Disk and Retrieving old pictures from disk. I had no trouble accomplishing these tasks.

The next page describes methods of moving the cursor about the screen without drawing a line. It works well, but here I must remark that it is more laborious than the same operations in IMAGEMAKER. This is because you must know the X and Y coordinates of the point you are moving to, while with IMAGEMAKER you just move visually to the point desired. This is true of many moving operations in DRAW!

The next page of documentation gives instructions on how "To Draw and Work With a Line." It tells (after moving from the Main Menu) how to set the mode, use the cursor arrows, change the line unit length, change the line pattern, and others. It is straightforward and allows the creation and manipulation of line figures in about any way the user may desire.

The next three pages of the document concern the creation of Figures from the Main Menu, and Sub Menu. Standard figures including circle, ellipse, arc, chord slice, pie slice, rectangle (forming from lower left hand corner), bird, face, irregular polygon, are available at the touch of the fifteen function keys. These can be enlarged or reduced in size and can be moved to specified coordinates or moved in line unit lengths with the cursor keys. Rotation can also be achieved.

Special figures are also available from the Main Menu. These include the creation of X and Y axes for graphs with a specified number of tics, framing or unframing the screen, creating stars with specified diameter and any number of points, viewports, and windows.

I had no real problems working with figures and producing all those described above. It does, however, require some study and repetition to learn the various commands, and because of their multiplicity, a continuing reference to the document until they are learned.

The next document page gives instructions on printing characters on the screen using the Sub Letter Menu. Any alphanumeric key will print on the screen and I found no difficulty in making it do so.

Characters may be entered or deleted at chosen locations. However, this is a weak capability of DRAW! when compared to IMAGEMAKER. The user has only the standard size character available in upper or lower cases, while IMAGEMAKER allows characters to grow, reduce, pivot, embolden, fill, alter spacing, lean, and move readily. On the positive side, one must look at the price differential. The lettering capabilities of DRAW! are adequate to do the job, but require more planning and offer less variety.

The next page of instructions covers what I feel is the most outstanding feature of DRAW! This is the capability of actually being able to draw using cursor-directed brushstrokes from the Sub Brush Menu. The brush width and line pattern may be varied and the stroke controlled to progress in any direction on the screen. As stated in the document, "The brush operation is extremely versatile, but the price is complexity." It requires a good deal of patience to master the many commands necessary, but does allow creativity in constructing the screen picture, which is an offsetting advantage.

Section 3

The final three instructional pages of the document describe the modification of an existing picture. I was successful in

carrying out the modification, but it is a complicated, time-consuming process which involves hunting down and making changes to the actual command involved in creating the part of the picture the user is altering. A keylist of all the commands used to create the original picture is made available and is printed to paper. The user must then find the section which contains the commands and pick those he wishes to change. This is the most complicated feature of DRAW!, requiring painstaking effort on the part of the user.

The rest of the document contains an appendix which gives the user additional support for most of the operations discussed above.

Improvements Ahead

In recent correspondence with Alan J. Warshawer, the creator of the software, he indicates that an additional feature has been added. This is the capability of recalling any picture into another picture. The user can develop a file of special pictures of standard shapes or articles, and then call them into a current picture.

We had made some other suggestions to Mr. Warshawer, and he has stated that he is already working on them. He indicated that he is quite receptive to constructive suggestions, and is willing to put in effort to improve DRAW! We also suggested a tutorial would be very helpful. He agrees, but thought it would be very time-consuming to write, and at this time is only thinking about it.

Summary

In summary, I feel DRAW! is well worth its price. It is not a luxury car, but for a user, whether hobbyist or commercial, who would like to create graphics and is willing to learn a number of commands, it does the job! You can get DRAW! from:

AJW Associates
2407 Brentwood Place
Alexandria, VA 22306
(703) 768-4804

# # #

Tinkering with your Horizon: Filters, 19,200, Drives

By Saul G. Levy
2555 E. Irvington Rd., Lot 47
Tucson, Arizona 85714
(602) 889-7753

Fan Filters

The major Compass articles on adding a filter to a Horizon Computer left me amused (Vol. IV, no. 1, p. 9, and no. 5, p. 13). Steven Hogan should be congratulated for bringing up the need for a filter. The dogged determination shown by our intrepid editor is useful for pointing out that adding a filter is really a very simple task!

The Perfect Filter Holder

Mr. Hogan's filter holder is too complicated and expensive (see Vol. IV, No. 5, p. 20 for a cheaper one). A friend in the Tucson Computer Group showed me an exact replacement filter holder for the fan guard on my 1978 Horizon. It is designed for instrument cases and comes in various sizes and filter porosities. The Horizon's mounting holes are four inches apart, center to center. Forty-five pores per inch is the standard porosity. One side snaps out to allow access to the thin plastic filter element which can be washed and reused. The snap-out panel has cut-out holes for the mounting screw heads which exactly match the hardware used on my Horizon. Once the filter holder is installed no screws need be removed to

gain access to the filter element!

Remove the Back Panel

Unless you have tiny hands it is far easier to remove the back panel to gain access to the fan! The only nuts you have to worry about are holding the fan (the back panel screws use threaded inserts in place of nuts). Mr. Hogan's instructions on p. 12 are excellent; follow steps 1 to 7 until it says: "Remount the fan..." Replace the fan guard (which Mr. Hogan calls the plastic finger guard in the exploded view on p. 11) with the new filter holder. Make sure the snap-out panel faces outward (away from the fan). Longer screws or gaskets are not needed. Remount the fan as instructed, insert the screws and nuts, and tighten everything down. Reattach the fan's wires and turn the blades several times to check for obstructions. Reattach the back panel and you're done.

If you're going to remove the power supply wires from the motherboard, you don't have to cut the cable tie on the back panel. I needed to add the second power supply for my new disk drives plus a few other additions to the motherboard. If you have the motherboard assembly manual, you should have no trouble reinstalling the power supply wires except for the confusion caused by having three, black ground wires. One of these goes to the medium-sized, 11,000-microfarad capacitor and shouldn't be confused with the other two black wires which go to the large, 180,000-microfarad capacitor. Either mark the odd black wire or you will need an ohmmeter to find it again (you could remove both cable ties, but who wants to do that?).

If you remove the power supply wires, use a long-nosed pliers on each connector to wiggle it free. DO NOT PULL ON THE WIRES! I'm not fond of crimped-on connectors. My crimping tool never makes a mechanically-sound crimp.

Filters for Sale

You can try a local filter supply company for the filter holder I described above. My friend and I both received our holders for free from nice salesmen. My local supplier has a $30.00 minimum order which is steep when you only want one $2.55 filter holder which includes the filter element (10 extra elements cost $2.50). As a service to my readers I will supply one filter holder and three filter elements for $5.00 including postage and handling. Each element should last a long time. If you need a new fan or hole plugs, please contact Mr. Hogan as I don't stock them.

Warning

I must repeat the warning that if you install a filter, you must periodically clean it (once a month is reasonable except in very dirty areas). The left, front corner of my Horizon now puts out a stream of warm air. If the filter is dirty and obstructs the air flow, it can shorten the life of, or destroy, components! The regulators, CPU, and RAM ICs get quite hot. Keep your filter clean!

Speeding it Up

When I rebuilt my computer I added Method Two's hardware modification for using 19,200 baud on my ADM-3A terminal (Compass, Vol. IV, No. 5, p. 3). I soldered a wire directly to pin 12 of IC 7D. This modification works fine except when I clear the screen and print a string in the same BASIC statement:

```
CHR$(26),"THIS IS A STRING TO PRINT"
```

The video RAMs take a little too long to be cleared! I get:

```
HIS IS A STRING TO PRINT
```
or
```
TIS IS A STRING TO PRINT
```

which looks strange! To fix this add one or two spaces to the beginning of the string:

```
CHR$(26)," THIS IS A STRING TO PRINT"
```

Alien Disk Drives

Shugart SA-400 and Tandon 100-2 disk drives are the correct size to fit the Horizon's drive opening and mounting plate layout. I bought two Shugart SA-455 half-height drives which are now made by Panasonic in Japan. These drives can be mounted with the original access

holes, but the left one will hit the small, 8,900-microfarad capacitor which should be loosened and moved back as far as possible.

Mount that drive with the signal ribbon cable attached (it is still a tight fit!). These drives are too long to be mounted together where the left drive normally goes and must be mounted on the right side instead. This means having to drill four new holes in the drive mounting plate and four new access holes in the bottom of your beloved Horizon. Two of the new access holes will overlap the older holes so take care when drilling them out. Does anyone out there need four extra mounting holes?!?!

I'm happy with the speed and capacity of my new drives, and even CP/M is more bearable! Having two drives on-line after 6 2/3 years of only one drive is truly amazing.

# # #

Which bit? Which Drive?

C. Pat Rooney
1360 North 3rd, NBU #61-10
Laramie, Wyoming 82070

I just got a double sided, fast stepping, TEAC disk drive to add to my computer. Let's see, where did I put the addendum to the North Star software manual? Found it. Now, bit 7 is one for double sided drive, and bit zero is one for fast stepping. How did I do it when I added the two used Tandon single sided drives to my computer a couple of years ago?

This is what prompted me to write this program to configure the personalization byte in DOS 5.2. It will also work for DOS 5.1.

If you are like me, you do next to nothing in programming at the bit level of machine language. Sure, we do modifications to DOS and BASIC at the machine level, especially ones that are printed in Compass. From time to time, we have written machine language subroutines to be called from our BASIC programs. But, up to this time, I haven't found the need to become proficient in programming at the bit level. I do not want to change the personalization byte to FF as I have a mixture of drives. My drive setup now consists of one double sided fast-stepper, two single sided fast-steppers, and one single sided slow-stepper flippy (I can use both sides of the disk as single sided).

Before I get to the program, let me tell you about the TEAC 55B drive that I got through my North Star dealer, Wyoming Computer Co., from Advanced Computer Products for $149.95. The TEAC 55B is a half height, double sided, fast stepping disk drive. The drive has a solenoid that loads the head. You have the option of having the head load on the drive select signal or the motor on signal.

This added function of loading the head, as opposed to having the head loaded all the time the door is closed as with the Tandon drives, does NOT slow the access time as far as I can tell. The drive is relatively quiet. You can hear the solenoid load the head, but it certainly isn't objectionable. When the access is finished, the head disengages from the disk. This feature makes irrelevant the question of whether you should have a disk in a drive with the door closed when you turn your computer on or off.

The stepper motor is markedly quieter than a good number of other drives that I have heard. It is especially quiet compared to the Tandon drives. The TEAC is of the new generation that has direct drive motors, in other words - **no belt.** One other plus is that the TEAC drive gives off noticeably less heat than either the Tandon or Siemens that I have connected to my computer.

One very nice feature is the motor coming on at about half speed when you insert a disk into the drive. The motor is only on for a few seconds. This helps the disk to center more consistently on the cone, and it should enhance the life of the center of the disk.

If you are wondering if I am pleased

with the TEAC 55B, let me put it this way: I going to place my order for another one. I am **very** pleased with the way it works, is constructed, and is finished. I can heartily recommend this drive.

A word about supporting your local dealer. If you are as fortunate as I am, you have a local North Star dealer like Andy Aronson, the owner of Wyoming Computer Company here in Laramie Wyoming. I may, at times, pay more for my equipment than I would from one of the discount advertisers, but Andy supports everything he sells. Over the years, Andy has given me help that has more than made up for the difference in price. Mr. Aronson is very knowledgeable when it comes to the workings of the North Star computer. If you do not have such a local dealer, then I feel sorry for you.

The Program

Now, on to the program. I am not going to describe the program line by line as I feel most of it is fairly self-explanatory. Be forewarned - you can take quite a few lines out of this program and it will still operate. **Don't do it!!** You will probably only use this program once every year or so when you add a drive to your computer, or when you may change different types of drives around that you already have in your system. The program was written with this in mind. (After I wrote the program, I had fun "playing" with it to see how the different drives would react when DOS is told that they are other than they actually are.)

Line 10 - C$ is used to clear the screen.

Line 20 to 50 - is the function to position the cursor. The formula is escape=x,y. This formula is used on the Soroc and Televideo terminals as well as on my Fulcrum video board. In the "x+32" and the "y+32" the "32" is the offset.

Line 140 - clean up the display in case of an alphabetic input.

Line 160 - C(x) array for the bit pattern.

Line 180 - using an 8 bit pattern, therefore 9-B=8 if B=1, 9-B=7 if B=2, 9-B=6 if B=3, etc.

Line 210 to 220 - convert the binary to a decimal number.

Line 230 to 290 - safeguard in case of an error during input. Only "Y" will let the program write to the disk. Any other input will cause the program to abort without writing to the disk.

Line 300 to 320 - write the personalization byte to the disk.

Line 330 - this is a call to E800 for an automatic system reset.

Warning

Try this on a COPY of your working disk with North Star DOS. I do not know if this will work on every configuration of terminal and computer that exists in the North Star world. If you are like me, the only part of your computer that is North Star is the disk controller and the CPU boards. The rest of my computer is "alien".

An Alien Board

Speaking of alien equipment, I would like to tell you of a 64K static memory board that I got through my local dealer from Advanced Computer Products. The board is fully populated with 64K of static RAM. As for the speed of the RAM chips, I do not know what it is. But, I **do know** that it works with the North Star CPU board in my system, and I tried it in a standard North Star computer at Wyoming Computer Co. You can disable a 2K section of memory board where the disk controller resides. I have "moved" the monitor to reside in memory starting at F000, although CF and CD will **not** work at this location as there is not enough room.

The board comes assembled and tested. The **best** part is the **price.** Advanced Computer Products has been advertising the board for $199.95. This is the best price that I have come across for a memory board that works with the North Star and is not in kit form.

# # #

```
10 C$=CHR$(26)
20 DEF FNC$(X,Y)
30 !CHR$(27),"=",CHR$(X+32),CHR$(Y+32),
40 RETURN ""
50 FNEND
60 !C$
70 !FNC$(5,10),"THIS PROGRAM SETS THE DISK DRIVE CONFIGURE BYTE IN DOS."
80 ! !TAB(10),"THIS PROGRAM WILL WORK WITH DOS 5.1 AND 5.2."
90 ! !TAB(10),"THE DISK WITH DOS MUST BE IN DRIVE #1."
100 ! !TAB(10),"PRESS ANY KEY TO CONTINUE.",
110 A$=INCHAR$(0)   !C$
120 !FNC$(4,10),   INPUT"ENTER THE NUMBER OF DRIVES IN YOUR SYSTEM - ",A
130 IF A[1 OR A\4 THEN 120
140 !C$   !FNC$(4,10),"ENTER THE NUMBER OF DRIVES IN YOUR SYSTEM - ",A
150 FOR B=1 TO A
160 ! !"DRIVE #",B,   INPUT" IS (0)SINGLE OR (1)DOUBLE SIDED - ",C(B)
170 IF C(B)\1 THEN 160
180 INPUT"          IS (0)SLOW OR (1)FAST STEPPING - ",C(9-B)
190 IF C(9-B)\1 THEN 180
200 NEXT B   !
210 F=(C(1)*128)+(C(2)*64)+(C(3)*32)+(C(4)*16)+(C(5)*8)
220 F=F+(C(6)*4)+(C(7)*2)+(C(8)*1)
230 FOR B=1 TO A
240 !TAB(10),"DRIVE",B,
250 IF C(B)=1 THEN !" IS DOUBLE SIDED, ", ELSE !" IS SINGLE SIDED, ",
260 IF C(9-B)=1 THEN !"FAST STEPPING." ELSE !"SLOW STEPPING."
270 NEXT B   !
280 INPUT1"   OKAY TO WRITE TO DISK ?(Y/N) ",A$
290 IF A$[\"Y" THEN !" ** ABORTED **" ELSE 300   END
300 OPEN #1 %0,"DOS,1"
310 WRITE #1 %52,&F,NOENDMARK
320 CLOSE #1
330 !C$   !CALL(59392)
```

DOCUPOWER! A Review

By Alan H. Nelson

Recently I was involved in a project which required writing descriptions for fifty different academic courses; these descriptions had to be merged into a text file which was generated not with a regular text processor, but with a database program. Although the office had a word-processing program with a mail-merge program, the only way the mail-merge program could integrate the text file with the descriptions was to create a separate file for each of the fifty descriptions, and then to read them into the master file one at a time.

This procedure required an awkwardly large number of separate files; though the project didn't hit the normal microcomputer limit of 256 files per disk, it's still hard to keep track of that many files, particularly since the filenames can't even be displayed all at once on the screen.

This is only one of numerous similar problems with merging selected textual information into a target file. Another common situation is boiler-plating, which is the process of placing standard paragraphs into a text, or indeed of making up an entire text out of pre-written parts. (Attention Lawyers!!!!!)

The solution to my problem turned out to be a program called DOCUPOWER! This program allows you to place multiple pieces of pre-written text into one or more large files, and later to extract selected paragraphs or groups of paragraphs from these pre-written files for insertion into a new text.

DOCUPOWER! Programs

Essentially DOCUPOWER is not one program but many. These programs can be used either by a master program-control module, or as individual modules. Though it is designed to be used with the master program-control modules, it is easier to explain and at times easier to use when taken program by program. So I will first describe the package program by program, and then I will explain how they are all integrated.

Process

The most important program of all is called PROCESS.COM. By typing

process sample result

you will cause the program PROCESS to look at a file called SAMPLE; PROCESS will follow instructions which have been placed in SAMPLE, and will produce a file called RESULT. RESULT will consist of any pure text which was typed into SAMPLE, plus text which was snatched from other files according to the instructions typed into SAMPLE.

Already it is evident that **process** assumes the pre-existence of at least two files, and it is evident that **process** will create a third, new file. Let's call these files the **control** file, the **resource** file, and the **target** file. The target file is of course the desired end-result. Though this file is important in the sense of being the desired end-product, it is less interesting than the others because it is a file that could have been created directly with a word-processor, though at a considerable expense of time and energy.

Resource File

The resource file, though technically more interesting than the **target** file, is still relatively simple. It consists of bits and pieces of text in any order whatever, each piece of text preceded by an identifying code which is unique in that file. The **code** is a five-digit number surrounded by markers. With five digits it is theoretically possible to have 99999 distinct identifiers, or 100,000 counting the code 00000. The **markers** are arbitrary deliminators, chosen no doubt because they make the codes stand out to the eye as well as to the computer, and are unlikely to occur in normal typing.

100,000 of anything is a very large number, almost ridiculously large, and in fact memory constraints place a limit of about 200 codes per file in CP/M.

However, the five-digit code encourages the use of numeric identifiers; thus, for example, a certain type of information can be given a first-digit code of 1, another type a first-digit code of 2, and so forth. Many human beings are perfectly capable of classifying things this way, as with telephone area codes or the Dewey Decimal System. In any case, if five digits are too many, the system will work just as well with fewer. Here is an example of a coded resource file:

[[12345]]
And so are you

[[23456]]
Roses are red

[[34567]]
Violets are blue

[[456789]]
Apples are yummy

The coding can give every marked paragraph or group of paragraphs in an entire microcomputer system a unique identifier. Depending on the circumstances, this identifier may consist of the disk drive designation, a unique file-name in any disk drive, and the unique code within the file. What's necessary of course is to have a program which will find that uniquely marked piece of text and insert it into another text at a designated spot. This is the job of **process.**

The Command and Target Files

The structure of the **command** file can now be guessed. It consists of a sequence of identification codes. Here is an example:

[[23456]]
[[34567]]
[[45678]]
[[12345]]

And here is the resulting **target** file after running **process:**

Roses are red
Violets are blue
Apples are yummy
And so are you

Here is another control file:

[[45678]]
[[34567]]
[[23456]]
[[12345]]

And here is the resulting target file:

Apples are yummy
Violets are blue
Roses are red
And so are you

So by simply switching around the codes, it's possible to produce a different result. Moreover, it's possible to intersperse text and codes, as in this control file:

It's possible to place codes in any order, like this:

[[23456]]
[[12345]]
[[34567]]
[[45678]]

That's poetry?

Here's the result:

It's possible to place codes in any order, like this:

Roses are red
And so are you
Violets are blue
Apples are yummy

That's poetry?

Filenames

Since filenames can be part of the identifier, and since it must be possible to choose from different files in the course of constructing a document, somehow it will be necessary to tell the control file which is the resource file. Here's the way the filename is designated in DOCUPOWER!:

[[0$d:filename.res]]

The name of the resource file, with its disk drive if required, is preceded by the

arbitrary string 0$, and all this is surrounded by the doubled square brackets. So if you want to have the program change resource files in the course of building a target document, all you have to do is to give the name of the new file in this manner. Of course it's always necessary to give the name of the resource file before the first calling code in the control file, so the sample resource files given above weren't really quite complete.

Paragraphs Only

Any program has its limitations, and the chief limitation of DOCUPOWER! is that it will only select from paragraph-like material, and will only insert the the extracted material into the target files as paragraphs. It will not insert a word (e.g. a name) into the middle of a sentence in the middle of a paragraph.

The individual paragraphs can be as long or as short as desired, and as many of these paragraphs can be grouped under a single code as desired. But the material will always be preceded and followed by a carriage return in both the resource file and in the target file.

Benefit

The chief benefit of DOCUPOWER! is that it allows an indefinitely large block of pre-written text to be inserted into a given place in a target file merely by entering a short piece of code into a control file. So several keystrokes worth of coding can replace hundreds or even thousands of strokes of copy-typing. Moreover, the copying will be error-free and should not require proofreading after the source text has once been judged correct.

Bs and Ws

The rest of DOCUPOWER! is bells and whistles, though there are a lot of them. One bell (or whistle if you prefer) is required for the **process** program to work at all; the rest of the bells and whistles are optional.

The one requirement is that the resource files must be pre-indexed by a program called **INDEX.COM** before they can be used at all. The need for indexing is not theoretically absolute, but was probably deemed a practical necessity by the programmers.

Consider that the only way for a computer to find material (including embedded codes) in a standard **text** file is to start at the beginning and to search character by character until it finds what it is looking for or gets to the end. Without pre-indexing the program would have to search the resource file beginning at the beginning for every code in turn, a time-wasting task for a computer however fast it can do the search. Since DOCUPOWER! searches for text not only in creating the target document, but in permitting a preview and inspection of the paragraphs in the course of constructing the control file, indexing is a practical necessity.

So the writers of DOCUPOWER! have written a program which will search a resource file for all the codes and make a mental note of where each code is located in the file. The previewing module and the **process** program therefore only need to search through a very short **index** file, find the code with its location in the resource file, and jump directly to the correct place in the resource file by calculating its memory address, thus taking advantage of the speed which is the chiefest beauty of Random Access Memory. The **index** program also has an error-checking function, since it will identify duplicate codes in a resource file as a protection against future error.

The essential steps are now in place. The user must create one or more resource files in advance, and then **index** these resource files. Next it is necessary to create a control file which will determine the contents of the final document. **process** will create the desired target file.

The Envelope: DOC1 and DOC2

Programs called DOC1.COM and DOC2.COM guide the user through the process of choosing resource files, inserting codes into any text file which is to be used as a resource file, indexing resource files, constructing control files, previewing paragraphs which are candidates for insertion into the target files, and much more. Still other

enhancements permit fine-tuning the end result by adding extra spaces, horizontal lines, and so forth.

DOC1 permits the creation of a **directory**, or, so to speak, a **menu** of resource files. Each resource file displayed on the screen can be provided with a description or reminder more informative than the eight- or eleven-character filename permitted by CP/M or MS-DOS. Since the resource files are shown on the screen in a numbered list, the file can be selected just by typing its one- or two-digit number.

DOC2 is a program which automates the construction of a control file. It allows the user to scroll through a chosen resource file, and to select material according to information summarized in a header. Finally, a keystroke issued from DOC2 will permit the indexing of a resource file and the creation of a new file by the **process** program.

Terminology

The biggest problem I found with DOCUPOWER! was in mastering the terminology, for example in understanding the exact difference between a resource file and a control file. After having worked with DOCUPOWER! for a while the distinctions are so clear to me that I can't understand how I could ever have been confused. But being a beginner means having to do something without knowing everything, and that's always hard.

Another problem I found was in understanding the menus in DOC1 and DOC2. The creators of the program faced a problem here: they obviously wanted to place as many functions at the user's fingertips as possible. For the most part, each function is invoked with a single key-stroke. But each potentially-available key-stroke has to be defined by a word or group of words which describes its function. Each available function therefore takes up space on the screen; every three functions take up a line on the screen; the more lines taken up, the smaller the amount of text that can be displayed from the resource file or from the command-file-under-construction; hence the need to make the descriptions of all possible processes as short as possible; but short descriptions tend to be cryptic; and cryptic descriptions are anathema to a beginner.

For my own part I think I would introduce a beginner not by using DOC1 or DOC2, but by showing the beginner how to create a resource file and a command file with a word-processor, and by demonstrating how the programs **index** and **process** will result in a third file, the warmly desired end-result. Only then would I try to get this now somewhat more experienced beginner to take advantage of the ingenious and time-saving features of DOC1 and DOC2. In other words, I would instruct a user about the same way as I have attempted to instruct you the reader.

Complaint Department

When I first used DOCUPOWER! I had one minor and one major complaint. The minor complaint is that it was nowhere clear in the documentation whether the character preceding the $ in the filename designator was a capital letter **O** or the number **0** (zero). I had to try it both ways before deducing that it was a zero.

Second, and more seriously, I discovered that **process** would allow up to five resource files to be specified in a control file, but would baulk at a sixth, giving an incorrect error message.

I got in touch with the authors by their hot-line, and discovered that they had never encountered anyone who wanted to select from so many resource files in a single control file. They were unaware of the problem because they hadn't encountered it before. They said they would fix the problem, and much to their credit, they did. So my only serious complaint was remedied almost immediately.

Compliment Department

DOCUPOWER! is an ingenious program which answers a very great need felt by many professional microcomputer users, of creating documents simply and easily out of pre-existing parts. It is very well thought out and very fast. Learning or teaching the program will take some patience. Users already experienced in creating documents with a word-processer will be able to master the program much faster than rank beginners because

understanding the program assumes some understanding of files and file structure. Though I find the documentation slightly hard to follow in spots, I think it is difficult because the concept is slightly difficult, and not at all because it is badly written. This is a sophisticated tool best suited to somewhat sophisticated users.

Integrating Software

The sign of a really good program is that it can be put to use in ways the authors have not anticipated. The documentation describes three methods of producing control files: first, using a separate word-processing program; second, using DOCUPOWER's DOC2 program; and third, using another program created in a computer language (e.g. BASIC). I decided that I wanted to create a control file using a **database program**, especially to take advantage of its sorting capability.

My task was to merge some two thousand paragraphs scattered throughout numerous text files into one long file; and these paragraphs had to end up in a coherent order. I gave each paragraph in each file a code which was nothing more than the year of the document followed by a zero as the fifth digit. The second document from the same year received a 1, the third a 2, and so forth. So in each file I could have ten documents from the same year, and an unlimited number of years. I also organized the documents into files to which I gave coded alphabetic names.

I created a long database file containing the numeric codes and the names of the files in which these were embedded. (This process was easily automated - see "Databases from On-Line Sessions" in the last Compass.) I then sorted on date as my first field and filename as my second field; I copied the resulting sorted database to a text file in which the filename was preceded by the string [[0$, followed by the string]][[, followed by the date code, followed by the string]], like this:

```
[[0$ibm]][[19790]]
[[0$ibm]][[19791]]
[[0$northstr]][[19790]]
[[0$osborn]][[19790]]
[[0$compupro]][[19800]]
[[0$ibm]][[19800]]
[[0$northstar]][[19800]]
[[0$northstar]][[19801]]
```

So the codes were sorted by year, and within each year, by company name. The heavy work of sorting and producing the control file was done by the database program; the heavy work of producing a text following the sort was done by DOCUPOWER!

You can see why I was frustrated by the fact that in its early form DOCUPOWER! could handle only five changes of filename in any control file: in this application the filename changed with virtually every code! And I succeeded in doing what I enjoy most, mixing and matching programs in ways not dreamed of by the programmers.

Availability

DOCUPOWER! is available from Computing!, 2519 Greenwich, San Francisco, CA 94123, at a price of $149.00.

# # #

Membership

On the following pages is a list of 1985-6 members of INSUA. The zip code or mailing code is intended to allow current members to contact one another for mutual assistance with micro-computing matters. No other use is to be made of this list.

00626 RAFAEL A. ALCOCER
01002 WALTER KOHLER
01201 JOHN G. BURNS
01545 DAVID STEFFEN
01757 BRUCE DE RIENZE
01775 JOSEPH E. SULLIVAN
01778 DONALD HAY
01886 RONALD E. TETREV
01904 ROBERT GILFORD
01915 RONALD J. SUBLER
02158 DAVID J. YATES
02173 MERTON KAHNE
02173 EDWARD GANSHIRT
02192 PETER J. DiCAMILLO
02632 BARCLAY CORP.
02719 DOROTHY CATE
02748 SILVINO C. FERREIRA
02871 JAMES M. MCLEAN
02901 WILLIAM D. COUNTS
03049 JOHN A. GALLANT
03824 RUSS HALEY
03840 DAVID L. RICHARDS
03902 WES BARBOUR PHOTO
04252 RAY SIEGLER
04558 K A POMROY
04697 JOHN H. LONGMAID
05201 LEE SUPOWIT
05401 FREDERICK H. RAAB
05676 PETER TOURIN
06050 J.C. LARKIN
06064 ROLAND MALBOEUF
06089 MICHAEL CRISAFULLI
06117 JOHN H. ADEE
06242 WARREN C. CAMPBELL
06404 MICHAEL E. BOUTIN
06470 WILL TOLL
06470 IRVIG C. ZACHER
06470 ROBERT MUNGER
06470 JACK TRAUB
06475 MICHAEL D. RICE
06489 GEORGE DOMBROWSKI
07470 BRUCE T. BUBELLO
07481 KAMAL G. BAHRI
07601 WARREN J. KAPS
07621 JAMES LAUSER
07646 ALAN SCHOFFMAN
07648 FRANK GEIGER
07670 RENO A. DEL BEN
07748 PAUL T. BRADY
07834 ROBERT H. MARTIN
07901 ALAN F. DICKEY
07960 G.F. HAYS
08003 GREGORY D. HICKIN
08003 J. PETERS
08028 RICHARD SEIDNER
08028 RONALD L. MARGESON
08060 KEN CURTIS
08109 JOHN R. GRANT
08540 MARK&PAMELA SHEINGORN
08540 OLAV REDI
08807 EDWARD HSI
08822 JOHN T. KALINOWSKI
08857 LEE DANIEL QUINN
08903 GENE A. GIACOMELI
08904 DAVID ROSENBERG
09011 LTC DAVID S. KLUNK
09102 RINALDO VACHINO
09333 BRIAN MAASS
09407 TIMOTHY L. SEAMAN
09667 OLIVER C. STOKES
09757 REX M. VENATOR
10007 ALAN J. STOPER
10010 ANTHONY S. NISKANEN
10011 JOEL KREMSDORF
10025 J.J. DANNENBERG
10028 GABRIEL KATONA
10028 DAN KONIGSBACH
10032 JAMES R. MILLER
10038 TED BLANK
10128 KATHLEEN L. KELLY
10128 HARRY DERWIG
10276 BILL SHOR/VENUS RECRD
11434 LAWRENCE J. CULLEN
11520 MORRIS GLADSTEIN
11550 E.J. AGNELLO
11552 MURRAY GILBERT
11576 EVAN KATZ
11693 STEPHEN A. FARMER
11701 GENE BROWNE
11715 GERALD STRICKLAND
11722 LEONARD SCHMIRL
11725 JOHN C. PEREIRA
11726 ALAN SHAW
11743 BRUCE W. HARKNESS
11793 DENNIS CASHTIN
11794 NICHOLAS CARNEVALE
11797 MOREY J. HERZOG
11901 DOUGLAS A. RANDALL
11935 ROSEMARY KLOS
12010 ROBERT SWENSON
12072 J. WM. WOYTHALER
12309 JAMES M LOMMEL
12590 RICHARD C. BROWN
12953 HARVEY FACTOR
13041 WALLACE B. WATSON
13057 WIN BABCOCK
13088 JAMES M. YURDIN
13214 DAVID J. RAPP
13350 LAUREN E. BULL
13760 PAUL F. CUSHING
13760 DANIEL NORMOLLE
13850 HARVEY ROEHL
13903 WILLIAM H. CONNOR
13905 JOHN B. WADE, INC.
14051 JAMES H. GUSTAFSON
14132 ROBERT APPLEGATE
14150 RICHARD LOCKWOOD
14150 ARTHUR KURTZ
14150 ROBERT G. DISPENZA
14214 ALAN GROSS
14215 R. ACKERHALT
14216 SPECTRUM SLIDES
14216 DAVID ANDERSON
15901 PAUL CARPENTER
16335 LEONARD HUNTER
17214 R. MACK
17325 LAWRENCE N. DIXON
17331 IVAN L. FILLMORE
17601 JOE MALAK
18103 GORGE CONLY
18105 E.A. LINEBRGER
18328 CHARLES PROHASKA
19072 STEPHEN FLINK
19103 ROBERT G. DEAN
19104 RICHARD T. BIDDLE
19341 GARY G. REYNOLDS
19355 ROBERT W. APPEL
19446 WILLIAM T. JOHNSON
19567 CLIFFORD B. SHERMAN
19707 H.F. PORTER
19901 RAY SCARBOROUGH
20007 GRAHAM ATKINSON
20009 FRANK J. PRUSS
20011 LOUIE BLALOCK
20317 BERYL X. SMITH
20601 AJ ELECTRONICS
20684 LLOYD D. STERLING
20686 DWIGHT A. JOHNSON
20740 ROBERT D. PARR
20770 ERIC SPEAR
20770 DONALD M. COAKLEY, JR
20782 JAMES E. INGE
20816 RICHARD BERG
20816 NORA M. TAYLOR
20850 ALAN H. CZARAPATA
20854 BURTON ANDREWS
20878 JACK HEYTENS
20895 WILLIAM E. WAGNER
20895 RON RASPET
20901 KENNETH W. PARKER
20902 R. ROWLAND
20910 GORDON RIEL
21001 HAROLD J. ASHCRAFT
21146 STEPHEN R. TROY

06497 MR. R. DONALD REED
06497 ANGELO D'ALESSIO
06611 ROBERT CARBONE
06611 LUSTEG TAX SERV.
06793 LEE A. NEUHAUS
06798 JOSEPH A. FOSTER
06836 BERKLEY HOLDINGS CORP
06852 A.J. GUSTAVSEN
06854 ALAN KEYWORTH
06855 THOMAS R. TIGHE
06877 ANDRE MCHOSE
06880 DONALD R. GORDON
07006 RALPH STEINHART
07042 FREDERICK D. SIBLEY
07054 WARNER HAUER
07094 JEROME HAGGART
07110 JEFF DUMANSKY
07306 FRANCIS X. DORRITY
07407 RONALD MACKAY
07446 WILLIAM T. KELLY
22301 WOLFHARD RAMM
22306 EUGENE C. BOUNDS
22306 ALAN J. WARSHAWER
23113 JESSE BAILEY
23454 WILLIAM R. MARTIN
23462 KATHERINE PITSILIDES
24017 STUART P. JACKSON
24018 GSS COMPUTER TECH
24024 HARRY G. NORRIS
24078 ROGER A. HOUGH
24153 HOWARD D. SACKETT
24503 WILLIAM H. PRESSLY
25401 JACK DEHAVEN PHOTO
26719 JAMES A. WHITMAN
27106 JERRY WHELAN
27106 HAROLD MENCK
27262 FRED THOMAS
27405 TERRY HOUGH
27502 HENRY O. COLOMB, JR.
27511 JERRY W. RICHARDSON
27695 DENNIS HERMAN
10301 STANLEY GOLDIN
10468 PETER MELZER
10471 PHILIP W. BRANDT
10512 JAY JUDELL
10514 HAL SKURNICK
10514 ROBERT FASTOVSKY
10533 PATRICK CORRY
10541 NORMAN F. BRAY
10549 L. MARK RUSSAKOFF
10583 ERIC LINDOW
10803 FREDERICK J. SKINNER
10977 HARRY SHAMES
10983 EDWARD H. MCAVOY
11040 HARRY ZAM
11214 ROBERT FALK
11236 JERALD ABRAMS
11354 EDWARD HO
11354 CHARLES PECK
11361 JIM BLAKE
11419 HARRY RIES
32961 BUD L. HOLMAN
33023 AL O'HARA
33138 DOUG INGRAM
33143 W. WALDO LYNCH
33144 YGNACIO MORENO
33156 ALPHA BUSINESS COMPUT
33157 JOSEPH FERREIRA
33165 RAYMOND S. KULZICK
33166 LATIN DATA M-238
33177 WILLIAM KLINE,JR
33328 DAVID MORRIS
33572 R.J. STURTZ JR.
33606 TONY LETO
33715 ARTHUR B. CHAUSMER
33860 WILLIAM W. STEWART
33881 MILTON P. CHARTER
33962 LELAND S. DENNING
35802 CARL H. PALMEK
35803 DOUG ELGIN
35807 ROBERT A. BROWN
36104 WILLIAM L. IRVIN
14216 RICHARD GREENBERG
14221 ROBERT J. SCHUDER
14226 MARK N. TAYLOR
14226 RAOUL NAROLL
14227 SCOTT S. CARTER
14469 JOHN A. BRYANT
14580 KENNETH L. CLUM
14612 DAVID STEWART
14614 HARVEY M. NUSBAUM
14618 ERWIN RAHN
14623 ROBERT ROSENBLUM
15061 RICHARD R. DUPREE
15068 DAVID REVILLA
15146 GREGORY J. ALHEID
15147 EDWARD JONCZAK
15207 GEOFFREY D. BLOCH
15641 SAMUEL VINCENT
15668 HARVEY HECKER
15801 J. THOMSON
15851 GARY L. SENIOR
44260 W.E. CLAXTON
44280 D.L. RINGWALT
44505 K.C. KUNIN
44646 VERYL E TURSKEY
44720 CHARLES MAKINSON
44805 JOHN ROWSEY
44883 COMPUTER WORKS
45202 EUGENE M. ROTHCHILD
45227 WM. E. JOHNSON
45322 RALPH M. WATROUS
45431 WILLIAM G. THORPE
45433 MARTIN RICHARDSON
45459 ROBERT D. GRUBBS
46168 TERRY E. PLANK
46176 TIM DEATON
46204 WILLIAM H. SHAKAL
46222 PAUL W. HERRING
46229 ELDON HAWKINS
46236 R.L. WOLFF
46250 PHIL ALEXANDER
46307 GEORGE M. POLLINGUE
21201 MARK A. KURZMACK
21207 STANLEY BARON
21209 STANFORD LAMBERG
21209 ELI FREEDMAN
21217 C.J. PENNINGTON
21229 JAMES SMITH
21401 WILLIAM BUTLER
21401 STODDARD"TED"KNOWLES
21701 FRED ABELES
22003 DON C. ECKHOLDT
22032 DAVID FISCHER
22039 ANDREW P. SAGE
22041 JAMES L. SKILTON
22090 AUDREY SUNSTROM
22094 JOHN P. RETELLE
22101 E.L. JOHNSON
22124 WARREN B. SAUDERS
22180 DANIEL D. ARDEN
22193 ROBERT L. PORTER
22201 ALLAN ARMUS
49017 STEVEN HARKE
49269 ROGER KLOEPFER
49301 IVAN BELYEA
49441 KENNETH KERR
49862 PETER JONAS
51334 MERLE JOHNSON
52240 JAMES FLUCK
52240 GERALD STAMP
52761 WALTER A. THROM
52806 RICK AHLGREN
53115 ARTHUR L. JOHNSON
53186 JEROME J. MONFRE
53211 MICHAEL B. SHEFFEY
53226 ARVIS A. KRAETSCH
53558 STAN SITTS
53703 LUNAR RADIATION CORP.
53705 DARRELL S. BRAUN
53706 GLENN SATHER
53711 J.S. LOWREY
53713 PAUL BENDER
54552 TOM SCHEIBL

27705 ROBERT C. CARSON
28403 ROBERT SANDERS
28532 JOHN KIEFFER JR.
28739 C.D. GIBBS
29205 RALPH A. RUSCETTA
29260 JOHN E. MC MURRAY
29407 GARY LUNSFORD
29445 STANLEY YOAKUM
29578 GEORGE D. SACKETT
29681 PETE HOOKS
30001 JOE FOPPIANO
30117 E. PERRY WALDREP JR.
30117 RICHARD T. JONES
30306 ROBERT H. WALLING
30340 TIM LINK
30458 JAMES W. TAHLER
30467 J. HILL
30542 JAMES SCARBOROUGH
31201 PAUL E. LYLES
31501 JAMES C. JOHNSON
31510 LORENZO E. HAND
31707 BEV. B. HARRIS
31709 CHAS W STRICKLAND, JR
32073 GENE W. ASCHENBECK
32205 JOHN H. SAARI
32221 RICHARD HARRELSON
32312 BEN JOHNSON ASSOC.INC
32405 JAN CRANE
32541 DENNIS A. BYRNE
32601 THOMAS E. BARRUP
32604 ALAN GREGALOT
32605 JOHN C. SODERSTUM
32714 W. PORTER WIGHTMAN
32731 BILL GALBREATH
32746 CHARLES N. RITTER
32771 GEORGE CHENEY
32792 DAVID B. DROSTE
32803 STEPHEN E. DOLIVE
32901 ROGER POLLOCK
32953 ARTHUR J. PHILLIS
36109 CHARLES A. HIGHTOWER
36111 JAMES C. MATTHEWS
36264 H. EUGENE TOLLESON
36582 ROY KEELEY
36590 MICHAEL R. WILLIAMS
36604 DAVID PATE
37115 STEPHEN D. KEEL
37133 WILLIAM Q. CRICHLOW
37214 S.H. PEARSALL
37215 JAMES ANDERSON
37220 THOMAS M. BROWN
37221 HEALTHCARE TELECOMM.
37662 STANLEY W. STILL
37830 JAMES B. BALL
37919 WARREN LAMBERT
37919 ERIC FAIRFIELD
37931 M. CARL BECKER
38025 S.P. WELBORN, JR.
38655 OXFORD SOFTWARE CO.
40118 IAN G. ELLIS
40223 JAMES A. HOUCHENS
40383 WAYNE D. THOMPSON
40506 DAVID ALLEN
41653 ERNEST BROOKHART
42301 J.A. BRYANT
43081 WM. P. MODRY
43085 PAUL WERNER
43205 CHILD ABUSE PROGRAM
43212 EWEN KING-SMITH
43537 FRANK T. ALLEN
43560 FRANK WALMSLEY
43606 H.BRADFORD THOMPSON
43611 JIM FRY
44017 PAUL HARLAMERT
44087 DAVID BURNS
44106 N.A. ANIS
44122 DEZSO LEVENDULA
44124 GARY L. WHEELER
44129 BARBARA SOLOMON
44256 ROBERT W. BROWN
46319 JOE ALONSO
46320 THIEL COHEN
46321 JOHN B. MALLOY
46506 ORTON MILLS
46590 VILAS E. DEANE
46615 DENNIS HEAD
46703 JIM SHEARER
46733 JIM HOWENSTINE
46807 DANIEL E. BERNING
46815 DON SLANE
46825 BRUCE A. WIGHT
47401 B & L COMMUNICATIONS
47906 DOMINICK ANDRISANI
47906 HOLLY L. MASON
48018 KEITH A. DEVORE
48020 CONNIE SIEH
48020 ROGER LA POINTE
48043 HENRY W. MILLER II
48050 ROBERT E. LAURION
48058 MICHIGAN REPROGRAPHIC
48063 THOMAS B. LIGHTBODY
48067 FRANZ H. BREIDENICH
48067 ANN BRUNK
48070 ERIK KIND
48075 JIM MITCHELL
48079 LEONARD HOOL
48084 MICHAEL B. ACHORN
48089 ROBERT CLYNE
48103 PETER W. MEEK
48104 JAMES L. TAYLOR
48105 MEL L. BARCLAY
48125 RAYMOND R. PASK
48161 DAVID L. SHADLE
48197 JOHN HUGHES
48198 CLIFFORD HARRISON
48236 CHARLES NAIRN
48453 BLUE WATER HOMES, INC
48746 REV. RON IRIS
49008 ERIC SCHREUR
49012 B.A. THUNMAN,
54901 COMP CARE COMPUTR SYS
54903 CARLSON'S FINE FOODS
54914 RONALD SAMSON
55108 DAVID DUGGAN
55113 BRUCE M. WEBER
55413 EUGENE PISHKO
55417 MICHAEL CANNY
55423 VICTOR H. HEINER
55433 DAVID E. MACDONALD
55438 G.M. CZAJKOWSKI
56241 RANDY JESERITZ
56601 STEWART & WALKER, INC
57701 T.J. DAHLQUIST
57702 JAMES A. KUNZ
58274 JOSEPH C. TATE
59604 OSCAR M. CARLSON
59624 CUSTOM COMPUTING
59802 PAUL A. SHUEY
59806 CARL RUMMEL
60030 JAMES BROWNE
60040 ARNOLD B. TONI
60068 ED COUDAL
60068 JAMES M. ULLMAN
60078 A. L. WUDI
60093 THOMAS A. WICK
60102 MARTY LEIDER
60115 WALLACE R. MCALLISTER
60183 ALLAN HIGGINS
60302 C.T. BOMBECK M.D
60439 WILLIAM S POINDEXTER3
60525 GREGORY LAURIANO
60604 CHEM. DE PAUL UNIV
60608 PHYLLIS MCDONALD
60616 TATE YOSHIDA
60620 E.E. ANDERSON & ASSOC
60645 EDWARD CRASS
60659 DANIEL D. STUHLMAN
61107 MATT CROWLEY
61462 ROBERT SHIMMIN
61604 STEVE REDIGER

61701 M.R. CONRAN
61801 HENRY E. MURPHY
62025 E.R. WILLIAMS
62901 RICHARD M. JOHNSON
63011 DENNIS L. DOEFLER
63031 KENNETH O. HENDERSON
63032 GENERAL SYSTEMS, INC.
63119 GEORGE A. REID
63121 J. WM. MUELLER
63121 CARL E. SCHAFFNER
63128 ERIC D. NULSEN
63301 FRANK BROOKS
63366 L.R. SLATTERY
63401 WM. REMILLONG
63801 LEE A. BOWMAN
64055 DON WEEKLEY
65201 ALFRED LLORENS
65202 D.J. MANSON
65656 STEPHEN C. BABBIT
65808 OZARK STRUCTURES, INC
66203 CRAIG BROWN
66208 MORTON JACOBS
66605 PAUL BARKLEY
66614 SPENCER SMITH
66839 O.K. HUDSON
67042 FRANCES HELLAR
67114 DENNIS KELSEY
67203 GEORGE W. LANDIS
67203 JOHN CAMPBELL
67337 JIM GARNETT
67401 GLEN E. EATON
67401 KENNETH CARMAN
67601 CHARLES VOTAW
68133 JERRY FRANCIS
68154 KIRK BENEDICT
69201 TED J. ORMESHER
70112 LARRY P. FEIGEN
70112 ELLIOTT M. BAIN
70381 RICK TEMPLE
70394 LAWRENCE J. MILLER
70422 ROBERT A. LANDRY, JR.
75067 JON JACOBS
75088 CONRAD ROMBERG
75090 STEVE MARUM
75149 STEVE LEAKE
75374 WALTER B. RICE
75711 DAN STEPHENSON
76011 MIKE STUTE
76043 MICHAEL J. RIGGS
76105 G.D. EGGER II
76302 JOSEPH E. HAYS
76707 JOHN R. BRASSARD
76801 PAT RUDESEAL
76904 ROBERT C. HICKS
76950 JAMES E. DOVER
77002 FRED L. ROBBINS
77005 BILL ARNOLD
77025 CHARLES SEIDEL
77034 GARY R. HARLOW
77042 JAMES G. VLETAS
77058 MARY C. FERGUSON
77071 RAY B. JOHNSON
77074 APPLIED METEOROLOGY
77074 J. T. McCORMACK INC.
77080 A. RAY-DEWITT&CO.,INC
77081 W.D. HAMMOND
77227 D. W. HUGHES
77373 JAMES TATE
77449 MERRY JOHNSON
77450 ROGER SHULKIN
77511 A.W. ADAMS, JR
77511 BOB ALEXANDER
77539 JAMES G. HALES
77571 SIMPLE OPERATING SYS.
77592 DENNIS KOI
77662 A.G. BLACKWELL
78102 OAK LAND COMMUNICA
78209 PEPOS S. DOUNSON
78221 FELIX I. BERNAL
78284 BENNETT DYKE
78539 RONALD C. KETTERING
78705 A. WILSON NOLLE
80521 GORDON WANG
80525 JERRY HUMMEL
80631 JOSEPH F. HAEFELI
80907 FLEMING ELECTRONICS
80918 DEAN HENDRICKSON
81001 DAVID M. PERKINS
81212 VERNON ESTES
81301 MEL MATIS
81639 LARRY COSTA
82070 C. PAT ROONEY
82071 RON JOHNSON
82426 BRUCE W. BREGSTROM
82801 GLENN DORSCH
82834 JAMES S. GUYTON
83201 ROGER WILLIAMS
83338 CHARLES H. CORRELL
83340 JOHN HERBERT
83402 DAVE SEVY
83642 J. R. KILLIAN
83642 JON CARTER
83704 JOHN D. MUTCH
84057 DELTAX CORPORATION
84058 ERIC N. SKOUSEN
84119 W.L. PENROSE
84319 EUGENE K. ISRAELSEN
84321 HITOSHI YOSHIMOTO
84401 NORMAN WORTH
84403 STEVEN E. KAMMEYER
84404 FONCEY TAYLOR
84532 E. J. CLAUS,
84604 WENDELL ALLRED
84663 CRAIG A. LARSEN
85013 MARGARET ZINKY
85014 METTEE/MCGILL/MURPHY
85015 DONALD H. SMITH
85020 ROBERT HEALEY
85023 L.W.(VERNE) DISNEY
85034 R. L. SANDERS
85224 WAYNE R. ANDERSON
85251 RICHARD ERIBES
85258 JACK R. RUSSELL
88003 JOHN A. LUDWIG
88004 PAT O'DEA
88005 M.H. BERNSTEIN
88201 WARREN EHLERT
89101 A.G. ZENGER
89102 ROBERT BOWEN
89107 GILBERT FIRMENICH
89107 JOSEPH I. SALGO
89120 JAMES DUNN
89121 FRANK A. WHITAKER JR.
89423 MARC ROMERO
89511 J.E. PEPPLE
89701 CUSTOM SERVICES, INC.
89702 JOSEPH P. SAMMUT III
90012 HIROSHI SAISHO
90026 SHERWOOD LEE
90045 E.E. ST. JOHN
90066 SAM EL-HAI
90066 WILLIAM PROUD
90066 ALFRED J. MONROE
90068 EDWARD P. ANCONA, JR.
90213 JOHN RILEY
90230 PHILIP MASSIE
90232 GLENN MCMURRY
90241 BRIAN HOLMES
90242 JOHN ARMSTRONG
90247 GEORGE POND
90260 PAUL W. PHISTER, JR.
90262 SNOW MFG. CO.
90266 G. R. TELLE
90272 NATHAN LOCKMAN
90272 JANETTE RAINWATER
90272 E.F. GRANT
90274 S.M. CLEMENTE
90274 GEORGE UEBLE
90291 JAMES MC GINTY
90402 J.V. HOLDAM
90405 HARRIS GELLMAN
90505 GERRY MAYER
90608 BOB HENDERSON
90630 CRAIG BOURNE

70433 LOUIS J. ULMER
70601 CONWAY STONE MAGEE
70663 CARL N BLAKE
70663 WADE WATTS
70802 WILLIAM J. WEMPREN
70809 RONNY J. CHAMPLIN
70815 W.A. FAVRE, SR.
72042 Q. D. LA FARGUE, JR
72745 JEFF BANKS
72801 DALE W. FINLEY
72853 SAM C. TURNER
73527 LARRY E. SMITH
74003 WAYNE McGINNIS
74011 FRANK LAUGHLIN
74011 DONALD WORTMAN
74114 RANDALL STAPONSKI
74561 PERRY GARST
75001 DAVID W. NOELL
75042 RICHARD L. SOLOMON
75067 DONALD F. WILHOITE
91750 DENNIS M. SHARPE
92008 GARY MOSELLE
92014 TOM TROZERA
92020 DONALD R. HENRY
92027 L.A. BEEL
92031 LEE YOUGHFLESH
92037 HARRY G. BLUESTEIN
92061 EDWARD L. HOPKINS
92065 DANIEL CROSS
92108 JOHN F. GERGURICH
92124 KEN MCINNES
92126 LEE L. STEADMAN
92344 DONALD D. FAWCETT
92373 WILLIAM E. JAVERT
92376 DORR STUART
92382 S.D. SAMPSON
92501 WILLIAM MELCHER
92507 WILLIAM H. ORTTUNG
92621 PAUL R. JONES
92627 JOHN T. LEWIS
92633 STEVEN A. HOGAN

78723 DOROTHY L. DAILEY
78759 WALLACE TUTEN
78934 JAMES H. WHITCOMB
79008 WILL GRAHAM
79106 NIEL NORWOOD
79606 RICHARD GERTH
79709 ROGER BARNHILL
79925 CHARLES E. VINSON
80026 WOLFRAN KASEMIR
80026 TOM ISGAR
80044 RICK DOWNS
80110 PAUL WHITAKER
80122 BRIAN PHILLIPS
80206 ART MILANO
80211 J.A. VULETICH
80220 F.B. ROGERS
80231 RONNIE SMITH
80303 RICHARD ROOK
80307 WILLIAM D. McCAA JR.
80501 JIM SWARTZ
94002 HERMAN THOMS
94010 D.W. CARLSON
94015 KURT H. HECKSCHER
94022 J.S. SIUDZINSKI
94022 ROGER SHERMAN CO.
94022 J.N. CHRISTIANSEN
94022 J.C. SPUOUT
94022 L.K. KOEHLER
94030 CHARLES MAYNARD
94030 DATA METHODS&SERVICE
94040 BRUCE J. EDMUNDSON
94040 KEN SMITH
94044 ROBERT LEE
94063 R. L. MCLEAN
94063 E. NOWAK
94066 JOAN RANKIN
94070 R. B. BRAUNSTEIN
94086 AMTEL SYSTEMS CORP.
94086 WALLACE E. LIN
94086 JEROME J. MILLON
94087 MIKE AHLMANN

85281 JESS HANSON
85282 GREG CLARK
85283 RALPH J. KETTER
85287 MICHAEL F. SHERIDAN
85308 ROBERT J. ALEKSA
85613 HAROLD ADAMS
85703 LOYD HOPPER
85704 PETER A. CENTO
85710 DUANE MILLER
85710 HILBERT SCHOUTEN
85712 SAUL LEVY
85715 A.T. COLEMAN
85716 D.M. HUNTEN
85716 DESERT DATA SYSTEM
85721 MICHAEL CUSANVICH
85732 GERARD BUNGE
87108 V.E. DUDLEY
87108 DOUG JACKSON/SIM, INC
87502 LAIRD GRAESER
87544 ROBERT R. SHOWALTER
94538 ROBERT BICKFORD
94539 DONALD R. GOTHHOLD
94539 DAVID DAVISON
94542 S.Y. TANG
94542 JOEL H. FINK
94544 MIKE JORY CREATIVE
94546 VIVIAN E. NESBITT
94550 ELMOND HOLBROOK
94556 SYLVIA WESSEL
94558 ERNEST LEE ABBOTT
94558 NORMAN DELEVZE
94558 FRANK THOMPSON
94559 GREG W KELLER CORP
94563 FRANKLIN LEW
94564 ALFRED DEL SIMONE
94566 ROBERT SCOTT
94566 DOUGLAS J. SEMON
94568 HEXCEL CORPORATION
94572 FRANK ERNST
94574 JAMES THOMPSON
94577 ED ELLEFSEN

90631 HUBERT H. LOVE JR.
90701 COMPUTER OPTIONS
90717 JOHN MADDOCK, INC.
90745 DONALD NIERAETH
90803 BARNEY FLAM
90814 DON E. APPLEBY
91001 WILLIE L. PATTERSON
91040 BOB KUETHER
91105 ROBIN WATKINS
91106 TONY B. ANDERSON
91107 W.F. PFEIFFER
91125 LORNE REID
91321 W.T. HICKS
91335 TODD ZERVAS
91344 J.W. CURTIS
91344 ROBERT C. RADFORD
91353 TONY LERNER
91364 JOSEPH KEMP
91403 GILBERT J. GILBERT
91711 PAUL J. KENKRICK
94806 WALTER E. LAUGHLIN
94901 DAVID KARP
94903 JOHN RICHARDS
94904 JIM FURMAN, SOUND INC
94939 FREDERICK D. SEITEL
94947 ROGER T. SMITH
94957 LHARY MEYER
94960 HENRY SPELMAN,III
95003 WILLIAM PROUDFOOT
95030 TOM MC DANIEL
95035 CLYDE CHESNEY
95037 MARTIN BROWN
95054 CALVIN L. WONG
95060 EDWARD C SUMPF
95065 PETER WILDE, KINNEDIC
95112 GEORGE EWERS
95117 RALPH GETSLA
95120 CHARLEY LICHTENSTEIN
95120 PHILIP E. BAILY
95120 S M SCHREIBER PROCOM
95123 DAVID TOMLINSON

92635 MARSHAL J. BRECHT
92645 EDWIN YARGA
92646 RICHARD T. McCARTNEY
92647 STEPHEN SHARON
92649 MARL RUBIN
92653 M.G. LOWE & COMP.
92661 DICK LEWIS
92670 PHILIP D. COREY
92675 DUNCAN MORRILL
92677 R.H. DAVIS
92680 CHRISTOPHER JONES
92683 KENNETH R. OLSEN
92691 ROBERT HOGG
92691 W.O. TAYLOR
92704 JAMES MCAULIFFE
92714 ROBERT MACINTYRE
92714 CHRISTOPHER PESAVENTO
92804 WALTER JELLUM
92804 B.A. WILKINSON
92805 LESTER L. WELCH
92805 HOWARD VIPPERMAN
92806 FUMIHIDE NAKAMURA
92807 PROCOMP SERVICES
92807 BEN LEE
93003 RENE RODRIGUEZ
93004 STEVE NOLL
93117 BRIAN HOOPER
93302 WEBSTER ELECTRONIC CO
93401 JEFFREY W. FISHER
93422 JIM FITZGERALD
93455 FUEL INJECTION
93501 ROBERT J. FLAKE
93534 TRINDEL J. FERGUSON
93555 DIXIE D. GREEN
93555 CARL H. MORLEY
93555 G.D. THIGPEN
93662 TORII MARKET
93704 JOHN M. KIRBY
93727 DOUG VAUGHN
93901 G. PATRICK STANFORD
94087 HUGH FROHBACH
94087 JIM MELLENGER
94087 JOE BYERLY
94102 FRED D. LONSDALE
94103 REID & AXELROD
94107 DENNIS J. LUNSFORD
94108 LINDA HARRINGTON
94109 JACK EASON
94109 RAYMOND S. CRANDELL
94109 LOUIS E. ALMGREN
94110 PAT MALONEY
94114 WILLIAM PARKER
94117 GEORGE GEE
94117 THOMAS G. BALDWIN
94118 ROGER P. FRIEDENTHAL
94118 JERED S. NELSON
94120 WILLIAM HARMON
94122 WALLACE FRIESEN
94122 RICHARD JUMPER
94123 CLYDE STEINER
94131 WILLAM WRAY
94134 JOHN T. ARNOTT
94134 RUTH SCOTT
94303 EUGENE DONG
94306 MILDRED ASH
94306 GLENN C. STEINER
94404 E.W. AMES
94508 RANDY FISCHER
94509 BOB BEAVER
94509 JAY M. HUBERT
94517 STEVEN C. HANSEN
94518 JAMES LIND
94518 LARRY ANUTA
94518 INDUST. GRAPHIC ARTS
94518 LARRY T. NICHOLSON
94519 JOHN FARNHAM
94523 BRUCE M. RAPPAPORT
94530 ARMANDO PICCIOTTO
94533 DICK WILLIAMSON
94536 ROY LATHAM
94578 KENNETH LETSCH
94578 FRANK KOHZAD
94580 CHARLES MCDOWELL
94583 WAYNE HATAYAMA
94585 GEORGE RIDDLE
94585 TONY MERLONGHI
94585 ROBERT GARY
94585 JOHN H BARCLAY
94590 JOE N. HINES
94596 SARAH WASSERMANN
94596 HAROLD JEFFREY
94596 FREEMAN E. GRAY
94598 JEFFREY STRAUS
94598 JOHN SEIDELL
94598 JERRY VAN OSENBRUGGEN
94601 THOMAS J. CROTHERS
94606 JOE TYSL
94609 BRUCE KLICKSTEIN
94609 ANTHONY F. CANTEA
94609 JO ANN LOOS
94611 M.H. CHENEY
94611 ANN GORDON MC STAY
94618 BOB CELLUCCI
94618 SF BAY CHAPTER SIERRA
94619 CLARENCE BOYD
94619 DON PROIA
94703 DANIEL BERKEY
94704 RICHARD M. BETTS
94705 ROBERT MILLS
94705 ARTHUR H. BAZELL
94705 WILLIAM D. LOUGHMAN
94708 JIM CRAIG
94708 ALAN NELSON
94709 RICK NEMCIK-CRUZ
94710 BOB SIMMONS
94720 ALAIN HENON
94803 MELODYE M YAMADA-COHN
94803 KENNETH SILVERSTEIN
94806 HUGH POWER
94806 JAMES M. KELLEHER
95126 LARRY G. GRIFFIN
95128 STANLEY W. GETSLA
95129 RONALD J. CANTONI
95133 RUSSELL D. OESER
95135 GORDON MACBETH
95139 WILLIAM W. KIRKNESS
95140 WAYNE EARTHMAN
95151 DANIEL BROWN
95157 DENNIS ALLEN
95205 COMPLEAT DATA SERV.
95209 ALAN REYBURN
95223 SHEILA STERLING
95340 DAVID D. BROADWATER
95381 JOHN E. BAKER
95401 DON JACKSON
95401 ROBERT LONG
95402 C.R. BRISCOE
95451 STOKES LADDERS, INC.
95476 KENNETH S. MCTAGGART
95540 ALBERT S. BURROWS
95608 IVAN F. GENNIS
95616 THOMAS D. LE CLAIRE
95628 ALAN J. ANTOS
95628 GARY BOYD HAFFNER
95676 ROY CHADWICK
95678 DICK WAGNER
95678 GEORGE ERIO
95688 HARVEY SHAPIRO
95730 LEONARD N. SHAHEEN
95821 ROBERT F. HUNT
95821 HANK SHAPIRO
95826 LUCIEN A. GRONDIN
95826 SACRAMENTO CNTY APCD
95831 GARY KAWAYE
95831 ALAN PRITCHARD
95842 RICK MURRAY
95927 MARC CULLEN
95928 S. JOSEPH TOY
96230 ARTHUR E. GARTNER
96328 STEVEN TOWNSEND

96720 EUGENE PETERSON
96778 W. B. HOWARD
96813 WALTER YOUNG
96817 PHILLIP M.K. LAU
97005 RONALD K. MELOTT
97042 REX BREUNSBACH
97113 PATRICK M. CASTLE
97115 RON MILLER
97116 ROBERT A. BROWNING
97130 RONALD G. LARSON
97138 WARREN KAN
97214 JACK C. RILEY
97225 C.K. SHANKS & ASSOC
97266 MR. JAMES E. MCCASTON
97339 JOE SNYDER
97403 JOHN ALAN HULL
97403 JOEL W. MC CLURE
97470 JIM BAQUE
98004 BOB ANDERSON
98005 J. FRANCIS SMITH
98006 JAMES A. BLOOMFIELD
98011 JOE S. CREAGER
98024 N.W. GRONLUND
98027 W.D. DRUMMOND
98056 FRANK C. SHELTON
98056 DON WERTS
98104 ROBERT SIGLEY
98115 RF SPECIALIATIES
98115 NEIL SMITH
98116 BURKE C. DYKES
98117 PAUL B. ONCLEY
98121 KENNETH L. WALL & CO.
98125 ROY GILLETTE
98136 G.E HUGHES
98145 PAUL S. PERSON
98250 ARTHUR HORNER
98372 WILLIAM LAWSON
98402 LARRY CLARK
98406 ROBERT SCHAEFER
98438 JIMMY GULLEKSON
98501 ROGER G. CALHOUN
98506 RICHARD BOEDEKER
98661 GREGORY HAZEN
98665 THOMAS G. PARSONS
98840 JOE THOMASON
98902 JAMES EVERT
99336 R. CHARLES SMITH
99345 SANDPIPER FARMS INC.
99362 TUCKER LAND CO.
99501 KILO MICROAIR
99502 DAN VANDEMENT
99502 R.D. REEVE
99504 JOSEPH MAGUIRE
99508 JOHN HERRING
99559 BASIC BUSINESS SERV.
99559 GALEN DIRKSEN
99577 EUGENE G. MORRIS JR.
99615 AKSALA ELECTRONICS

B.G. BURFIELD, 5000 AUSTRALIA
DAVID YATES, 4067 AUSTRALIA
BOB WILLSON, 4067 AUSTRALIA
ARMSTRONG & PARTNERS, 2283 AUSTRALIA
JOHN DENT, 3039 AUSTRALIA
TECTRON INDUSTIES, 5000 AUSTRALIA
SCHOOL OF APPLIED SCI, 2616 AUSTRALIA
D & J GALLAGHER, 2519 AUSTRALIA
N.J. FORREST, 2534 AUSTRALIA
K ANDREWS, 2902 AUSTRALIA
JOHN C. BONNETT, 5041 AUSTRALIA
BRIAN COMPUTER AID, 3940 AUSTRALIA
JORDAN COMPUTER BUREAU, 3930 AUSTRALIA
DE CEULAER G J B, B-2530 BELGIUM
ALFREDO DELABARRA SANTIAGO, CHILE
JUAN R. SOTOIONAL, SAN JOSE 1000 COSTA RICA
ALLAN DYSTRUP NIELSEN, DK-2300, DENMARK
JOHN THOMPSON, W1M 7LF UK
T.W. MILLER/MAZUMA LTD, IP15 5PD UK
DALE TRUMBLE, WD3 UK
STANLEY V. HELM, SE9 5PR UK
GORDON J. MITCHELL, 1N13 ONW UK
THEODORE O. ALLEN, HU6 7HJ UK
ROBERT MCLEAN, G77 5NN UK
PIERRE CHAVET, 75782 PARIS FRANCE
RUDI SCHMIT, 5060 WEST GERMANY
HOMER YU HONG KONG
KELVIN K.O. WONG HONG KONG
NETRACO (HONG KONG) LTD HONG KONG
K.C. LAU HONG KONG
ALBERTO GIRLANDO, 35100 ITALY
CHONG PANG, 22-23 MALASIA 25946,24664
CARLOS KUBLIGARFIAS, MEXICO CITY 73, MEXICO
J.J.C.M. BAR, AKERSLOOT, THE NETHERLANDS
JACOBUS SMIT, AMSTELVEEN,NETHERLANDS
BUNSU CHUNG, PHILIPPINES 3117
RUBY U. CASTRO, MANILA, PHILIPPINES
M.J.C. DE MELLO, RIYADH 11423, SAUDI ARABIA
PETER GEHBAUER, RIYADH 11132, SAUDI ARABIA
DJORDJE BLAGOJEVIC, YU-11000, YUGOSLAVIA

Canada

BILL DARKER, L3V 6H3
WILLIAM SCOTT, NOK 1WO
HIROTO SAKA, T6J 3P1
RICHARD DERKSEN, R3N OB3
RONALD HAYTER, V6J 1L5
BOB DAWSON, R9A 1R8
THE FLYER PEOPLE, V6B 3X9
T.M. GUEST, VOJ 1ZO
PETER J. GAUTHIER, E1C 8H7
DAVID H. MCCORMICK, SOK 2KO
CLAY SCHNURR, 72J 2N2
WILLIAM ALLAN JAMES, P7A 7E4
FRED J. LOOKER, N7A 3W9
HAL ARTHURS, COB 1EO
CLAUDE LAMONTAGNE, J3Y 4Z6
NORMAN GOLDSTEIN, H4R 1M3
LORNE GRONER, V9W 4Y9
O. M. FULLER, J4P 3B1
HOWARD LAWRENCE, V1B 1H7
ANTOINE MURRAY, G8Y 2H6
DOROTHY A. JEFFERY, V6T 1W5
G.H. LONGO & CO., T2M 4M2
A.D. LATTA, M5S 1A1
W.C. WARD, S6H 2B8
JAMES E. BOVDHARD, H1N 2G4
KEVIN GRAHAM, H7P 1N3
GARY ENNS, V34 6L4
KEITH MARTINSEN, KOA 3GO
E.F. BURROWS, V5H 2M6
J.D. POPE, B3H 3E3
ROBERT LEDUC, J8H 4G6
MURRAY N. CHARLTON, L7R 4A6
F.C. LEBRON, P. ENG., N6A 2H9
R. WALKER, LOK 2BO
DAVID S. YOUNG, V8R 5X5
KUZIAK LAW FIRM, S4T 1Z4
G.M. NOVAK, M1E 4R5
JOHN F. OHRT, KOE 1XO
ANDRE DANIS, E1C 8T8
OWEN LOW, TOE OMO
BARRY ROBINSON, K8N-4L2

FOR SALE

Dual quad Horizon (64K) with qume QVT-100 terminal (amber). Very good condition with liberal amounts of CP/M and DOS software, documentation and disks thrown in. $1200 or best offer. Also: 2 NorthStar HRAM 64K memory boards (working) $100 each. Call Robert Cowart (415) 540-6667.

Insua

International NorthStar Users Association
Publishers of The Compass Newsletter
PO Box 2910 • Fairfield, CA 94533

Bulk Rate
U.S. Postage
PAID
Walnut Creek
Permit No. 203

TATE 2761
JAMES TATE
23914 SPRING DAY LN.
SPRING, TX 77373

Postmaster: Address correction requested.
Please enter key number on form 3579.