NORTHERN BYTES

Volume 6 Number 3

GREETINGS! Welcome to yet another issue of NORTHERN BYTES. We have a couple of "near announcements" to cover this month.

What's a "near announcement?" It means I'm "jumping the gun" a bit to let you know about some of the things we are working on for your benefit. These announcements are of things not yet finalized, but that seem too important to keep to ourselves until the last minute. Support for the TRS-80 is not dead - in fact, there may be more of it around now than ever before!

Near announcement #1: NORTHERN BYTES and The Alternate Source are working on putting up an international bulletin board for TRS-80 users, to be available via a major packet-switching network here in the U.S. (no, probably not the one you're thinking of), and indirectly accessible (with added surcharges, unfortunately) via other packet networks both here and in other countries. There's even an 800 "toll-free" access number, but it's no bargain, since it carries an $18 per hour access surcharge. If we do manage to get this going, you will have to have a VISA or MasterCard account to which we can bill your usage (we're open to other suggestions!). We hope to be able to make access less expensive than by comparable means (such as the information utilities we're all familiar with). At this point, nothing's been finalized - in fact, we're still in the initial talking stages. If you'd really like to see something like this, why not drop us a line and let us know if you'd be willing to support such a system (provided the costs are reasonable)? If you're really anxious to see something like this, drop us a self addressed stamped envelope (outside the U.S., forget the stamp but enclose an International Reply Coupon or a Canadian quarter) and we will mail you full details if and when we are ready to implement this system.

Near announcement #2: Here's one for NEWDOS/80 users - we will soon be releasing the most significant NEWDOS/80 enhancement package ever. Space does not permit me to go into all the details here, but the feature I have most appreciated about this package is that it has a SUPER Automatic Disk Format Determination routine that just about lets you forget about resetting PDRIVEs to read different disk formats. It will even read a 40 track disk on an 80 track drive, or a single-sided disk on a double-sided drive without manually resetting PDRIVEs. It will even read from and write to disks formatted by an "alien" DOS such as DOSPLUS, LDOS, MULTIDOS, or TRSDOS 6.x (it will not automatically read a TRSDOS 1.3 disk, but the former method of resetting the PDRIVE to copy files to and from a TRSDOS 1.3 disk is still available). This little goodie (which has saved me considerable time in the preparation of NORTHERN BYTES) is just ONE of a whole package of enhancements by Mr. Alan Johnstone. At the present time there are still a few bugs to be worked out, but when we release the package the enhancement programs will be in the public domain, albeit possibly priced a little higher than our normal series of PD disks (we may include some sort of user's manual which will help justify the increased cost).

Not-so-near announcement #3: "The end of NORTHERN BYTES?" I'd hate to have to discontinue it, but I've had a problem. You see, as I've mentioned before, NORTHERN BYTES is basically a one-person operation, and this winter I have been literally swamped with mail (much of it article submissions, for which I thank you all). Trouble is, I simply don't have time to go through all the mail and process it properly. Sometimes it seems like I am spending all my waking hours on NORTHERN BYTES, and I've haven't had any time to do much of anything else - including work on the Public Domain Library disks, or even any programming. This tends to take the fun out of it! And if we get the bulletin board system mentioned above up and running, that will tie up a couple more hours of my time each day.

Up to this point, I have been doing NORTHERN BYTES more for fun than profit (maybe it would be more accurate to say "for the fun of it" and let it go at that!). But what started out as a one or two hour a day project has expanded to take considerably more time. So, it is imperative that some changes be made.

Here's change #1: Effective immediately, HARDCOPY ARTICLE SUBMISSIONS WILL NOT BE ACCEPTED - PERIOD! If you send an article submission, put both the text file and any programs on disk or tape, or send them via MCI Mail. Letters to the editor should also be sent on disk. With disks widely available for under $1.50, this policy shouldn't hurt anyone. Besides, most of us have a few disks that were rejected during formatting - why not try formatting one of those single density and copying your article to that? I am not asking you to use a brand-new, quad-density certified disk - any old, used disk will do so long as it keeps me from having to re-type your article or letter! SuperScripsit users, PLEASE save your text files in ASCII format!

Change # 2: If we are to continue NORTHERN BYTES, we need to generate some money. Why? So we can make this a more-than-one person operation! Unfortunately, most people like to get paid for the work they do (actually, I don't mind it myself) and right now we aren't generating much money! So here is what I am asking you to do: Tell all your TRS-80 using friends about NORTHERN BYTES. Tell them it is only $2.00 per issue and that we will take their VISA or MasterCard number (and expiration date) and send them each new issue as it appears, and bill them $2.00. No big investment, no paying a bunch of money up front and then getting stuck when the publication folds. We bill only for the issues we've sent, after we mail them. On top of that, we've been around long enough that you know we're not just a fly-by-night operation. If you belong to a TRS-80 users group, PLEASE tell the members of your group about us! We don't advertise in any of the TRS-80 magazines (no big ad bucks to spend, especially at the rates they get!) so there are lots of people who never heard of us. If you like NORTHERN BYTES, help spread the word!

By the way, I'm not trying to scare anyone, but I've seen many magazines that have just folded with no advance warning. BASIC COMPUTING (formerly 80-U.S.) disappeared very suddenly, leaving many people (including the folks at Logical Systems, Inc.) holding the bag! I don't want to do that, so I'm letting you know that I'm very seriously thinking about discontinuing NORTHERN BYTES in order to give you a chance to help change my mind. You may have heard of the "Newsletter Editor Burnout" syndrome. Sometimes it's brought on by lack of appreciation (which is not the problem here!), and sometimes it's brought on by the newsletter editor being overworked (ahem!). I really do hate to make NORTHERN BYTES more commercial, but unfortunately, unless we can afford to get some help, your editor's burnout may be terminal. The thing that would be most effective in saving this publication would be if a whole bunch of people suddenly send in their card numbers.

One point to make here - even though The Alternate Source does the actual shipping of and billing for NORTHERN BYTES, I maintain the mailing list in Sault Ste. Marie. So, send your card number and expiration date to the return address for NORTHERN BYTES shown on the back page of this newsletter.

I suppose that eventually, a bulletin board type system could become a replacement vehicle for the type of news provided in NORTHERN BYTES, but I would hate to see that happen, because many users would find it less convenient (not to mention more expensive!) to get their TRS-80 related news that way. I prefer to see the BBS and NORTHERN BYTES complementing each other. But a BBS is generally self-maintaining, in that when a user posts a message, the SYSOP need only read it, determine if its contents are suitable for the BBS, and delete it if necessary. Articles sent to NORTHERN BYTES usually require LOTS of editing to maintain a uniform format. For example, you may have noticed that most assembly-language source code listings that appear in NORTHERN BYTES are in a more-or-less standard format. Well, they very seldom come to us in that format! Similarly, text files usually need to be massaged to obtain a more or less standard format and pleasing appearance. Then everything needs to be checked for spelling and printed out (a very slow process - if we ever start making money, one of the first things I am going to get is a faster printer!). None of this usually needs to be done on a BBS. Granted, there is a certain amount of maintenance-type work that is peculiar

to BBS's, but I suspect it is easier than what is involved in preparing a newsletter (maybe this is a case of "ignorance is bliss" on my part).

In any case, it's up to you, dear reader. If you have any comments on the future of NORTHERN BYTES, the proposed BBS, or anything else, please drop me a line. Or better yet, a text file on a disk or tape, or an MCI Mail letter!

Speaking of MCI Mail, an error by the folks at that organization put us (both NORTHERN BYTES and The Alternate Source) "off the air" for a few days around the end of February and beginning of March. So, if you tried to send us a message and the system wouldn't let you do it, please try again - all should be well now!

THE EXTERMINATOR

With the re-appearance of warm weather, the BUGS come swarming around! Dig out your back issues and your red pencils, and note this month's batch:

There were some problems with Michael Brotherton's WHERE program that appeared in NORTHERN BYTES Volume 5 Number 7 (and on TAS Public Domain Library disk # 005). Please make the following changes in WHERE/ASM and re-assemble:

```
00330        LD    (4020H),HL      ;MOVE CURSOR.....
00331        LD    A,(54H)         ;MODEL 1 OR 3?
00332        DEC   A
00333        RET   Z               ;RETURN IF MODEL 1
00334        LD    HL,40E6H        ;FIX "LAST BYTE EXECUTED"
00335        LD    (BTFIND+2),HL   ;REPLACE MODEL 1 SETTING
00336        LD    (K11+7),HL      ;IN TWO PLACES
00340        RET                   ;LINES

01621        LD    A,1
01622        LD    (WONE),A

01860 DOSCM DEFM 'BASIC,64512,DEFUSR0=&HFC13:X=USR(0):
                                   DEFUSR0=&HFD83'
```

Also, in line 1880 of the source code, change the program version number to 2.2. Thanks to Michael Brotherton for providing these fixes. I will also mention here that Michael has a "shareware" program called FMU that works in a manner somewhat similar to the VFU/CMD program of MULTIDOS, except that FMU works with NEWDOS/80. If you want a copy of this program, send a blank disk and return postage to Mike (Michael Brotherton, 5 Saddle Ridge Road, Wilton, Connecticut 06897). If you like the program, you'll be asked to contribute a small amount of money to Mike, otherwise just erase the disk and you owe nothing.

One more note about WHERE, this from A.F. West, editor of the TRS-80 SYSTEM 80 Computer Group club newsletter (MacGregor, Queensland, Australia). Mr. West points out that if you have WHERE/CMD installed and type in the following line:

```
10 FOR I=1 TO 1O :NEXT
```

(note that the "O" in the number 10 is a letter "O", not a zero as it should be), and then type RUN and <ENTER>, the result will be:

```
10 FOR I=1 TO 1O >>>:NEXT
```

What this means is that in some circumstances, the actual error may be just BEFORE the pointer rather than just after.

Finally, should you decide to change the ORG address in line 100 of WHERE/ASM, keep in mind that the addresses in line 1860 must also be changed by a corresponding amount!

The next three BUGS were imported from Australia! In the article UPGRADING THE TRS-80 MODEL I TO 64K OF DRAMs which appeared in Volume 5, Number 8, page 18, under "Part 2 Straps to be added", line (c) should read as follows:

(c) *RAS to pins 2 & 14 at Z73 strap #9

In MODEL III SELF-BOOTING DISK USING NEWDOS/80 which appeared in Volume 6, Number 1 (page 3), insert the following line in step 3:

Byte 06 - Track number of the /CMD file

Then, on page 7 of Volume 6, Number 1, an article entitled RANDOM I/O should be totally disregarded. Not only are there errors in the circuit diagram, but author Dave Kennedy says that the circuit (as it should have been published) doesn't work in all situations anyway, and that he will be submitting a corrected circuit in the near future.

Turning to last issue's article on the new Model 4/4P ROM image differences, Vern Hester (author of MULTIDOS) passed along some corrections and clarifications that may be of interest. First of all, at 02B5H a LD SP,4288H instruction has been changed to a LD SP,42E8H instruction. Under the "old" ROM, if you were in BASIC and used the SYSTEM command, because the Stack Pointer was set at 4288H the DOS vectors were clobbered. MULTIDOS and DOSPLUS avoided this by leaving the Stack Pointer where it was. This bug would only show up if you were a disk system user, but for some reason decided to load a machine language program from tape using the BASIC "SYSTEM" command.

Second, I noted that at 1BC2H and other places, references to memory location 40B0H were changed to references to 409FH. I had said that 409FH was a previously unused location. Wrong! Vern notes that 409FH is used by the Model III BASIC LIST and LLIST commands, to enable listing of "packed strings", that is, to display graphics as graphics characters when bit 0 is set. Also, bit 2 is set whenever a REM or ' token is encountered. This error (stating that 409FH is unused) also appears on page 67 of TRS-80 ROM Routines Documented (oops!!).

Finally, some corrections to the NEW RS-232 BOOT ROUTINE CODE. At memory locations 3180H-3183H a LD A,6DH instruction followed by a OUT (0EAH),A instruction is found. These initialize the UART for an 8 bit word, one stop bit, parity disabled, transmitter enabled, DTR on, and RTS off. When operating full duplex, RTS off maintains COMM in non-transmit mode, while under half duplex, RTS off maintains COMM in receive mode. Then, a little later on at memory locations 318AH-318DH, a LD A,6CH instruction followed by a OUT (0EAH),A instruction is found. These initialize the UART again with the same settings EXCEPT that RTS is set ON (note that in this case, when bit 0 is at logical 0, RTS is on, while bit 0 = 1 turns RTS off). When operating full duplex, RTS on maintains COMM in transmit mode, while under half duplex, RTS on maintains COMM in transmit mode and inhibits receive. "What is COMM?", you ask. That's whatever is connected to the RS-232 (the TRS-80 is TERM).

Also in the RS-232 BOOT ROUTINE CODE, the comment for line 570 (memory location 37D5H, a PUSH AF instruction) said 'SAVE STATUS & CHARACTER'. Change that to just 'SAVE CHARACTER'. And, at line 640 (memory location 37E0H, a LD A,6DH instruction), the comment given is 'SET "DTR" BACK ON'. Change that to 'SET "RTS" OFF'.

Bob Grommes of the Christian Computer Users Association passed along some information that he got off of CompuServe. Part of this information was in a file on the Writers & Editors SIG, which in many respects duplicates the information in our article on the new ROM image. But I thought I'd pass along an excerpt anyway:

"The Model III ROM is actually three ROMs. ROM A holds the contents of addresses 0000 to 1FFF (all values are given in hexadecimal), ROM B contains 2000 to 2FFF, and ROM C has 3000 to 37FF. There have been several versions of ROM C, but recently, for the first time since the Model III was introduced, Radio Shack has changed ROM A.... The only major change is the printer driver (the interface between the computer and a parallel printer), although there are several minor changes elsewhere.

"Fortunately, the procedures for setting the printer's line length and page length haven't changed. Also, the ROM printer driver is still at 03C2 to 0451, but now occupies other areas as well. The routine to wait for either "printer ready" or the <BREAK> key, formerly at 0440, is now at 01DC, and there are jump vectors at the formerly unused locations 0043 and 0063.

"The new driver handles things slightly differently from its predecessor. With the exception of the carriage return (0D) and the form feed (0C), all control characters (below 20 hex) and non-ASCII characters (above 7F) had been sent to the printer unchanged. These increased the driver's count of the number of characters printed on the current line, although a new line would not be started. Characters from 20 through 7F were handled by a ROM translation table, and would force the start of a new line when necessary.

"The ROM translation table is no longer in the new driver, but all characters below A0 are treated in the same way as before. From A0 to FF, however, the character actually printed depends on two flags, both of which are initially zero but can be changed by the

user. All possibilities will add one to the character count; a new line, with one exception, will be started if that count has reached its limit.

"If the value in location 41FB is greater than one, all characters are sent to the printer unchanged. If that value is zero, characters from C0 to FF have 20 hex subtracted before being printed, while for characters from A0 to BF, the second flag, at 41FC, is checked. If the value in 41FC is not zero, then characters from A0 to BF aren't changed; if is is zero, then 40 hex is added before those characters are printed.

"If the value in 41FB is equal to one, then A0 to BF are unchanged, and E0 to FF are also unchanged but will not cause a new line to be started (this is the one exception mentioned before). If the character falls within the range of C0 to DF, then things get interesting. The address of the translation table is taken from 4220 and 4221 (low-order byte in 4220), and the character to be printed is determined from this table. The translation table used here is not in ROM, but is user-defined. It's 20 (32 decimal) bytes long, with the first byte corresponding to the character C0 and the last to DF.

"There's one change in the new printer driver that I can't understand. The routine at 01DC, as mentioned before, checks for either "printer ready" or the <BREAK> key. Previously, if the printer was not ready, and <BREAK> was pressed, control would return directly to the byte I/O routine at 0694. Now, control returns to the printer driver, and the character and line counters will be updated as if the character had been printed!"

I don't know the author of the above text, so I can't give credit. But Bob mentions that LDOS Support, commenting on the text file, said that the ROM A described in the file is what Radio Shack calls the "international ROM". This ROM A, together with the appropriate ROM C for the target country, has been distributed on Model 4's shipped to foreign markets ever since the earliest days of the Model 4. Apparently they have just decided to use this ROM A domestically also.

LETTERS DEPARTMENT

Starting with this issue, we are requesting that persons sending letters intended for publication send them on magnetic media or via MCI Mail (especially if longer than a couple of paragraphs). If you are NOT using Allwrite (or Newscript) and your word processor offers the option to save your file in ASCII format, please do so (especially if using SuperScripsit!). Your cooperation in this matter will help us to bring you a better newsletter!

Dear Jack,

Why is it that so few monitor programs have search commands where you can specify wildcards? I need this feature often, but except for expensive hardware logic analyzers I have only seen a single program where it was included. The feature is so easy to implement, that it should be a standard feature in all search routines. The letter X could be used as a hexadecimal wildcard character. For example to find calls to file open or close routines you could search for CD2X44. This would also include any call to the Kill routine, but it would save searching for CD2044, CD2444 and CD2844. Similarly, searching for almost all DOS routine calls could be done with CDXX44.

The search is easy to implement. Instead of just a search string, you have a search and a mask string. For each valid digit in the specified search string you put a zero in the mask string and the digit into the search string. For each wildcard you put digit F in the search string and the mask string. When comparing characters, first OR the destination with the mask, and then compare with the search string. Alternatively the AND function could be used by putting F in the mask string for valid digits, and 0 in the search and mask strings for wildcard digits.

If B contains a byte count, HL points to the next byte in memory, IX to the next search byte, and IY to the next mask byte, then the sequence for comparison could be:

```
FIND    EQU  $
        LD   A,(HL)        ;get memory byte
        OR   (IY)          ;mask for wildcards
        CP   A,(IX)        ;compare with searched
        JR   NZ,NOMATCH    ;not found
        INC  HL            ;next destination byte
        INC  IX            ;next search byte
        INC  IY            ;next mask byte
        DJNZ FIND          ;loop for next character
FOUND   EQU  $             ;continue if found
```

Perhaps it could be considered for the next version of TASMON?

..... Tandy has started making their computers convertible from one power source to another. The Model 4 needs to have a jumper moved on the switched mode power supply. I don't know if the same applies to monitors and other peripherals. The main problem with US purchases is the warranty and repairs, which [local authorized repair agencies] are reluctant to perform because too many are bypassing the official importers. I usually restrict overseas purchases to software and small units such as memory expansions or density doublers. All except my LNW 5/8" doubler have been reliable, and now it seems that LNW has gone out of business and taken my doubler with them. My model I is therefore restricted to single density. Perhaps one of your NORTHERN BYTES readers has a spare LNW 5/8" doubler which they would part with for a reasonable amount, say up to $35 plus postage? Mine originally cost about $160 with the Dosplus system.

Kindest regards, Arne Rohde
13 Gwenand Place, Howick, Auckland, NEW ZEALAND

[If any NORTHERN BYTES readers have an LNW doubler for Arne, but don't want to ship it overseas (not that it's difficult), you can send it c/o NORTHERN BYTES and I will re-ship it - but drop a line to Arne first and make sure he still needs it, please! As for the TASMON "wildcard character" question, I've forwarded it to Jonathan Yarden (who is the latest of many people to do work on TASMON!) for his consideration. We'll keep you posted if we actually implement this idea!]

Dear Jack!

Thanks for doing such a good job on Northern Bytes. I enjoy it and use a lot of what you print.

Question! Reference NB Volume 5 Number 4, page 3. Section "TRS-80 Tidbits, Trash, Treasure, and Trivia". At mid-page, John says "Before these patches can be done, the patch to disable password checking must have been applied." And he goes on to list

PATCH #2 (ADD=4ED4,FIND=20,CHG=18)

This patch on my TRSDOS 1.3 does not disable the password challenge on backup requests. What have I missed? Is there a typo error in the patch or is further modification needed? I use a Model 4 in the III mode.

New subject. In Volume 5, Number 5 you asked about interest in Sanyo's. The largest Sanyo SIG in the U.S. is here in Moscow - some 500 members, I am told. I passed your name and address on to one of their group to see if there is interest here.

Chuck Hudson
P.O. Box 9163, Moscow, Idaho 83843

[Haven't heard a word from the Sanyo SIG, so if you want more info, contact Chuck. As for the TRSDOS 1.3 problem, I passed it along to John Hallgren. His reply follows:

"Your inquiry about the "password patch" was referred to me by Jack, and after some research, I think I have the answer. At first glance, there seemed to be nothing amiss, but upon re-reading your letter, the reason became clear. The key words are "backup requests". The 4ED4 patch has a different function, which is hinted at in the next line in the article ('...we can modify any file...'). Since the 4ED4 patch modifies the OPEN/INIT overlay ('*2'), it only works at the file level. The BACKUP/FORMAT routines ('*7') also contain password tests. Using my copy of 'TRSDOS COMMENTED' by Soft Sector Marketing (now out of business), I have come up with the following patch:

PATCH #7 (ADD=55A8,FIND=28,CHG=18)

Please note that this is NOT a tested patch, as I have a Model I and have limited access to a III/4, but it should work. What it does is change a 'JR Z' to 'JR' so that the 'INVALID MASTER PASSWORD' error routine cannot be executed. You will still be prompted for the password as it is a common routine used for checking the source disk password on a BACKUP operation and getting the password for the destination disk on FORMAT operations; however, the results of the source disk password check will be ignored, and the BACKUP will continue as if you had given the proper password. I cannot recall seeing this particular aspect of the TRSDOS 1.3 'protection' scheme discussed before, but the patch for file access has been around since 1982. At this rate, the perfect TRSDOS will be patched together by the year 2010!!

"Hope this answered your question, and if you have any problems with this or any more inquiries about TRSDOS 2.3/2.3B

(Mod I), 2.7/2.8 (Mod I double density), or 1.2/1.3 (Mod III/4), please feel free to contact me at:

John Hallgren
1939 Atlantis Drive
Clearwater, Florida 33575"]

Dear Jack:

I recently upgraded from a Model I to a Model 4 TRS-80. I bought a 16K cassette Model and installed drives from my Model I in it. They have a Track Step Rate of 20 ms.

I acquired a copy of TRSDOS 6, which has a Track Step Rate of 6 ms. built into it. After doing a (STEP=3) and a (BSTEP=3) to it, it still would not load. I had to change the step rate in the boot sector to the proper value. The change is at Sector 1, Byte 9D (memory location 439D): 18=6 ms., 19=12 ms., 1A=20 ms., 1B=30 ms. I made the change with MULTIDOS' ZAP in the Model III mode. With MULTIDOS you don't have to play with PDRIVE configurations.

Kaz Sokolowski
5960 Everwood Road, Toledo, Ohio 43613

OVERSEAS EXPERIENCE

by "Computer Nut"
(Part 3 of a series)

Hardware is not an end in itself, it is simply a means of achieving the desired end result of running suitable software. The most important part of any computer system is the software which is available to run on it.

A common problem for many computer users, especially those living overseas or outside the normal range of computer stores, is finding out exactly which software is suitable for a particular purpose. There are several methods available for assessing the suitability.

The first, and simplest, method is to read computer magazines and look for software reviews. The method is not foolproof, for a number of reasons. Some magazines always print positive reviews, no matter how bad the software. Others only print reviews of software which they have found to be good. Often a review never appears for a particular item of software. Even when a review does appear the reviewer often misses out on some particular point which can be of importance to you, such as catering for special printer drivers, allowing lower case input on a Model I, or respecting the reservation of high memory for other routines.

Of course you cannot expect a reviewer to cover all features of a complex software system, so the next place to go is probably the software publisher. Magazine advertisements are often not a reliable guide, but if you do have specific questions then the software publisher should be able to answer them. This could also be a good test of the support which can be expected after you have bought an item of software. If the publisher does not bother to answer your letter before you buy, then the odds are that you will never get an answer to any questions or problems after you buy.

A short note of advice to advertisers. Please include a postage rate in your advertisements for overseas orders. And if you don't want the problems or the profits involved, then please state clearly that you will only accept orders from within the country.

If you do expect a written answer from another country then you could send along one or two international reply coupons. They should be available in Post Offices in most countries, and can be exchanged in other countries for the postage value of a surface mail letter. Send two coupons if you expect a reply by air mail. If you live in the same country you could send a stamped, self-addressed envelope for the reply. But please don't do what several Americans have done to me. They send an envelope with American stamps on. As far as I know, the only country where U.S. stamps are valid is in the U.S.

If you live close enough to a computer club with members having the same computer system as yours, then you should be able to share software experiences. I have never been in this situation, but I have lived in an area with one or two other TRS-80 users, and I have corresponded with others for, I hope, mutual gain. This is probably the method which gives the best results when evaluating software, but also the area which can cause the most problems.

Let me say at once that I am not an advocate of program piracy as a means of reducing software costs. I have several acquaintances who have bought software packages from the Far East which were obviously pirated. My living is made by writing programs, including programs for TRS-80 users, so obviously I do not approve of piracy. However, I have found that the best method of evaluating software is to try it, and the least expensive method of trying it is to borrow a copy from someone else. There are probably many who would call this piracy, but I should add that if I find a software package useful and intend to continue using it, then I will buy my own copy.

The current state of the TRS-80 marketplace is making it increasingly difficult to obtain suitable software packages. The number of magazines and publications for the TRS-80 has declined rapidly, and the size of the remaining magazines is also diminishing. Reduced size means fewer advertisers, which again means that some of the programs which were once available are no longer on the market. The same, of course, is also true for hardware. A recent experience has left me with a defective disk doubler for a model I somewhere unknown, after being sent for replacement to the manufacturer who apparently has gone out of business. My letters to them remain unanswered.

Perhaps someone else has a solution, but sometimes I feel the need for a listing of all the programs currently available for the TRS-80 series, a listing of all companies still trading, and a listing of companies as they go out of business. One of the best sources I know of for availability of TRS-80 software is a British company called Molimerx Ltd. The address is 1 Buckhurst Road, Bexhill-on-sea, East Sussex, England. They have a catalog of about 190 pages, containing a number of programs I haven't seen advertised elsewhere for some time. There is a nominal charge for the catalog.

A method of testing new software packages, if no other options are available, is to buy them and try them. This can be an expensive method, and can leave you with a number of unused, expensive pieces of software. For example I have a large number of operating systems, but only two of them are used regularly. The others mostly gather dust on the shelf. Probably less than half of the software packages I have purchased are used more than two or three times altogether. For the TRS-80 it seems that very few software publishers produce a demonstration version of their systems. Perhaps this is because software prices are generally lower for the TRS-80 than for CP/M and MS-DOS systems, and perhaps it is because demo versions require extra work to produce. Whatever the reasons, more demo versions of the more expensive systems, with a cost (partly) refundable on purchase, would be appreciated.

USE OF THE SPOOLER ON TRSDOS 6.2.0

[Editor's Note: This file was downloaded from CompuServe by Bob Grommes of the Christian Computer Users Association, who passed it along to us. I assume that it was authored by someone at Logical Systems, Inc.]

It has come to our attention that some users are having difficulty using the system spooler on TRSDOS 6.2.0. These problems fall into two categories:

1) The spooler won't run at all in the back bank (extra memory).

2) The spooler runs fine at the DOS level but blows up in BASIC.

Problem #1:

This is generally a result of a hardware difficulty. We have two machines here which exhibit this problem. Once the spooler is installed, the machine may lock immediately, or run until printer output is attempted.

It appears that the CPU can't reliably execute programs that reside in the extra 64K RAM. Both are very early release machines with one or two wait states, and the problem may be related to the PAL chip not inserting M1 wait states for memory accesses in the alternate banks. Solution: get your CPU board swapped out. It is likely that Radio Shack will charge you for this.

Problem #2:

This one is a software problem. Contrary to the Radio Shack 6.2 manual, back bank spooling was supposed to work with BASIC. It turns out, however, that there is a conflict between BASIC's use of vectored <break> and the spooler. The following patch will allow the use of the spooler in the back bank from BASIC:

PATCH BASIC/CMD.BASIC (D52,83=00 00:F52,83=06 7D)

It may be possible to prevent this conflict in a future release of TRSDOS 6.

In either case, spooling by using a disk file contained on a memDISK in the back bank should work.

SET/RESET AND DIRECT VIDEO CONTROL FOR MODEL 4 BASIC

by Bob Grommes -- B & G Microsystems
1733 Eastern S.E., Grand Rapids, Michigan 49507
CompuServe 74126,72

It has always been something of a challenge to me to achieve absolute control over the hardware using lowly old BASIC. Without the documentation to create some kind of interface with the DOS, this had been relatively difficult on earlier machines like the Model I and III until IJG and others provided the documentation that Tandy wouldn't -- such as Microsoft BASIC Decoded and BASIC Faster and Better. About the time the Model 4 was introduced, Tandy's attitude of paranoid secrecy began to soften, and even the "official" Tandy coverage of the Model 4 hardware and DOS (I refer the the Model 4 Technical Reference Manual) was superb in comparison to the past. I've since collected a wealth of other tidbits from my investigations and the work of others, but am amazed how little has been published from the perspective of the Model I/III programmer trying to shift his mental gears to work on the Model 4. If you're like me, you've developed some favorite techniques you'd like to use on the 4, but just haven't the time or resources (or perhaps the heart) to start over from scratch and get those techniques working on the Model 4. Well, here's a fairly satisfying solution to this lack. Hang in here with me and when we're all done, you'll have a simple method of calling machine language routines from Model 4 BASIC that will give you much more power, flexibility and sophistication in designing displays than you have had until now.

We're going to concern ourselves here with taming the Model 4 video display. While some of you have been happily PEEKing and POKEing your Model I/III video display, pointing strings to it and so forth, you find this quite impossible with the Model 4, because the video RAM is not "memory mapped". A fair amount of writing has been done about the Model 4 video, including a fairly decent article in 80 Micro a few months back, so I won't go into great detail here. To make a long story short, there is a 3K RAM in the Model 4 that has the keyboard and the video display "mapped" to it. In order to read or write to this RAM, we send certain values out port 84H. The hardware then "bank switches" this RAM so that it can be addressed in high RAM starting at F400H and extending all the way to FFFFH, the top of physical memory. The video RAM actually goes from F800H through FF7FH (1920 bytes). The area from F400H through F7FFH is your keyboard RAM, and the area ABOVE video RAM (FF80H through FFFFH) is somewhat mysterious. Under TRSDOS 6.2, it is said to hold the keyboard type-ahead buffer and "other system buffers". Under previous versions, nothing I know of was said about this area but I believe it was unused by TRSDOS. A very few applications stashed data and short machine language routines here and won't work under TRSDOS 6.2 for this reason.

Anyway, while this 3K RAM is switched in, the previous contents of the high memory it shadows are "lost" until the video/keyboard RAM is swapped back out. Since most users have some kind of high memory drivers or filters active in this area, keyboard and video access routines must be very careful or they will clobber code that the system needs to function when the video/keyboard RAM is switched in. A little reflection will tell you that the DOS has quite a task keeping all the various system and user requests from colliding with the need to almost constantly scan keyboard and video. Also, our ability to directly access this RAM from BASIC is really quite restricted and inflexible because there are so many things we just can't to at the same time as video RAM is available. Any kind of I/O to or from system devices or the disks will cause a conflict. Any BASIC function which happens to call DOS routines may cause trouble. And because BASIC wasn't written with our kind of shenanigans in mind, IT may undo our configuration at any time in the course of its normal housekeeping.

Still, for certain purposes, a direct video access method from BASIC could be useful -- for example, the old trick of POKEing a value in the very last screen position to avoid scrolling. So before we go to a discussion of how to get TRSDOS 6 to handle video access FOR us, we'll demonstrate it the HARD way first. At the same time, we'll demonstrate the logic behind SET and RESET. These functions work too slow when written in BASIC, but it's easier to understand the logic behind them if we work through them that way and then explain the assembly language version.

DGRAPH/BAS is actually two demonstration programs in one. The lines from 500 on are a self-contained program; to use it you have to load DGRAPH/BAS and RUN 500. But for now let's look at the code prior to line 500, which demonstrates direct video access from BASIC.

The purpose of this module is to draw a border around the outer edge of the Model 4 display, beginning in the upper left hand corner, and then erase it, using pixel SET and RESET. Program logic actually begins at line 100, where the first thing we have to do is keep BASIC from using addresses above F3FFH, where video/keyboard RAM will soon reside. The CLEAR statement is equivalent to entering BASIC with the command BASIC (M=X'F3FF'). In your applications, you could have machine-language routines or data stored in the area F400H to HIGH$, as long as you won't use them during the direct video access.

Next, we do what I feel BASIC should do by default, and consider all variables an integer unless otherwise specifically defined. This is assumed by all our variable passing techniques later on, so be forewarned. Next, in an effort to squeeze every bit of speed we can out of this program, we DIMension all the variables we will use in the order we want them in the variable lookup table. Finally, we set the variable VIDEO to the starting RAM address of video memory. Then the GOSUB 10 instruction will define several user functions. Ignore these for the moment, we'll get back to them shortly.

Now in lines 110 - 130 we do the actual bank switching. The idea is: (1) disable interrupts. This will get rid of TRSDOS' constant access of the video to blink the cursor, and will also stop any interrupt driven tasks currently active in high memory so we won't clobber them. (2) Determine the current status of port 84H, tweak a couple of bits, and output that value to switch in video RAM.

Unlike the Model I, which has CMD functions to enable/disable interrupts, we need a short machine-language routine (Wait! Come back!) to do the job. This is a huge, two-byte routine, so we store it in an integer variable in line 110. The assembler mnemonics for this routine is simply DI followed by RET. The VARPTR of DI.Z80 will serve as the entry address to this routine. Although unused in this demo, EI.Z80 is loaded with the instructions to re-enable interrupts (EI followed by RET). Naturally you have to switch video memory back out and re-enable interrupts once you're done fooling around in video RAM, but using a normal video output instruction (such as PRINT) or an END statement will do this anyway, so you usually don't have to explicitly bother with the chore.

In line 130, we take a look at the byte in memory location 78H. In all releases of TRSDOS 6 to date (including 6.2.0) this location has contained the last value sent to port 84H. There is no absolute guarantee that this location won't move in future releases of the DOS; it is part of the system flags table which is supposed to be located with a machine-language call to the @FLAGS routine. (Officially, it's the "O" flag, usually labelled OFREG$). This is the one big thing they didn't tell you in the 80 Micro article; if you want to insure your program will run under any future release of TRSDOS 6, you have to at least call a machine-language routine to pinpoint the location of this flag. Anyway, in this case we'll risk future portability for the sake of illustration. We take the value at 78H, and send that value out port 84H after first adjusting the proper bits which will tell the hardware to bring in the video RAM. At this point we have temporarily lost any previous contents of memory above F3FFH, and this area is now addressable as keyboard and video. Note that we didn't actually change the value at 78H; if we were to send that value back out port 84H we would de-select the keyboard video RAM.

Now we have to draw our border around the edge of the display. How do we turn on the proper pixels? First, we have to poke graphics blanks (128 decimal) around the edges of the display. For those who haven't gotten into the theory behind direct pixel addressing, it goes like this: Assuming we have a video location with a value in the range 128 through 191 decimal in it, we can turn any of the six pixels at that position on or off by setting or resetting the appropriate bit. If we number the pixels like so:

```
0 1
2 3
4 5
```

. . . then setting bit 0 would turn on pixel 0, setting bit 4 would turn on pixel 4, and so on. In lines 132 and 134, we are accomplishing 2 things: we are making sure that no non-graphics characters are at the screen edge, and that all pixels are off.

The balance of this first demo through the END statement in line 220 draws and then erases the border using user-defined functions which mimic Model III SET and RESET statements as closely as possible. Again, although unused in this demo, the function POINT has been defined to describe how it would work. Let's take a closer look at the functions.

The VIDRAM function calculates the memory address of a given screen location, given the screen row and column. SET, RESET and POINT then use this function to PEEK at the proper byte on the video display and modify it. With SET and RESET we must then poke the changed byte back into the same location.

We pass the X and Y coordinates of the pixel we want to work with to these functions, so their first task is to convert X and Y to the appropriate row and column to pass to the VIDRAM function. Second, we must determine which bit WITHIN that byte we are going to deal with. At this point I'm probably going to loose you if you don't get a pencil and paper and try a few examples of what I'm about to discuss to convince yourself it works. It's easier to SEE how it works than to understand the theory behind it. Once you SEE it, the theory isn't so bad.

We can determine the video character row by dividing Y by 3 and ignoring any remainder (each video character position is three pixels high). Similarly, we can calculate the video column by dividing X by 2 and ignoring any remainder, because each video character is two pixels wide. Fortunately, since Model 4 BASIC supports integer division (note the backslashes we used instead of the normal forward slash) it's easier and much faster than using FIX, as we would have had to under Model III BASIC.

Now that we have the proper video byte, how do we find the correct bit within that byte? Well, just pretend that the byte we are dealing with is a tiny video display, two columns wide by three rows deep. Remember those remainders we just threw away? Well, we shouldn't have, because they are our pixel row and column. Again, Model 4 BASIC to the rescue. The MOD (short for modulus) operation gives us the remainder of any integer division, so we don't have to develop a remainder function. So Y MOD 3 yields the pixel row (0-2) and X MOD 2 yields the pixel column (0-1). Just to be difficult (and speed things up a bit), I didn't actually use X MOD 2 to arrive at the pixel column; I used the alternate method of ANDing X with 1. ANDing any number with 1 will yield 1 if the number is odd, 0 if it's even. Since the only two possible outcomes are 0 or 1, this method works for the pixel column and is faster since it avoids a software-intensive division operation.

Finally, we just take the pixel row, multiply it by two and add the pixel column to it. This yields a number 0 through 5, which happens to be the bit we are looking for. Think of this operation as a mini-VIDRAM function for our imaginary mini-video display.

Now the only dirty work left is to set, reset or test the bit we have just calculated. We do this with our logical operators. The method is borrowed from Rosenfelder's book, Basic Faster and Better which, if you don't have a copy of it, you should. (Lousy English, but you get the point). To make a long story short:

To set any bit in a byte: New.byte = Old.byte OR 2 to the power of bit
To reset any bit in a byte: New.byte = Old.byte AND NOT 2 to the power of bit
To test any bit in a byte: Value = Old.byte AND 2 to the power of bit

So, believe it or not, we went through all that just to address ONE pixel! Now run the 1st demo and see how long it takes to address ALL the pixels at the outer screen edge TWICE! Hmmm . . . clearly not anywhere near as fast as Model III built-in pixel functions, since it takes about 40 seconds to draw and un-draw our border. Looks like we need machine-language speed on this one.

Well, if we're going to all that trouble, we might as well cover all the other things we can do to video from Model III BASIC that we can't directly do from Model 4 BASIC. The assembler code for VDCTL/OBJ which accompanies this article gives us complete control over the Model 4 video, with SET, RESET, POINT, video PEEK and POKE, scroll-protect, and the facilities to copy back and forth between video and RAM (or BASIC string variables). The second BASIC demo (line 500 to the end) loads VDCTL/OBJ and calls two of its several routines to do the same job as the first demo in about one tenth the time. In fact it works out about perfect, by my benchmark . . . about twice as fast as Model III SET and RESET, which is just about how fast it needs to be, since we have almost twice as many pixels to deal with. Let's bypass the assembler code for a bit and concentrate on the BASIC interface. If you don't know (or don't care to know) Z-80 assembly language, just concentrate on the methods for making practical use of VDCTL. Hopefully those of you interested in assembly language -- beginners and old-timers alike -- will learn a few tricks and get more comfortable "talking" with TRSDOS 6.

Taking a look at line 500, the entry point for our second demo, you will notice we are reserving memory above F3FFH again. This is not because we have to protect video memory (TRSDOS will take care of that matter for us this time) but only because we decided our VDCTL machine code will reside in memory starting at F400H. We could have put it anywhere within reason, but I selected this location because it's low enough to stay below the system HIGH$ pointer (unless you have a lot of drivers or filters in high memory) and still high enough to keep from stealing too much memory from BASIC.

In line 510 we load VDCTL/OBJ and define several variables. These all represent the entry points to the various routines supported by VDCTL. Here's one good thing about long variable names -- it's immediately apparent what the purpose of each routine is.

Line 520 clears the screen and turns the cursor off, and the balance of the program gets down to business and draws, then erases, our border -- ah, THAT'S more like it!!

All the VDCTL routines use the BASIC keyword CALL for access, rather than USR. Why CALL? It's more flexible and self-documenting. It's also easier to understand and use. About the only advantage I can see of USR over CALL is that USR can be used in expressions -- for example, IF NOT USR7 THEN . . . or even DEF FNX=Y*USR7(Z). USR, however, has several limitations: you can't directly pass more than one variable to the subroutine, or receive more than one value back from it. You can't directly pass string variables to or from a routine called by USR. You are limited to 10 simultaneously defined USR functions in a program. Another factor is that, at least in Microsoft's thinking, USR seems to be on the way "out". The Microsoft BASIC manual that comes with MS-DOS, for example, says that USR is provided "mainly for compatibility with older programs" and that CALL "is the recommended method" for interfacing with machine-language routines. Thus, if you use USR today, it may be obsolete tomorrow. Nuff said.

CALL allows us to pass any number of variables of any type (including strings) to and from our machine language subroutine in a very simple and flexible manner. The format is CALL ROUTINE(variable list . . .) where ROUTINE is an integer variable that points to the entry address of the desired routine and "variable list" is one or more variables to be passed to the subroutine and/or receive values back from it. (If you don't provide a variable list, no values are passed). There are some limitations: arguments to CALL must be variables, not constants. Also, they must be declared variables -- that is, when you use the command CALL SET(X,Y), X and Y must have been previously assigned a value somewhere in the program. BASIC normally assumes a value of zero in such cases, but not with CALL -- you'll get an Illegal Function Call error.

Below are the syntax and operation of each of the functions in VDCTL -- the entry points for each are as defined in line 510 of the demo program. A few points to keep in mind: Except for VIDTORAM, all values passed to VDCTL are character values (0 to 255 decimal). VDCTL ignores the Most Significant Byte of these variables and thus "sees" them as modulo 256. Except for VLINE, absolutely no error checking is done for invalid values passed (for example, a SET or RESET outside the boundaries of the display). In turn, the TRSDOS routines used by VDCTL do minimal error checking also. It's your responsibility to pass VALID values to VDCTL! VLINE checks for a target string that's too short but doesn't check for an invalid video display line.

SET -- Turns on a pixel at a specified location on the video display. The call format is CALL SET(X,Y) where X is the X coordinate (0 thru 159) and Y is the Y coordinate (0 thru 71). If there is a non-graphics character at that particular position on the display, it will be be treated as if it had been a graphics "all off" character (128 decimal).

RESET -- The inverse of SET; turns off a pixel at a specified location. Same call format as SET. If there is a non-graphics character at that particular position on the display, it will be replaced with a graphics "all off" character (128 decimal).

POINT -- Test a pixel at a specified location on the video display. The call format is CALL POINT(X,Y,PIXEL) where X is the X coordinate, Y is the Y coordinate, and PIXEL is a variable to hold the results of the test. On entry, X may be 0 thru 159, Y may be 0 thru 71, and the contents of PIXEL is unimportant. On exit, X and Y are unchanged and PIXEL will be 0 if the tested pixel was off. If the tested pixel was on, the variable PIXEL will be non-zero.

VPEEK -- Read the value at a given position on the video display. This is like a PEEK to video ram on the Model III. The format is CALL VPEEK(RW,CL,CHAR) where RW is the video row (0-23) and CL is the video column (0-79). The value at that position will be returned in CHAR.

VPOKE -- The inverse of VPEEK, works like a POKE to video RAM on the Model III. Format: CALL VPOKE(RW,CL,CHAR) where RW is the video row, CL is the video column and CHAR is the value to place there. Note that this function completely bypasses the video driver so it does not move the cursor and all values less than 32 and greater than 191 placed on the display with VPOKE will result in the corresponding special character being displayed.

CURSOR -- Change the cursor character and store the old cursor character. You can change the cursor character with the statement SYSTEM "SYSTEM (BLINK=x)", but the CURSOR call is faster (no disk access involved; it's instantaneous) and you can also save the current cursor character for later recall. For example, suppose your program wants to use a large cursor such as 191, but when your program is finished, you want to restore the cursor character to whatever it was when the program was executed. You could just assume it was 95 (the default cursor under TRSDOS 6.2) but it might have been 176 (the default cursor under earlier TRSDOS 6 releases) or the user might have changed it to some other character of his own preference. Our calling format for this one is CALL CURSOR(NEW.CURSOR,OLD.CURSOR) where NEW.CURSOR is the new cursor character, and OLD.CURSOR is a variable to hold the old character. NEW.CURSOR must, of course, be a value of 0 to 255; as with VPOKE, all values result in a character of some kind. On entry, the contents of OLD.CURSOR are unimportant and on exit, OLD.CURSOR will have the cursor character just replaced. Later, a call such as CALL CURSOR(OLD.CURSOR,NEW.CURSOR) will reverse the action of the first call. Important note: TRSDOS Version 6.0.0 doesn't return the old cursor value, so using this call under that particular release of the DOS will return garbage values for OLD.CURSOR. All releases from 6.0.1 on return the old cursor value correctly. (Quick aside: Two undocumented features of Model 4 BASIC: You can legally use a period in a variable name, and you can substitute square brackets for parentheses when dealing with array variables).

PROTECT -- Scroll protect a specified number of lines at the top of the video display. Format: CALL PROTECT(LINES) where LINES is the number of lines you want to protect. This value must be in the range 0 thru 7 (larger numbers are treated as modulo 8). This call does not alter the appearance of the display in any way, but subsequent scrolling will not have any effect on the top number of lines you specified. CLS will still clear the entire screen and you can still PRINT normally to positions in the scroll-protected area; however, the ONLY way to remove the scroll protect is to CALL PROTECT(LINES) with LINES=0, or reboot the system. By the way, the 7 lines is a limitation of TRSDOS, not VDCTL. Break for an editorial: why this seemingly arbitrary limitation? Why not let you protect any number of lines? Why not allow top AND bottom scroll protects? For that matter, why not specify the four corners of your actual display area so you could create a window -- even the primitive Apple II allows you to do that!!

VLINE -- Copy a line of the video display into a BASIC string variable. Format: CALL VLINE(VL$,LN) where VL$ is a string which has been pre-defined to a length of at least 80 (actual contents irrelevant) and LN is the line of the video display you want stored in the string. VLINE returns without doing anything if the string is less than 80 bytes long. If you have some reason to do so, you can reverse the direction of the copy by issuing the command POKE VLINE+17,0. Once you do this poke, your 80 byte string VL$ will be placed on line LN of the display. Of course for most purposes you can do the same thing by just using the PRINT @ statement. To restore VLINE to its normal function, POKE VLINE+17,1. IMPORTANT NOTE: This function works ONLY under TRSDOS 6.2!

RAMTOVID -- Copy a 2K area of RAM to the video display. With some skillful manipulation, you can create a buffer area in high memory (say, in a string array), construct your desired screen there, then instantaneously copy that area to the display. The format is CALL RAMTOVID(BUFFER) where BUFFER is the starting address of the RAM area to be copied to the display. BUFFER must be greater than 23FFH and less than EC00H.

The RAMTOVID function can be reversed with a POKE RAMTOVID+7,6. (To restore normal operations, POKE RAMTOVID+7,5). You must be careful with this one. You have to properly reserve a RAM buffer that will hold the video display. The buffer must begin BETWEEN 23FFH and EC01H. For the purposes of interfacing with BASIC, you might as well locate it right under your machine language routines and reserve the needed memory with the CLEAR statement. That's another reason I chose F400H as the starting address for VDCTL. The BASIC statement CLEAR, &HEBFF would allow you to have your video display buffer from EC00H through F3FFH. This is exactly 2048 bytes. If you had issued the above mentioned CLEAR statement at the start of your program, and BUFFER=&HEC00, you would instantly have a copy of the video display just below VDCTL in memory.

The method used in the paragraph above will work with any release of TRSDOS 6. There are some wrinkles, though. Releases PRIOR to 6.2.0 copied 2048 bytes into the buffer, so you would get the 1920 bytes of video and the 128 "mystery bytes" immediately following video. Version 6.2.0, however, copies only the actual 1920 bytes of video RAM. This means you only need reserve 1920 bytes, instead of 2048; HOWEVER, the revised Technical Reference Manual STILL gives EC00H as the highest possible starting address for a RAM video buffer. Was this an oversight in documentation, or a bug in the DOS itself?

After some testing, I found that, at least under release 6.2.0, you can have your buffer start as high as EC80H, which is exactly 1920 bytes below F400H. In fact, no error trapping is done if you specify a higher address, so you CAN do strange, dangerous and mostly useless things to your program if you aren't careful.

Anyway, for the benefit of those who want to customize (excuse me, hack) on the assembly language source code, I'm going to mention a few points regarding the DOS interface. Due to the space we've already taken, I'm not going into great detail; you should have a copy of the Model 4 Technical Reference Manual if you seriously want to write this kind of code, anyway.

All of the routines in VDCTL use just one TRSDOS 6 supervisor call (SVC). The name of it is @VDCTL (ViDeo ConTroL) and it's probably the most multi faceted SVC in the entire DOS. Actually, this SVC has several functions we haven't used, but they are easily available from BASIC (positioning the cursor and determining the current cursor position).

The following should be kept in mind about all our calls to @VDCTL: None of the @VDCTL functions used in our code changes the current cursor location; AF and B are destroyed by all the functions of @VDCTL; and all calls to @VDCTL are done with 15 decimal loaded into the A register (to select @VDCTL, which is SVC #15) and the desired @VDCTL function number (1 thru 9) in the B register. All calls to @VDCTL which deal with a particular row and column location on the display are entered with H=desired row and L=desired column.

Since neither TRSDOS nor BASIC provide any facilities for direct pixel addressing on the Model 4, I had to create a routine which I call CALCADDR to handle the number-crunching involved. SET, RESET and POINT all call it as the first order of business. This routine gets the X and Y coordinates passed by BASIC's CALL verb and translates them into the appropriate video row and column, handily left in HL, just where @VDCTL needs them to be. It also leaves the number of the bit we have to address at that location (0 thru 5) in the C register. SET, RESET and POINT then take the appropriate action on the proper pixel by performing the correct logical operation on the byte at video row, column. The method of using a lookup table for this operation, as well as most of the math in CALCADDR, is adapted from a routine devised by Barden for the Model III in More TRS-80 Assembly Language Programming, page 201.

How does BASIC pass variables with the CALL verb? Upon entry to our assembly code, HL points to the VARPTR of the first BASIC variable; DE points to the VARPTR of the second variable and BC points to the VARPTR of the third variable. If there are more than three variables, BC will point to the start of a block of memory containing the 3rd through Nth VARPTRs, one after another. To understand how to read or alter these variables, you have to understand the structure of BASIC VARPTRs, which is beyond the scope of this article. However, studying the source to VDCTL/OBJ should give you an idea or two of how to go about it.

As a final note, notice that we gave the last RET instruction in the code a label, DOSENTRY, and used that label in the END statement. If the user attempts to execute the object code module from TRSDOS Ready, all that happens is that the program is loaded, control transferred to DOSENTRY, and the RET instruction takes him right back to TRSDOS Ready.

Your comments and/or enhancements to this code are appreciated. Have fun!

[DGRAPH/BAS]

```
1 'Demonstration of direct access of Model 4 video RAM from BASIC
2 'Bob Grommes, 1733 Eastern SE, Grand Rapids, MI 49507-2029
3 '    "RUN" for Demo #1; "RUN 500" for Demo #2
5 GOTO 100
10 DEF FNVIDRAM(RW,COL)=VIDEO+((RW*80)+COL)
20 DEF FNSET(X,Y)=(PEEK(FNVIDRAM(Y\3,X\2))) OR 2^(((Y MOD 3)*
2)+(X AND 1))
```

```
30 DEF FNRESET(X,Y)=(PEEK(FNVIDRAM(Y\3,X\2)) AND NOT 2^((Y
 MOD 3)*2)+(X AND 1))
40 DEF FNPOINT(X,Y)=(PEEK(FNVIDRAM(Y\3,X\2)) AND 2^((Y MOD
 3)*2)+(X AND 1))
50 RETURN
100 CLEAR,&HF3FF:DEFINT A-
Z:DIM X,Y,VIDEO:CLS:VIDEO=&HF800:GOSUB 10
110 DI.Z80=CVI(CHR$(&HF3)+CHR$(&HC9)):EI.Z80=CVI(CHR$(&HFB)+C
HR$(&HC9))
120 DI=VARPTR(DI.Z80):CALL DI
130 OUT &H84,(PEEK(&H78) AND &HFC) OR 2
132 FOR X=0 TO 79:POKE FNVIDRAM(0,X),128:POKE FNVIDRAM(23,
X),128:NEXT
134 FOR X=0 TO 23:POKE FNVIDRAM(X,0),128:POKE FNVIDRAM(X,7
9),128:NEXT
140 FOR X=0 TO 159:POKE FNVIDRAM(0,X\2),FNSET(X,0):NEXT
150 FOR Y=0 TO 71:POKE FNVIDRAM(Y\3,79),FNSET(159,Y):NEXT
160 FOR X=159 TO 0 STEP -
1:POKE FNVIDRAM(23,X\2),FNSET(X,71):NEXT
170 FOR Y=71 TO 0 STEP -
1:POKE FNVIDRAM(Y\3,0),FNSET(0,Y):NEXT
180 FOR X=0 TO 159:POKE FNVIDRAM(0,X\2),FNRESET(X,0):NEXT
190 FOR Y=0 TO 71:POKE FNVIDRAM(Y\3,79),FNRESET(159,Y):NEXT
200 FOR X=159 TO 0 STEP -
1:POKE FNVIDRAM(23,X\2),FNRESET(X,71):NEXT
210 FOR Y=71 TO 0 STEP -
1:POKE FNVIDRAM(Y\3,0),FNRESET(0,Y):NEXT
220 END
500 CLEAR,&HF3FF:DEFINT A-Z:DIM X,Y,SET,RESET,POINT,PIXEL
510 SYSTEM "LOAD VDCTL/OBJ":SET=&HF400:RESET=&HF424:POIN
T=&HF448:VPEEK=&HF46B:CURSOR=&HF478:VPOKE=&HF481:PROTE
CT=&HF491:VLINE=&HF498:RAMTOVID=&HF4AB
520 CLS:PRINT CHR$(15);
530 Y=0:FOR X=0 TO 159:CALL SET(X,Y):NEXT
540 X=159:FOR Y=0 TO 71:CALL SET(X,Y):NEXT
550 Y=71:FOR X=159 TO 0 STEP -1:CALL SET(X,Y):NEXT
560 X=0:FOR Y=71 TO 0 STEP -1:CALL SET(X,Y):NEXT
570 Y=0:FOR X=0 TO 159:CALL RESET(X,Y):NEXT
580 X=159:FOR Y=0 TO 71:CALL RESET(X,Y):NEXT
590 Y=71:FOR X=159 TO 0 STEP -1:CALL RESET(X,Y):NEXT
600 X=0:FOR Y=71 TO 0 STEP -1:CALL RESET(X,Y):NEXT
610 PRINT@0,CHR$(14);
```

[Source code for VDCTL/OBJ]

```
                00001 ; VDCTL/OBJ — @VDCTL interface for Model 4 BASIC -- 02/26/84
                00002 ;
000F            00003 @VDCTL    EQU   15
F400            00004           ORG   0F400H
F400 CD85F4     00005 SET       CALL  CALCADDR      ; Get video pos & bit addresses
F403 3E0F       00006           LD    A,@VDCTL      ; Get video character
F405 C5         00007           PUSH  BC            ; BC altered by @VDCTL
F406 0601       00008           LD    B,1
F408 EF         00009           RST   40            ; Video character now in A
F409 C1         00010           POP   BC            ; Restore C
F40A E5         00011           PUSH  HL            ; Save row,column
F40B FE80       00012           CP    128           ; If A<128 or >191, make it 128
F40D 3806       00013           JR    C,FORCE128
F40F FEC0       00014           CP    192
F411 3002       00015           JR    NC,FORCE128
F413 1802       00016           JR    VALID
F415 3E80       00017 FORCE128  LD    A,128
F417 21D1F4     00018 VALID     LD    HL,SETMASK    ; Point to start of table
F41A 09         00019           ADD   HL,BC         ; Index to desired mask
F41B B6         00020           OR    (HL)          ; Set pixel
F41C 4F         00021           LD    C,A           ; Put the character back
F41D 3E0F       00022           LD    A,@VDCTL
F41F 0602       00023           LD    B,2
F421 E1         00024           POP   HL            ; Restore row,column
F422 EF         00025           RST   40
F423 C9         00026           RET                 ; Back to BASIC
                00027
;===============================================================
F424 CD85F4     00028 RESET     CALL  CALCADDR      ; Get byte and bit addresses
F427 3E0F       00029           LD    A,@VDCTL      ; Get video character
F429 C5         00030           PUSH  BC            ; BC is altered by @VDCTL
F42A 0601       00031           LD    B,1
F42C EF         00032           RST   40
F42D C1         00033           POP   BC            ; Restore C
F42E E5         00034           PUSH  HL            ; Save row,column
F42F FE80       00035           CP    128           ; If A<128 or >191, make it 128
F431 3811       00036           JR    C,PUT128      ;  and skip the reset
F433 FEC0       00037           CP    192
F435 3810       00038           JR    NC,PUT128
F437 21D7F4     00039           LD    HL,RESETMASK  ; Point to start of table
F43A 09         00040           ADD   HL,BC         ; Index to desired mask
F43B A6         00041           AND   (HL)          ; Reset pixel
F43C 4F         00042 PUTBYTE   LD    C,A           ; Put the character back
F43D 3E0F       00043           LD    A,@VDCTL
F43F 0602       00044           LD    B,2
F441 E1         00045           POP   HL            ; Restore row,column
F442 EF         00046           RST   40
F443 C9         00047           RET                 ; Back to BASIC
F444 3E80       00048 PUT128    LD    A,128
F446 18F4       00049           JR    PUTBYTE
                00050
;===============================================================
F448 C5         00051 POINT     PUSH  BC            ; Save pointer to T/F variable
F449 CD85F4     00052           CALL  CALCADDR      ; Get byte and bit addresses
F44C 3E0F       00053           LD    A,@VDCTL      ; Get video character
F44E C5         00054           PUSH  BC            ; BC altered by @VDCTL
F44F 0601       00055           LD    B,1
F451 EF         00056           RST   40
F452 C1         00057           POP   BC            ; Restore C
F453 FE80       00058           CP    128           ; If char <128 or >191, return false
F455 3810       00059           JR    C,PIXELOFF
F457 FEC0       00060           CP    192
F459 380C       00061           JR    NC,PIXELOFF
F45B 21D0F4     00062           LD    HL,POINTMASK  ; Point to start of table
F45E 09         00063           ADD   HL,BC         ; Index to desired mask
F45F A6         00064           AND   (HL)          ; A=0 if pixel off, <>0 if on
F460 E1         00065 LOADVAR   POP   HL            ; Point HL to BASIC T/F variable
F461 77         00066           LD    (HL),A        ; Load the BASIC T/F variable
F462 23         00067           INC   HL
F463 3E00       00068           LD    A,0
F465 77         00069           LD    (HL),A
F466 C9         00070           RET                 ; Back to BASIC
F467 3E00       00071 PIXELOFF  LD    A,0
F469 18F5       00072           JR    LOADVAR
                00073
;===============================================================
F46B C5         00074 VPEEK     PUSH  BC            ; Save 3rd argument
F46C 46         00075           LD    B,(HL)        ; B = Row
F46D EB         00076           EX    DE,HL
F46E 4E         00077           LD    C,(HL)        ; C = Column
F46F C5         00078           PUSH  BC            ; Move BC to HL
F470 E1         00079           POP   HL
F471 0601       00080           LD    B,1           ; Setup for SVC
F473 3E0F       00081           LD    A,@VDCTL
F475 EF         00082           RST   40            ; Take a peek
F476 18E8       00083           JR    LOADVAR       ; Now return result & back to BASIC
                00084
;===============================================================
F478 4E         00085 CURSOR    LD    C,(HL)        ; C = new cursor character
F479 D5         00086           PUSH  DE            ; Save return variable
F47A 0608       00087           LD    B,8           ; Set up for SVC
F47C 3E0F       00088           LD    A,@VDCTL
F47E EF         00089           RST   40            ; Old cursor character now in A
F47F 18DF       00090           JR    LOADVAR       ; Return old cursor & back to BASIC
                00091
;===============================================================
F481 C5         00092 VPOKE     PUSH  BC            ; Save pointer to value to be poked
F482 46         00093           LD    B,(HL)        ; B = Row
F483 EB         00094           EX    DE,HL
F484 4E         00095           LD    C,(HL)        ; C = Column
F485 C5         00096           PUSH  BC            ; Save Row & Column in AF
F486 F1         00097           POP   AF
F487 E1         00098           POP   HL            ; Point to value to poke
F488 4E         00099           LD    C,(HL)        ; Put poke value into C
F489 F5         00100           PUSH  AF            ; Get row, column into HL
F48A E1         00101           POP   HL
F48B 0602       00102           LD    B,2           ; Set up for SVC
F48D 3E0F       00103           LD    A,@VDCTL
F48F EF         00104           RST   40            ; Poke it
F490 C9         00105           RET                 ; Back to BASIC
                00106
;===============================================================
F491 4E         00107 PROTECT   LD    C,(HL)        ; C = number of lines to protect
F492 0607       00108           LD    B,7           ; Set up for SVC
F494 3E0F       00109           LD    A,@VDCTL
F496 EF         00110           RST   40            ; Do it to it
F497 C9         00111           RET                 ; Back to BASIC
                00112
```

```
;*****************************************************************
F498 7E      00113 VLINE    LD    A,(HL)      ; Get length of target string
F499 FE50    00114          CP    80          ; Make sure length is >= 80
F49B D8      00115          RET   C           ; Do nothing if < 80
F49C D5      00116          PUSH  DE          ; Save 2nd argument
F49D 23      00117          INC   HL          ; Get actual string address into DE
F49E 5E      00118          LD    E,(HL)
F49F 23      00119          INC   HL
F4A0 56      00120          LD    D,(HL)
F4A1 E1      00121          POP   HL          ; HL points to 2nd argument
F4A2 66      00122          LD    H,(HL)      ; H has video line to get
F4A3 0609    00123          LD    B,9         ; Set up for SVC
F4A5 3E0F    00124          LD    A,@VDCTL
F4A7 0E01    00125          LD    C,1
F4A8         00126 POKEADDR1 EQU  $-1         ; Poke 0 here to copy string to video
F4A9 EF      00127          RST   40          ; Get the line
F4AA C9      00128          RET               ; Back to BASIC
             00129
;*****************************************************************
F4AB 5E      00130 RAMTOVID LD    E,(HL)      ; Get pointer to RAM buffer into HL
F4AC 23      00131          INC   HL
F4AD 56      00132          LD    D,(HL)
F4AE EB      00133          EX    DE,HL
F4AF 3E0F    00134          LD    A,@VDCTL    ; Set up for SVC
F4B1 0605    00135          LD    B,5
F4B2         00136 POKEADDR2 EQU  $-1         ; Poke 6 here to make this VIDTORAM
F4B3 EF      00137          RST   40
F4B4 C9      00138          RET               ; Back to BASIC
             00139
;*****************************************************************
             00140 ;      CALCADDR -- Translate X,Y coordinates to absolute character
             00141 ;                  position and bit position within the character
             00142 ;            ENTRY:  HL Points to the X coordinate, DE points to Y coord.
             00143 ;            EXIT:   H=video row, L=video column, C=bit position
             00144
;*****************************************************************
F4B5 4E      00145 CALCADDR LD    C,(HL)      ; Get the X coordinate into C
F4B6 EB      00146          EX    DE,HL
F4B7 7E      00147          LD    A,(HL)      ; and the Y coordinate into A
F4B8 69      00148          LD    L,C         ; Now L has X coordinate
F4B9 CB3D    00149          SRL   L           ; L/2=Video column now in L
F4BB 0E00    00150          LD    C,0         ; Set bit column to 0
F4BD 3001    00151          JR    NC,BITCOL0  ; Go if bit column=0
F4BF 0C      00152          INC   C           ; Else bit column=1
F4C0 06FF    00153 BITCOL0  LD    B,-1        ; Initialize counter
F4C2 04      00154 VIDROW   INC   B           ; Bump quotient (B= video row #)
F4C3 D603    00155          SUB   3           ; Successive subract for /3
F4C5 F2C2F4  00156          JP    P,VIDROW    ; Loop until negative
F4C8 C603    00157          ADD   A,3         ; A=Bit row (Y MOD 3)
F4CA 07      00158          RLCA              ; bit row * 2
F4CB 81      00159          ADD   A,C         ; (bit row * 2)+bit column
F4CC 4F      00160          LD    C,A         ; C = bit position (0 thru 5)
F4CD 60      00161          LD    H,B         ; H=Video row
F4CE 0600    00162          LD    B,0         ; Zap B for later ADD BC,HL instruction
F4D0 C9      00163 DOSENTRY RET
F4D1 81      00164 SETMASK  DB    81H,82H,84H,88H,90H,0A0H      ; Mask tables
     82 84 88 90 A0
F4D7 FE      00165 RESETMASK DB   0FEH,0FDH,0FBH,0F7H,0EFH,0DFH
     FD FB F7 EF DF
F4DD 01      00166 POINTMASK DB   1,2,4,8,16,32
     02 04 08 10 20
F4D0         00167          END   DOSENTRY      ; Do nothing if executed from DOS ready
F4D0 is the transfer address
00000 Total errors
```

NORTHERN BYTES ARTICLE SUBMISSION FORMAT

Occasionally we hear from someone who wants to contribute an article to NORTHERN BYTES, but isn't sure what format to put it in or how to send it. Here's a quick guide:

1) Send your article on disk, tape, or electronically via MCI Mail. We can read just about any TRS-80, CP/M, or MS-DOS compatible disk (if we can't read it directly, we can at least pull the files off the disk using a conversion program such as Hypercross). If you use tape on a Model III or 4, save the article and/or program to tape at the low (500 baud) cassette speed.

2) If you are using Allwrite or Newscript as your word-processor, you may send text files in your word-processor's native format. If you are using SuperScripsit, please do NOT send a SuperScripsit format file - convert it to ASCII first (SuperScripsit has a way to do this). In most other cases, we prefer a plain-vanilla ASCII text file. If you don't have a word processor, use any text editor you do have - the editor of an editor-assembler program will do, or you can even type text into BASIC lines (just be sure to put an apostrophe or REM statement at the beginning of each line). We can easily strip off leading line numbers and/or other garbage if necessary.

3) Programs listing formats: BASIC programs may be sent in regular (tokenized) format OR in ASCII format IF you are using Model I/III BASIC. Programs created under other BASICs (Model 4 BASIC, etc.) should be saved using ASCII format (the command is: SAVE "filename",A). Editor-Assembler listings should be sent in BOTH the normal format of your editor-assembler AND in list file format if your assembler has that option (if you don't know what "list file format" is, your assembler probably doesn't have it and in that case, we probably don't need it since we can generate it here. In particular, we do NOT need the "list file" if your program was created using Microsoft's Editor-Assembler Plus or the Radio Shack or Apparat EDTASM programs). If you do send the list file and have the option to include the symbol table, please do so. Programs in other languages should be sent in ASCII format (you should assume that the editor has never even heard of the language you've used, and write your article and prepare your program listing accordingly).

4) DON'T FORGET to send BOTH the article text file and any program files. We no longer accept hardcopy submissions!

5) You may use any ALLWRITE-compatible control words or emphasis marks within your text (I may remove them if you get carried away, but go for it anyway). In particular, if you want something to be underlined, do it like this:

... here are the @$words to be underlined@% ...

will translate to:

... here are the words to be underlined ...

6) Finally, if you are mailing disks, either use a regular size disk or photo mailer, or an envelope that is not too large (about 6"x9" is a good size) and then sandwich the disk between two pieces of very stiff cardboard. Write "DO NOT BEND" on the outside of your envelope. Two no-no's are very large envelopes (I have a regular size mailbox, and my mailman invariably tries to bend such envelopes to make them fit in the box) and "Jiffy" type bags (very nice for mailing fragile items, but very poor protection for disks since they can be easily bent).

MORE CHANGES TO SETDATE
by Jack Decker

If you live in the U.S.A., you'll probably want to skip this article. Otherwise, read on...

In NORTHERN BYTES Volume 6, Number 2, I printed a letter from Clifford S. Richards of Sydney, Australia, asking for a version of SETDATE that would work with a patched version of NEWDOS/80 which stores the date in DD/MM/YY format (rather than the original MM/DD/YY format normally used here in the U.S.). I provided a list of changes to SETDATE version 1.3 which would accomplish this, however, Clifford has contacted me again and asked for a version which also writes the date to the file DATE/TXT, similar to what DS/CMD (on TAS Public Domain Library Disk #006) does, but once again modified to work with a DOS that stores dates internally in the DD/MM/YY format.

So - here's how to modify the already-modified SETDATE once more, to also write the date string to file DATE/TXT.

1) Start with a copy of SETDATE/ASM version 1.3 (do not use an earlier version, as these instructions will not make sense. If you need a copy of version 1.3, it can be found on any recent copy of TAS Public Domain Library Disk #001. If you have an older copy of PD#001, send it to me [not TAS] along with an adequate amount to cover return postage, and I will upgrade it for you).

2) Apply the changes from NORTHERN BYTES, Volume 6, Number 2, pages 4 and 5. Save and assemble this program since you may want to use it on your non-wordprocessing disks.

3) Once you have ascertained that you have installed the previous changes correctly, apply the changes shown below. Please note that you can save yourself a bit of effort if you read everything before you do anything. If you are observant, you will notice that I tell you to delete a large block of code in one place, and then re-enter almost exactly the same code elsewhere. If your assembler permits, you might save yourself some effort by just moving the code and making the required changes.

```
******** DELETE the following lines:
01820           LD    A,(BC)          ;Get day (1 - 31)
01830           LD    L,A             ;Put day in L
01840           LD    H,0             ;HL = day
01850           PUSH  BC              ;Save date storage pntr
01860           CALL  PRTNUM          ;Print day
01870           POP   BC              ;Restore date storage ptr
01880           CALL  PRTSPC          ;Print space character
01890           DEC   BC              ;Point to month byte
01900           LD    A,(BC)          ;Get month (1 - 12)
01910           ADD   A,6             ;Offset for string table
01920           CALL  PRTSTR          ;Print month string
01930           CALL  PRTSPC          ;Print space character
01940           DEC   BC              ;Point to year byte
01950           LD    A,(BC)          ;Get year (0 - 99)
01960           LD    C,A             ;Put year in C
01970           LD    B,0             ;BC = last 2 digits year
01980           LD    HL,(CNTURY)     ;Get century offset
01990           ADD   HL,BC           ;HL = Year (all 4 digits)
02000           CALL  PRTNUM          ;Print year
******** REPLACE with this line:
01820           CALL  PRTDAT          ;Print day, month & year

******** DELETE this line:
02420 EXIT2     POP   HL              ;Restore input buffer ptr
******** REPLACE with these lines:
02411 EXIT2     CALL  GETBFR          ;Get loc memory date stor
02412           PUSH  DE              ;Save memory date storage
02413           LD    B,0             ;Logical Record Lngth=256
02414           LD    DE,FCB2         ;File Control Block pntr
02415           LD    HL,FILBUF       ;File I/O Buffer pntr
02416           CALL  4420H           ;DOS INIT routine
02417           JR    NZ,ERREXT       ;Go if error
02418           POP   BC              ;Get memory date storage
02419           INC   BC              ;Point to month byte
02420           INC   BC              ;Point to day byte
02421           LD    A,1BH           ;ROM "byte output" addr
02422           LD    (OUTCHR+1),A    ;Change program vector
02423           CALL  PRTDAT          ;Output date to file
02424           LD    A,0DH           ;Carriage return in A
02425           CALL  OUTCHR          ;Output it to file
02426           CALL  4428H           ;CLOSE file
02427           POP   HL              ;Restore input buffer ptr

******** CHANGE lines 2520 & 2540 (JR to JP):
02520           JR    C,GETKEY        ; less than ASCII 18H
CHANGE TO:
02520           JP    C,GETKEY        ; less than ASCII 18H

02540           JR    NC,GETKEY       ; than ASCII 1CH
CHANGE TO:
02540           JP    NC,GETKEY       ; than ASCII 1CH

******** CHANGE lines 3300 & 3400 (33H to OUTCHR):
03300           CALL  33H             ;Display it on video
CHANGE TO:
03300           CALL  OUTCHR          ;Display it on video

03400           CALL  33H             ;Display it on video
CHANGE TO:
03400           CALL  OUTCHR          ;Display it on video

******** DELETE the following lines:
03350 PRTNUM    CALL  0A9AH           ;Number in HL to ACCUM
03360           CALL  0FBDH           ;Convert # to string
03370           INC   HL              ;Skip leading space char
******** REPLACE with these lines:
03350 PRTDAT    LD    A,(BC)          ;Get day (1 - 31)
03351           LD    L,A             ;Put day in L
03352           LD    H,0             ;HL = day
03353           PUSH  BC              ;Save date storage pntr
03354           CALL  PRTNUM          ;Print day
03355           POP   BC              ;Restore date storage ptr
03356           CALL  PRTSPC          ;Print space character
03357           DEC   BC              ;Point to month byte
03358           LD    A,(BC)          ;Get month (1 - 12)
03359           ADD   A,6             ;Offset for string table
03360           CALL  PRTSTR          ;Print month string
03361           CALL  PRTSPC          ;Print space character
03362           DEC   BC              ;Point to year byte
03363           LD    A,(BC)          ;Get year (0 - 99)
03364           LD    C,A             ;Put year in C
03365           LD    B,0             ;BC = last 2 digits year
03366           LD    HL,(CNTURY)     ;Get century offset
03367           ADD   HL,BC           ;HL = Year (all 4 digits)
03368 PRTNUM    PUSH  DE              ;Save DE
03369           CALL  0A9AH           ;Number in HL to ACCUM
03370           CALL  0FBDH           ;Convert # to string
03371           INC   HL              ;Skip leading space char
03372           POP   DE              ;Restore DE

******** CHANGE line 3420 (JP 33H to JR OUTCHR):
03420           JP    33H             ;Display on video & RET
CHANGE TO:
03420           JR    OUTCHR          ;Display on video & RET

******** CHANGE the following lines:
03670 FCB       DEFM  'AUSDATE/CMD';File Control Block area
03680           DEFB  3               ; with program filename
03690           DEFS  20D             ; (total 32 bytes)
******** ADD/REPLACE (3661-3663 added, 3670 & 3690 changed):
03661 OUTCHR    JP    33H             ;Adr modified for dsk I/O
03662 FCB2      DEFM  'DATE/TXT'      ;Second FCB area (uses
03663           DEFB  3               ; part of original FCB)
03670 FCB       DEFM  'RD/CMD'        ;File Control Block area
03680           DEFB  3               ; with program filename
03690           DEFS  25D             ; (total 32 bytes)
```

INSTRUCTIONS FOR THE INSTALLATION OF ADDITIONAL 64K MEMORY IN THE NEW TRS-80 MODEL 4A
by Mike Santana

[Reprinted from The INTERFACE newsletter of the San Gabriel Valley TRS-80 Users Group.]

This new Model 4A has one single board with CPU, Disk Driver Circuit, RS232 and Printer Driver in one single P.C. Board. It also has a new keyboard and a Green Screen.

1. Locate U33 SN74LS273N.
2. Next to U33 Pin 17 is Capacitor C39.
3. Locate wire running from Capacitor C39 connection in front U33 Pin 17 to U5 Pin 16.
4. Disconnect wire soldered to C39 and solder to U33 Pin 16.
5. Jumper wire is now connected between U5 Pin 16 and U33 Pin 16.
6. Install new 64K Memory in sockets U67 - U74.

INTERNATIONAL NEWDOS/80 USERS' GROUP

by Greg Small

Welcome to the second installment of this column devoted to the understanding, use, and modification of NEWDOS/80.

For those of you coming in late, let me explain the background of this feature.

I attempted to start an International NEWDOS/80 Users' Group last year. My original intention was to coordinate the group from my location, issue a newsletter, and possibly distribute a diskette containing public-domain NEWDOS/80-specific programs and/or text files containing patches, etc.

I had visions of getting perhaps 100 inquiries from the publicity Northern Bytes gave me in the Spring of 1984. This did not happen.

I had letters from Australia, Japan, Spain, Canada, and even the U.S.A., but these totalled fewer than ten. I also had three or four people logon to my BBS from the U.S.A.

So I felt the support was NOT great, and had been dragging my heels on doing anything further with the group idea. My intention was to ignore it and hope it went away quietly. However, my experience with Jerry Vabulas from New York City kept getting in the way.

Jerry logged onto my BBS, and we exchanged a number of messages about NEWDOS/80 Version 2.5, which I did not use at that time. He was even good enough to send me a diskette with a few patches he had developed. I, in turn, did nothing with them.

Finally, for the previous issue I put together some thoughts and ideas for the future. I also shared my commented disassembly of BOOT/SYS for the Model I Version 2.0 and some ideas about enhancements this commented code could allow. As well, I sought answers from you people on a re-boot problem many Model I owners seem to have.

This second column continues on that theme, and I hope it will be a productive tool for NEWDOS/80 users.

I have also opened a section on my BBS for the International NEWDOS/80 Users' Group. The section has a message base, and a download area for programs and text files. The text of these columns is also stored there.

We now have at least 55 active boards in the Toronto area, so BBSing tends to be a little hectic from time to time and the "children" try to take their toll on serious boards.

Because of this, you MUST read the board policies and register online to obtain full access. This takes 7-10 minutes. I am at present only running at 300 bps but hope to have a 1200 or maybe even a 2400 bps modem shortly.

By the time you read this I hope to have devised a method to allow Northern Bytes readers access to the section without a long-drawn-out preamble. More details will be in the next column.

Apparat also appears to have "increased" its support of NEWDOS/80. It recently announced a Profile 3 Plus Upgrade for the DOS, and stated it would continue support of NEWDOS/80 Version 2.0 & 2.5. The details include its technical support and update service as outlined in the manual.

As well, Apparat made this announcement:

"24-HOUR DATA-LINE

"Apparat's TBBS #1 of Denver Electronic Bulletin Board now offers a NEWDOS/80 support section. Register for support on your first call, we will try to get your registration active on the NEWDOS/80 support section within a week. Call (303) 741-4071."

At this writing, I have logged onto their board but my registration has not yet been activated (see sidebar).

As promised in the last issue, the theme of this issue's column is the theory behind, and the formulation of, ZAPs or patches to NEWDOS/80. The discussion will not be limited to the DOS alone but will give even the rank amateur enough background in preparing and installing simple ZAPs to become just slightly dangerous to his disks.

First, a disclaimer!

ALWAYS, ALWAYS, ALWAYS work on COPIES of backup disks.

NEVER, NEVER, NEVER work on your backups or your working copies or your masters.

NEVER, NEVER, NEVER test a ZAP with ANY live data or programs online.

None of us will take any responsibility for your lost data or programs or disks or hair if you disobey this simple rule. I used to ignore this rule frequently, but got fed up with having to re-create working copies and hard-drive directories.

I trust I have made my point!

ZAPs for the DOS and many programs can be found in magazines, newsletters and on BBS's and commercial networks. Or they can be generated by you. I know, I know, you say you know nothing about machine code. Well, sit tight. That's what this is all about.

Once the ZAP is either discovered or written it MUST be tested. Your system may not be 100% identical to that of the author of the patch. The source of the patch may not have provided ALL the necessary parts of the ZAP or they may be incorrect. Thus the need for testing -- on a COPY of your working diskette.

To install and test a ZAP there are just a few simple steps.

First, we make sure the patch is for the version of the program or DOS you are using. Next, we protect ourselves from doom.

The safest method is to use the COPY command with the BDU option to make copies of any working diskettes that will be involved in the testing of the ZAPs. To copy your working SYSTEM diskette to a blank diskette in drive 1 you would issue the following command:

COPY,0,1,,,,FMT,BDU,DPDN=0<CR>

where <CR> means press your ENTER key.

This will format the diskette in drive 1 and make an EXACT duplicate of the diskette in drive 0. The files DIR/SYS and BOOT/SYS will be copied exactly from the source to the destination. This type of COPY can now be done for any other disks needed for the tests.

If, later, a patch fails and causes damage to a sector or file, or to BOOT/SYS or DIR/SYS on the copy of the working diskette, it will be a simple procedure to use SUPERZAP to copy over a few sectors from the working diskette to the copy of the working diskette and all will again be ready for continuation of the testing.

Finally, a little extra bit of safety for you hard-drive users.

If you run your hard drive from floppies, make a special TEST SYSTEM diskette that defines a very small portion of the drive as the ONLY live part of the drive.

If your BOOT floppy transfers SYSTEM control to the hard drive make a special TEST BOOT diskette that defines a very small portion of the drive as the ONLY live part of the drive. You must have the SYSTEM on that test drive if you expect to run strictly from the hard drive.

Then, if a patch fails, you will not have the heads writing myriad patterns across the full 5 or 10 (or more) meg drive.

(Note that it is NOT easily possible to copy a hard drive DIR/SYS to another filespec on the hard drive. But it is possible to copy it from the hard drive to a floppy. I have a special /JCL file

APPARAT'S TBBS - AN ACT NOT QUITE TOGETHER

In the accompanying column I make reference to APPARAT's TBBS #1 of Denver Electronic Bulletin Board.

I have now made contact, after a fashion. I first called around 6 AM on March 11, 1985. The transmission was atrocious but I persisted. After re-reading many menus two or more times I was able to register for the NEWDOS/80 support section and log off.

I called back seven days later, again around 6 AM. This time I listened to the incoming modem carrier and felt it was weak -- but the line was clean so I connected and was rewarded with a seemingly good transmission.

After the initial logon messages I was at the Main Menu and garbage transmission again. I was able to make my way to the NEWDOS/80 support section and was dismayed to see that I was NOT recognized as having registered previously. Apparat said in their recent press release, ".... we will try to get your registration active ... within a week."

Obviously this is not always possible. I feel that an organization that truly wants to offer support should be able to allow their (previously) paying customers access to the support section without this type of delay. It is simply a matter of priorities.

[I also would be interested in other users experience with Apparat's carrier signal strength.]

I will continue logging on to the Apparat BBS but would suggest you wait longer than one week and also that you ensure the signal strength and line quality are excellent before trying to complete your connection. Otherwise you will merely be wasting your time and money.

- Greg Small

that I run regularly that backs up all DIR/SYS's. Then if a DIR/SYS goes bad it is a simple process to SUPERZAP the sectors needed from the floppy to repair the hard drive.)

Next, a list of the tools we will need for any ZAP creation. In brackets are the specific tools I use:

- a monitor (TASMON/CMD)
- a list of Z-80 opcodes (How to Program the Z80 by Rodnay Zaks - available from Radio Shack - 62-2066)
- an intimate knowledge of the operation of the program to be patched. This is critical for easy creation of ZAPs.

The following are useful but NOT essential:

- a diskette zapper (SUPERZAP/CMD)
- an editor assembler (EDTASM/CMD)
- a disassembler (DISASSEM/CMD)
- a byte finder (FINDLOC/CMD)
- a word processor (SCRIPSIT/CMD)

With the preliminary setup and list of tools out of the way we can get on to the generation of a ZAP of our own. The ZAP I outline below was created to solve a specific problem I had. This may be useless to YOU but will serve to illustrate how easily a ZAP can be created.

To help pay for this hobby I maintain the mailing list of a charitable organization on one of my Model I's. I wrote the programs needed in BASIC using NEWDOS/80. As the list grew I found that a certain amount of file reorganization was going to be required so I wrote a program to sift a file of names into many other files.

When I first got a disk system I felt that the maximum of 15 files allowed by TRSDOS and all the other DOS's was slightly ridiculous. Who could ever need all those files? However, I now found myself needing 16 files. I rewrote the program to do the task in two passes and forgot about the limitation.

A year later I was faced with a similar task. This time I decided I would investigate a little further, as the time required for the previous run was excessive because of the need for the double pass.

Following is the thinking I used to prepare my plan of attack:

1 - If BASIC is invoked without any parameters three files are allocated.

2 - If BASIC is invoked with a parameter from 0 to 15 then the parameter is assumed to be the number of files to be allocated.

3 - If BASIC is invoked with a parameter less than 0 or greater than 15, and the parameter is NOT analyzed and found to be a valid memory size, then an exit to DOS READY is taken.

4 - If I could find an exit to DOS READY, then the code prior to it would most likely be a check for a request for allocation of more than 15 files.

5 - There are, however, other exits to DOS READY in BASIC/CMD. If we enter BASIC * an exit to DOS READY will be taken if HIMEM is set differently than it was during the previous run.

(Any single references to memory locations or disk file locations are for both Model I and III. If the references are different, then the Model III reference is given in square brackets after the Model I reference, thus: 6543H [6521H].)

Next I loaded TASMON and relocated it to high memory. (I could have used a previously relocated copy that had been relocated and DUMPed earlier.) I then used the LD command of TASMON to load BASIC/CMD. The specific command was:

```
LD<CR>
BASIC/CMD<CR>
```

TASMON reported that BASIC/CMD resided from 5700H through 684DH and that the entry point was 66BEH.

I did not want to have to disassemble the entire BASIC/CMD to a printer and then manually search through and comment the code, so decided to use a little logic — and the machine. (Besides, this article is supposed to show how a non-assembly language type can do this sort of thing with a little bit of logic and reading of some references. I do not specifically refer to Zak's book in this discussion but if you are not familiar with Z-80 opcodes you will find yourself using it to look up the hex byte(s) that represent the various instructions or to determine what an instruction does.)

- If BASIC 16 is used to invoke BASIC we end up at DOS READY.
- DOS READY is defined as 402DH in section 3.2 of the manual.

If I could find a reference to 402DH in the code, I might be somewhere near where the maximum number of files test is made.

Using the FIND command of TASMON, I searched (starting at 5700H, as that is where TASMON reported BASIC/CMD started) for the byte combination 2D 40. (The Z-80 reverses byte pairs). The specific command was:

```
F 5700 2D 40 <CR>
```

The only reported location of this was at 57CAH. Using the disassemble command of TASMON, I disassembled the code, starting at 57C3H, as I wanted to see a little of the code prior to this address.

I found the following code and added the comments:

```
57C3  7E      LD    A,(HL)    ;get a byte in A
57C4  FE53    CP    53        ;is it an "S"?
57C6  E1      POP   HL        ;get HL back
57C7  2003    JR    NZ,57CC   ;jump around if not "S"
57C9  C32D40  JP    402D      ;jump to DOS READY
```

I could see that this was most likely the test for the "S" in the CMD"S" command from BASIC. But using the FIND command again showed it is the ONLY reference to DOS READY in BASIC/CMD so I proceeded a little further.

The bytes at 57C9 are a jump to DOS READY, so I searched for references to those bytes. TASMON reported they were found at 6FDEH [64B7H], 64E6H [64BFH], 6558H [6531H], and 673EH.

I noted that the entry point to BASIC is not at the beginning of the file (as is usual for most TRS-80 programs) but at 66BEH. Presumably, this is some sort of setup routine.

So my examination was started at 673EH. Disassembling and examining the code around that address showed the following, with the comments again added by me:

```
6722  FE10    CP    10        ;compare to 16
6724  3017    JR    NC,673D   ;jump to 673D if CARRY not set
.............
673D  C3C957  JP    57C9      ;jump to the jump to DOS READY
```

I had found a place where a compare was occurring. The value was near the value of the maximum number of files. I had found a place to test.

I recorded the address (6723H) of the 10H (or 16), which I was assuming was the maximum number of files plus one . I then exited to DOS READY and typed DEBUG<CR>. At the next DOS READY prompt I typed:

```
BASIC 16<CR>
```

and promptly entered DEBUG as expected. Once there I typed M6723<CR> which put the modify cursor over the 1 in the 10. I changed the 10 to 11 and pressed ENTER. I then typed G<CR> and was rewarded with the normal BASIC herald.

The test! I typed:

```
OPEN"I",16,"BOOT/SYS"<CR>
```

and was rewarded with BASIC's familiar READY prompt - not an error message. I then modified an existing BASIC program by changing all references to buffer 1 to read buffer 16. I experimented to prove that buffer 16 actually existed and was in use and that all the other buffers worked as expected. All seemed well.

The next task was to find the maximum number of buffers that could be created and used. I played with various numbers and discovered it was possible to have 128 buffers in a 48K floppy system with HIMEM set at FFFFH. However, this is extreme! For all practical purposes 100 buffers seems to be the maximum that should be allocated, to allow for some programming space.

Finally I had to write a file to document this ZAP. You may not feel this is necessary, but believe me, if you do NOT do it you will surely discover later that you need it.

I had to determine where on diskette this patch was to be applied. I knew it was to go at 6723H in memory. I also knew that the sequence of bytes around the ZAP was FE 10 30 17.

I invoked SUPERZAP and typed DFS<CR>. At the next prompt I entered BASIC/CMD<CR> followed by 0<CR>. I then typed L,FE,10,30,17<CR>, which is SUPERZAP's locate command, followed by the bytes I was seeking. The find cursor appeared at byte 66H [3FH] of relative sector 16. So I had a possible ZAP location. To

verify that it was the correct location, I typed F<CR>. No other matches were found.

Now, a far simpler method is to type:

FINDLOC BASIC/CMD 6723<CR>

The result looks like this:

```
FINDLOC - V1.0  (c) 1982  Bob Withers
Locate address = 6723
File relative sector =    16 (Decimal),   10 (Hex)
Byte displacement    =   183 (Decimal),   B7 (Hex)
(Byte displacement   =    64 (Decimal),   40 (Hex))
```

I found that a lot simpler than the SUPERZAP method. (FINDLOC/CMD is, or will soon be, available on a TAS Public Domain Disk. It has also been available on some of the CompuServe SIGs and is available in the NEWDOS/80 Users' Group area on TBBS - The Business Board System (416) 640-3434.)

APPARAT set the standard for documentation of a ZAP. I followed their lead.

First, we prepare the Model I ZAP by naming and dating it:

******** BASIC1 ******** 03/07/85 ******** V2M1 ********

Next we describe the purpose and function of the ZAP:

> This patch enables more files than 15 on Model I NEWDOS/80 Version 2 BASIC

Now we document the actual ZAP:

```
BASIC/CMD 16,66
        change FE 10 30 to FE xx 30
```

And finally we add the commentary about the value to insert:

> The value used to replace xx must be one more than the maximum number of files required and cannot be greater than 81H in a 48k system and, for all practical purposes, should not be greater than 64H, because it restricts programming space.

Following is the complete ZAP for the I and the III. Also included is a ZAP to change the default number of files from 3 to a number selected and patched into BASIC.

******** BASIC1 ******** 03/07/85 ******** V2M1 ********

This patch enables more files than 15 on Model I NEWDOS/80 Version 2 BASIC

```
BASIC/CMD 16,66
        change FE 10 30 to FE xx 30
```

This patch sets the default number of files on Model I NEWDOS/80 Version 2 BASIC

```
BASIC/CMD 14,46
        change 3E 03 32 to 3E yy 32
```

The value used to replace xx must be one more than the maximum number of files required and cannot be greater than 81H in a 48k system and, for all practical purposes, should not be greater than 64H, because it restricts programming space.

The value used to replace yy should be the default number of files to be allocated when BASIC is invoked.

******** BASIC1 ******** 03/07/85 ******** V2M3 ********

This patch enables more files than 15 on Model III NEWDOS/80 Version 2 BASIC

```
BASIC/CMD 16,3F
        change FE 10 30 to FE xx 30
```

This patch sets the default number of files on Model III NEWDOS/80 Version 2 BASIC

```
BASIC/CMD 14,1F
        change 3E 03 32 to 3E yy 32
```

The value used to replace xx must be one more than the maximum number of files required and cannot be greater than 81H in a 48k system and, for all practical purposes, should not be greater than 64H , because if restricts programming space.

The value used to replace yy should be the default number of files to be allocated when BASIC is invoked.

In the next column I will provide a number of ZAPs that I have created over the past few years.

I want to thank Bob Blackburn for the editing he has done on this column. Bob is a local journalist who turned to a Model III two years ago to replace his aging Underwood. He has now started an online editing service using the TBBS software.

Simply, Bob's clients upload their "raw" text to his system at their convenience. He then edits the text for style, clarity, spelling and punctuation. When he is satisfied with the results he places the file online again and will phone the client to notify him the text is ready for retrieval. The client then can download the text, again at his convenience.

Anyone wanting more information on this unique service can contact Bob at the following:

Bob Blackburn
1694 Gerrard Street East
Toronto, Ontario, CANADA
M4L 2B2

Phone: (416) 466-3587 (voice)
MCI Mail: Username: BBLACKBURN MCI Mail ID #: 254-9871

If you have any specific requests for this column or wish to contact me full details are in the last issue of Northern Bytes. The two major changes are that my MCI Mail account has finally been activated and a section has been created on TBBS - The Business Board System for the Users' Group as outlined at the beginning of this column.

Here is a summary of the previous details:

Mail: Greg Small, Box 607, Stouffville, Ontario, Canada L0H 1L0
Phone: (416) 640-4400 - 7PM-9PM/week nights/10AM-6PM/weekends.
Dataline: (416) 640-3434 - 24 hours a day.
MCI Mail: Username: GSMALL/EAGLE MCI Mail ID #: 251-7579
CompuServe EasyPlex: PPN 72335,560

MORE ON AUTO-BOOTING A MODEL 4P

In past months we have published much in these pages about making a self-booting disk for the Model 4P, using a DOS other than LDOS or TRSDOS. There are a couple of comments that should be added to this discussion.

First, the question has been raised, how does the Model 4P know whether it is booting a TRSDOS 6 (or other Model 4 DOS) disk as opposed to a Model III DOS disk? Simple, the 4P checks the boot code on the disk for a CALL 00nnH instruction (specifically, the bytes CD xx 00 where xx can be anything). TRSDOS 1.3 has a "CALL 0033H" instruction in the boot code, and all other Model III DOSes apparently also have at least one CALL below 0100H in their boot code. Model 4 DOSes, on the other hand, do NOT have calls below 0100H in their boot code. Simple!

Second, if you've gone and installed double-sided disks in your 4P, you may wonder what the secret is to making a double-sided auto-booting disk. Well, both LDOS and TRSDOS use relative byte CD of the GAT sector (first sector of the directory) to store information about the disk. If bit 5 of relative byte CD is set, that indicates that the disk is double-sided. NEWDOS/80 doesn't use that byte, so it just puts a zero byte there. If you used a method similar to the one described in Tony Domigan's letter in NORTHERN BYTES, Volume 6, Number 2 (page 3) to attempt to create a double-sided self-booting NEWDOS/80 disk but had no success, try zapping relative byte CD of the dummy directory (the one that is part of the created MODELA/III file, not the real one used by NEWDOS/80) to FFH (all bits set, including bit 5). Unfortunately, I am unable to actually try this because I do not have access to a Model 4 with a double-sided drive zero.

Credits: Thanks to John Hallgren, Jon Yarden, and The INTERFACE (newsletter of the San Gabriel Valley TRS-80 Users Group) for providing bits and pieces of the above information.

FIXGAT/ASM FOR MODEL III NEWDOS/80 VERSION 2

by Tony Domigan

This routine is designed to correct the Granule Allocation Table of NEWDOS/80 version 2 diskettes. As I explained with my previous routine to fix the HIT sector, Super Utility plus will do a far better job than my routine, however, SU+ does not support Double-Sided NEWDOS/80 configurations. This routine, however, will work similarly with mono and double-sided NEWDOS diskettes.

As with FIXHIT/ASM, FIXGAT requires that the DIRectory should be readable by the system as it is accessed by file I/O. The PDRIVE settings for the drive to be tested should be correct as I check only the memory PDRIVE settings. A final limitation is that I have not included checking for non-standard track counts. If the track count is standard, that is, can be apportioned to a whole number of 'lumps' all is OK, however, if you chose say a TC of 42 or 82, then using FIXGAT will result in an error in the last allocation byte.

The Granule Existence Table (GET) proved to be an awkward problem for me to solve. The NEWDOS manual states that it will not be formed if the number of lumps exceeds 60H, however I have noticed that with double-sided ops the GET is fully allocated. DIRCHECK doesn't seem to mind whether it is initialized or allocated so I opted to initialize the GET as well as the GAT.

The command line for this program is

FIXGAT<CR> . . . . . .defaults to drive 0
FIXGAT dn<CR> . . . . . .selects requested drive number
FIXGAT :dn<CR> . . . . . .selects requested drive number

Tony Domigan
P.O. Box 150, Thomastown, Victoria, Australia, 3074.
MCI-ID : 254-5121

```
                00100 ;             FIXGAT/ASM - Version 1.0
                00110 ;               NEWDOS80/V2  (III)
                00120 ;                 3rd March 1985
                00130 ;         For NORTHERN BYTES & the PUBLIC DOMAIN
                00140 ;                   by Tony Domigan
                00150 ;  PO Box 150, Thomastown, Victoria, 3074, Australia
                00160 ;MCI-ID:254-5121. SOURCE-ID:BCT039. TAB-ID:DOMIPOBOWINN
                00170 ;-------------------------------------------------------
                00180 ;
5200            00190          ORG    5200H          ;Anywhere abv 5200H
                00200 ; ------------- TEST FOR DRIVESPEC ----------------------
5200            00210 PARSER   EQU    $
5200 7E         00220          LD     A,(HL)         ;Trailing chars?
5201 FE00       00230          CP     00H            ;Assume drive 0?
5203 2002       00240          JR     NZ,DN          ;Yes, print msg
5205 3E30       00250          LD     A,30H
5207 FE3A       00260 DN       CP     3AH            ;Colon
5209 2002       00270          JR     NZ,NUMBER      ;Skip if not colon
520B 23         00280          INC    HL             ;Yes, colon so ck next
520C 7E         00290          LD     A,(HL)         ;Get next character
520D FE34       00300 NUMBER   CP     34H            ;> Drive 3?
520F 3004       00310          JR     NC,BADNUM      ;Yes, bad drive number
5211 FE30       00320          CP     30H            ;>=0and<=3
5213 3005       00330          JR     NC,POSTDR      ;Yes, use drive number
5215 3EA0       00340 BADNUM   LD     A,0A0H         ;Illegal drive number
5217 C30944     00350          JP     4409H          ;Error exit
521A 32D655     00360 POSTDR   LD     (DRIVE),A      ;Post drv num to FCB
521D 327D54     00370          LD     (DRVNUM),A     ;Post drv num to banner
                00380 ; ------------- CALCULATE PDRIVE POINTER ----------------
5220 D630       00390 CALCS    SUB    30H            ;Convert to Binary
5222 210A00     00400          LD     HL,000AH       ;Gap between PDRIVES
5225 CD374C     00410          CALL   4C37H          ;NEWDOS Multiply Rtne
5228 3E91       00420          LD     A,PDRIVE       ;LSB PDRIVE address
522A 85         00430          ADD    A,L            ;Adjust Pointer
522B 6F         00440          LD     L,A            ;lsb of pointer
522C 2642       00450          LD     H,42H          ;msb of pdrive
522E E5         00460          PUSH   HL             ;Save :dn pdrive ptr
522F DDE1       00470          POP    IX             ;Xfr it to IX
                00480 ; ------------- RETRIEVE DISK ALLOC BYTES ---------------
5231 DD7E00     00490 PTDIR    LD     A,(IX+0)       ;Directory
5234 32C755     00500          LD     (DIRPT),A      ;Store it
5237 21EF54     00510          LD     HL,NDIR        ;Convert to..
523A CD0744     00520          CALL   4407H          ;ASCII string
523D DD7E01     00530 SUMLMP   LD     A,(IX+1)       ;Total lumps
5240 32C855     00540          LD     (LUMPS),A
5243 21AC54     00550          LD     HL,NLUMP
5246 CD0744     00560          CALL   4407H
5249 DD7E03     00570 TCOUNT   LD     A,(IX+3)       ;Track count
524C 32C955     00580          LD     (TC),A
524F 218054     00590          LD     HL,NTC
5252 CD0744     00600          CALL   4407H
5255 DD7E04     00610 SPTNO    LD     A,(IX+4)       ;Sectors/track
5258 32CA55     00620          LD     (SPT),A
525B 219954     00630          LD     HL,NSPT
525E CD0744     00640          CALL   4407H
5261 DD7E05     00650 GPLVAL   LD     A,(IX+5)       ;Grans per lump
5264 32CB55     00660          LD     (GPL),A
5267 F5         00670          PUSH   AF
5268 21C954     00680          LD     HL,NGPL
526B CD0744     00690          CALL   4407H
526E F1         00700 SDIR     POP    AF
526F F5         00710          PUSH   AF
5270 6F         00720          LD     L,A            ;Place GPL in HL
5271 2600       00730          LD     H,00H
5273 DD7E00     00740          LD     A,(IX+0)       ;DIR lump
5276 CD374C     00750          CALL   4C37H          ;Multiply gplxdir
5279 3E05       00760          LD     A,05H          ;5 Sectors/gran
527B CD394C     00770          CALL   4C39H          ;5xtotal grans
527E EB         00780          EX     DE,HL          ;Sce string number
527F 21F354     00790          LD     HL,DIRS        ;Dest string
5282 CD0244     00800          CALL   4402H          ;Post to message
5285 F1         00810          POP    AF             ;Restore GPL
                00820 ; ------------- MAKE STARTING GRAN. ALLOC. BYTE ---------
5286 47         00830 FGAB     LD     B,A            ;Loop=GPL
5287 3EFF       00840          LD     A,0FFH         ;All Allocated byte
5289 CB27       00850 FGLOOP   SLA    A              ;Zero gran
528B 10FC       00860          DJNZ   FGLOOP         ;Loop til GPL
528D 32CC55     00870          LD     (GAB),A        ;Post initz byte
                00880 ; ------------- DISPLAY BANNER --------------------------
5290 CDC901     00890 BANNER   CALL   01C9H          ;CLS
5293 215254     00900          LD     HL,BANMSG      ;Pt to msg
5296 CD6744     00910          CALL   4467H          ;Display msg
                00920 ; ------------- OPEN DIR/SYS OF NOMINATED DRIVE ---------
5299 210058     00930 OPEN     LD     HL,BUFF1       ;FCB buffer
529C 11CE55     00940          LD     DE,FCB         ;Pt to my FCB
529F 0600       00950          LD     B,00H          ;LRL=Full sector I/O
52A1 CD2444     00960          CALL   4424H          ;OPEN  DIR/SYS
52A4 C21954     00970          JP     NZ,EREXIT      ;Z=No error
52A7 3ADA55     00980          LD     A,(LEN)        ;File length from FCB
52AA D602       00990          SUB    02H            ;REL Count=EOF-HIT
52AC 4F         01000          LD     C,A            ;Post page counter to C
                01010 ; ------------- SETUP POINTERS --------------------------
52AD 0608       01020 SETUP    LD     B,08H          ;FPDE Slots 0-7
52AF ED43C555   01030          LD     (PCOUNT),BC    ;Store slot/page
                01040 ; ------------- POSITION TO HIT SECTOR ------------------
52B3 CD2454     01050 INIT     CALL   READ           ;Read GAT sector
                01060 ; ------------- PREPARE BUFFERS -------------------------
52B6 110059     01070 SETBUF   LD     DE,BUFF2       ;Dest=NewGAT
52B9 010001     01080          LD     BC,100H        ;Move sector
52BC C5         01090          PUSH   BC             ;Save count
52BD EDB0       01100          LDIR                  ;Move it
52BF C1         01110          POP    BC             ;Restore Count
52C0 EDB0       01120          LDIR                  ;Move also to OldGAT
                01130 ; ------------- PREP ALLOC TABLE ------------------------
52C2 210059     01140          LD     HL,BUFF2       ;New GAT
52C5 3AC855     01150 DOALLO   LD     A,(LUMPS)      ;Lump Count
52C8 FE60       01160          CP     60H            ;Existence Table?
52CA 3808       01170          JR     C,INITZ        ;Yes if carry
52CC 3ECA       01180 BIG      LD     A,0CAH         ;No existence table
52CE 32F752     01190          LD     (SIZE+1),A     ;post to fill rtne
52D1 32CD55     01200          LD     (FLAG),A       ;Save BIG flag
52D4 CDE852     01210 INITZ    CALL   INZ            ;Init Allocation Table
52D7 3ACD55     01220          LD     A,(FLAG)       ;Big flag
52DA FE00       01230          CP     00H            ;GET?
52DC 2036       01240          JR     NZ,SKPHIT      ;Skip if no GET
52DE 3E6C       01250          LD     A,6CH
52E0 32CD55     01260          LD     (FLAG),A       ;No-more flag
52E3 32F752     01270          LD     (SIZE+1),A     ;Post to fill rtne
52E6 18D0       01280          JR     DOALLO         ;Do GET
52E8 3ACC55     01290 INZ      LD     A,(GAB)        ;Formed init alloc
52EB 32F053     01300          LD     (PBLOOP+1),A   ;Place in loop
52EE 3AC855     01310          LD     A,(LUMPS)      ;Number of Lumps...
52F1 47         01320          LD     B,A            ;into loop counter
52F2 C5         01330          PUSH   BC             ;Save lump counter
52F3 CD0E53     01340          CALL   PBLOOP         ;Do unallocated GAT
52F6 3E61       01350 SIZE     LD     A,61H          ;Standard GAT
52F8 C1         01360          POP    BC             ;Restore counter
52F9 48         01370          LD     C,B
```

```
52FA 0680       01380           LD      B,80H
52FC 5F         01390           LD      E,A
52FD 1600       01400           LD      D,00H
52FF EB         01410           EX      DE,HL
5300 ED42       01420           SBC     HL,BC
5302 7D         01430           LD      A,L
5303 EB         01440           EX      DE,HL
5304 47         01450           LD      B,A             ;loop val
5305 3EFF       01460           LD      A,0FFH          ;Allocated byte
5307 320F53     01470           LD      (PBLOOP+1),A    ;Place in loop
530A CD0E53     01480           CALL    PBLOOP          ;Make table
530D C9         01490           RET
530E 3600       01500 PBLOOP    LD      (HL),00H        ;Fill byte
5310 23         01510           INC     HL              ;Inc Table pointer
5311 10FB       01520           DJNZ    PBLOOP
5313 C9         01530           RET
                01540 ; ---------- SKIP HIT SECTOR ----------
5314 CD2454     01550 SKPHIT    CALL    READ            ;Read HIT sector
                01560 ; ---------- BEGIN LOOP FPDE PAGES ----------
5317 ED4BC555   01570 PAGE      LD      BC,(PCOUNT)     ;Page/Slot Count
531B FD210058   01580           LD      IY,BUFF1        ;FPDE pointer
531F CD2454     01590           CALL    READ            ;Read fpde page to buff1
                01600 ; ---------- LOOP FOR 8 FPDE SLOTS PER FPDE PAGE --
5322 FDE5       01610 FPDE      PUSH    IY              ;save FPDE pointer
5324 FD7E00     01620           LD      A,(IY+0)        ;First byte of FPDE
5327 E690       01630           AND     90H             ;Mask for alloc FP/FXDE
5329 FE00       01640           CP      00H             ;Not allocated?
532B CA8853     01650           JP      Z,NXSLOT        ;Skip if no file
532E 111600     01660 SLOTCK    LD      DE,22           ;Rel byte 22H
5331 FD19       01670           ADD     IY,DE           ;Pt to first extent
5333 1600       01680 SLOTCK    LD      D,00H           ;Zero D
5335 DD210859   01690           LD      IX,BUFF2        ;New GAT
5339 218353     01700           LD      HL,GTABLE       ;SET Table
533C FD7E00     01710           LD      A,(IY+0)        ;Starting Lump
533F FEFF       01720           CP      0FFH            ;None?
5341 CA8853     01730           JP      Z,NXSLOT        ;Skip if so
5344 FEFE       01740           CP      0FEH            ;FXDE?
5346 CA8853     01750           JP      Z,NXSLOT        ;Skip if so
5349 5F         01760           LD      E,A             ;DE = Lump
534A DD19       01770           ADD     IX,DE           ;IX = Lump pos in GAT
534C FD7E01     01780 GCOUNT    LD      A,(IY+1)        ;Allocation
534F F5         01790           PUSH    AF
5350 E61F       01800           AND     1FH             ;Mask for No. grans
5352 3C         01810           INC     A               ;Real grans
5353 4F         01820           LD      C,A             ;Save in C
5354 F1         01830 GSTART    POP     AF
5355 E6E0       01840           AND     0E0H            ;Mask for start Gran
5357 07         01850           RLCA                    ;Convert..
5358 07         01860           RLCA                    ; to
5359 07         01870           RLCA                    ;  binary
535A F5         01880 SGLOOP    PUSH    AF
535B 3ACB55     01890           LD      A,(GPL)         ;Number grans/lump
535E 5F         01900           LD      E,A             ;Placed in E
535F F1         01910           POP     AF              ;Restore Start
5360 F5         01920           PUSH    AF
5361 BB         01930           CP      E               ;Less than GPL
5362 3805       01940           JR      C,SGOK          ;Skip if so
5364 F1         01950 ZSB       POP     AF              ;Restore Stack
5365 AF         01960           XOR     A               ;Zero A
5366 F5         01970           PUSH    AF              ;Save Gran Zero
5367 DD23       01980           INC     IX              ;Bump GAT pointer
5369 5F         01990 SGOK      LD      E,A             ;E=Start gran
536A 1600       02000           LD      D,00H
536C 218353     02010           LD      HL,GTABLE       ;SET table
536F 19         02020           ADD     HL,DE           ;HL=SET bit val
5370 7E         02030           LD      A,(HL)          ;Load it to A
5371 327753     02040           LD      (SB+3),A        ;Mod SET bit
5374 DDCB00C6   02050 SB        SET     0,(IX+0)        ;Allocate Gran
5378 F1         02060           POP     AF
5379 3C         02070           INC     A               ;Bump gran ptr
537A 0D         02080           DEC     C               ;More Grans?
537B 20DD       02090           JR      NZ,SGLOOP       ;Yes, loop for more
537D FD23       02100           INC     IY              ;Pt to the..
537F FD23       02110           INC     IY              ;  next Extent
5381 18B0       02120           JR      SLOTCK          ;Ck Next Extent
5383 C6         02130 GTABLE    DEFB    0C6H            ;BIT 0
5384 CE         02140           DEFB    0CEH            ;  1
5385 D6         02150           DEFB    0D6H            ;  2
5386 DE         02160           DEFB    0DEH            ;  3
5387 E6         02170           DEFB    0E6H            ;  4
5388 EE         02180           DEFB    0EEH            ;  5
5389 F6         02190           DEFB    0F6H            ;  6
538A FE         02200           DEFB    0FEH            ;  7
                02210 ; ---------- POSITION TO NEXT FPDE SLOT IN PAGE --
538B            02220 NXSLOT    EQU     $
538B FDE1       02230           POP     IY              ;Restore FPDE ptr
538D 112000     02240           LD      DE,32           ;Slot Gap
5390 FD19       02250           ADD     IY,DE           ;New FPDE
5392 05         02260           DEC     B               ;Slot count
5393 C22253     02270           JP      NZ,FPDE         ;Loop til end of page
                02280 ; ---------- LOOP FOR ALL FPDE PAGES ----------
5396 ED4BC555   02290 NXFPDE    LD      BC,(PCOUNT)     ;Page/Slot count
539A 0D         02300           DEC     C               ;Next FPDE
539B ED43C555   02310           LD      (PCOUNT),BC     ;Save Pointers
539F C21753     02320           JP      NZ,PAGE         ;Loop for rem pages
                02330 ; ---------- COMPARE OLD & NEW GAT SECTORS ----------
53A2            02340 HITCK     EQU     $
53A2 DD210000   02350           LD      IX,0000H        ;Non-match counter
53A6 210859     02360           LD      HL,BUFF2        ;New GAT table
53A9 11005A     02370           LD      DE,BUFF3        ;Old GAT table
53AC 0600       02380           LD      B,00H           ;256 bytes
53AE 1A         02390 CKLOOP    LD      A,(DE)          ;Get old GAT code
53AF BE         02400           CP      (HL)            ;Compare with new
53B0 2802       02410           JR      Z,SAME          ;Skip if the same
53B2 DD23       02420           INC     IX              ;Bump error count
53B4 23         02430 SAME      INC     HL              ;Next new GAT code
53B5 13         02440           INC     DE              ;Next old GAT code
53B6 10F6       02450           DJNZ    CKLOOP          ;Loop for whole table
53B8 DDE5       02460           PUSH    IX              ;Move error count
53BA D1         02470           POP     DE              ;to DE
53BB 7B         02480           LD      A,E             ;Errors always <256
53BC 21FA54     02490           LD      HL,ERRNUM       ;Point to error num $
53BF F5         02500           PUSH    AF              ;Save error count
53C0 CD0744     02510           CALL    4407H           ;Load to string
53C3 21FA54     02520           LD      HL,ERRNUM       ;Point to error msg
53C6 CD6744     02530           CALL    4467H           ;Display num errors
53C9 F1         02540           POP     AF              ;Restore num errors
53CA FE00       02550           CP      00H             ;No errors?
53CC 2850       02560           JR      Z,CLOSE         ;Yes, omit fix rtne
53CE 211955     02570           LD      HL,WRPRMT       ;Fix Y/N Msg
53D1 CD6744     02580           CALL    4467H           ;Display msg
53D4 CD4900     02590           CALL    49H             ;KBWAIT
53D7 F620       02600           OR      20H             ;Cvrt to l/c
53D9 FE79       02610           CP      79H             ;yes?
53DB 2808       02620           JR      Z,YES           ;then fixit
53DD 217555     02630 NO        LD      HL,WAMSG        ;Abort msg
53E0 CD6744     02640           CALL    4467H           ;Display it
53E3 1839       02650           JR      CLOSE           ;JP over write rtne
53E5 215855     02660 YES       LD      HL,WRMSG        ;Write message
53E8 CD6744     02670           CALL    4467H           ;Display it
                02680 ; ---------- POSITION TO GAT SECTOR ----------
53EB 11CE55     02690           LD      DE,FCB          ;Pt to FCB
53EE CD3F44     02700 SOF       CALL    443FH           ;Move to start of file
                02710 ; ---------- MOVE NEW HIT TO FCB BUFFER ----------
53F1 210859     02720 MOVE      LD      HL,BUFF2        ;New GAT buffer
53F4 110058     02730           LD      DE,BUFF1        ;FCB buffer
53F7 010001     02740           LD      BC,256          ;Bytes to move
53FA ED80       02750           LDIR                    ;Xfer it
                02760 ; ---------- PREPARE FCB FOR READ-PROTECT WRITE ----
53FC 3ACE55     02770           LD      A,(FCB)         ;FCB 1st byte
53FF F601       02780           OR      01H             ;Make read-protect
5401 32CE55     02790           LD      (FCB),A         ;Amend FCB
5404 3ACF55     02800           LD      A,(FCB+1)       ;FCB 2nd byte
5407 F640       02810           OR      40H             ;Do not update EOF
5409 32CF55     02820           LD      (FCB+1),A       ;Amend FCB
                02830 ; ---------- WRITE WITH VERIFY ----------
540C 210058     02840 WRITE     LD      HL,BUFF1        ;New GAT
540F 11CE55     02850           LD      DE,FCB          ;DE --> FCB
5412 0600       02860           LD      B,00H           ;LRL=256
5414 CD3C44     02870           CALL    443CH           ;Write with verify
5417 2805       02880           JR      Z,CLOSE         ;Exit if no error
5419 F680       02890 EREXIT    OR      80H             ;Make long err msg
541B CD0944     02900           CALL    4409H           ;Display error msg
541E CD2844     02910 CLOSE     CALL    4428H           ;Close FCB
5421 C32D40     02920           JP      402DH           ;Exit to DOS
                02930 ; ---------- READ SECTOR ----------
5424            02940 READ      EQU     $
5424 C5         02950           PUSH    BC              ;Save slot count
5425 210058     02960           LD      HL,BUFF1        ;FCB buffer
5428 11CE55     02970           LD      DE,FCB          ;FCB
542B 010000     02980           LD      BC,0000H        ;LRL=256
542E CD3644     02990           CALL    4436H           ;Read sector
```

```
5431 C1       03080         POP    BC              ;Restore slot count
5432 F5       03010         PUSH   AF              ;Save error count
5433 FE1C     03020         CP     1CH             ;EOF?
5435 CA1954   03030         JP     Z,EREXIT        ;Error exit
5438 FE1D     03040         CP     1DH             ;Past EOF?
543A CA1954   03050         JP     Z,EREXIT        ;Error exit
543D F1       03060         POP    AF              ;Restore error code
543E C0       03070 NOERR   RET    NZ              ;Continue if NZ
              03080 ; ---------- READ-PROTECT ERROR ----------
543F 3AD855   03090 NRPERR  LD     A,(NEXT)        ;Get next pointer
5442 3D       03100         DEC    A               ;Pt to last sector read
5443 E5       03110         PUSH   HL              ;Save FCB ptr
5444 21A855   03120         LD     HL,RPSR         ;Sector num
5447 CDD744   03130         CALL   44D7H           ;Post to $
544A 219D55   03140         LD     HL,NRPMSG       ;Nr msg
544D CD6744   03150         CALL   4467H           ;Display it
5450 E1       03160         POP    HL              ;Restore FCB ptr
5451 C9       03170         RET                    ;Continue processing
              03180 ; ---------- STRINGS AND STORAGE ----------
5452 46       03190 BANMSG  DEFM   'FIXGAT for NEWDOS80/V2 - Drive on Test --> '
     49 58 47 41 54 20 66 6F 72 20 4E 45 57 44 4F 53
     38 30 2F 56 32 20 2D 20 44 72 69 76 65 20 6F 6E
     20 54 65 73 74 20 2D 2D 3E 20
547D 30       03200 DRVNUM  DEFM   '0'
547E 0A       03210         DEFB   0AH
547F 0A       03220         DEFB   0AH
5480 44       03230 STATS   DEFM   'Disk Stats : '
     69 73 6B 20 53 74 61 74 73 20 3A 20
548D 2020     03240 NTC     DEFW   2020H
548F 48       03250         DEFM   'H Tracks, '
     20 54 72 61 63 6B 73 2E 20
5499 2020     03260 NSPT    DEFW   2020H
549B 48       03270         DEFM   'H Sectors/Track, '
     20 53 65 63 74 6F 72 73 2F 54 72 61 63 6B 2E 20
54AC 2020     03280 NLUMP   DEFW   2020H
54AE 48       03290         DEFM   'H Lumps/Disk,'
     20 4C 75 6D 70 73 2F 44 69 73 6B 2E
54BB 0A       03300         DEFB   0AH
54BC 20       03310         DEFM   '                '
     20 20 20 20 20 20 20 20 20 20 20 20
54C9 2020     03320 NGPL    DEFW   2020H
54CB 48       03330         DEFM   'H Grans/Lump, '
     20 47 72 61 6E 73 2F 4C 75 6D 70 2E 20
54D9 44       03340         DEFM   'DIRectory Lump/Sector '
     49 52 65 63 74 6F 72 79 20 4C 75 6D 70 2F 53 65
     63 74 6F 72 20
54EF 2020     03350 NDIR    DEFW   2020H
54F1 48       03360         DEFM   'H/'
     2F
54F3 2020     03370 DIRS    DEFW   2020H
54F5 20       03380         DEFM   ' H'
     20 48
54F8 0A       03390         DEFB   0AH
54F9 0D       03400         DEFB   0DH
54FA 20       03410 ERRNUM  DEFM   ' H '
     20 48 20
54FE 47       03420         DEFM   'Granule Allocation Errors'
     72 61 6E 75 6C 65 20 41 6C 6C 6F 63 61 74 69 6F
     6E 20 45 72 72 6F 72 73
5517 0A       03430         DEFB   0AH
5518 0D       03440         DEFB   0DH
5519 52       03450 NRPRMT  DEFM   'Reply Y/N to Repair Diskette Granule Allocation Table'
     65 70 6C 79 20 59 2F 4E 20 74 6F 20 52 65 70 61
     69 72 20 44 69 73 6B 65 74 74 65 20 47 72 61 6E
     75 6C 65 20 41 6C 6C 6F 63 61 74 69 6F 6E 20 54
     61 62 6C 65
554E 0A       03460         DEFB   0AH
554F 0D       03470         DEFB   0DH
5550 57       03480 NRMSG   DEFM   'Writing the Corrected GAT Sector Now'
     72 69 74 69 6E 67 20 74 68 65 20 43 6F 72 72 65
     63 74 65 64 20 47 41 54 20 53 65 63 74 6F 72 20
     4E 6F 77
5574 0D       03490         DEFB   0DH
5575 2A       03500 NAMSG   DEFM   '**********  ABORTING  *** Per Request'
     2A 2A 2A 2A 2A 2A 2A 2A 20 20 20 41 42 4F 52 54
     49 4E 47 20 20 20 2A 2A 2A 20 20 50 65 72 20 52
     65 71 75 65 73 74
559C 0D       03510         DEFB   0DH
559D 44       03520 NRPMSG  DEFM   'DIR/SYS FRS : '
     49 52 2F 53 59 53 20 46 52 53 20 3A 20
55AB 30       03530 RPSR    DEFM   '00 '
     30 20
55AE 69       03540         DEFM   'is not Read-Protected'
     73 20 6E 6F 74 20 52 65 61 64 2D 50 72 6F 74 65
     63 74 65 64
55C3 0A       03550         DEFB   0AH
55C4 0D       03560         DEFB   0DH
55C5 0000     03570 PCOUNT  DEFW   0000H
55C7 00       03580 DIRPT   DEFB   00H
55C8 00       03590 LUMPS   DEFB   00H
55C9 00       03600 TC      DEFB   00H
55CA 00       03610 SPT     DEFB   00H
55CB 00       03620 GPL     DEFB   00H
55CC 00       03630 GAB     DEFB   00H
55CD 00       03640 FLAG    DEFB   00H
0091          03650 PDRIVE  EQU    91H
              03660 ; ---------- F C B ----------
55CE 44       03670 FCB     DEFM   'DIR/SYS:'      ;Filespec
     49 52 2F 53 59 53 3A
55D6 30       03680 DRIVE   DEFM   '0'             ;Drivespec
55D7 0D       03690         DEFB   0DH
55D8 00       03700 NEXT    DEFB   00H             ;Next sector pointer
55D9 00       03710         DEFB   00H
55DA 00       03720 LEN     DEFB   00H             ;LSB length of file
55DB 00       03730         DEFB   00H
55DC 00       03740 LUMP    DEFB   00H             ;Starting lump
55DD 00       03750 GRAN    DEFB   00H             ;Number of grans
0010          03760 EXT     DEFS   10H             ;Extents
              03770 ; ---------- STORAGE BUFFERS ----------
5800          03780         ORG    5800H
0100          03790 BUFF1   DEFS   256             ;FCB buffer
0100          03800 BUFF2   DEFS   256             ;New GAT CODE buffer
0100          03810 BUFF3   DEFS   256             ;Orig GAT sector
5200          03820         END    PARSER
00000 TOTAL ERRORS
```

```
BADNUM 5215   BANMSG 5452   BANNER 5290   BIG    52CC   BUFF1  5800
BUFF2  5900   BUFF3  5A00   CALCS  5220   CKLOOP 53AE   CLOSE  541E
DIRPT  55C7   DIRS   54F3   DN     5207   DOALLO 52C5   DRIVE  55D6
DRVNUM 547D   EREXIT 5419   ERRNUM 54FA   EXT    55DE   FCB    55CE
FGAB   5286   FGLOOP 5289   FLAG   55CD   FPDE   5322   GAB    55CC
GCOUNT 534C   GPL    55CB   GPLVAL 5261   GRAN   55DD   GSTART 5354
GTABLE 5383   HITCK  53A2   INIT   5283   INITZ  5204   INZ    52E8
LEN    55DA   LUMP   55DC   LUMPS  55C8   MOVE   53F1   NDIR   54EF
NEXT   55D8   NGPL   54C9   NLUMP  54AC   NO     5300   NOERR  543E
NRPERR 543F   NRPMSG 559D   NSPT   5499   NTC    548D   NUMBER 5280
NAMSG  5575   NXFPDE 5396   NXSLOT 5388   OPEN   5299   PAGE   5317
PARSER 5200   PBLOOP 530E   PCOUNT 55C5   PDRIVE 0091   POSTDR 521A
PTDIR  5231   READ   5424   RPSR   55AB   SAME   53B4   SB     5374
SDIR   526E   SETBUF 52B6   SETUP  52A0   SGLOOP 535A   SGOK   5369
SIZE   52F6   SKPHIT 5314   SLOTCK 5333   SLOTOK 532E   SOF    53EE
SPT    55CA   SPTNO  5255   STATS  5480   SUMLMP 5230   TC     55C9
TCOUNT 5249   WRITE  540C   WRMSG  5550   WRPRMT 5519   YES    53E5
ZSB    5364
```

ALLWRITE! BUG(?)

We almost got bit by this one last issue, and wanted to warn our readers who use ALLWRITE! We published an article entitled ANCHOR SIGNALMAN MARK XII DTR FIX. At the top of the parts list, the following appeared:

1 - 2N4401 transistor

But this part wasn't there in the final printout! Apparently a carriage return got left out, and the ALLWRITE! formatting line just before the parts list looked like this:

¦foof¦in+6¦ll70 1 - 2N4401 transistor

You would think that ALLWRITE! would complain about a formatting line like this, but it doesn't. I see that as a BUG in ALLWRITE!, since some really important lines of text could get omitted this way. The folks at PROSOFT get NORTHERN BYTES, so I hope they'll decide to fix ALLWRITE! so that it will protest w/ coming across lines like the one above.

As for our last issue, we caught the bug after the master copy had been made, but (fortunately) before we actually printed the page in question. We had to make up a new master copy, but that was the only damage. Had we failed to notice this, some folks might have wondered why their MODEM fix wouldn't work!

PACKET SWITCHING NETWORKS
Revised 4-Mar-85
Contains former file GERMAN.TXT (modified)
by Hans G. Michna (CompuServe I.D. 74776,2361)

[This article was downloaded from CompuServe by Greg Small. I'm reprinting it here because it could be of considerable interest to many of our readers outside the continental U.S.A., as well as to those of our readers who travel overseas.]

If you are interested in CompuServe access and file transfer through packet switching networks like those found in some 50 countries all around the world - read on. I have solved the problem of downloading from CompuServe through DATEX-P (Germany) and jumped 3 meters high when it finally worked. The solution applies to other countries also.

Contents:

[1] QUICK INTRODUCTION

If you are not interested in the details - the next time you enter CompuServe via a packet switching network do this:

- Logon until you see the "User ID:" prompt. (Any time later than this will also work.)
- Enter a Ctrl-P (hold down the Ctrl key and press P once). You are now talking to your network instead of CompuServe.
- Enter "set 3:126,4:0,5:1,9:0,12:0,118:8,119:21,120:22,125:10" without the quotes and press Return.
- Press Return a second time.
- Enter your user ID and continue as usual.

This gives you

- prompt response to Ctrl command characters like Ctrl-C and Ctrl-O,
- uninterrupted text uploads
- no disturbing fill characters,
- cheaper local PAD line editing with Backspace, Ctrl-U and Ctrl-V and mostly clean lines when in an online conference.

IMPORTANT NOTES:

(1) Check if your network's command prefix is really Ctrl-P by entering Ctrl-P, then a nonsense command, then one Return. If you get a network error message Ctrl-P is all right. If you get a CompuServe message like ?XXXXXX - INVALID USER ID - TRY AGAIN / User ID: then Ctrl-P is wrong for you and you have to ask the network operator for the correct network command prefix or attention character.

(2) Your computer may freeze especially after finishing a connection. Type a Ctrl-Q (hold down Ctrl and press Q) and continue normally.

(3) See chapter --- XMODEM FILE TRANSFERS --- if necessary.

[2] DETAILED EXPLANATION

THE SYSTEM: Connection to CompuServe is normally done in the following way:

Async Terminal or Microcomputer - PAD - Packet Switching Network - Gateway - CompuServe Network

THE PROBLEM: Using a microcomputer I can hardly afford a direct X.25 channel to the network. So I use the public PAD (Packet Assembly and Disassembly) unit provided by the "Post" which accepts the standard async signals, 300 or 1200 bps full duplex. The PAD's behaviour is the source of all potential trouble.

Fortunately the PAD can be controlled by the user to such an extent that even XMODEM and similar file transfers can be facilitated. The standard settings, however, are completely inadequate, especially for binary file transfers.

Uploading (from you to CompuServe) is more difficult than downloading because the PAD normally interprets some special characters send by your asynchronous terminal or microcomputer which do not occur in the other direction, e.g. X-ON, X-OFF and PAD command prefixes.

THE SOLUTION: Let us look at the problems in detail. In the following the PAD parameter numbers and settings apply directly to the German DATEX-P network. The international parameters as well as the basic problems and processes, however, apply to all packet switching networks that are accessed by asynchronous terminals through PADs.

PAD COMMANDS: "SET parameter_no : value , parameter_no : value , ..." sets PAD parameters. "PAR?" lists the current parameter settings. "PROF profile_no" resets all parameters to predefined values and "PROF? profile_no" lists the predefined values of profile_no without applying them. Do not key in the quotes. In most cases you will only need the SET command.

[3] INTERNATIONAL PAD PARAMETERS

PAD COMMAND PREFIX: When connected with CompuServe you can still issue commands to the PAD. A special character (DLE=^P, Ctrl-P, check for your particular network) switches the PAD into command mode. The following lines are not sent to CompuServe any longer but are taken as commands by the PAD. Two consecutive Returns get you back into the connection. For example to set parameter 3 to the value of 126 and parameter 4 to zero you have to do this: Key in ^P to switch the PAD into command mode, then key in "set 3:126,4:0". Do not key the quotes. Finally press Return twice to get out of command mode again.

This enables you to change PAD parameters while you are already connected. We need this facility to adjust the PAD to our needs, especially because setting the PAD parameters before establishing the connection does not always work. In DATEX-P the parameters change when the connection is established and these changes are not always favourable.

You can tell the PAD to let DLE (^P) characters pass unnoticed with "set 1:0". However you will be able to get into command mode never again during the course of that connection. For binary uploads "set 1:0".

PAD ECHO: The PAD will usually echo everything you send back to you. This enables you to use full duplex transmission so you can see all transmission errors. For file transfers switch the echo off with "set 2:0". Afterwards switch the echo back on with "set 2:1".

FORWARD DATA CHARACTER: The PAD can be told to form a packet and forward it before the packet is filled completely. This is necessary because often you will not fill up a packet. Imagine you want to enter a menu selection. You key just one digit and a Return. Without a Forward Data Character setting the PAD would now wait for you to fill the remaining 126 bytes of this packet before it is sent on its way to CompuServe. You will also want the PAD to forward control characters like ^C and Escape immediately.

For file transfers, especially binary uploads, it is not desired to forward packets that are not completely filled for economic reasons. After all you pay for the packet, not the characters in it (more exactly for the segment). "set 3:126" to forward data after all control characters and DEL. "set 3:2" to forward data after Return characters only. "set 3:0" (no Forward Data Character) for file transfers.

FORWARD DATA TIME LIMIT: If you have no Forward Data Character you have to tell the PAD to forward data anyway after a certain time because the transfer protocol (e.g. XMODEM) will not always fill the packet completely. "set 4:8" for file transfers. This yields a .32 s limit (8 * 40 ms). "set 4:0" for normal operation with a Forward Data Character (no time limit). The maximum value for this parameter is 255.

X-ON/X-OFF FROM PAD TO DTE: This parameter enables the PAD to stop and restart your transmission by sending X-OFF and X-ON bytes to your computer. "set 5:0" if this is not desired. "set 5:1" for uploads.

There is a minor problem when you allow the PAD to send X-OFFs to you. Sometimes, especially after finishing or breaking a connection, the PAD sends an X-OFF and your computer seems to freeze. Simply key a Ctrl-Q (X-ON) and everything is all right again.

PAD MESSAGES: You may forbid the PAD to send its own messages to you with "set 6:0" during a straight text download if you are afraid of "Parity Error" or similar messages in the middle of a received file. "set 6:1" normally.

BREAK: There are different PAD reactions to a break signal sent by you. Parameters 7 and 8 deal with these. Leave them alone, we do not normally use break signals.

NUMBER OF FILL CHARACTERS AFTER RETURN: Always "set 9:0" unless you have a real Teletype that cannot return the carriage in time.

LINE LENGTH: Always "set 10:0" unless you want the PAD to break long lines with additional Returns and Linefeeds. (Maximum value 255)

X-ON/X-OFF FROM DTE TO PAD: "set 12:0" for binary uploads to make the PAD ignore X-ON and X-OFF characters. "set 12:1" otherwise.

[4] NATIONAL PAD PARAMETERS (DATEX-P AND OTHERS)

DELETE CHARACTER, DELETE LINE, REPEAT LINE, ADDITIONAL FORWARD DATA CHARACTERS: These parameters allow local line editing performed by the PAD. With "set 118:8,119:21,120:22" the PAD can be instructed to perform the duties of the Backspace, Ctrl-U and Ctrl-V commands locally which saves you money whenever you use these commands. If in doubt "set 118:0,119:0,120:0,121:0,122:0".

PARITY: If you use 7 bit with parity you may "set 123:1" to make the PAD check your parity bit. "set 123:0" for 8 bit character length and for all binary file transfers.

DELAY OUTPUT DURING INPUT: Parameter 125 can make the PAD hold incoming data until you have finished typing a line, a very handy feature for online conferences. "set 125:10" for a maximum hold time of 10 seconds. Don't worry, the PAD will not hold all incoming data for 10 seconds when you type. As soon as you press Return all upheld data will start flowing again. "set 125:0" for file transfers under protocol.

INSERT LINEFEED: Try whether "set 126:0" works with your equipment. If the Return key does not advance to the next line let the PAD echo a linefeed after each Return sent by you with "set 126:4" which is the standard setting. (Other settings are: 1 = insert linefeeds after Returns sent by host through PAD to DTE, 5 = both 1 and 4.)

[5] XMODEM FILE TRANSFER

CompuServe's XMODEM has a special problem. When you initiate an XMODEM file transfer CompuServe automatically sets a Transparent Profile (which is nice) but does this just an instant too late. Thus the initial handshaking is spoiled and the file transfer always gets stuck.

Fortunately we now know enough to take things into our own hands and control the PAD ourselves until the CompuServe programmers get this fixed.

Being too lazy to SET all those parameters individually we can make use of the Transparent Profile our networks offer. In Germany and Canada and probably many or all other countries the Transparent Profile is called "PROF 3".

To start an XMODEM file transfer do this:

1. Go to the point in CompuServe approximately one command before starting the download or upload.
2. Enter Ctrl-P (^P, DLE, the network's command prefix, check for your particular network).
3. Type "prof 3" without the quotes and press Return.
4. Press Return a second time. You are now back in connection with CompuServe.
5. Issue the last command(s) to start the download or upload process. There is no echo any more, i.e. you don't see what you are typing. Don't worry, just carry on.
6. After the transfer, if you don't like the standard parameter setting, SET the parameters again by using a ^Pset command.

[6] PLEASE WRITE

Please drop me a line if you have used the information in this file unsuccessfully or successfully! I will update this file whenever new knowledge becomes available. Do not send me SIG messages since I do not come here often, use electronic mail. I would especially like to know:

Does Ctrl-P work in your network?
Is PROF 3 the transparent profile in your network?
Could you upload straight text?
Could you download with XMODEM?
Could you upload with XMODEM?
Could you locate any error or missing information in this text?
Do you have any other information you think should be included here?
Do you have any other information that might be of interest to me?

Thank you very much for your interest and co-operation.

NOTE: CompuServe can be reached directly with "0 3132" and also through Tymnet and Telenet. The numbers are "0 3106,CPS01", "0 3106,CIS02", "0 3106,CIS03", "0 3106,CIS04", "0 3106 001133", "0 3106 001134", "0 3106 00337300" for Tymnet and "0 3110 20200202" and "0 3110 61400227" for Telenet. Do not key the quotes. You may have to substitute the leading 0 by your network's international prefix like 1 or C or P 1. Often you may omit spaces. You may substitute "0 3107" for "0 3106" which presumably enforces the use of an ITT gateway. There is a surcharge for all connections except 0 3132.

Hans G. Michna 74776,2361

A PATCH OF A PATCH OF A PATCH

by Lawrence C. White VE3FNE
(36 Kempster Avenue, Ottawa, Ontario, CANADA K2B 6M1)

To answer Nate Salsbury's question in NORTHERN BYTES Volume 5 Number 4 (pages 9-11) his changes will read in a tape at 500 baud but will not write tapes. I made additions to Nate's program that allow you to read and write 500 baud tapes on the Model III. These revisions should work with Versions 1.06 and 1.07 of EDTASM PLUS. I made these changes on Version 1.06. There was one error in the printout on Page 11 of the aforementioned article - line 02140 should read as follows:

```
LD          HL,4383    ; TO EA WARM REENTRY
```

Since I did not put in the patch for the use of disks (to tell the truth, I could not find the correct place to insert Arne's program due to a problem in my disassembler), I started my patch at 7284H and completed the first patch at 72C8H (this is the patch that is in Nate's article with the exception of everything from line 02430 to the end of the patch).

I started my patch at 72C9H, this is a repeat of Nate's patch but is called from location 4FD4H (in version 1.06). The patch follows:

```
72C9    CD0844    CALL    4408H
72CC    AF        XOR     A
72CD    321142    LD      (4211H),A
72D0    210342    LD      HL,4203H
72D3    11C572    LD      DE,SAVBRK
72D6    010300    LD      BC,03H
72D9    EDB0      LDIR
72DB    3EC3      LD      A,0C3H
72DD    320342    LD      (4203H),A
72E0    218343    LD      HL,4383H
72E3    220442    LD      (4204H),HL
72E6    0ED3      LD      C,0D3H
72E8    C9        RET
72E9    E972      (BUFST)
72EB    E972      (BUFEND)
72ED    FFFF      (ENDTXT)
```

For the above patch to work address 4FD4H must be changed to CALL 72C9H and address 4459H must be changed to 72E9H as follows:

```
4FD5  C9 72
4459  E9 72
```

As written the patches will read and write source code at 500 baud on the Model III. After the source code has been read or written, enter ZBUG and change the value 0D3H at locations 72A2H and 72E7H to 55H, the program can now be assembled and the object code written to tape and ZBUG can now write and read object tapes.

These patches and changes work but if anyone can find an easier method I would appreciate hearing from them. Also I would like to know where to start the disk patch, written by Arne Rohde, for the version 1.06.

ZAPS FOR USERS OF VIDEO4 WITH NEWDOS/80

Contributed by Bob Brumley

1. To fix the directory display so as to display filenames 5 across, and to use all 24 lines of the video display:

```
SYS8/SYS, FRS 01, byte 0C  change  0D  to  15
SYS8/SYS, FRS 01, byte 12  change  05  to  06
SYS8/SYS, FRS 02, byte 8E  change  04  to  05
SYS8/SYS, FRS 02, byte 96  change  0F  to  17
```

2. Zaps to EDTASM and DISASSEM to use all 24 lines of the video display during screen paging:

```
EDTASM/CMD,   FRS 10, byte 72  change  10  to  18
DISASSEM/CMD, FRS 01, byte 75  change  10  to  18
```

INPUT@ DISK BASIC ENHANCEMENT

by Gil Spencer VK2JK

1. My Philosophy for Disk BASIC Enhancements

Lots of routines which enhance Disk BASIC have been published (for example, the "Improved Ampersand Function" in Appendix VI of TRS-80 ROM Routines Documented). Authors generally tuck the extra code at high addresses in memory, thus requiring HIMEM protection as well as re-assembly for 32K, 48K, etc. [unless you take the extra effort to make the program self-relocating and to protect itself in memory -editor]. This approach has always bothered me because it means you have to remember to set HIMEM and because other bits are often kept at these high addresses. It seems to me that a better place to hide these extra chunks of code...those which are used only in conjunction with Disk BASIC...is before the beginning of BASIC's Program Statement Table (PST). Routines take up no more room there. Often they consume less because initialization code is only needed for the Cold Start. This "setup" code can be discarded after initialization. For examples of what I'm trying to say cf. my paper "Enhance Your Level II BASIC" on pages 202-214 in "80-Micro" for July, 1981. See also the assembler code for TATUPNI below. Summarized, the steps are these:

a. Divide your procedure into two parts: ACTIVE part and SETUP part.

b. Write your code in this sequence: ACTIVE part (last byte must be a '00'), then SETUP part.

c. At the very end of SETUP part do this: Force "BASIC Begin" pointer (40A4H-40A5H) to point to SETUP; perform a BASIC "NEW" (CALL 1B4DH); exit to BASIC 'READY'.

The only unanswered question now is where to ORG your assembly. The answer is found by peeking at "BASIC Begin" pointer for your BASIC in its "natural state".

2. INPUT@ Disk BASIC enhancement

When you write BASIC programs (particularly business programs) where input is to be entered by accounting clerks, etc., one of the BIG problems is to make your program sufficiently "bullet proof". How do you prevent the input operator from hitting unwanted <BREAK>, <CLEAR> or <ENTER> keys? How do you allow only numerals (plus, perhaps, "+", "-" or ".") in fields which must contain only such input? How does the input operator abort if a mistake is realized during input? How do you insist on, say, five characters in the field (no more, no less) if that's what the program requires? Lewis Rosenfelder explored this problem in his masterwork "BASIC Faster and Better & Other Mysteries", Chapter 13 "Data Entry Handlers". A potentially even better solution is presented by Roger A. Smith, Jr. in his paper "Easy Input" (page 109, November 1984 issue of "80-Micro"). I've embroidered upon Smith's paper and the result is TATUPNI (Input At, spelled backwards!). The assembler source code (suitable for the Model III running NEWDOS/80) is printed below, along with a BASIC demonstration program to show how TATUPNI works. Presently, the CMD"S" return to DOS is unpredictable; I've not yet tried to figure out why.

[Editor's note: This program was originally assembled using NEDAS v4.1B (a version of Misosys's EDAS rewritten for NEWDOS/80 that Gil uses). Since most NORTHERN BYTES readers will not have that assembler available, I have slightly modified the source code that Gil sent me so that it will run under the standard Apparat EDTASM (supplied with NEWDOS/80). If you wish to use a version of EDAS to assemble, simply change all DEFB's to DB, all DEFS's to DS, and all DEFW's to DW.]

Source code for TATUPNI/CMD

```
00100 ;TATUPNI (INPUTAT, Backwards!)          v 1.2-850112
00110 ;From: Gil Spencer / VK2JK / Phone 61 (02) 969-7860
00120 ;      Box 300 / Spit Junction /NSW  2088 / Australia
00130 ;Syst: 48k 2-disk TRS-80 Mod3 / Fax-80 Printer.
00140 ;DOS : NEWDOS80 v2.0 (through Zap 866).
00150 ;
00160 ;A dynamic DISK BASIC patch, adding 2 new cmds:
00170 ;       "INPUT@ printat,flag,USING string ;variable".
00180 ;       "&POS", Fn returning current cursor position.
00190 ;
00200 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00210 ;xx  From:          "EASY INPUT"              xx
00220 ;xx  By:              Roger Smith             xx
00230 ;xx  "80 Micro" Nov 1984 issue, Page 109-120  xx
00240 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00250 ;Enhancements: by Gil Spencer.
00260 ;TATUPNI resides before BASIC PST & consumes <600 bytes.
00270 ;To use, execute Disk BASIC. Then execute TATUPNI from
00280 ;BASIC with CMD"TATUPNI", or similar (depends on DOS).
00290 ;
00300 ;<Pgm TATUPNI for TRS-80 Model I/III.>'
00310 ;<Dynamically activates 'INPUT@' and '&POS' in DISK BASIC.>'
00320 ;<For basic details cf. "Easy Input" in Nov 84 "80 Micro",
00330 ; pg 109.  For upgrade info cf. TATUPNI/BAS.>'
00340 ;
00350 ;"Un-comment" applicable ORG below.
6A46          00360          ORG     6A46H   ;NEWDOS/80 begin PST Disk BASIC.
              00370 ;        ORG     6A7DH   ;TRSDOS M3 begin PST Disk BASIC.
00380 ;
00390 ;Global Equates:
005F          00400 CURCHR   EQU     5FH     ;Cursor char = Underscore.
00EE          00410 BCKCHR   EQU     0EEH    ;Background char = "Shadow X".
              00420 ;ROM Equates:
19AE          00430 BASIC    EQU     19AEH   ;Decker's BASIC entry (Mod1/3).
1B4D          00440 BASNEW   EQU     1B4DH   ;Mod3 BASIC "NEW" rtn.
19A2          00450 ERROR    EQU     19A2H   ;Error Handling.
2337          00460 EVAL     EQU     2337H   ;Evaluate Expression.
1E4A          00470 FCERR    EQU     1E4AH   ;FC Error.
2B81          00480 GETINT   EQU     2B81H   ;Evaluate Integer Expression.
1F33          00490 HDLSTR   EQU     1F33H   ;"Handle" $ & put it with $ vars.
2B6B          00500 INPUTF   EQU     2B6BH   ;Finish of INPUT ROM rtn.
002B          00510 KBCHAR   EQU     002BH   ;Check keyboard for char.
1997          00520 SNERR    EQU     1997H   ;Syntax Error.
0AF4          00530 TEST     EQU     0AF4H   ;TM Error if NRA1 NOT $.
0AF6          00540 TMERR    EQU     0AF6H   ;TM Error.
2680          00550 VARPTR   EQU     2680H   ;Returns VARPTR.
              00560 ;Reserved RAM Equates:
40A4          00570 BASBGN   EQU     40A4H   ;Pointer to beginning of BASIC.
4019          00580 CAPLOK   EQU     4019H   ;( )=0=uplc. ( )<>0=CAPS only.
40A6          00590 CURPOS   EQU     40A6H   ;Cursor Position (0-63) on line.
4020          00600 CURSAD   EQU     4020H   ;Cursor Address.
4099          00610 INKEY    EQU     4099H   ;Last Key Pressed.
40A7          00620 KEYBUF   EQU     40A7H   ;Pointer to Keyboard Buffer.
40B5          00630 LSPADR   EQU     40B5H   ;Literal $ Pool Table (L$PT).
40B3          00640 LSPNXT   EQU     40B3H   ;Nxt Avail Loc in L$PT.
4194          00650 VAND     EQU     4194H   ;& Vector.
41D6          00660 VINPUT   EQU     41D6H   ;Input Vector.
40AF          00670 VTF      EQU     40AFH   ;Variable Type Flag.
4121          00680 NRA1     EQU     4121H   ;NRA1 (Integer & $).
              00690 ;
6A46          00700 NUVAN    EQU     $       ;New '&' Vector points here!
6A46 23       00710          INC     HL              ;Check next char.
6A47 7E       00720          LD      A,(HL)
6A48 2B       00730          DEC     HL              ;Restore Ptr.
6A49 FEDC     00740          CP      0DCH            ;Is it "POS" token?
6A4B 2803     00750          JR      Z,NUVAN1        ;-If Yes, JP.
0003          00760 ANDPTX   DEFS    3               ;-If No, Exit new code.
              00770 ;NOTE: ANDPTX is modified to Reserved RAM JP by SETUP.
6A50 E5       00780 NUVAN1   PUSH    HL              ;Store Line Location.
6A51 3E02     00790          LD      A,2             ;Store "INT" in Type Loc.
6A53 32AF40   00800          LD      (VTF),A
6A56 2A2040   00810          LD      HL,(CURSAD)     ;Get Cursor Address.
6A59 0180C4   00820          LD      BC,-3C80H       ;Subtract 3C80h.
6A5C 09       00830          ADD     HL,BC
6A5D 222141   00840          LD      (NRA1),HL       ;Store result in REG1.
6A60 E1       00850          POP     HL              ;Restore Ptr.
6A61 23       00860          INC     HL              ;Bump past token.
6A62 D7       00870          RST     10H             ;Skip any spaces.
6A63 C9       00880          RET
              00890 ;
6A64          00900 NUVIN    EQU     $       ;New INPUT vector points here.
6A64 FE40     00910          CP      '@'             ;Is it INPUT
6A66 2803     00920          JR      Z,NUVIN1        ;-If Yes, JP
0003          00930 INPPTX   DEFS    3               ;-If No, exit new code.
              00940 ;NOTE: INPPTX is changed to Reserved RAM JP by SETUP.
6A6B CD812B   00950 NUVIN1   CALL    GETINT          ;Evaluate Integer.
6A6E FE04     00960          CP      4               ;Is it >1023?
6A70 D24A1E   00970          JP      NC,FCERR        ;-If yes, JP
6A73 E5       00980          PUSH    HL              ;-Else, Store Ptr.
6A74 21003C   00990          LD      HL,3C00H        ;Start of Video.
6A77 19       01000          ADD     HL,DE           ;Add Offset.
6A78 222040   01010          LD      (CURSAD),HL     ;Cursor Address.
6A7B E1       01020          POP     HL              ;Restore Ptr.
6A7C CF       01030          RST     8               ;Test if next $ byte =...
6A7D 2C       01040          DEFB    ','             ;...comma.
6A7E 7E       01050          LD      A,(HL)  ;Get char after ',' in $.
6A7F FEBF     01060          CP      0BFH            ;Is it USING token?
```

```
6A81 2810   01070        JR    Z,USING     ;-If yes, JP.
6A83 2B     01080        DEC   HL          ;-Else Decrement Ptr.
6A84 CD012B 01090        CALL  GETINT      ;Evaluate Integer.
6A87 7B     01100        LD    A,E         ;Get LSB of Integer.
6A88 32836C 01110        LD    (FLAG),A    ;Store @ Flag.
6A8B CF     01120        RST   8           ;Test if next $ byte =...
6A8C 2C     01130        DEFB  ','         ;...a comma.
6A8D 7E     01140        LD    A,(HL)      ;Get next char.
6A8E FEBF   01150        CP    0BFH        ;Is it USING token?
6A90 C29719 01160        JP    NZ,SNERR    ;-If No, JP.
6A93 23     01170 USING  INC   HL          ;Increment Ptr.
6A94 E5     01180        PUSH  HL          ;Store Ptr.
6A95 CD3723 01190        CALL  EVAL        ;Evaluate Expression.
6A98 E1     01200        POP   HL          ;Restore Ptr.
6A99 23     01210 LOOPU  INC   HL          ;Increment Ptr.
6A9A 7E     01220        LD    A,(HL)      ;Get char.
6A9B B7     01230        OR    A           ;Is it zero?
6A9C CA9719 01240        JP    Z,SNERR     ;If yes, JP.
6A9F FE3B   01250        CP    ';'         ;Is it semi-colon?
6AA1 20F6   01260        JR    NZ,LOOPU    ;-If No, then loop.
6AA3 23     01270        INC   HL          ;Increment Ptr.
6AA4 7E     01280 LOOPU2 LD    A,(HL)      ;Get char.
6AA5 B7     01290        OR    A           ;Is it zero?
6AA6 CA9719 01300        JP    Z,SNERR     ;-If Yes, JP.
6AA9 FE20   01310        CP    ' '         ;Is it a space?
6AAB 2003   01320        JR    NZ,EOL      ;-If No, then EOL.
6AAD 23     01330        INC   HL          ;-Else, Increment Ptr...
6AAE 18F4   01340        JR    LOOPU2      ;...and loop.
6AB0 22886C 01350 EOL    LD    (PTR),HL    ;Store Ptr.
6AB3 3AAF40 01360        LD    A,(VTF)     ;Get variable type.
6AB6 FE03   01370        CP    3           ;Is it a $?
6AB8 C2F60A 01380        JP    NZ,TMERR    ;-If No, JP.
6ABB ED5B2141 01390      LD    DE,(VRA1)   ;Get VARPTR.
6ABF 1A     01400        LD    A,(DE)      ;A=$ length.
6AC0 B7     01410        OR    A           ;Is it zero?
6AC1 CA4A1E 01420        JP    Z,FCERR     ;-If Yes, JP.
6AC4 47     01430        LD    B,A         ;-Else B= $ length.
6AC5 32FD6B 01440        LD    (TEMP),A    ;Store length.
6AC8 13     01450        INC   DE          ;Increment VARPTR.
6AC9 1A     01460        LD    A,(DE)      ;Get LSB of Location.
6ACA 6F     01470        LD    L,A         ;L=LSB
6ACB 13     01480        INC   DE          ;Increment VARPTR.
6ACC 1A     01490        LD    A,(DE)      ;Get MSB of Location.
6ACD 67     01500        LD    H,A         ;HL--> $.
6ACE 22846C 01510        LD    (STRING),HL ;Store HL.
6AD1 0E00   01520        LD    C,0         ;Counter=0.
6AD3 7E     01530 LOOP   LD    A,(HL)      ;Count the $s in the $.
6AD4 FE23   01540        CP    '#'
6AD6 2001   01550        JR    NZ,NEXT
6AD8 0C     01560        INC   C
6AD9 23     01570 NEXT   INC   HL
6ADA 10F7   01580        DJNZ  LOOP
6ADC 79     01590        LD    A,C         ;A=Number of $s.
6ADD 32FF6B 01600        LD    (LEN),A     ;Store A.
6AE0 B7     01610        OR    A           ;Is it zero?
6AE1 CA4A1E 01620        JP    Z,FCERR     ;-If Yes, JP.
6AE4 06EE   01630        LD    B,BCKCHR    ;B=Background char.
6AE6 ED5B2040 01640      LD    DE,(CURSAD) ;Get Cursor Address.
6AEA 2A846C 01650        LD    HL,(STRING) ;Get Ptr to $.
6AED E5     01660        PUSH  HL          ;Store $ Ptr.
6AEE D5     01670        PUSH  DE          ;Store Cursor Location.
6AEF 7E     01680 LOOP1  LD    A,(HL)      ;Get char.
6AF0 FE23   01690        CP    '#'         ;Is it #?
6AF2 2001   01700        JR    NZ,CONTZ    ;-If no, JP.
6AF4 78     01710        LD    A,B         ;-Else A=Background.
6AF5 12     01720 CONTZ  LD    (DE),A      ;Send A to video.
6AF6 CD266B 01730        CALL  INCDE       ;Increment cursor loc.
6AF9 23     01740        INC   HL          ;Point to next char.
6AFA 3AFD6B 01750        LD    A,(TEMP)    ;Get length of $.
6AFD 3D     01760        DEC   A           ;Decrement it.
6AFE 32FD6B 01770        LD    (TEMP),A    ;Store decremented len.
6B01 20EC   01780        JR    NZ,LOOP1    ;Loop til TEMP=0.
6B03 3AFF6B 01790        LD    A,(LEN)     ;Get number of $s.
6B06 4F     01800        LD    C,A         ;Store in C.
6B07 ED532040 01810 CONT1 LD   (CURSAD),DE ;Store Cursor Address.
6B0B 7B     01820        LD    A,E         ;Compute CURPOS & store.
6B0C E63F   01830        AND   3FH
6B0E 32A640 01840        LD    (CURPOS),A
6B11 D1     01850        POP   DE          ;Move Curs Loc to DE.
6B12 E1     01860        POP   HL          ;Move Start $ loc to HL.
6B13 ED53086C 01870      LD    (SCREEN),DE ;Store curs loc @ SCREEN.
6B17 79     01880 LOOP2  LD    A,C         ;A=# INP chars left.
6B18 B7     01890        OR    A           ;Is it zero?
6B19 2813   01900        JR    Z,MATCH     ;-If Yes, JP.
6B1B 7E     01910        LD    A,(HL)      ;-Else Get char.
6B1C FE23   01920        CP    '#'         ;Is it #?
6B1E 280E   01930        JR    Z,MATCH     ;-If Yes, JP.
6B20 23     01940 LOOP3  INC   HL          ;-Else, Next char.
6B21 CD266B 01950        CALL  INCDE       ;Increment curs loc.
6B24 18F1   01960        JR    LOOP2       ;Loop til finished.
6B26 13     01970 INCDE  INC   DE          ;Increment Curs Loc rtn.
6B27 7A     01980 NORM   LD    A,D         ;Keep DE on Screen.
6B28 E63F   01990        AND   3FH
6B2A F63C   02000        OR    3CH
6B2C 57     02010        LD    D,A
6B2D C9     02020        RET
6B2E 79     02030 MATCH  LD    A,C         ;Get # INP chars left.
6B2F B7     02040        OR    A           ;Is it zero?
6B30 2004   02050        JR    NZ,MATCH2   ;-If No, JP.
6B32 3E20   02060        LD    A,' '       ;-Else A=space.
6B34 1801   02070        JR    MATCH3
6B36 78     02080 MATCH2 LD    A,B         ;Get Background
6B37 ED53846C 02090 MATCH3 LD  (LOC),DE    ;Store location.
6B3B 12     02100        LD    (DE),A      ;Display cursor.
6B3C CD026C 02110        CALL  KEYIN       ;Get char from keyboard.
6B3F FE08   02120        CP    8           ;Is it a Backspace?
6B41 201F   02130        JR    NZ,NOTBK    ;-If No, JP.
6B43 3AFF6B 02140        LD    A,(LEN)
6B46 B9     02150        CP    C           ;CP to chars left.
6B47 28E5   02160        JR    Z,MATCH     ;-If equal, JP
6B49 0C     02170        INC   C           ;Increment chars left.
6B4A 3E01   02180        LD    A,1         ;A=1.
6B4C B9     02190        CP    C           ;Is there 1 char left?
6B4D 2804   02200        JR    Z,ONE       ;-If Yes, JP.
6B4F 78     02210        LD    A,B         ;-Else, get Background.
6B50 12     02220        LD    (DE),A      ;Display it.
6B51 1803   02230        JR    LOOPM       ;Loop.
6B53 3E20   02240 ONE    LD    A,' '       ;A=space.
6B55 12     02250        LD    (DE),A      ;Display it.
6B56 1B     02260 LOOPM  DEC   DE          ;Backspace rtn.
6B57 CD276B 02270        CALL  NORM        ;Keep DE on Screen.
6B5A 2B     02280        DEC   HL          ;Decrement Ptr to $.
6B5B 7E     02290        LD    A,(HL)      ;Get char.
6B5C FE23   02300        CP    '#'         ;Is it a #?
6B5E 20F6   02310        JR    NZ,LOOPM    ;-If No, Loop.
6B60 18CC   02320        JR    MATCH       ;-Else, JP.
6B62 FE0A   02330 NOTBK  CP    10          ;Is it a dwn-arrw?
6B64 2804   02340        JR    Z,TEN       ;-If Yes, JP.
6B66 FE5B   02350        CP    91          ;Is it an up-arrw?
6B68 2019   02360        JR    NZ,NOT91    ;-If No, JP.
6B6A 08     02370 TEN    EX    AF,AF'      ;Store A in A',
6B6B 3A836C 02380        LD    A,(FLAG)    ;Get FLAG byte.
6B6E CB57   02390        BIT   2,A         ;Test bit 2.
6B70 200D   02400        JR    NZ,BADCHR   ;-If bit2=1, char is Bad.
6B72 08     02410        EX    AF,AF'      ;-Else get value.
6B73 1EC6   02420        LD    E,0C6H      ;E=Error 198.
6B75 FE5B   02430        CP    91          ;Is it up-arrw?
6B77 CAA219 02440        JP    Z,ERROR     ;-If Yes, JP.
6B7A 1EC8   02450        LD    E,0C8H      ;-Else, E= Error 101.
6B7C C3A219 02460        JP    ERROR
6B7F 78     02470 BADCHR LD    A,B         ;Get background char.
6B80 12     02480        LD    (DE),A      ;Display it.
6B81 18AB   02490        JR    MATCH       ;Continue.
6B83 FE0D   02500 NOT91  CP    13          ;Is it ENTER?
6B85 2013   02510        JR    NZ,NOT13    ;If No, JP.
6B87 3A836C 02520        LD    A,(FLAG)    ;Get FLAG byte.
6B8A CB5F   02530        BIT   3,A         ;Test bit 3.
6B8C 281B   02540        JR    Z,EXIT      ;-If bit 3=0, JP.
6B8E 3AFF6B 02550        LD    A,(LEN)     ;Check length.
6B91 B9     02560        CP    C           ;-Should be All...
6B92 2815   02570        JR    Z,EXIT      ;--If Yes, JP.
6B94 AF     02580        XOR   A           ;...or None.
6B95 B1     02590        OR    C
6B96 20E7   02600        JR    NZ,BADCHR   ;--If NOT None, JP.
6B98 180F   02610        JR    EXIT        ;--Else exit.
6B9A 32FD6B 02620 NOT13  LD    (TEMP),A    ;Store char.
6B9D AF     02630        XOR   A           ;A=0.
6B9E B1     02640        OR    C           ;Does C=0?
6B9F 28DE   02650        JR    Z,BADCHR    ;-If Yes, JP.
6BA1 3AFD6B 02660        LD    A,(TEMP)    ;Get char.
6BA4 12     02670        LD    (DE),A      ;Display it.
6BA5 0D     02680        DEC   C           ;Decrement Counter.
```

```
6BA6 C32B6B  02690          JP    LOOP3
6BA9 2A846C  02700 EXIT     LD    HL,(STRING)   ;Get Ptr to $.
6BAC 3E20    02710          LD    A,' '         ;A=space.
6BAE 12      02720          LD    (DE),A        ;Display space.
6BAF ED5B006C 02730         LD    DE,(SCREEN)   ;Get Starting Cursor Pos.
6BB3 3AFF6B  02740          LD    A,(LEN)       ;Get # of INP chars.
6BB6 4F      02750          LD    C,A           ;Store # in C.
6BB7 AF      02760          XOR   A             ;A=0.
6BB8 329940  02770          LD    (INKEY),A     ;Erase INKEY$.
6BBB FD2AA740 02780         LD    IY,(KEYBUF)   ;IY--> Keyboard Buffer.
6BBF 7E      02790 LOOP4    LD    A,(HL)        ;Get char.
6BC0 FE23    02800          CP    '#'           ;Is it a #?
6BC2 2806    02810          JR    Z,MATCH1      ;-If Yes, JP.
6BC4 23      02820 LOOP5    INC   HL            ;Get Next char.
6BC5 CD266B  02830          CALL  INCDE         ;Increment video ptr...
6BC8 18F5    02840          JR    LOOP4         ;...and loop.
6BCA 1A      02850 MATCH1   LD    A,(DE)        ;Get INP char.
6BCB B8      02860          CP    B             ;Is it a background char?
6BCC 2003    02870          JR    NZ,CONT5      ;-If No, JP.
6BCE 3E20    02880          LD    A,32          ;-Else A=space.
6BD0 12      02890          LD    (DE),A        ;Display the space.
6BD1 FD7700  02900 CONT5    LD    (IY),A  ;Store INP char in INP buffer.
6BD4 FD23    02910          INC   IY            ;Increment Buffer Ptr.
6BD6 0D      02920          DEC   C             ;Decrement Counter.
6BD7 20EB    02930          JR    NZ,LOOP5      ;Loop til Counter=0.
6BD9 21B540  02940          LD    HL,LSPADR     ;Restore temp $ Stor Ptr.
6BDC 22B340  02950          LD    (LSPNXT),HL
6BDF FD360000 02960         LD    (IY),0        ;Mark End of Input.
6BE3 2A886C  02970          LD    HL,(PTR)      ;Get Ptr to Pos in pgm.
6BE6 F1      02980          POP   AF            ;Clear stack.
6BE7 CD0D26  02990          CALL  VARPTR        ;Find VARPTR.
6BEA CDF40A  03000          CALL  TEST          ;If not $, then FC Error.
6BED E5      03010          PUSH  HL            ;Store Ptrs.
6BEE D5      03020          PUSH  DE
6BEF 2AA740  03030          LD    HL,(KEYBUF)   ;Get Buffer loc.
6BF2 2B      03040          DEC   HL            ;Point to Buffer-1.
6BF3 0600    03050          LD    B,0           ;B=0.
6BF5 CD6828  03060          CALL  INPUTF        ;Use ROM INPUT rtn now.
6BF8 E1      03070          POP   HL
6BF9 AF      03080          XOR   A
6BFA C3331F  03090          JP    HOLSTR        ;JP to ROM $ "handler".
             03100 ;
0002         03110 TEMP     DEFS  2
0001         03120 LEN      DEFS  1
0002         03130 SCREEN   DEFS  2
             03140 ;
6C02 D5      03150 KEYIN    PUSH  DE            ;Get char from keybd rtn.
6C03 C5      03160          PUSH  BC
6C04 E5      03170          PUSH  HL
6C05 2A866C  03180 LOOPK    LD    HL,(LOC)      ;Get video location.
6C08 7E      03190          LD    A,(HL)        ;Get current char.
6C09 32FD6B  03200          LD    (TEMP),A      ;Store current char.
6C0C 3A836C  03210 NOFLSH   LD    A,(FLAG)      ;Get Flag.
6C0F E6C0    03220          AND   0C0H          ;Mask all bits but 6 & 7.
6C11 47      03230          LD    B,A           ;Store mask in B.
6C12 365F    03240          LD    (HL),CURCHR   ;Display Cursor char.
6C14 CD386C  03250 LOOPK1   CALL  SCAN          ;Call Scan subrtn.
6C17 B7      03260          OR    A             ;Is it a zero?
6C18 201A    03270          JR    NZ,KEYRET     ;-If No, JP.
6C1A 10F8    03280          DJNZ  LOOPK1        ;-Else, Loop til B=0,
6C1C 3A836C  03290          LD    A,(FLAG)      ;Get Flag.
6C1F CB67    03300          BIT   4,A           ;Test bit 4.
6C21 20E9    03310          JR    NZ,NOFLSH     ;-If bit 4=1, JP.
6C23 E6C0    03320          AND   0C0H          ;-Else Get Blink Rate.
6C25 47      03330          LD    B,A           ;B=Blink delay.
6C26 3AFD6B  03340          LD    A,(TEMP)      ;Get char.
6C29 77      03350          LD    (HL),A        ;Display it.
6C2A CD386C  03360 LOOPK2   CALL  SCAN          ;Call Scan subrtn.
6C2D B7      03370          OR    A             ;Is it zero?
6C2E 2004    03380          JR    NZ,KEYRET     ;-If No, JP.
6C30 10F8    03390          DJNZ  LOOPK2        ;-Else Loop until B=0.
6C32 18D1    03400          JR    LOOPK
6C34 E1      03410 KEYRET   POP   HL            ;Clear Stack.
6C35 C1      03420          POP   BC
6C36 D1      03430          POP   DE
6C37 C9      03440          RET
             03450 ;
6C38 3A836C  03460 SCAN     LD    A,(FLAG)      ;Get FLAG byte.
6C3B E620    03470          AND   20H           ;Mask all bit but bit 5.
6C3D 321940  03480          LD    (CAPLOK),A    ;Load CAPLOK with mask.
6C40 CD2B00  03490          CALL  KBCHAR        ;Check KB for char.
6C43 B7      03500          OR    A             ;Is it zero?
6C44 C8      03510          RET   Z             ;-If Yes, RET.
6C45 FE08    03520          CP    8             ;Is it Backspace?
6C47 C8      03530          RET   Z             ;-If Yes, RET.
6C48 FE0D    03540          CP    13            ;Is it ENTER?
6C4A C8      03550          RET   Z             ;-If Yes, RET.
6C4B FE5B    03560          CP    91            ;Is it Up-Arrw?
6C4D C8      03570          RET   Z             ;-If Yes, RET.
6C4E FE0A    03580          CP    10            ;Is it Dwn-Arrw?
6C50 C8      03590          RET   Z             ;-If Yes, RET.
6C51 FE20    03600          CP    ' '           ;Is it Space?
6C53 382C    03610          JR    C,BAD         ;-If <Space, JP.
6C55 C5      03620          PUSH  BC            ;Store BC.
6C56 47      03630          LD    B,A           ;Store char in B.
6C57 3A836C  03640          LD    A,(FLAG)      ;Get FLAG byte.
6C5A CB47    03650          BIT   0,A           ;Test bit 0.
6C5C 78      03660          LD    A,B           ;A=char.
6C5D C1      03670          POP   BC            ;Restore BC.
6C5E C8      03680          RET   Z       ;If FLAG byte bit0=0, RET (A=KB).
6C5F C5      03690          PUSH  BC
6C60 47      03700          LD    B,A
6C61 3A836C  03710          LD    A,(FLAG)      ;Get FLAG byte.
6C64 CB4F    03720          BIT   1,A           ;Test bit 1.
6C66 78      03730          LD    A,B
6C67 C1      03740          POP   BC
6C68 280E    03750          JR    Z,SCAN1 ;-If FLAG byte bit1=0, JP (A=KB).
6C6A FE2B    03760          CP    '+'           ;-Else test for '+'.
6C6C 3813    03770          JR    C,BAD         ;--If <+, JP.
6C6E FE2C    03780          CP    ','           ;Is it comma?
6C70 280F    03790          JR    Z,BAD         ;-If Yes, JP.
6C72 FE2F    03800          CP    '/'           ;Is it virgule?
6C74 280B    03810          JR    Z,BAD         ;-If Yes, JP.
6C76 1804    03820          JR    SCAN2
6C78 FE30    03830 SCAN1    CP    '0'           ;Else test for numeral.
6C7A 3805    03840          JR    C,BAD         ;If <0, JP.
6C7C FE3A    03850 SCAN2    CP    '9'+1
6C7E 3001    03860          JR    NC,BAD        ;If >9, JP.
6C80 C9      03870          RET
             03880 ;
6C81 AF      03890 BAD      XOR   A             ;A=0.
6C82 C9      03900          RET
             03910 ;
6C83 00      03920 FLAG     DEFB  0       ;Default=Blnk slwst,uplc,any char.
6C84 0000    03930 STRING   DEFW  0
6C86 0000    03940 LOC      DEFW  0
6C88 0000    03950 PTR      DEFW  0
6C8A 00      03960          DEFB  0       ;Byte before BASIC Begin must=0!
             03970 ;
             03980 ;NOTE: Code from here used ONLY to initialize TATUPKI.
             03990 ;DISK BASIC PST will begin here.
6C8B         04000 SETUP    EQU   $       ;1st, relocate orig vectors.
6C8B 2AD741  04010          LD    HL,(VINPUT+1) ;Get orig INPUT Vector.
6C8E ED5B9541 04020         LD    DE,(VAND+1)   ;Get orig '&' Vector.
6C92 3AD641  04030          LD    A,(VINPUT)    ;Get orig INPUT V opcode.
6C95 47      04040          LD    B,A           ;Store it in B.
6C96 3A9441  04050          LD    A,(VAND)      ;Get orig '&' V opcode.
6C99 324D6A  04060          LD    (ANDPTX),A    ;Put it @ ANDPTX.
6C9C 78      04070          LD    A,B           ;Restore orig INPUT v op.
6C9D 32686A  04080          LD    (INPPTX),A    ;Put it @ INPPTX.
6CA0 ED534E6A 04090         LD    (ANDPTX+1),DE ;Put orig INPUT Vector.
6CA4 2Z696A  04100          LD    (INPPTX+1),HL ;Put orig '&' Vector.
             04110 ;2nd, replace orig vectors with new data.
6CA7 3EC3    04120          LD    A,0C3H        ;0C3=JP.
6CA9 329441  04130          LD    (VAND),A
6CAC 32D641  04140          LD    (VINPUT),A
6CAF 21646A  04150          LD    HL,NUVIN      ;Patch into vectors.
6CB2 22D741  04160          LD    (VINPUT+1),HL
6CB5 21466A  04170          LD    HL,NUVAN
6CB8 2Z9541  04180          LD    (VAND+1),HL
             04190 ;3rd, Fix up BASIC pointers and jump to BASIC.
6CBB 218B6C  04200          LD    HL,SETUP      ;Point to SETUP.
6CBE 22A440  04210          LD    (BASBGN),HL   ;Store it @ BASBGN.
6CC1 CD4D1B  04220          CALL  BASNEW        ;Perform a BASIC "NEW".
6CC4 011181A 04230          LD    BC,1A18H      ;Required for JP BASIC.
6CC7 C3AE19  04240          JP    BASIC
6CBB         04250          END   SETUP
00000 TOTAL ERRORS

ANDPTX  6A4D   BAD     6C81   BADCHR  6B7F   BASBGN  40A4   BASIC   19AE
BASNEW  1B4D   BCKCHR  00EE   CAPLOK  4019   CONT1   6B07   CONT5   6BD1
CONTZ   6AF5   CURCHR  005F   CURPOS  40A6   CURSAD  4020   EOL     6AB0
```

```
ERROR   19A2   EVAL    2337   EXIT    6BA9   FCERR   1E4A   FLAG    6C83
GETINT  2B01   HDLSTR  1F33   INCDE   6B26   INKEY   4099   INPPTX  6A68
INPUTF  2B68   KBCHAR  002B   KEYBUF  40A7   KEYIN   6C02   KEYRET  6C34
LEN     6BFF   LOC     6C86   LOOP    6AD3   LOOP1   6AEF   LOOP2   6B17
LOOP3   6B20   LOOP4   6B8F   LOOP5   6BC4   LOOPK   6C05   LOOPK1  6C14
LOOPK2  6C2A   LOOPN   6B56   LOOPU   6A99   LOOPU2  6AA4   LSPADR  4065
LSPNXT  4083   MATCH   6B2E   MATCH1  6BCA   MATCH2  6B36   MATCH3  6B37
NEXT    6AD9   NOFLSH  6C0C   NORM    6B27   NOT13   6B9A   NOT91   6B83
NOTBK   6B62   NUVAN   6AA6   NUVAN1  6A50   NUVIN   6A64   NUVIN1  6A6B
ONE     6B53   PTR     6C88   SCAN    6C38   SCAN1   6C78   SCAN2   6C7C
SCREEN  6C00   SETUP   6C8B   SNERR   1997   STRING  6C84   TEMP    6BFD
TEN     6B6A   TEST    0AF4   THERR   0AF6   USING   6A93   VAND    4194
VARPTR  260D   VINPUT  41D6   VTF     40AF   WRA1    4121
```

BASIC demo program to show how TATUPNI/CMD works

```
0 'TATUPNI     VK2JK / Box 300 / Spit Jct NSW 2088 / AUSTA -v1.0-841228
1 'This is a demo pgm to show how TATUPNI performs a dynamic
2 'patch on a NEWDOS TRS80 Mod3.  Patch adds 2 cmds to Disk
3 'BASIC. They are:
4 '     1. INPUT@ printat,flag,USING string; variable.
5 '     2. &POS   function which returns current cursor POS.
6 'For more details, see "80 Micro" story "EASY INPUT" Nov 84.
7 '
8 'NOTE: This v of TATUPNI (INPUTAT, backwards!) has several
9 'enhancements over v in article.  They are:
10 '    a. TATUPNI stored at beginning of BASIC PST.
11 '    b. Beginning of BASIC PST then adjusted to accomodate.
12 '    c. Thus, no MEMSIZE reservation required.
13 '    d. One v TATUPNI fits all RAM sizes (16k, 32k, 48k).
14 '    e. Source has 2 ORGs (a= TRSDOS (1.3), b= NEWDOS (2.0).
15 '    f. After invocation TATUPNI, CMD"S" is unpredictable.
16 '    g. Arrangement of bits in FLAG improved:
17 '            Bit Wt. Legend
18 '             0   1  Permit numerals only?
19 '             1   2  If numerals only, permit (.,-,+)?
20 '             2   4  No return on Up-Arrw or Dwn-Arrw?
21 '             3   8  Max OR no chars only?
22 '             4  16  No flashing cursor?
23 '             5  32  Permit capitals only?
24 '             6  64  Cursor flash speed (cf article).
25 '             7 128   "     "    "    "    "
100 CLEAR1000: CLS
110 PRINT"BEFORE RUNning this pgm, be sure you have installed"
120 PRINT"the TATUPNI rtn.  If not already done, you can now"
130 PRINT"issue the BASIC cmd:"
140 PRINT,"CMD";CHR$(34);"TATUPNI";CHR$(34): PRINT
150 PRINT"Then re-LOAD and RUN TATUPNI/BAS. Else <CONT>"
160 STOP
170 ON ERROR GOTO380
180 CLS: PRINT"This is a demonstration of TATUPNI, a formatted input rtn."
190 PRINT"When 'PRINT Using?' appears, enter a format string."
200 PRINT"Try 'PHONE (###) ###-####'."
210 PRINT"    When 'FLAG?' appears enter a Flag Value. Try '9'."
220 PRINT"(numerals only, maximum or no chars only)."
230 PRINT"    When 'PRINT@?' appears enter a print@ location."
240 PRINT"Try '860'."
250 PRINT"After entering the print@ location, the input statement is"
260 PRINT"executed.  Experiment with the backspace, up and down arrows,"
270 PRINT"and the <ENTER> key.": PRINT
280 PA=860: A$="####": F=0
290 PRINT@704,;: INPUT"PRINT Using";A$
300 INPUT"Flag";F
310 INPUT"PRINT@ ";PA
320 '
330 '--------------------
340 INPUT@PA,F,USINGA$;B$
350 '--------------------
360 '
370 T=0: PRINT: PRINT"b$=";B$: GOTO290
380 E=ERR/2+1: IF E=100 THEN PRINT"START": RESUME290 ELSE IF E=101 THEN
PRINT"*END*  ": RESUME290 ELSE ON ERROR GOTO0
```

INTERFACING BASIC AND TRSDOS 6 SVC ROUTINES

by Mike Orr

In Volume 5 Number 8 one of your readers has asked for some assistance in interfacing Basic and the TRSDOS 6.? (pick a version) SVC routines. P.G. Raeth was specifically interested in accessing SVC 15,1 which is the "GET" character from screen routine. Printed below are two sample programs that I hope will answer the questions that he had.

The assembly language listing is used to create an object code file called VARPAS/OBJ. The BASIC program (called VARPAS/BAS) will load the VARPAS/OBJ when it executes. The only requirement is that memory size be set to FF00H to protect the assembler routine.

I have not included any detailed instructions as the programs are extremely simple. I might mention the one problem that P.G. Raeth may have been encountering. His method of setting up for the basic CALL seems to be causing the problem. For example:

He had been trying to access SVC 15,1 using CALL M%(RP%,CP%,BP%) where:

M% = address of object code
RP% = pointer to R% which holds screen row to read
CP% = pointer to C% which holds screen column to read
BP% = pointer to B% which should hold the byte located at R%,C% on the screen on return from the assembler routine.

The TRSDOS manual does not explain very clearly, but on execution of a CALL from BASIC, the HL, DE, and BC registers ALREADY point to desired bytes. In other words, by using RP%, CP%, and BP%, P.G. Raeth has just introduced a POINTER TO A POINTER! If I understand him correctly he wants to pass a row and column coordinate to an assembler routine and return with the character from the screen. This can be accomplished quite simply with a CALL M%(R%,C%,B%) !

On entry to the assembler routine HL is pointing at the row (R%) variable, DE is pointing at the column (C%) variable and BC is pointing to the DUMMY (B%) variable which will contain the character on return from the routine. Hope this explains it a little better than TRICKYDOS has done in the manual.

-Mike Orr
449 Hamilton Avenue, Nanaimo, British Columbia V9R 4E7, CANADA

```
                00001 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                00002 ;
                00003 ;     Passing variables via the CALL instruction
                00004 ;     of TRSDOS 6.2 Basic. For demonstration
                00005 ;     purposes only 3 variables are passed.
                00006 ;     Written by Mike Orr, 449 Hamilton Ave.
                00007 ;     Nanaimo B.C. Canada, V9R 4E7.
                00008 ;
                00009 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                00010 ;
                00011 ;
                00012 ; Entry: HL===>Parameter 1  (Pointer to ROWSZ Screen row)
                00013 ;        DE===>Parameter 2  (Pointer to COLSZ Screen col)
                00014 ;        BC===>Parameter 3  (Pointer to BYTEZ Dummy var.)
                00015 ;
                00016 ; Exit:  BYTEZ holds byte at Row/Column
                00017 ; No error checking for screen limits
FF01            00018          ORG     0FF01H
FF01 7E         00019 START    LD      A,(HL)          ;get LSB row pos'n
FF02 67         00020          LD      H,A             ;put here for SVC setup
FF03 1A         00021          LD      A,(DE)          ;get LSB col pos'n
FF04 6F         00022          LD      L,A             ;put here for SVC setup
FF05 C5         00023          PUSH    BC              ;save pointer to BYTEZ
FF06 3E0F       00024          LD      A,0FH           ;@VDCTL svc
FF08 0601       00025          LD      B,1             ;"get" char function
FF0A EF         00026          RST     28H
FF0B C1         00027          POP     BC              ;pointer to BYTEZ
FF0C 02         00028          LD      (BC),A          ;char to BC
FF0D C9         00029          RET                     ;back with char in BYTE$
FF01            00030          END     START
00000 TOTAL ERRORS

START   FF01
```

```
1 REM TRSDOS 6.x SVC/Basic interface demonstration
2 REM by Mike Orr (02/18/85)
5 SYSTEM"LOAD VARPAS/CMD"
10 CLS:FOR X=1 TO 79:PRINT CHR$(RND(26)+96);:NEXT
20 ROWS%=0:COLS%=0:BYTE%=0:REM BYTE% will hold char on
return from CALL
25 M%=&HFF01:REM Address of Assembler routine
26 FOR COLS%=0 TO 79
70 CALL M%(ROWS%,COLS%,BYTE%)
75 FOR X=1 TO 500:NEXT:REM Delay loop
80 PRINT @1200,"The character at Row ";ROWS%;"and Column
";COLS%;"is ";CHR$(34);CHR$(BYTE%);CHR$(34);
85 NEXT COLS%
90 PRINT"The Row and Column counts each start at ZERO!"
100 PRINT:PRINT
```

SOFTWARE!

Perhaps you have heard the term "expert program". An "expert program" is a program that generally contains all information necessary for competence in a given field. The Alternate Source is now offering the first in a new series of expert programs: The Algebra Program.

The Algebra Program can be used to help you with homework, whether you are a student or an engineer. Currently an MSDOS computer with 256k is required for operation.

The Algebra Program does for algebra what the hand calculator did for arithmetic. On your computer. The Algebra Program will completely automate the solving of algebraic and, with additional modules, trig and calculus problems! Simply key in the equation and out comes the solution, instantly, along with all intermediate steps. Enter your algebra expressions just as you see them in the book, or use computer notation.

An example expression might look like "xyz" or "x*y*z" or "x * y * z". All these and other forms are acceptable. If you type in a problem like "10a+12a", The Algebra Program returns the proper answer, "22a". Perhaps you would like to try a more complicated problem, like "x**2+4x+2x**2+9x". Almost instantly, The Algebra Program returns the proper answer, "3x**2+13x". The Algebra Program is different from hundreds of other math programs now available because the answers contain all algebraic terms. Other math programs return only numeric answers. The only exception is MuMath, but unless you know the Lisp programming language, you will find minimal applications for that package.

The Algebra Program is very user-friendly and forgiving about syntax. With very few documented exceptions, problems can be entered just as they appear in the textbook. With your MSDOS keyboard "PRINT-SCREEN" key, you can optionally route the step-by-step detailed algebraic solutions to the printer. The Algebra Program contains "core" information and is a requirement for all additional modules in this series (most additional modules will be under $50). Coming up: The Trig Program and The Deriative Program. Versions will soon be available in French, German and Spanish, too.

The Algebra Program is $99.95 and includes simple documentation. The program was designed so that beginning algebra students can start using the program within minutes. Can you think of a cheaper way to provide your kid with an "expert" Algebra tutor for an unlimited number of hours?

****

Programmers. BAS34 will convert your Model I/III BASIC programs to a more generic form of BASIC which may be used with a wide variety of other operating systems. There are now five versions of this package available: TRSDOS 6.x, Model I/III, CP/M, MSDOS (general) and MSDOS (2000). Please specify which version you desire when you place your order. Certain modifications are specifically to translate code to a format used with systems other than TRSDOS 6.x.

We believe Dennis Allen's BAS34 utilities to be the most powerful translation packages on the market at this time. If you are a registered owner and would like to receive an update to the new version of BAS34, the price is $10 for both a new disk and the new manual. The new price of the package is $49.95. Be sure to specify which operating system you wish the program to run under.

****

UNIKEY is a machine language program for use on the Radio Shack Model 4 using TRSDOS 6.x. It is used in conjunction with BASIC to permit single key entry of 85 key words and phrases. In addition, UNIKEY offers three "programmable" key combinations, each of which can be assigned by the user. When installing UNIKEY, the user has the option of including a HELP screen that quickly shows the key combinations to produce any of the UNIKEY substitutions. A /JCL file is included on the master disk for easy installation of UNIKEY. UNIKEY, the Keystroke Saver, is only $19.95, complete with a simple 10-page manual.

****

MACRO TYPING is a typing practice program featuring high speed video action and sound. The term "macro" simply means "large". With MACRO TYPING you may practice typing with characters that are many times normal size. This is very valuable if someone in your family is visually impared.

MACRO TYPING is excellent for all skill levels. Difficulty is determined by the practice text. Practice text from simple upper case alphabet to "real" text containing upper and lower case letters and punctuation is included on the program disk. Text generating utilities are also included. The program has been tested at speeds over 100 words per minute, so you're not likely to outgrow it. A frequently updated "speedometer" tells how many words per minute you are typing. Versions are included for Models I/III and 4 in the native modes. MACRO TYPING is only $29.95 and includes documentation, the TYPE/CMD program, lots of sample practice text and two sample programs for generating more sample text.

Order from:
The Alternate Source Information Outlet
704 North Pennsylvania Avenue
Lansing, MI 48906
(517) 482-8270

GOING ONLINE?

Want a FREE terminal package? Not just ANY terminal package. One of the best available for the TRS-80 Model 4! READ ON!

LTERM started out as a small dumb terminal program intended for personal use with Bulletin Board Systems (BBS). It was soon realized that more advanced features were required to communicate with the outside world. Several versions of LTERM were developed for the TRS-80 Models I and III and have found their way into public BBS systems all over the country. LTERM IV marks the latest version of this popular terminal program with several new features not found on other more expensive programs or in earlier versions of LTERM.

LTERM takes special advantage of many of the Model 4 advanced features, such as the function keys, reverse video, the extra bank of memory (if you have it installed) and more. LTERM supports AUTOLOGON, TRUE BREAK (required by some computers) and custom CONVERSION TABLES (INCOMING, OUTGOING and a PRINTER table) that may optionally be saved on disk. Display and EDIT the transmit/receive buffer in both ASCII and HEXADECIMAL (HEX) modes, automatic FILE OPTIONS, CR/LF GROUPING, custom VIDEO FILTERs, ECHO/FEEDBACK TOGGLE, LOGON STRING OUTPUT (for BBS systems that don't support auto logon) and MACRO-STRINGS are also supported. The main BUFFER may be transmitted using either "prompted" line output or using standard XON/XOFF protocol. PRINTER OPTIONS are available to route text to your line printer. All RS-232 options are within easy control of the user. Type directly into the buffer for "quick and dirty" messages. The main buffer is 65534 bytes in a 128k machine and over 33,000 bytes in a 64k machine. Files may be transmitted using the DIRECT FILE TRANSFER "MODEM" protocol. DOS COMMANDS may be executed without leaving the LTERM program and, of course, there's even more! Online HELP is available simply by pressing [F3].

So, how do you get this nifty package without paying the $49.95 list price? Simple! Just order a 1200 Password modem from The Alternate Source and mention this ad! The Password is Hayes compatible, completely programmable, auto-answer and auto-dial and a fantastic bargain at only $299. Place your order TODAY and receive LTERM for your Model 4 absolutely FREE. All cables included! Phone (517) 482-8270 for fastest response, or mail order from The Alternate Source Information Outlet, 704 North Pennsylvania Avenue, Lansing, MI, 48906. Just to make sure you get the best deal around, we'll pay shipping on this and any products ordered with this offer. As always, a copy of Northern Bytes will be included absolutely FREE!

NORTHERN BYTES
c/o Jack Decker
1804 West 18th Street
Lot # 155
Sault Ste. Marie, Michigan 49783
MCI Mail Address: 102-7413
Telex: 6501027413
(Answerback: 6501027413 MCI)

POSTMASTER: If undeliverable return to:
The Alternate Source, 704 N. Pennsylvania, Lansing, MI 48906

To: