NORTHERN BYTES

Volume 6 Number 7

GREETINGS! Welcome to the "fall has fell" edition of NORTHERN BYTES. You may not like my corny opening lines, but after all, it isn't autumn without 'em. Ahem...

I really do have to apologize to our readers in Australia and New Zealand. Not for making bad jokes about fall while they're celebrating spring, but for the late delivery of NORTHERN BYTES Volume 5 Number 5. After that experience, I can understand why some U.S. firms are so hesitant to deal with customers outside of this country. I can also understand why electronic mail is becoming such a popular topic.

It all started out with the best of intentions - we wanted to give ALL of our subscribers in Australia and New Zealand speedy delivery, not just those that could afford to pay the extra dollar per issue for airmail postage. Tony Domigan had agreed to make copies of a master copy of NORTHERN BYTES, and to mail those copies to our southwest Pacific area readers. So we sent him our master copy of NORTHERN BYTES Volume 5 Number 5 by air mail - and, of course, the post office promptly managed to lose it (actually, all indications are that they pocketed our air mail postage, then put the package on a slow boat to China). The master copy finally turned up on Tony's doorstep about a month and a half later.

In the meantime, we discovered an overseas mailing service that seemed to have the answer to our problems. They operate as follows: You send them your entire overseas mailing, bundled according to countries, at a New York address. Your mailing is then airlifted to London, England (along with all the other mailings they have received that day) where it is deposited into the British postal system for airmail delivery to the various destination countries. The price per piece is about one-third of U.S. printed matter postage rates, presumably because the British post office charges so much less for international mailings of printed matter.

When we discovered that Tony had still not received the masters after about three weeks, we decided to send a direct mailing using this overseas mailing service. We figured it would make a good test mailing for that service. Suffice it to say that the results were something less than spectacular.

It appears that the British either didn't know or didn't care that the issues of NORTHERN BYTES were supposed to be sent by air mail. You guessed it -- we got more "slow boat" delivery! In addition, by this time a strike was in progress at one of the Australia Post offices that processes incoming international mail. The net result was that some readers got their copies more than two months late, and for this we apologize.

As I write this, we are planning to give this overseas mail service one more try, for Volume 6 Number 6 (the issue prior to the one you're reading). But this time we are going to mail each issue in an envelope that is preprinted with the legend "AIR MAIL/PAR AVION". I have also written the mailing service to advise that this is their last chance to get it right! If you are on our regular mailing list, live outside North America, and receive this issue with U.S postage affixed, you'll know that the experiment didn't work.

You have, no doubt, already noticed the "new look" of NORTHERN BYTES. This is due to the recent purchase of a Tandy DMP-2100P dot matrix printer for use with NORTHERN BYTES. Yes, I did say dot matrix! The secret is that the DMP-2100P uses 24 pins in the printhead rather than the usual 7 or 9. The printer itself is made by Toshiba, but the interesting thing about that is that I have heard stories to the effect that some of the Toshiba printers (sold under the Toshiba brand name) have some sort of bug in their ROMs that inhibit proper operation under certain conditions, whereas the Tandy units do not have this bug. I've even heard stories of Toshiba printer owners purchasing replacement DMP-2100(P) ROMs from Tandy and installing them in Toshiba printers, just to fix whatever this problem is. If you own the Allwrite word processing program from Prosoft, you'll notice that it supports the DMP-2100P but not the Toshiba printers, apparently because of this bug. If anyone can give me a more specific explanation of the difference between the units, I'd be happy to print it here. In the meantime, Tandy has lowered the price of the DMP-2100P to the point that there really isn't much of a saving in buying the comparable Toshiba printer, especially when you consider that Tandy will service printers you buy from them.

Incidentally, two other features of the DMP-2100P that don't receive much attention are that it supports downloadable fonts and has an "IBM Emulation Mode". Now, think hard, an "IBM" printer is really which popular printer is disguise? In other words, bunky, if you have programs written to work on an "Epson-compatible" printer, there's a good chance they will also run on the DMP-2100P, if you enter the "IBM Emulation Mode" and maybe do a little bit-fiddling. The manual doesn't indicate that the TRS-80 block graphics are available in memory, but a clever programmer could download these to the printer and use them as a "downloadable font". Speaking of downloadable fonts, Prosoft has four disks full of these (see their ad in 80-Micro for details). These fonts can be used by any program, but Allwrite can download them to the printer, intermix them with normal text and/or other downloadable fonts, and center or right-justify them properly.

I might mention here that if you have coded a downloadable font set for the DMP-2100P or for any other printer that supports downloadable fonts, you might consider contributing it to the TAS Public Domain Library so that others can also make use of it. If it's for any printer other than the DMP-2100P, please send a sample printout so that we can see what it looks like. If you have written a program that helps design such fonts, we'd be very interested in having it for NORTHERN BYTES and/or the PD Library.

Some portions of Northern Bytes may still appear in our "old" typeface for a while, because we have some articles that have already been printed but were squeezed out of previous issues due to lack of space. We could take the time to reprint them, but there aren't enough hours in the day as it is. Besides, mixed typefaces may help add a bit of variety to this publication.

We are making a slight change in the way we handle electronic mail. Starting now, we are checking our electronic mailboxes on a weekly basis, usually at some time during the weekend or on Monday morning. We currently use the following services: MCI Mail (102-7413), Compuserve (72167,161), and Delphi (TASIO). The numbers in parenthesis are our I.D.'s for each service. Please note that due to the low volume of mail that comes to us via MCI Mail, we may drop that service when our renewal time comes up in January (we haven't decided for sure yet, just wanted to let you know that we're thinking about it).

I want to conclude with a warm welcome to our new readers, especially those that first learned about NORTHERN BYTES through our electronic "mass mailing" on Compuserve. If you want to see this support for your TRS-80 Models I/III/4/4D 4P continue, please encourage your friends to sign up. This is probably the only publication that you can be sure won't take your money and then fold, since we bill you for each issue as it is published - not before! If you've ever lost money on a magazine subscription before, at least you know that it can't happen here.

Just in case we don't get another issue out before the end of the year, I hope that all of you have a very happy holiday season, and extend our best wishes to you for the coming year!

LETTERS DEPARTMENT

Reminder: Persons sending letters intended for publication should send them on magnetic media or via Compuserve, Delphi, or 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!

This month our letters column has a slightly different format. Although we are still receiving a huge volume of mail, we have not received any letters specifically intended for publication recently. Seems like it's feast or famine around here (remember the large number of letters we published last issue?).

We recently did a promotional "mailing" via Compuserve, offering a free sample issue of NORTHERN BYTES to those who had never seen a copy. Many of those who responded to this offer took the time to include a few comments, and we'd like to share a few of those with you. Since the folks that sent these comments may not have intended to have their names in print, only their city and state will be used as the "signature". We hope you find these comments enlightening and thought-provoking.

Jack,

Your Northern Bytes Newsletter sounds good. I have a Model I and am very unhappy with the lack of coverage that another magazine (80-MICRO) is not giving to this fine machine. Please send a copy.

Hooks, Texas

To Jack Decker:

I am a TRS-80 MODEL I user and I hope your newsletter will have some information pertaining to my system (so many of the publications promise that they do but, in reality they only mention compatibility as a courtesy and to get subscribers.)

After reading your offer about the free copy of your newsletter on COMPUSERVE, I have decided to take you up on the offer and sample your newsletter. Believe me when I say that I hope to find your publication to be informative for us long forgotten MODEL I users.

Brentwood, New York

Dear Jack,

One of the things that I most dislike about Tandy is the way that they abandon the owners of their older computers. I own a Model I that I have expanded as much as I can. I think it is still a good computer and I have no intention of "trading up". I appreciate any opportunity to learn more about my computer so please send your copy for me to examine. I am looking forward to reading it.

Marrero, Louisana

Dear Jack,

O.K., I'll accept the offer for a free copy of NORTHERN BYTES. If it helps you, I have several Model Is with Omikron CP/M modifications. They are falling into disuse due to the availability of better equipment (i.e. faster, more reliable, more disk space, and on and on and on). I would be especially interested in information on overcoming the many shortcomings of this old equipment, or uses that would not be so hindered by them (e.g. use in home control, intelligent print buffers, etc.). If you try to cover other elements of the Radio Shack product line, I am currently using several TRS Model 100s.

New Orleans, Louisana

[Two comments that we saw frequently were complaints about the lack of support for the Model I, and complaints about 80-Micro. Well, I'm sure that 80-Micro has the same problem that we do, which is that you can't please everyone. No matter what direction they (or we) take, there are going to be some folks (hopefully not too many!) that aren't happy with it. We aim NORTHERN BYTES toward the more experienced TRS-80 user, so we sometimes tend to get complaints from beginners who say that they can't understand half of what we print.

As for which models we will support, that depends solely on the types of articles that we receive. But I think we are in a slightly better position to provide Model I support than some other publications, for two reasons. One is that your editor owns and uses a Model I on an almost daily basis (I also have a Model 4P that sees somewhat less use). The other reason is that we frequently receive articles from places like Australia and New Zealand, where there are more Model I TRS-80s (and Model I clones, mostly the Dick Smith SYSTEM-80) in use than Model III or 4 machines. Finally, many programs will run on both the Models I and III with just a little tweaking, and I've been known to go in and slightly modify a program prior to publication to make it Model I/III/4(III mode) compatible (however, if you're writing an article for NORTHERN BYTES, please don't count on me doing this!).]

Jack:

I'd sure like to see your newsletter. Please send a sample copy, along with arrangements you might make for contributors, e.g. Compuserve transmissions, compensation (in trade?), the kind of stuff you want. I have a Model I TRS-80 and am fairly proficient in its use.

Mt. Vernon, Washington

..... Do you accept contributions to the newsletter by EMAIL? Looking forward to the sample issue.

Nevada City, California

[We accept contributions to NORTHERN BYTES via any method you can get them to us, except hardcopy. That means you can send them on disk, tape (500 baud only, please), or by direct MODEM upload at 300 or 1200 baud (call (906) 632-3248 to pre-arrange this), or via Compuserve, Delphi, or MCI Mail. Please note that we do not always check the electronic mailboxes on a regular basis, and usually not more than once a week in any event, so don't send time-critical material by that method.

Contributions is the key word in the above paragraph. NORTHERN BYTES is written by and for TRS-80 users, and we are usually fortunate if our income covers our expenses. If our income ever rises (which will only happen if we get more paying readers!) then we may start paying for articles. However, for the present, the only compensation that we provide for articles is a free "subscription", usually for the next six issues (assuming that we publish that many more issues!). If you require additional payment, you must go through The Alternate Source. But your article should be really spectacular if you expect to receive additional compensation for it. The other side of the coin is that I have some reluctance to use articles that have been sent to TAS rather than directly to me (I can't explain why, except that the direct author contact means a lot to me), so please don't send an article by way of TAS unless you think it merits additional payment (or you don't care if it never appears in print). Electronic mail is the exception to this rule, however, because it can be retrieved at either location.]

Jack,

I'd like to get a copy of Northern Bytes -- solid info on the III and 4/4P is tough to come by these days. I recently got a batch of 6 TAS public domain disks, and there saw Northern Bytes mentioned for the first time. Couldn't tell from that whether you were in business any longer, though. Anyway, I'll bite (byte?).

Winfield, Kansas

Dear Jack,

I would indeed enjoy receiving a free sample of Northern Bytes - I've heard several favorable comments about NB, but never found the publisher.

I look forward to receiving what I hope will be the first of many. Thanks.

Springfield, Pennsylvania

Dear Jack Decker,

I have never seen an issue of Northern Bytes and am eager to take you up on an offer of a free issue per your letter on CompuServe. I did subscribe to The Alternate Source some time back and received one issue of it before it folded. If Northern Bytes is half as good as The Alternate Source, I'm sure I'll subscribe.

Mendota, Illinois

YES YES YES!!! Please send a sample copy of Northern Bytes. I originally subscribed to TAS at the 80-Micro TRS-80 show that Wayne Green put on in New York City, about 5 (??) years ago. The ONLY magazine that I really regret losing was TAS. If NB is anything like it I want it.

Seattle, Washington

[NORTHERN BYTES is NOT a direct successor to The Alternate Source Programmer's Journal, though we're flattered by the comparison. Actually, this newsletter originally was a computer club publication (with the same editor) until the club folded. Back in those days, it covered all makes and models of computers (not just the TRS-80 Models I/III/4/4D/4P as we do now), and was a lot thinner and less attractive than it is today. That's why we

don't offer back issues prior to Volume 5 - in fact, we don't even have the master copies of most of those issues anymore.

And, yes, we're still in business, even if a lot of TRS-80 users have never heard of us. If we are to stay in business, though, we must build up our readership, so if you want us to continue, PLEASE be sure to mention us to your friends that own or use a TRS-80.]

Please forward a sample copy of Northern Bytes. My immediate need is to convert a BASIC program to machine language to speed it up. I score gymnastics meets and it's a great program, but slow. Any help? Thanks again for the sample. I'll be looking forward to it.

Ft. Wayne, Indiana

[We published an article featuring a simple BASIC compiler program in NORTHERN BYTES Volume 5, Number 9. Whether it would compile your program or not depends on how your program is written. If you'd like to try it, you can get the back issue for $2.00, or TAS Public Domain Library Disk #004 (which contains the compiler program and documentation file) for $10.00 plus $3.00 shipping/handling from The Alternate Source.

Before you go that route, however, you might try adding a hardware speed-up modification to your computer, and/or revising your BASIC program for greater speed. A program such as THE ANALYST (by Modular Software Associates) or FASTER (by Prosoft) can really help cut the execution time of your program. In addition, if the slowness of the program is due to "garbage collection" of string variables, try to reduce the use of strings, or buy and use a program such as THE COLLECTOR (by Modular Software Associates) or TRASHMAN (by Prosoft) to eliminate garbage collection delays.

Yet another option would be to purchase a commercial compiler program. This should be considered as a last resort, however, since good compilers tend to be expensive and usually require you to modify your BASIC program (sometimes substantially) before you can get a working machine language program out. And the compiled program is always larger than the original (a factor to consider if your program uses a lot of memory) and usually requires a "run-time" portion of the compiler to be memory-resident whenever the program is run. However, use of a compiler will usually (but not always) provide the greatest speed increase.]

Dear Jack:

I have purchased Montezuma Micro's CP/M 2.2 package for the Model 4, and T, Turbo Toolbox, and Turbo Tutor. So I want to know more of how to use these software programs, as well -- looking forward to hearing more of how to use all these things, and more, new stuff, suggestions, etc., in Northern Bytes. Thanks for your help.

Tucson, Arizona

[This is another case of "We'll print it if somebody sends it to us." However, you should be aware that CP/M is a world unto itself, and you may find that you have a lot more in common with other CP/M users (no matter what brand computer they may be using) than with the rest of the TRS-80 world. Which is a polite way of saying that we're sorry, but there are probably other publications around that will be more helpful to you than NORTHERN BYTES, at least in helping you learn and use CP/M. Perhaps one of our readers could recommend a really good CP/M oriented publication, or perhaps write some articles on the subject for us.]

Please send sample copy of the "Northern Bytes" newsletter. P.S. If UM puts a "Northern" Bite on OSU this year, forget getting a subscription from me.

Columbus, Ohio

[Quick, somebody, whip up a extremely realistic computerized football simulation where OSU always wins. Then all we need is an experienced hardware hacker to patch the output into this guy's cable TV line...]

EL CHEAPO EPROM ERASER

An article written by Mike L. Simon in an early issue of Kilobaud magazine (sorry, I don't recall which issue) describes how the author constructed an inexpensive EPROM eraser. Basically, he used a GE germicidal lamp number G4T4/1, a four-pin tube socket (Amphenol part number 77MIP4 or equivalent), and a GE ballast number 89G435 (a number 58G827-60 ballast will also work and will extend bulb life, but will increase the amount of time required to completely erase an EPROM from 15-20 minutes to 20-35 minutes). He also used a SPST pushbutton switch (or FS-5 starter) and mounted the whole works in an inverted bread pan using a homemade "L" bracket.

When wiring to the AC line, one side of the line is connected to one wire of the ballast. The other wire from the ballast is connected to pin 1 of the socket. Pins 2 and 3 of the socket are connected to the pushbutton switch or starter, and pin 4 of the socket is connected to the other side of the AC line. It is suggested that a grounded (3 prong) cord set be used and that the green ground wire be connected to the bread pan to avoid accidental shock. The bulb should be mounted so that its surface is 1⅛" from the surface that the EPROMS will rest on. The article gives additional construction details and the WARNING that you must NOT look at the ultraviolet lamp (when it is lit, of course) without eye protection. The original Kilobaud article suggested that for further information (and a source for the lamp, socket and ballast) you could contact Mike L. Simon of Chicago, Illinois, however, that article is probably about five years old now, and the Chicago telephone directory no longer lists a Mike L. Simon at the address given in the original article.

This article is adapted from an early (pre-1984) issue of NORTHERN BYTES. We have not built or tested one of these EPROM erasers, and if you do, you do so at YOUR OWN RISK. Use all of the normal precautions necessary when constructing electrical equipment, and if you've never done anything like this before, get some help from a qualified electrician or an experienced hardware hacker.

EPROM PROGRAMMER KIT

Robotron Industries, Inc. (7041 West Maple Terrace, Wauwatosa, Wisconsin 53213) is said to have a low-cost EPROM programmer kit for use with the TRS-80 Model III (it should work with the Model 4 as well). The EP8401 kit comes with a printed circuit board, all components, software (on tape), interface cable, and instructions. EPROMs are not included. It can be used to read or program 2716, 2732, 2732A and 2764 type EPROMs. The EP8401 kit price is $49.95, and the power supply kit (+5V regulated and 30V supplies are required) is $14.95. You can also get a set of plans for $5.00, and a bare printed circuit board for $14.95. Add $4.00 for shipping and handling, plus 5% sales tax if you live in Wisconsin.

The above information was extracted from an article in the Milwaukee Area TRS-80 User's Group newsletter. We at NORTHERN BYTES have no additional information about this company.

HELP WANTED - CAN YOU HELP?

If anyone has (or knows where one can get) data sheets for the Japanese Integrated Circuits mentioned below, I'd appreciate it if you'd send me a copy of the data sheets (at least the pinout diagram), or the information as to where this data can be obtained. All of these IC's are used in imported electronic telephones, and the first two are made by Sharp. The numbers on these are LR 40992 (an 18-pin IC) and LR 4089 (16 pins). The third chip is a 40-pin job and I cannot determine the name of the manufacturer by visual inspection, however, the following markings appear on the IC:

ET 3388
8 JUNE 83
44801B41
4H1 JAPAN

If you can help, please contact me (Jack Decker) at 1804 West 18th Street #155, Sault Ste. Marie, Michigan 49783-1268.

This article contains a proposal for a new way of making BBS calls, information on a new BBS program for the TRS-80 Models III and 4, news about a 10,000 bits per second modem (not a typo!), and details on MCI Mail's new method of Canadian access. But before I get into any of those items, please let me clear up one error that I made last issue.

In the article entitled "CUT THOSE HIGH TELEPHONE BILLS", I talked about a telephone company service called Remote Call Forwarding. I then went on to state that, when using this service, "calls can be forwarded to a local or long distance number". Well, in some areas that may be true, but not here in Michigan. The tariffs here are such that the telephone company will only set up Remote Call Forwarding to a number outside the local calling area. In other words, it can only be used for toll calls. The whole point is to prevent people from doing the very thing that I was suggesting they do, namely, to use a Remote Call Forwarding line to receive local calls from a greater distance than that which would normally be part of their local calling area. If this last sentence didn't make much sense to you, it's because you didn't read the original article in our last issue (and if that's the case, you might as well skip the next paragraph).

However, there is NO such restriction on calls forwarded from a normal residential or business line that is equipped with regular call forwarding service. The monthly charge for such a line is usually less expensive than that for a Remote Call Forwarding line as well. The only catch is that you have to have a friend that will permit you to use his house or place of business as the location for the telephone associated with that line. You don't even need to keep a phone permanently connected to the line - just plug one in temporarily, program the call forwarding feature, unplug the phone, and forget about the line until such time as you need to reprogram it. You probably wouldn't even have to have any telephone wiring installed inside your friend's house, since many phone companies are now using a lightning arrestor that includes a modular test jack. In this application, there would be no reason to run any wiring beyond that point, since a phone only needs to be plugged into the jack for the few seconds required to program the call forwarding feature. If the phone company asks, just tell them you intend to do your own inside wiring (which is legal now in most areas of the U.S.).

The main thrust of last issue's article was GTE Telenet's new PC PURSUIT service. Part of the method of operation of that service is to call the user back - in other words, you dial the PC PURSUIT access number, and it then calls you back (but only if you're a local call from their number, which is what got us into the discussion of Remote Call Forwarding in the first place). However, it has occurred to me that this callback feature could be utilized by small BBS operators, particularly those who have the facilities to provide multiple access lines. In fact, even the big services such as Compuserve, The Source, Delphi, etc. ought to consider using a variation on this theme.

You see, there is a problem that many of us face. We don't live in the big cities, or even in medium sized towns. Sure, there are packet switching networks that crisscross the country, but they only offer service in the major-to-medium sized cities. If you're not near one of the packet network nodes, you have to call long distance to access the online service of your choice, and those long distance charges can add up pretty fast. And, of course, if you're calling a BBS that isn't on a packet network, even being in a major city won't help.

But suppose that the online service that you wanted to call operated as follows:

1) You dial a toll-free 800 number. You are then given 45 seconds to enter your account ID, password, and the telephone number you are at (if not the one you normally use).

2) The system hangs up, checks your ID information, and if it's not valid, does nothing (this has an added benefit of enhancing system security). If, however, the account info all checks, the system calls the user back using the cheapest long distance service available.

3) The total connect time charges (for the initial 800 number call and the outward-dialed long distance charges) are billed to the user's credit card, along with any charge for using the service.

Why not just use an 800 number for the entire call? Because 800 number toll charges are fairly expensive. They may be a little less expensive than dialing direct, but not that much less.

Why offer 800 number access at all? Because some people may wish to initiate a call from their home, or from some other location (such as a hotel room) where they may not wish to or be able to place a direct dialed call, and have all the connect time charges billed on one account (perhaps to their business).

Would having the system call back really reduce charges that much? Well, the potential for savings are considerable. For one thing, if the online service is located in a major city, it will likely be able to pick from among many long distance carriers not available to callers in smaller towns. Then, too, because of the volume of outbound calls that will be generated, the online service will probably qualify for generous volume discounts from the chosen carrier. Finally, the online service may be able to take advantage of calling plans that are not feasible for the average caller to use, such as AT&T outward WATS lines, or similar services offered by other carriers.

As an example of the latter, GTE Sprint offers a service called "Advanced WATS" in some areas of the country. The difference between "Advanced WATS" and plain old AT&T WATS is that under the AT&T plan, you buy WATS lines in one of five or six "bands" that cover an area ranging from a small area surrounding your home state to the entire nation. The problem is that all calls on a given WATS line are charged at the same rate, so if you buy a single WATS line that serves the entire nation, you'll pay for calls to nearby states at the same rate as calls to distant states. In addition, you'll usually have to have a separate line to call within your home state. With "Advanced WATS" from GTE Sprint, you get the entire country (including your home state unless local tariffs prohibit it) and all calls are charged according to the band you call - no paying cross-country rates to call a neighboring state. I phoned GTE Sprint for rate information, and discovered that while there is no installation charge for this service, there is a $70/month line charge plus you must make a minimum of $350 per month per line in calls. This would be no problem for a commercial online service or even for a fairly popular BBS, but would certainly prohibit the average home user from having such service. In addition, it's only available in certain major cities (only in the Detroit/Ann Arbor area of Michigan, for example).

Presumably, if you lived outside of GTE Sprint's normal service area, you might be able to get service by paying an additional mileage charge for a private line into their switch. But even if you were forced to rely on AT&T WATS service, volume discounts are available after the first few hours of use. And keep in mind that most alternate long distance services offer either volume discounts or a service similar to WATS, or both. The point is that because the online service would be the originator of the majority of the toll calls, it would be eligible for "volume discounts" and other reductions in toll charges not available to the average caller. And, if the online service were a "biggie" such as Compuserve, they could conceivably arrange to send the callback over their own packet network as far as possible, and only use the outward WATS line for the final few miles of the call. Perhaps some of the big packet networks might think about offering such a "callback" service to their mainframe customers that don't have their own networks.

What's to stop a BBS that's currently operating from offering this type of service? Two things: hardware and software. And it would be best if both were designed as an integrated package, though this isn't absolutely necessary. Actually, as we will see, a small BBS operator could get by without using any additional hardware, but will probably wind up paying slightly higher toll charges. However, the lack of software that will support this type of operation is a universal problem. At present, no BBS software that runs on a personal computer (to my knowledge) supports any type of callback scheme. It should not be difficult for the designer of a BBS program to implement such a feature, but it is probably beyond the programming ability of the average SYSOP to add such a capability to his system.

The hardware portion of this scheme is a more thorny problem. Consider that even the bare minimum system as described above would require three lines: incoming (800) line, outgoing WATS line, and a local line (which, presumably, could be used for both incoming and outgoing calls under certain circumstances). However, in many states, intrastate and interstate WATS service must be on separate lines (especially if you're getting this service from AT&T), so to provide full service, you might want an additional incoming (800) service line

and an additional outgoing WATS line to provide full intrastate service. And, if you have a really high outgoing call volume to a certain area, you may wish to purchase an additional WATS or special service line (such as a "foreign exchange" line, which gives you unlimited incoming and outgoing service to a particular city at a flat monthly rate). It's easy to see that you may wish to connect your modem to any one of many different lines. The hardware interface must be capable of directing your modem to the proper outgoing line (under software control) and, during idle periods on the BBS, scan all incoming lines for a ringing signal, and if one is found, connect that line to the modem (and ignore all other ring signals once a connection is made). And if that isn't complicated enough, consider the question of a multiple computer (or multiple port on a timesharing computer) installation, where many modems might be competing for the same lines (this problem isn't as difficult to resolve as it might at first sound, especially for those familiar with the operation of the old rotary "step-by-step" connector switches used in older telephone company exchanges).

Even if NO additional hardware were added to a system, however, a callback feature could be implemented using software only. In this case only a normal telephone line would be used. Outgoing calls would be handled by routing them through an alternate long distance company using their normal dial-up service or their "Dial 1" service (if you have selected that carrier as your primary carrier). Incoming calls could be dialed direct (which would mean that the caller would have to charge the first minute to his own or to someone else's phone bill, possibly incurring an "operator-handled" or third number billing charge), or you could make arrangements with an alternate long distance carrier for a 'speed dial" number.

"Speed dial" numbers work in this manner: You dial a long distance carrier on an 800 number (or on a regular local access number that goes into their switch). You then dial in a code of 4 to 8 digits in length (using a tone-dial telephone or a modem capable of tone dialing), which automatically routes the call to your telephone number, and bills the call to you. Note that callers can only call you (or your BBS) using this number, so you can give the number out freely. The charge is usually the normal long distance charge from the long distance company's switch to your phone, plus a surcharge (usually 35 to 50 cents per minute). This charge is still less than using a credit card or operator-assisted call, and a BBS operator using this scheme would bill the calls back to the caller anyway. Note that long distance companies serving a small region ("WATS line resellers") are much more likely to offer this type of service than the major firms.

If you're looking into this type of service, here's a few pointers: First, if most of your incoming calls are from a certain geographic area, try to use a long distance reseller that has local access numbers in that same area. This way you can avoid the surcharges associated with the 800 lines. Second, you don't necessarily have to use a carrier that offers service in your city - in fact, you could use any carrier in the U.S. that offers this service. Since "speed dial" calls are billed just like any call originating from the carrier's switch, the calls can terminate anywhere. Naturally, it's to your advantage if the carrier's switch is located nearby (both in terms of cost and of quality of the connections), but not absolutely necessary. Go to your library and look up the long distance companies serving nearby cities, and contact all of them. Also, keep in mind that if you are located near a state line, it may be less expensive to use a carrier located in an adjacent state, since you will be paying the (usually) lower interstate rates for the portion of the call from their switch to you (however, note that in some states the intrastate rates may be lower for short distances, so don't automatically exclude carriers within your home state).

Note that by using a "speed dial" number for incoming (800) service, and an alternate long distance company (that offers volume discounts) for outgoing service, you could get by using only a single, regular telephone line for your BBS. Your software, however, would still have to have the callback feature, and of course would have to be able to keep track of usage for billing purposes. A couple of advantages to the SYSOP would be greatly increased security (since he would always have a record of the number called) and the ability to bill for all or part of his service on a per-minute or per-use basis.

One final caveat - if you use a single line for both incoming and outgoing calls, a potential for abuse does exist, especially if you have to dial a local number and account code to reach your long distance carrier. The reason is that if the incoming caller does not hang up, the line would remain open and the outgoing call from your modem would go out over this still-open circuit. A dedicated phone phreak could feed your system a "phoney" dial tone and then record your long distance access number and billing code! One way to defeat this (though I don't guarantee it to be foolproof in all systems) is to wait about 40 seconds before originating the outgoing call (the local phone company equipment will usually break the circuit automatically about 30 seconds after you hang up, unless the switching equipment in your telephone central office is the older mechanical variety). You should also reset the 40-second timer if anyone else calls in during the delay, in which case the outgoing call should not be made until about eight seconds after the last incoming ring signal is received (this prevents the phreak from hanging up, then calling back a few seconds later to receive the outgoing call).

On to other things. George Matyaszek (SYSOP of the Chicago Syslink BBS [formerly the Chicago Greene Machine], which can be reached at (312) 622-4442) tipped me off to a new BBS program called Syslink III. Still in the development stages, Syslink III supports automatic transfer of electronic mail and other types of files between systems (at night when the phone rates are lowest), in a somewhat similar manner to the FIDO-NET systems that run on other computers. Even at its present stage of development, this system shows great potential. Here's a portion of slightly edited text that was downloaded from the Syslink III headquarters system:

"Syslink III is a powerful yet easy-to-use networking program designed for communities, offices, professionals, and other organizations that need easy access to a central computer via telephone line or direct connection.

"Syslink operates on the TRS-80 Model 3 or 4 microcomputer under NEWDOS 2.0/2.5 operating system with a minimum of two 5-1/4" disk drives and a Hayes-language compatible telephone modem. To provide responsive operation, the software is written in SIBASIC, Software Interphase's custom communications language (supplied). Syslink fully supports a line printer or hard disk drive if connected. No additional software is required.

"Syslink's features include 3 complete message bases, each capable of storing up to 3000 messages among 30 sub-sections (total message capacity: 9000), 2-way file transfer with ASCII or XMODEM error-checking protocol, private Electronic mail with reply and multiple send, eight distinct status reports, customized terminal settings for each user, three levels of assistance with the system Features, an extensive surveying/voting section with statistical analysis, and fourteen universal library commands, as well as sophisticated yet easy to use Text and line editors with global search/replace. All commands are full-featured, menu-prompted, and supported with online Help.

"Other advanced features of Syslink include a merchandising section for individual advertisers, computer dating service, private FileMail to exchange of files instead of letters, multi-level Trivia section, and Common Interest Groups. The Common Interest Groups (CIGs) offer the flexibility of allowing only certain members to access bulletins, conferences, voting, or file transfer within a CIG. Syslink will also have networking capabilities which allow the automatic transfer of messages and files between systems.

"System operator Features include detailed caller's log with monitor of all user's activity, complete maintenance of up to 999-member user base, eight adjustable priority levels for each Feature and user, multiple Sysop capability, remote system operations, plus much more. Optional HouseCleaner function allows Syslink to operate completely unattended according to a maintenance schedule specified by the System Operator. System Operator also has complete control over each Feature in the system, and can deny or enable users access to certain areas of Syslink according to priority.

"Located in Providence, Rhode Island, the headquarters may be reached at (401) 272-1138. The system is accessible at 300 or 1200 baud 24 hours a day. Feel free to contact us there if you have suggestions or questions about our software. Call the headquarters to see the latest improvements and features in Syslink online in beta-testing stage.

"Syslink is now available to the public for $150.00. The Common Interest Groups (CIG) option (recommended for HARD DISK only) is an additional $50.00. The package includes a menu-driven System operator disk with complete maintenance utilities, run-time disk, data disk, and SIBASIC communications language as well as Sysop, user, and setup manuals. Custom

versions, updates to t software, and extra user manuals are offered at a nominal cost.

"For more information, contact Software Interphase, 5 Bradley Street, Providence, Rhode Island 02908, or call our headquarters system.

"Coming soon! Syslink version 4.0 - Totally machine language, fast, works with Model III or Model 4 in 4 mode, adapts to most any DOS, multi-tasking too! - run Sysop utilities while someone is online!"

[End of downloaded text.]

What's interesting about Syslink III is that it is a fairly sophisticated BBS program that could be an alternative to to the popular TBBS software now used on many bulletin boards. TBBS is a very powerful system, but has some very irritating "features" that the author seems unable or unwilling to fix. At the top of the list is that the call time limit counter sometimes fails to give adequate warning that you're about to be kicked off the system, and boots you off just as you're entering typing in line 24 of a 25 line message (even if it's a termination message to the SYSOP!). If anything will make you want to put your fist through your CRT screen, I can guarantee that will!!! I know for a fact that many SYSOPs and TBBS users have complained about this, yet the problem remains uncorrected at this writing. If Syslink III does not have any similar "user-hostile" features, it might be a viable alternative to TBBS - in fact, early reports indicate that some folks who have used both systems consider Syslink III to be the superior system! Comments from any of our readers that have made extensive use of Syslink III would be appreciated.

According to an article in a recent issue of the Toledo Area Tandy Special Interest Group newsletter, you can now get a modem that operates at 10,000 bits per second, even on ordinary dial-up telephone circuits. The name of this modem is called Fastlink and it is made by Telebit of Cupertino, California. Whereas conventional modems monitor two signals in the telephone bandwidth, Fastlink monitors 512 frequency intervals. It has the ability to "pick and choose" the frequencies to create a complex waveform that allows the high speed operation even over "dirty" telephone lines.

Telebit says that the 10,000 bps rate is the average rate to be expected over ordinary telephone lines. Furthermore, Fastlink can adapt to changing line quality, decreasing or increasing speed until it finds the highest dependable operating speed. For example, if noise is present on a line, it will lower its speed in 50 bps increments until it can establish reliable communications.

Right now there are two things that might dissuade the average computer hobbyist from purchasing a Fastlink modem. One is that none of the commercial online services use this type of modem yet, and the other is the price: $1995. Other modem manufacturers are undoubtedly hard at work on similar products, though, so the price should drop as the competition heats up - and even the $1995 price might be a bargain for a large business, since it could potentially save many times that amount in telephone toll charges.

Finally, a word for MCI Mail users (or potential MCI Mail users) in Canada. It's now possible for Canadian users (and U.S. subscribers travelling through Canada) to directly access their MCI Mailbox. Previously, Canadian users had to have a separate account with the Canadian packet network, DATAPAC. Now, they only need their MCI Mail account. MCI claims that this access is cheaper, easier and more direct for the Canadian users, who will now receive only one bill for using their MCI Mail account. MCI Mail will bill these users at 15¢ per minute for connect-time through DATAPAC (this is the same rate at which U.S. users are billed for use of the (800) WATS line access, however, U.S. users accessing MCI Mail through TYMNET are billed only 5¢ per minute and if one of MCI Mail's own ports in a major U.S. city is used, there is no per-minute charge).

The procedure to access MCI Mail from Canada is as follows:

1. Dial the local access number. For a list of access numbers, type HELP PHONES <province> (For example, HELP PHONES BRITISH COLUMBIA) and press ENTER.

2. At the tone, connect your modem; type two periods (..) and press ENTER. The two periods will not appear on your screen, but you will see the Datapac identifier. For example:

DATAPAC: 9240 4317

3. Type 13106 and press ENTER. You will not see what you type on your screen, but you will see Datapac's response. For example:

```
DATAPAC: call connected to 1 3106
         (29D) (1, n, remote charging, packet size: 128)
```

4. At "tymnet: please log in:" type MCIMAILUSA and press ENTER. Now, the letters you type and Datapac's response will appear. For example:

```
tymnet: please log in: MCIMAILUSA
P 25
MCI- IS ON LINE
```

5. At "Please enter your user name:" type your MCI Mail u name and press ENTER. For example:

```
Port: 25.
Please enter your user name:  mbeal
```

6. At "Password:" type your MCI Mail password and press ENTER. For security, your password is not displayed. For example:

```
Password:  _______
Connection initiated. . . Opened.

Welcome to MCI Mail!
```


64K **$59.95 PPD**

INSTALLED IN KEYBOARD

TRS-80* Model I-LII

Send us your Keyboard and we will convert it to full 64K memory (48K RAM). Improved performance with or without Interface. 90 day warranty. Satisfaction guaranteed. Quick return. Free return freight within U.S.A.

ICE

International Carbide & Engineering, Inc.
100 Mill St. • P.O. Box 216
Drakes Branch, VA 23937
(800) 424-3311
(804) 568-3311 TWX: 910-997-8341
*TM TANDY CORP.

DOUBLE PRECISION IN 5 BYTES

by Darrel Lewis

[Reprinted from the TRS-80 SYSTEM 80 Computer Group newsletter, Queensland, Australia.]

Storing amounts of money on disk files has always been a problem for me. Do I use single precision and allow amounts only to $9999.99 accurately, or do I use double precision and waste space as the amounts are seldom large enough to need 16 digits of precision?

Here is one answer to the problem. It provides for amounts up to $49,000,000.00 and only takes 5 bytes per amount - the last 5 bytes of a double precision variable are all that are actually used. To pack the value it is converted to a string and the function PK$(value$) is used. To unpack, the function UP$(packed$) is called. The following is an example, showing the length of the packed string and proving the original figure after unpacking:

```
10 DEFFN PK$(A$) = RIGHT$(MKD$(FIX(VAL(A$) * 100 + 0.50001#)),5)  '
PACK
20 DEFFN UP$(A$) = STR$(FIX(CVD(STRING$(3,0) + A$))/100)  ' UNP.
30 CLS : INPUT "ENTER AMOUNT (UP TO $49 MILLION - 49000000.00)
";M$
40 B$ = FNPK$(M$) : PRINT LEN(B$)
50 PRINT "THE AMOUNT WAS " FNUP$(B$)
```

FOUR DRIVES FOR THE MODEL 4P
by Dave Peters

[This article is actually a combination of two articles written by Dave Peters, which are reprinted with permission from the CAPATUG newsletter. This article is Copyright ©1985 by the Capitol Area TRS-80 Users Group, and may not be reprinted without written permission of the CAPATUG officers (the club President's name is Tim Sukay, and he may be reached at (717) 763-9112, or you may send your request to the club's mailing address: 340 Lewisberry Road, New Cumberland, Pennsylvania 17070. The club also maintains a BBS at (717) 774-6543).

Please note that the first paragraphs and the diagrams pertain to the modification on the original (black-and-white screen) Model 4Ps, while the final paragraphs deal with the newer "green screen" models. However, the screen color should not be considered an absolute test for which mod is required, because the changeover to the new CRT color and the changeover to the Gate Array Logic board (which requires the second mod) may not have occurred at exactly the same time. In particular, Model 4Ps purchased outside the U.S.A. have always had the green screen CRT, yet may require the first modification described below.

As is the case with any hardware project described in NORTHERN BYTES, the information presented here is to be used by experienced hardware hackers only. We have NOT tested or in any way verified these modifications, so you must install them at your own risk. NORTHERN BYTES, the Capital Area Tandy Users Group, and or the author of this article will NOT be responsible in any way for any damages (including consequential damages and/or loss of income) sustained as a result of attempting to install these modifications in your computer.]

Since I got my 4P last summer, I have been hampered somewhat in my copying ability due to the lack of more than two drives. Because Radio Shack did not see fit to add the extra traces to the circuit board to enable drives 2 and 3, I was stuck with two drives and the idea that the external drives could be added easily if only I could come across a schematic of my machine. Well, I finally came up with a book on the 4P (with the help of Tim Sukay) and succeeded in modifying my machine. The mod is relatively simple and requires only seven wires, two diodes and one resistor and one simple trace cut. The entire modification should take less than two hours for an experienced hardware hacker and no more than an evening for the lesser experienced in the club. Let me warn you, however, that this modification does take some soldering skill in that the traces are small and difficult to work with. If you don't think that you can trust yourself with a fine tipped soldering iron, then ask someone else to assist you. I have included a schematic of the modification and a pictorial of both sides of the printed circuit board with the necessary added wires and the trace cut marked. Take your time and study the pictures and your disassembled board carefully and mark the necessary spots with a magic marker if you wish, just to be sure. Now on with the fun.

You will need the following to perform the mod: Seven pieces of wire wrap wire in the following lengths, 2", 2 1/2", 5 1/2" (2 each), 6" (2 each) and 8 1/2". You will also need two 1N914 or 1N4148 diodes and one 4.7K resistor.

1) Start by disassembling the computer. If you aren't sure about how to open your machine, then stop here and get help.

2a) Using the 2" wire, solder one end to Pin 6 of J5 (the connector for the disk drives.)

2b) Solder the other end of this wire to U20 Pin 8.

3a) Take the 2 1/2" wire and solder one end to Pin 14 of J5.

3b) Solder the other end to U20 pin 10.

4a) Take a 5 1/2" wire and solder one end to U20 pin 9.

4b) Solder the other end to U32 pin 10.

5a) Take the other 5 1/2" wire and solder one end to U20 pin 11.

5b) Solder the remaining end to U32 pin 7.

6a) With a 6" wire, solder one end to U32 pin 7 (careful or the other wire will come off).

6b) Solder the other end to U24 pin 11.

7a) With the other 6" wire, solder one end to U32 pin 10 (be careful again).

7b) The other end gets soldered to U24 pin 12.

8a) With the remaining 8 1/2" wire, solder one end to the anode end (the end without the band) of the diodes and cover the diode and the joint with a piece of heat shrink tubing (cut the lead of the diode to a length of 1/4" first). Form the cathode end of the diode into a small foot and solder it to pin 10 of U24.

8b) Solder the free end of the last wire to pin 13 of U74.

9) Bend the leads of the 4.7K resistor so that they line up with pin 13 of U74 and pin 14 of U75 (+5V). Then solder it in place after you've removed the excess lead length.

10) Carefully examine the solder side of the board for solder splashes or shorts on the wires just installed. The remaining work will be done on the component side.

11) Locate the trace going from U75 pin 1 and U74 pin 13 on the component and cut it carefully with a sharp knife (I use an X-acto knife).

12) Now take the remaining diode and bend the leads so that one end touches pin 1 of U75 and the other touches pin 13 of U74. Once the diode is formed, solder the cathode end (with the band) to pin 1 of U75 and the anode end to pin 13 of U74.

This completes the wiring part of the modification. Carefully examine all work for solder bridges, splashes and shorts, and most importantly, correct installation. Once you are satisfied, reassemble your machine, with the exception of the rear panel and case. Now we will check out the modification and see if you made a mistake. Turn on your machine and look for erratic operation. If you receive the "DISK DRIVE NOT READY" prompt, insert a disk in drive 0 and boot the system. If you don't receive the prompt, or something else is not as it should be, then turn the machine off and carefully check your work. If all works OK, we are ready to hook up the external drives.

To hook up the external drives, you will need a new cable. I made one from an old two drive cable left over from my Model I. If you don't have the know how to make a cable, find some one who does or get ready to bite the bullet and buy one made to your specs. You will find a diagram attached with the dimensions to each connector. If you ever intend to disconnect the external drive, you will need some kind of connector mounted on the rear of your machine. I used a 37 pin "D" connector which as it turns out is slightly large, but with a little work, it came out OK. I cut a slot in the rear panel large enough to accommodate the back of the connector and slid it in from the opening for the modem. My problem was that the mounting ear extended over the card guide for the modem and I would not have been able to remove the modem board without disassembling the computer. I solved the problem by removing the mounting ear and by using small self tapping screws, securing the flanged edge of the connector to the rear panel. Another self tapping screw was used on the other mounting ear to finish the job.

The cables you will need (one for inside the machine and one to go to the external drives) can be bought or made at home (all you need is a vice) from purchased connectors and cable. The first cable will consist of two 34 pin edge connectors, one 34 pin double row pin connector, one 37 pin "D" female connector and an 18" length of 34 conductor ribbon cable. One of the edge connectors will get mounted on the end of the cable, the second, four inches from the first, seven inches after this connector, is where the double row pin socket gets mounted and the 37 pin "D" connector gets put on the end. That is all there is to this cable. The external drive cable will have to be made to your particular needs since you may want the drive mounted someplace different from where I mount mine. This cable is basically the same as an old Model I or a Model III cable except that it will have a 37 pin "D" male connector on the end that usually has the 34 pin edge connector.

Once you have the cables made, you can install the internal cable in your machine and reassemble it. Then all that is left is to hook up the external drive(s) and enjoy.

The above instructions (and the diagrams that accompany this article) were published in the CAPATUG newsletter back in March. At about the same time, the original white screen 4Ps were being replaced with the newer version known as the Gate Array Logic 4P. The telltale difference was the green screen of the latter. Several CAPATUG members who purchased "green screen" 4Ps wanted the external drive mod on their machines, so I went to work with my trusty soldering iron and a schematic of

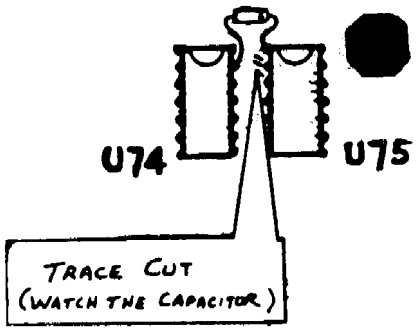
the machine and succeeded in adding the mod to this version of the 4P as well. The mod is much easier to install since only the addition of four wires is needed. You will still need to fabricate a new internal drive cable as described above. What follows is a description of the addition of the necessary four wires to make your machine complete.

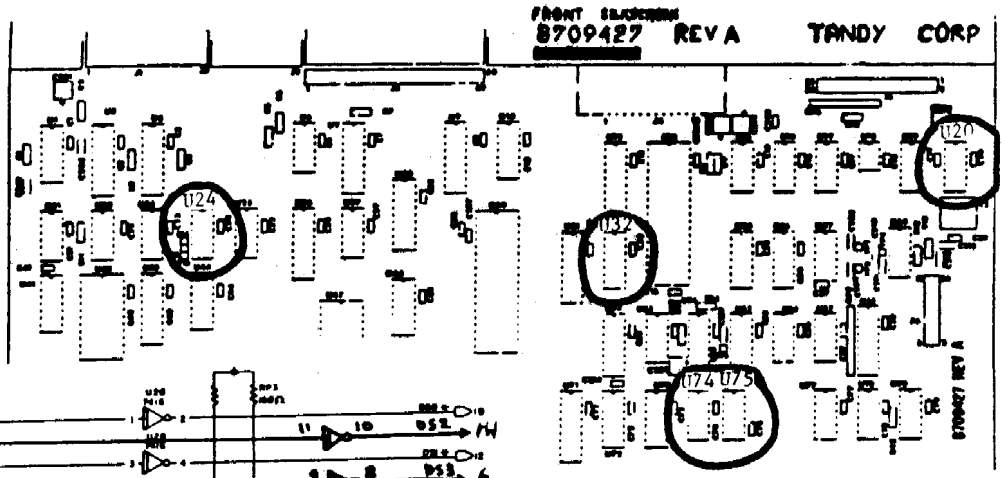
The most difficult part of the job is being able to identify the proper IC's and finding the right pins on those IC's. Oh, yes, and soldering to them! All of the IC's are numbered on the top side of the board. If you don't know which is pin one, seek help. Well, here goes:

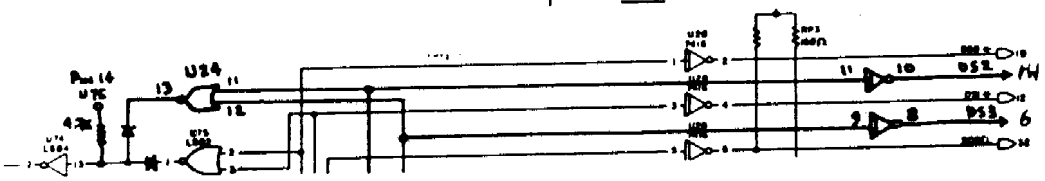
Locate U34 (74LS174) and solder a wire (I use 30 gauge wirewrap wire) to pin 7. The other end is connected to U14 (74LS16) pin 13. Now solder a wire to pin 10 of U34 and the other end to U14 pin 9. Take the third wire and solder one end to pin 12 of U14 and the other end to pin 14 of the double row header which goes to the floppy drives (J5). The pin numbers for J5 are on the top of the board and go in odd rows and even rows. The last wire will go from U14 pin 8 to J5 pin 6. That's all there is to it. Check your solder joints and reassemble your computer. You will have to make a new internal drive cable and a matching external drive cable as described above.

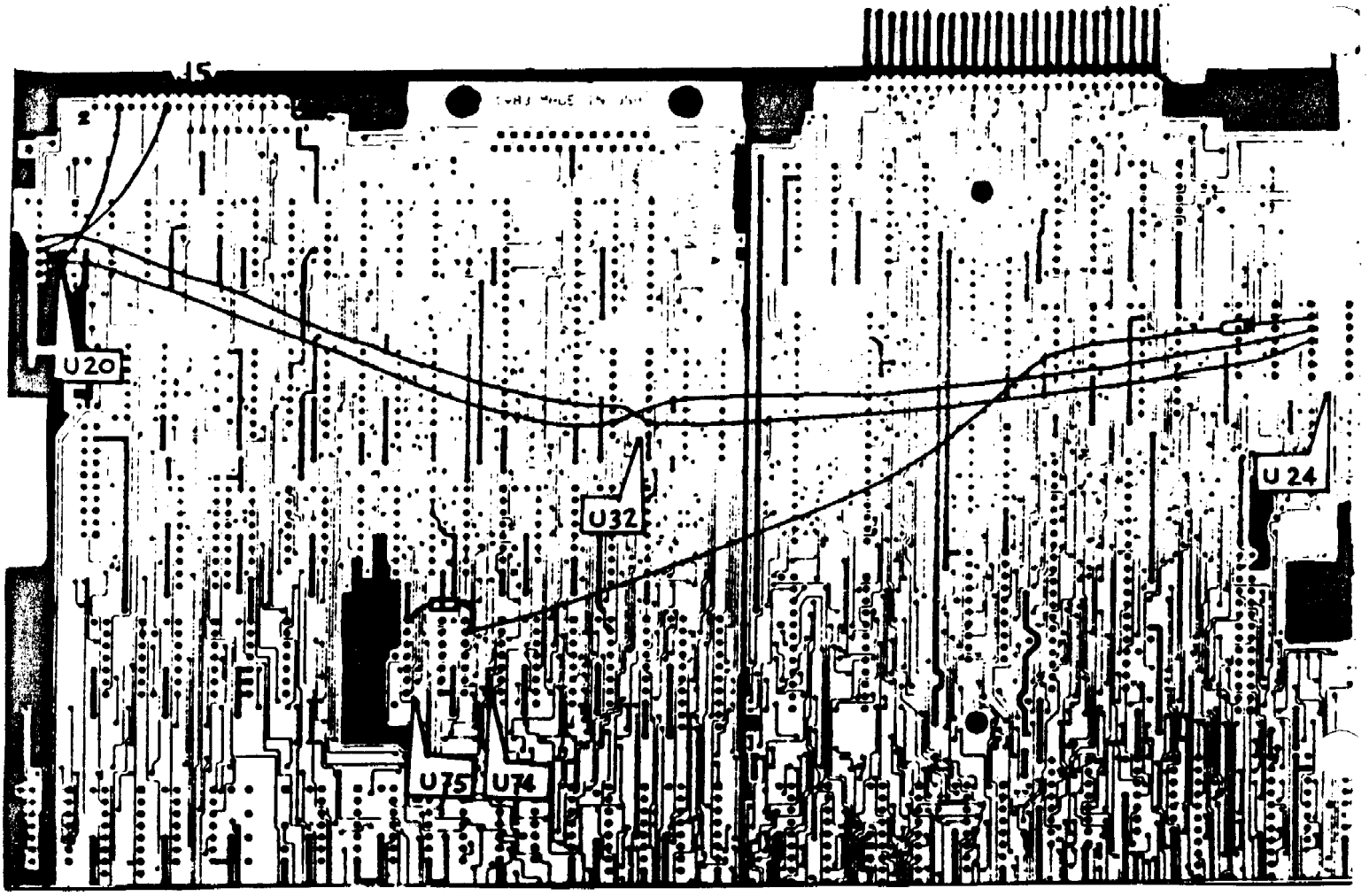
If you have any difficulties trying to figure this mod out, please feel free to call me at home at (717) 766-8192. I'm always glad to help.

[Editor's note: Please note once again that the diagrams below pertain to the installation in the original Model 4Ps, not the newer Gate Array Logic models.]









KEYBOARD PROCESS ROUTINE FOR USE FROM BASIC with SAMPLE PROGRAM

by Ron Zajac

Here is a relocatable keyboard routine which you can easily modify and use from a BASIC program. It places significant numeric values into consecutive integer array locations. The routine is called like this:

```
X = USR(VARPTR(Y(0))
```

-- where Y(0) is an integer array into which the keyboard values are to be placed. When the routine returns, Y(0) and Y(1) will contain values corresponding to key responses for key clusters on the right and left sides of the keyboard, respectively. Also, Y(2) will contain -1 if the space bar is pressed; this allows a pause feature in the game. Note that the documented routine will only alter the array values if pressed keys impose new values; this serves the interest of the QUIK program documented later on. If you want 0 values returned if no keys are pressed (more like an INKEY$ function), remove the source code lines with the "*" in the first comment column position.

Next, a highly optimized BASIC program is shown which makes use of the Z80 code generated from the DUALKEY assembly code. The code is put into an integer array, and the DEFUSR is set to the address of the zeroth element. The statement "G=USR0(BV)" performs the scan. BV contains the address of the key results array B().

QUIK is yet another version of the 2-player entrapment game where you try to be the last person to run your playing piece into a wall or "trail". The trails are left by the playing pieces as they move inexorably around the board. You can only control the direction of your piece by pressing the appropriate key for left, right, up and down movement. There is also a "warp" feature; if you see a special "warp" token appear in the playing field, you can gobble it up and use the credit to warp yourself to another random location on the board when you get into a tight spot. You activate warp by pressing a special key. Here is a table of the keys used by the two players:

Direction	Player 1	Player 2
Left	S	J
Right	F	L
Up	E	I
Down	D X or C	K M or , (comma)
WARP	3 or 4	8 or 9
PAUSE	Space Bar	

[Assembly language listing in ALE format:]

```
                ;               Process DUAL keyboard   02/16/85

5200                    ORG     5200H           ;Code is relocatable
0001            ROW1    EQU     01H             ;LSB of Keyboard Row ONE
0002            ROW2    EQU     02H             ;LSB of Keyboard Row TWO
0004            ROW3    EQU     04H             ;LSB of Keyboard Row THREE
0008            ROW4    EQU     08H             ;LSB of Keyboard Row FOUR
0010            ROW5    EQU     10H             ;LSB of Keyboard Row FIVE
0020            ROW6    EQU     20H             ;LSB of Keyboard Row SIX
0040            ROW7    EQU     40H             ;LSB of Keyboard Row SEVEN

0020            P1%UP1  EQU     00100000N       ;E - 3801H
0001            P1%DN1  EQU     00000001N       ;X - 3809H
0008            P1%DN2  EQU     00001000N       ;C - 3801H
0010            P1%DN3  EQU     00010000N       ;D - 3801H
0008            P1%LF1  EQU     00001000N       ;S - 3804H
0040            P1%RT1  EQU     01000000N       ;F - 3801H
0008            P1%WP1  EQU     00001000N       ;3 - 3810H
0010            P1%WP2  EQU     00010000N       ;4 - 3810H
0080            P1%PAU  EQU     10000000N       ; - 3840H

0002            P2%UP1  EQU     00000010N       ;I - 3802H
0020            P2%DN1  EQU     00100000N       ;M - 3802H
0010            P2%DN2  EQU     00010000N       ;, - 3820H
0008            P2%DN3  EQU     00001000N       ;K - 3802H
0004            P2%LF1  EQU     00000100N       ;J - 3802H
0010            P2%RT1  EQU     00010000N       ;L - 3802H
0001            P2%WP1  EQU     00000001N       ;8 - 3820H
0002            P2%WP2  EQU     00000010N       ;9 - 3820H
0080            P2%PAU  EQU     10000000N       ; - 3840H

0A7F            HLLOAD  EQU     0A7FH
0A9A            HLSAVE  EQU     0A9AH
0008            WHERE   EQU     08H             ;PC ==> HL

5200            ENTRY   EQU     $

5200 CD7F0A             CALL    HLLOAD
5203 E5                 PUSH    HL              ;Save address of array

5204 CD0800             CALL    WHERE
5207 1830               JR      EXEC

                ;       Note the clock orientation of the movement codes:

                ;                               4
                ;                               |
                ;                       3 -- + -- 1
                ;                               |
                ;                               2       Also, 5 is WARP

5209 012004     TABLE   DEFB    ROW1,P1%UP1,4
520C 080102             DEFB    ROW4,P1%DN1,2
520F 010802             DEFB    ROW1,P1%DN2,2
5212 011002             DEFB    ROW1,P1%DN3,2
5215 040803             DEFB    ROW3,P1%LF1,3
5218 014001             DEFB    ROW1,P1%RT1,1
521B 100805             DEFB    ROW5,P1%WP1,5
521E 101005             DEFB    ROW5,P1%WP2,5

5221 020204             DEFB    ROW2,P2%UP1,4
5224 022002             DEFB    ROW2,P2%DN1,2
5227 201002             DEFB    ROW6,P2%DN2,2
522A 020802             DEFB    ROW2,P2%DN3,2
522D 020403             DEFB    ROW2,P2%LF1,3
5230 021001             DEFB    ROW2,P2%RT1,1
5233 200105             DEFB    ROW6,P2%WP1,5
5236 200205             DEFB    ROW6,P2%WP2,5

5239 E5         EXEC    PUSH    HL
523A DDE1               POP     IX              ;IX points to TABLE-2
523C 0602               LD      B,2             ;2 tables to process

523E 0E08       LOOP1   LD      C,8             ;8 key entries/table
5240 1E00               LD      E,0

5242 DD6E02     LOOP2   LD      L,(IX+2)        ;LSB of key matrix
5245 2638               LD      H,38H           ;MSB of key matrix
5247 7E                 LD      A,(HL)          ;Get key char
5248 DDA603             AND     (IX+3)          ;Use bit mask
524B DD7E04             LD      A,(IX+4)        ;Get movement code
524E DD23               INC     IX
5250 DD23               INC     IX
5252 DD23               INC     IX              ;Point to next key entry
5254 2801               JR      Z,CHEKC         ;If no key pressed, dont xfer
5256 5F                 LD      E,A             ;xfer

5257 0D         CHEKC   DEC     C
5258 20EB               JR      NZ,LOOP2        ;Check each key entry

525A E1                 POP     HL              ;Get back array VARPTR
525B 7B                 LD      A,E             ;Posit key response
525C B7                 OR      A               ;*
525D 2805               JR      Z,CHEKB         ;*
525F 77                 LD      (HL),A
5260 23                 INC     HL
5261 AF                 XOR     A
5262 77                 LD      (HL),A
5263 2B                 DEC     HL
                ;                               Omit * lines for INKEY$-like function
5264 23         CHEKB   INC     HL
5265 23                 INC     HL              ;Next integer array element
5266 E5                 PUSH    HL
5267 1005               DJNZ    LOOP1           ;Process both tables

5269 E1                 POP     HL              ;Clean stack
526A 3A4038             LD      A,(3840H)       ;Key SPACE BAR address
526D FE80               CP      10000000N       ;Check for space bar
526F C29A0A             JP      NZ,HLSAVE       ;If not, don't pause
```

```
5272  36FF          LD    (HL),-1      ;Pause flag
5274  C39A0A        JP    HLSAVE
                    END

SYMBOL  TABLE
ROW1    0001    ROW2    0002    ROW3    0004    ROW4    0008    ROW5    0010
ROW6    0020    ROW7    0040    P1UP1   0020    P1DN1   0001    P1DN2   0008
P1DN3   0010    P1LF1   0008    P1RT1   0040    P1WP1   0008    P1WP2   0010
P1PAU   0080    P2UP1   0002    P2DN1   0020    P2DN2   0010    P2DN3   0008
P2LF1   0004    P2RT1   0010    P2WP1   0001    P2WP2   0002    P2PAU   0080
HLLOAD  0A7F    HLSAVE  0A9A    WHERE   0008    ENTRY   5200    TABLE   5209
EXEC    5239    LOOP1   523E    LOOP2   5242    CHEKC   5257    CHEKB   5264
```

```
100 CLEAR 1000: DEFINT A-N: DEFSTR O-Z: GOSUB 240: GOTO 220
110 IF B(2) THEN GOSUB 320
120 H=0: IF B(A)=5 THEN IF E(A)=0 THEN H=-1: GOTO 310 ELSE
E(A)=E(A)-1: GOTO 200
130 D(A)=D(A)+C(B(A)): IF PEEK(D(A))=32 THEN PRINT@D(A)-
CR,O(A);: RETURN
140 IF PEEK(D(A))=M THEN AW=0: CW=RND(AR): E(A)=E(A)+1 ELSE
H=-1
150 GOTO 310
160 CW=CW-1: IF CW>0 THEN RETURN ELSE CW=RND(AR): IF AW
THEN AW=0: PRINT@LW,Z0;: RETURN ELSE AW=-1: GOSUB 210:
PRINT@LW,WP;: RETURN
170 G=USR0(BV): A=1: GOSUB 110: IF H THEN F(0)=F(0)+1: GOSUB
310: GOTO 220 ELSE GOSUB 160
180 IF NP THEN G=USR0(BV): A=0: GOSUB 110: IF H THEN
F(1)=F(1)+1: GOSUB 310: GOTO 220 ELSE GOSUB 160
190 IF NP THEN 170 ELSE PRINT CHR$(28);:
PRINTUSING"###";CT;: CT=CT+1: GOTO 170
200 B(A)=RND(4): GOSUB 210: GOSUB 310: D(A)=LW+CR: GOSUB
310: GOTO 170
210 G=USR0(BV): LW=RND(19)*3-1+RND(13)*64: IF
PEEK(LW+CR)<>32 THEN 210 ELSE RETURN
220 PRINT W;: POKE 16383,191: IF NP THEN PRINT@0,"";:
PRINTUSING"P1-###";F(0);: PRINT@58,"";: PRINTUSING"P2-
###";F(1);
230 C(1)=3: C(2)=64: C(3)=-3: C(4)=-64: D(0)=15423: D(1)=16318:
B(0)=1: B(1)=3: B(2)=0: E(0)=0: E(1)=0: AW=0: AR=40: CT=1: GOTO 170
240 DIM
D(1),B(2),O(1),C(4),F(1),A(59),E(1),A,BV,H,CW,CR,NP,G,AW,Z0,LW,AR,W
P,W,M,W3,CT,W1,W4,W2,W0,Z,L,C,D,E,F,B,Z1
250 FOR A=0 TO 59: READ A(A): NEXT: DEFUSR0=VARPTR(A(0)):
BV=VARPTR(B(0)): F(0)=0: F(1)=0
260 O(0)=STRING$(3,166): O(1)=CHR$(179)+CHR$(140)+CHR$(179):
Z=STRING$(3,36): WP=CHR$(165)+CHR$(143)+CHR$(154): M=ASC(WP):
Z0="  ": CR=15360
270 W0=CHR$(28)+CHR$(191): W1=STRING$(19,191): W2="..!-!-!-Q-
U-I-K-!-!-!..": W3=STRING$(4,191)+CHR$(252): W4=STRING$(8,191):
W=W0+W1+W2+W1+W3+W3+W3+W3+W3+W3+W3+W3+W3+W3+W3+W3+W3+W3
+W3+W1+W1+W1+W4
280 RANDOM: CW=RND(AR): CLS: PRINT "Q-U-I-K, by Ron A.
Zajac";STRING$(3,13);"How many players (1 or 2)?"
290 Z1=INKEY$: IF Z1="" THEN 290 ELSE NP=VAL(Z1): IF NP<1 OR
NP>2 THEN 290 ELSE NP=1-NP: PRINT CHR$(13);"Do you want
instructions (Y/N)?";
300 Z1=INKEY$: IF Z1="" THEN 300 ELSE IF NOT(Z1="Y"ORZ1="y")
THEN RETURN ELSE IF NP THEN 580 ELSE GOTO 330
310 FOR B=0 TO 20: PRINT@D(A)-CR,Z0;: G=USR0(BV): G=USR0(BV):
PRINT@D(A)-CR,O(A);: G=USR0(BV): G=USR0(BV): NEXT B: RETURN
320 Z1=INKEY$: IF Z1="" THEN 320 ELSE B(2)=0: RETURN
330 CLS: PRINTCHR$(210);"Q U I K  s o l i t a i r e";CHR$(13)
340 PRINT"You will be propelled around a 20-by-14 square board.
Your"
350 PRINT"playing piece (which looks like this: ";O(1);") will always
be"
360 PRINT"moving, but you may change it's direction of motion by
using"
370 PRINT"these keys:";CHR$(13);CHR$(222);"[up]"
380 PRINT"Note that you";CHR$(210);"I";CHR$(211);"Putting your"
390 PRINT"may use either";CHR$(228);"middle finger"
400 PRINT"K,M or ','     [left] J";CHR$(203);"L [right]    on the
'K'"
410 PRINT"to move your";CHR$(228);"key aligns your"
420 PRINT"piece downward.";CHR$(204);"K,M or
',';CHR$(205);"other fingers"
430 PRINTCHR$(220);"[down]";CHR$(13)
440 PRINT"PRESS ENTER TO CONTINUE";: LINEINPUT Z1
450 CLS: PRINTCHR$(210);"Q U I K  s o l i t a i r e";CHR$(13)
460 PRINT"If you run into a wall (or your own trail) the game is
over."
470 PRINT"The number of squares you 'painted' on the screen will
be shown"
480 PRINT"in the upper left-hand corner of the display";CHR$(13)
490 PRINT"If you see one of these . . . ";WP;" ,";CHR$(13)
500 PRINTCHR$(209);". . . RUN INTO IT!  This is a 'Warp Credit'."
510 PRINT"Every time you 'bag' one, you are entitled to a
complementary"
520 PRINT"'warp'.  If you get into a tight situation, just press
either"
530 PRINT"'8' or '9'.  You will be 'warped' to another random place
on the"
540 PRINT"screen.  When you see your piece warping back onto the
screen,
550 PRINT"be sure to decide a direction and press the
corresponding key.";CHR$(13)
560 PRINT"PRESS ENTER TO CONTINUE";: LINEINPUT Z1: RETURN
570 '
580 CLS: PRINT"                    Q U I K  d u e l";CHR$(13)
590 PRINT"The two persons playing will be propelled around a 20-
by-14"
600 PRINT"square board.  Your playing pieces (shown in the diagram,
below)"
610 PRINT"will always be moving, but you may change direction by
using"
620 PRINT"these keys:"
630 PRINTCHR$(205);"[up]";CHR$(206);"|";CHR$(206);"[up]"
640 PRINT"Player 1     E";CHR$(208);"|";CHR$(207);"I      Player 2"
650 PRINTCHR$(223);"|"
660 PRINT"[left] S   ";O(0);"   F [right] | [left] J   ";O(1);"   L
[right]"
670 PRINTCHR$(223);"|"
680 PRINT"Warp is  D,X or C";CHR$(205);"|";CHR$(203);"K,M or','
Warp is"
690 PRINT"3 or 4     [down]";CHR$(206);"|";CHR$(204);"[down]      8
or 9";CHR$(13)
700 PRINT"PRESS ENTER TO CONTINUE";: LINEINPUT Z1
710 CLS: PRINTCHR$(216);"Q U I K  d u e l";CHR$(13)
720 PRINT"If one of you hits a wall or trail your opponent gains a
point.";CHR$(13)
730 PRINT"If you see one of these . . . ";WP;" ,";CHR$(13)
740 PRINTCHR$(209);". . . RUN INTO IT!  This is a 'Warp Credit'."
750 PRINT"Every time you 'bag' one, you are entitled to a
complementary"
760 PRINT"'warp'.  If you get into a tight situation, press one of
your"
770 PRINT"warp keys (just grope for the number keys; you'll hit
it!)."
780 PRINT"You will be 'warped' to another random place on the
screen."
790 PRINT"When you see your piece warping back onto the screen,
be sure"
800 PRINT"to decide a direction and press the appropriate direction
key.";CHR$(13)
810 PRINT"PRESS ENTER TO CONTINUE";: LINEINPUT Z1: RETURN
820 '
830 DATA 32717,-6902,3021,6144,304,1056,264,258
840 DATA 520,4097,1026,776,16385,4097,1288,4112
850 DATA 517,1026,8194,8194,528,2050,514,772
860 DATA 4098,8193,1281,544,-6907,-7715,518,2062
870 DATA 30,28381,9730,32312,-22819,-8957,1150,9181
880 DATA 9181,9181,296,3423,-6112,31713,10423,30469
890 DATA -20701,11127,8995,4325,-7723,16442,-456,-15744
900 DATA 2714,-202,-25917,10
```

HELP WANTED - CAN YOU HELP?

Daryl Heggarty of New South Wales, Australia wants to contact Laredo Systems, now believed defunct, about their 5mb hard disk LS 525. If you know anything anything about them (or maybe who has taken them over), please contact Daryl c/o Clifford S. Richards, 3 Boronia Road, Bellevue Hill, Sydney, New South Wales 2023, AUSTRALIA (Cliff can also be reached via MCI Mail, the ID number is 266-3283, or you can send your message to NORTHERN BYTES and we will forward it to Cliff).

EPROM PROGRAMMER
by Rich Balas

[Reprinted from NORTHERN BYTES Volume 3 Number 11, which in turn reprinted this article from the ABC Micro-Computer Newsletter (Volume 1, Number 2, July 1982). The ABC Micro-Computer Newsletter is a publication of the ABC-TV Owned and Operated Stations Division that nowadays caters mostly to the IBM-PC. Please note our standard disclaimer for this type of hardware project - we have NOT tested it, and for all we know it could cause serious damage to your computer, especially if you don't follow the instructions exactly. In any event, we assume absolutely no liability for any damage (including consequential damages, loss of earnings, etc.) that might occur as a result of you attempting to build or use this project.]

If you own a TRS-80™ Model I computer and have a need or desire to enter the world of firmware, here is a simple and low cost way to start.

Construction: I am not going to write a long thesis on how to build this programmer. Electronic fabricating and a background in engineering will be assumed. Layout is not critical except for the length of cable between the 8255 chip and the EPROM socket (zero insertion force recommended).

Without using pull-up resistors on the data lines try to keep cable under 6". All good engineering practices should be followed.

Interfacing: Connection to the TRS-80 is easy. A 40 conductor ribbon cable with a 40 pin mating connector is all that is needed.

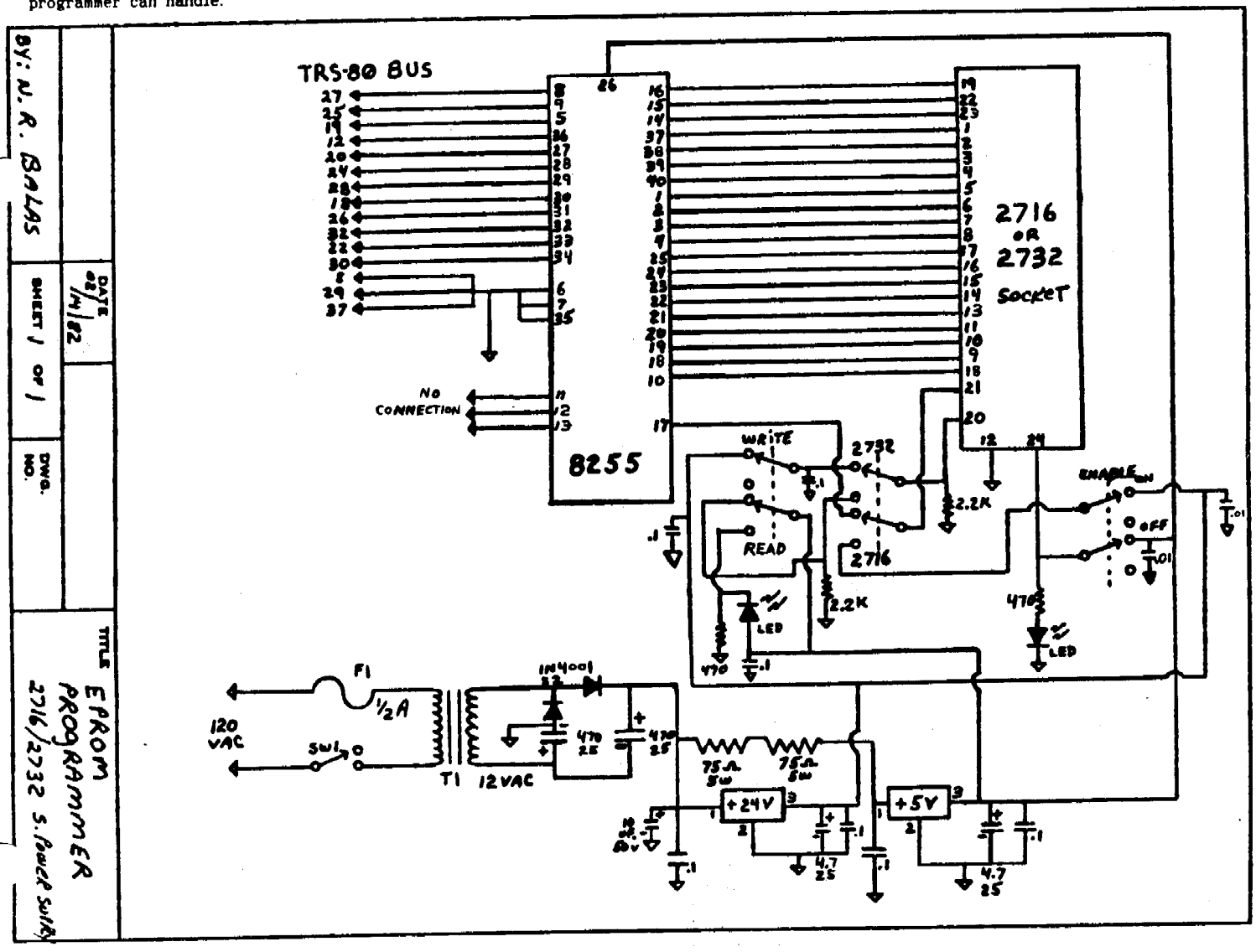
Software: Two BASIC listings are shown. One is for 2716/2516 single voltage devices, and one for 2732/2532 single voltage devices. Single voltage devices are the only type this programmer can handle.

Operation: With the power to the EPROM programmer off, connect the unit to the TRS-80 expansion bus port. The software uses locations 7000 hex to 77FF hex for 2x16 devices, and 7000 hex to 7FFF hex for 2x32 devices. Load the machine language program which is to be transferred to the EPROM device. Select EPROM type 2716 or 2732 on the programmer. This is important, damage will result if not correct. Place the read/write switch in the read position when inserting the EPROM device. Insert the EPROM in the programmer and apply power to the unit. Run the BASIC program applicable to the device being programmed. Instructions and prompting are provided. You will be asked to turn on the chip enable and write switches. Next you depress the ENTER key on the keyboard and the programming session begins. The decimal locations are displayed. When the write is complete you must switch the read/write switch to the read position. Again hit the ENTER key and the EPROM data will be compared to the data in memory, all errors will be displayed.

In closing I want to point out that if you want to read data from an already programmed 2716 device, you should add another switch to reduce the +24 volt programming line to pin 21 to +5 volts.

™A registered trademark of the Tandy Corporation.

```
5 '     PROGRAM FOR 2716 AND 2516 DEVICES
10 OUT 3,128:OUT 2,0:CLS
20 PRINT@64,"WILL BEGIN TRANSFER OF DATA"
30 PRINT@128,"STARTING AT 7000 HEX"
40 PRINT@128,"ENDING    AT 77FF HEX"
50 PRINT@256,"WRITE SWITCH AND CHIP ENABLE SWITCH ON"
60 PRINT@320,"PRESS <ENTER> TO START ";:INPUT Q
70 PRINT@448,"NOW PROGRAMMING LOCATION:"
```



```
80 FOR I = 28672 TO 30719
90 X=I-28672:MB=INT(X/256):LB=X-MB*256:Z=PEEK(I):PRINT@475,X
100 OUT 0,LB:OUT 2,MB:OUT 1,Z:OUT 2,MB+128:FOR J = 1 TO
11:NEXTJ:OUT 2,MB
110 NEXT I
120 PRINT@512,"DISABLE WRITE SWITCH, ";:INPUT"PRESS <ENTER> TO
CONTINUE ";Q
130 E=0
140 PRINT@640,"LOCATION","MEMORY","EPROM"
150 FOR I = 28672 TO 30719
160 X=I-28672:MB=INT(X/256):LB=X-MB*256:Z=PEEK(I)
170 OUT 3,130:OUT 0,LB:OUT 2,MB:U=INP(1):PRINT@704,X,Z,U, "TOT. ERR
";E
180 IF Z=U THEN 200
190 E=E+1
200 NEXT I
210 PRINT@768, "2716 WRITE AND VERIFY COMPLETE"
220 END
```

```
5 '     PROGRAM FOR 2732 AND 2532 DEVICES
10 OUT 3,128:OUT 2,0:CLS
20 PRINT@64,"WILL BEGIN TRANSFER OF DATA"
30 PRINT@128,"STARTING AT 7000 HEX"
40 PRINT@128,"ENDING   AT 7FFF HEX"
50 PRINT@256,"WRITE SWITCH AND CHIP ENABLE SWITCH ON"
60 PRINT@320,"PRESS <ENTER> TO START ";:INPUT Q
70 PRINT@448,"NOW PROGRAMMING LOCATION:"
80 FOR I = 28672 TO 32767
90 X=I-28672:MB=INT(X/256):LB=X-MB*256:Z=PEEK(I):PRINT@475,X
100 OUT 0,LB:OUT 1,Z:OUT 2,MB:FOR J = 1 TO 11:NEXT J:OUT 2,MB+128
110 NEXT I
120 OUT 2,MB
130 PRINT@512,"DISABLE WRITE SWITCH, ";:INPUT"PRESS <ENTER> TO
CONTINUE ";Q
140 E=0
150 PRINT@640,"LOCATION","MEMORY","EPROM"
160 FOR I = 28672 TO 32767
170 X=I-28672:MB=INT(X/256):LB=X-MB*256:Z=PEEK(I)
180 OUT 3,130:OUT 0,LB:OUT 2,MB:U=INP(1):PRINT@704,X,Z,U, "TOT. ERR
";E
190 IF Z=U THEN 210
200 E=E+1
210 NEXT I
220 PRINT@768, "2732 WRITE AND VERIFY COMPLETE"
230 END
```

USING 256 CYCLE REFRESH RAMS

by Errol Rosser

[Reprinted from SYDTRUG NEWS, P.O. Box 297, Padstow, New South Wales 2211, AUSTRALIA.]

In the December 1984 issue of SYDTRUG NEWS (Volume 5, Issue 4) [and in NORTHERN BYTES Volume 6, Number 1], Dave Kennedy showed a way of generating a 256 cycle refresh on the TRS-80 Mod I for those 4164 dynamic RAM's which need it.

That modification, whilst appearing logically correct, was in fact unreliable, and was found to be the cause of many unexplained crashes. It seems to be a timing fault causing delays in the A7 line to the RAMs.

The following circuit was found to be 100% reliable, even at 3 times speed-up. It is reasonably simple to install if you 'piggy-back' the extra chips.

To do the modification, you will need one each of 74LS30, 74LS74, 74LS367 and 74LS04.

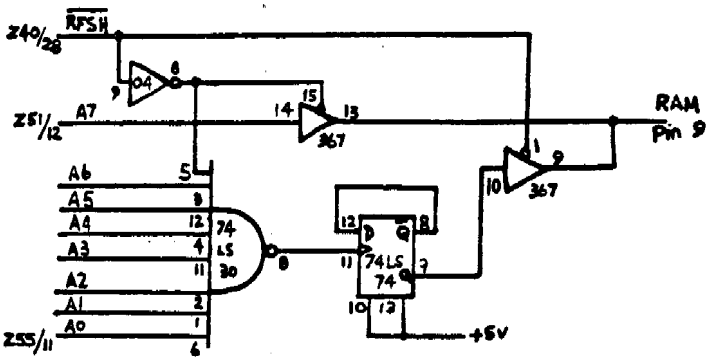
(1) The 74LS30 is piggy-backed over Z54 with pins 5, 6, & 8 bent out and all the rest soldered to Z54.

(2) The 74LS367 goes over Z51 with pins 8 & 16 soldered and all the rest bent out.

(3) The 74LS74 goes over Z53 with pins 7 & 14 soldered and all the rest bent out.

(4) The 74LS04 goes over Z52 with pins 7 & 14 soldered and all the rest bent out.

NOTE: The pins that are bent out on the piggy-back chips should be cut back very short so that they don't touch the piggy-back chip next to them.



Two Model 4 Utilities

*** REF ***

Cross reference for BASIC programs
In sorted order

REF$ All integers, line numbers, variables
REF$ Same with printer output
REF$xx Start full list at xx
REF$xx Same with output to printer
REF xx References for only xx
REF"xxx All references to string "xxx

* Variables up to 8 characters
* Locates ANY string (even in REMs)
* All machine language
* Interfaced fully to TRSDOS 6.x
* Does NOT provide the Edit function
* Accessible from BASIC

*** UNIKEY ***

Provides these functions

* One key entry for over 80 BASIC key words and phrases. Some examples are:

KEY	/SHIFT	/CTRL
D	DATA	DEF
W	WHILE	WEND
O	ON ERROR GOTO	OPEN "
R	RIGHT$(	RSET
L	LEFT$(	LSET

* HELP screen with all key combinations
* Self relocating
* Each Function key programmable for up to 80 characters. Change at any time.

REF $ 24.95 UNIKEY $ 19.95 BOTH $ 39.95 Add $1.00 Postage USA $2.00 elsewhere

Salsbury Associates Inc. 610 Madam Moore's Lane New Bern, NC 28560

CONVERTING COPYCAT3 INTO /CMD FORMAT
by Arne Rohde

1. Introduction

COPYCAT3 is a disk copy program for the TRS-80 Model III and Model 4. It will copy many protected disks, but, like Super Utility, it will not copy itself.

Using the disk access program TRAKCESS, it is possible to make a backup copy of the COPYCAT3 disk, and one method of doing this has already been published in Northern Bytes. The disadvantages of using self-booting disks, even if they can be copied, is that they take up a complete disk for just a single program, and that they cannot be called directly when operating in a DOS environment.

The procedure outlined here can be used to convert COPYCAT3 to a /CMD file which can be called like any other DOS program. After the program has been run, however, it will be necessary to reboot the system since the DOS has been destroyed. The method has been used for Version 3.02 of COPYCAT3, but since I neither own nor use COPYCAT3 I have not tried it on other versions.

To use this method of conversion to a /CMD file you will need a backup copy of COPYCAT3, a copy of TRAKCESS, and a little knowledge of disk formats.

2. Protection

COPYCAT3 uses a number of different methods to protect the program against prying eyes and monitor programs. But like all other programs the initial loader sector must be in readable format and the program can be followed from here.

The first attempt at foiling detection is that the boot sector reads another sector on top of itself. The sector which is read consists partly of the same information as the boot sector, and partly some extra code for loading the rest of the program. If the boot sector is modified to jump to a monitor program after loading a sector (by a breakpoint, for example) then the jump will never occur since the code has been overlaid by the data from the new sector.

The next trick is to identify the sectors on the disk with unusual track and sector numbers. A sector on a disk contains information besides the data contained in the sector. It contains a track number and a sector number, and the track number need not be the same as the number of the physical track on which it resides. Thus on track 0 you will find 4 sectors if you use the scan track feature of TRAKCESS. The first sector is the boot sector, and is identified with track 00H, sector 01H. Then follows a sector (the first one read) with track number 69H and sector number 69H. The sector identified as track 40H, sector 40H contains the code for configuring the program, and the sector with track number 30H, sector number 30H contains the data for the disk configuration.

The sector lengths are also unusual for TRS-80 disks, with sector 69H and 40H having a length of 1024 bytes each, and sector 30H a length of 128 bytes.

When the main program is to be loaded the loader program from sector 69H does a few steps backwards and forwards with the disk head, before landing on physical track 19 (decimal). This track contains two identical sectors, one with identification track 40H, sector 40H, and the other with track and sector number 30H. The code in these sectors has been hidden by the simple expedient of exclusive or'ing each byte with a fixed value, in this case 53H. Thus if you want to decipher the code in these sectors, you can recreate the original by performing an exclusive or on each byte with the value 53H.

The code on track 19 is used to load the main program which is stored on tracks 2, 3, 4 and 5. This code is also stored in a special format, since each track is written as a complete track without being subdivided into sectors. These are the tracks which will not be copied correctly with Super Utility, COPYCAT3, or TRAKCESS, but which can be copied by being edited with TRAKCESS.

To check whether these tracks have been read correctly, COPYCAT3 searches for the first occurrence of the sequence 4CH, 4EH in two consecutive bytes. The remainder of the data in the track consists of the length of the code, the load address, a checksum, and the actual code. Only the least significant four bits of each byte are used, and these are combined to form the actual data. The first byte in each pair of bytes contains the least significant 4 bits of the result, the next byte contains the most significant 4 bits.

Thus the sequence:

00 00 08 00 02 00 07 04 0F 09 01 02 00 00 00 00

found in track 2 converts to the sequence:

00 08 02 47 9F 21 00 00

which signifies a length of 800H, a load address of 4702H and a checksum for the block of 9FH. The instruction starting at 4702H is LD HL,0.

Track 5 contains a length of zero, signifying that the transfer address (4DE9H) is found in the next 2 bytes.

The tracks 0, 2, 3, 4, 5 and 19 are the only ones on the disk which contain relevant information.

3. Converting to /CMD

To perform the conversion from self-loading to /CMD format, a few changes will need to be made to the COPYCAT3 disk. These changes can easily be made with TRAKCESS or any other program which allows for modification of sectors of any size. Three modifications are required. The first two suppress the clearing of all memory when COPYCAT3 is first loaded. The third change gives a jump to a predetermined memory location after the full program has been loaded and before execution takes place.

Before COPYCAT3 is loaded, the loader will clear all of memory. Since we want to have a short program resident during the loading process, we will restrict the memory clear so the last address cleared will be 0BFFFH. The memory clear is actually done twice, and the second clear will be limited to the area ending at address 7FFFH.

Remember, these changes should only be made on a backup copy of the disk. The first two changes are on physical track 0. Load the sector with track number 0, sector number 1. Relative locations 0AH to 0CH are changed from 01 FE BB to 01 FE 7B. Rewrite the sector to disk, and load the sector with track number 69H, sector number 69H (still on physical track 0). Change relative locations 219 to 21C from BC C2 17 45 to CB 7C 28 FA, and rewrite the sector.

Now move the head to physical track 19 (decimal) and read the sector with track I.D. 30H, sector I.D. 30H. At relative locations 60H to 62H change the values 14 96 9A to 90 13 93. This is the sector which has been exclusive or'ed with 53H, so the code being inserted is actually C3 40 C0, which is a jump to the entry point of the accompanying program. Rewrite the sector to the disk.

The accompanying program can be entered and assembled with any editor/assembler, although some of the definition lines may have to be changed if anything other than EDAS is used. The object file should be assembled to disk with the name COPYCATD/CMD, or any other suitable name.

This short program consists of two parts. The first, from address C000H to C03FH, is the code which will be executed when the /CMD file is executed. It moves the COPYCAT3 program back into place, moves the screen display into place, and sets up the environment in the same way as the self-loading version. It will be moved to and executed from location A000H.

The second part of the program, beginning at location C000H is executed when the modified version of COPYCAT3 is run. It moves the program and screen memory into locations which can be dumped from DOS, clears the screen and displays a message, and then goes into a loop waiting for a reset.

The sequence for making the /CMD file is then as follows:

1. From the DOS ready prompt, issue a LOAD COPYCATD/CMD command.

2. Insert the COPYCAT3 disk as system disk and reset the computer.

3. The program should load, and the message to reset and dump should be displayed.

4. Insert the DOS system disk and reset the computer.

5. Issue the command DUMP COPYCAT3/CMD,8000H,0A03FH,0A000H or the equivalent for your operating system. The command shown is for the NEWDOS/80 system.

The COPYCAT3 program can now be executed directly from DOS simply by issuing the command COPYCAT3. If you want to reconfigure the program, you will have to repeat the sequence, but this time asking for the reconfigure option when executing step 2.

4. Further modifications

For users with disks of different sizes or differing copying requirements, it can be a nuisance that reconfiguration cannot be done with the /CMD version. It may be useful to have several distinct /CMD versions stored, each with its own configuration. It would perhaps be useful if COPYCAT displayed the selected options on the main display screen. Part of the display screen can be modified in the /CMD file if desired.

If you do modify the screen display, please do not move or change the serial number. Any tampering with the displayed serial number will result in a non-operating program. Since the information in this article is presented only for purposes of making copies for your own personal use, there should be no reason or cause to modify the serial number, and hence the information needed to change the number will not be given here.

Some of the customization could also be done directly in the /CMD file using some appropriate zap program. The information will be found at load address 8400H. The first four bytes are binary select codes for the source drive, 01H, 41H, 81H and C1H for drive 0; 02H, 42H, 82H and C2H for drive 1; 04H, 44H, 84H and C4H for drive 2; and 08H, 48H, 88H and C8H for drive 3. The next four bytes are the equivalent select codes for the destination drive.

Load address 8408H contains the step rate as a binary code with the possible values 00H, 01H, 02H and 03H.

Location 8409H contains the number of tracks on the source and destination disks, 28H for 40 tracks, 23H for 35 tracks and 50H for 80 tracks.

Locations 840CH and 840DH are the ASCII representations for the source and destination drives. The values are 30H for drive 0, 31H for drive 1, 32H for drive 2 and 33H for drive 3.

5. Plea

Since my living is made by writing and selling software I am interested in protecting the rights of software authors. As a user of software written by others, I am also annoyed by protection schemes. None of the protection schemes I have seen have resisted more than a few hours of concentrated effort, so their efficacy seems very doubtful. Perhaps the ultimate irony is that programs whose purpose is to make copies of protected programs are themselves protected, as for example Super Utility and COPYCAT.

But please, do NOT use the information presented here to make copies of the program for others. If you wish to use COPYCAT, then buy it for yourself and make a /CMD file for operating convenience only.

- Arne Rohde, Box 82-211, Highland Park, Auckland, New Zealand

```
                00100 ;
                00110 ;Extension file for COPYCAT3 to /CMD file
                00120 ;Version 1985-06-09
                00130 ;Written by arne
                00140 ;
8000            00150 PRGSAV  EQU   8000H          ;program save area
9B00            00160 ROMSAV  EQU   9B00H          ;low memory save
9C00            00170 SCRSAV  EQU   9C00H          ;save screen
A000            00180 RSTSAV  EQU   0A000H         ;save code for restore
C000            00190         ORG   0C000H         ;above cleared memory
C000            00200 RESTOR  EQU   $
C000 F3         00210         DI
C001 310000     00220         LD    SP,0           ;replaced by ptr
C002            00230 STKPTR  EQU   $-2
C004 3E01       00240         LD    A,01H          ;screen select
C006 D384       00250         OUT   (84H),A        ;screen at 3c00h
C008 AF         00260         XOR   A
C009 D3E4       00270         OUT   (0E4H),A       ;disable int
C00B 3E48       00280         LD    A,48H
C00D D3EC       00290         OUT   (0ECH),A       ;display type
C00F 010004     00300         LD    BC,1024        ;screen size
C012 11003C     00310         LD    DE,3C00H       ;screen address
C015 21009C     00320         LD    HL,SCRSAV      ;saved screen
C018 EDB0       00330         LDIR                 ;xfer data to screen
C01A 010001     00340         LD    BC,256         ;saved low mem
C01D 110000     00350         LD    DE,0           ;save from addr 0
C020 21009B     00360         LD    HL,ROMSAV
C023 EDB0       00370         LDIR
C025 01001B     00380         LD    BC,ROMSAV-PRGSAV ;saved length
C028 110040     00390         LD    DE,4000H       ;dest for code
C02B 210080     00400         LD    HL,PRGSAV      ;source
C02E EDB0       00410         LDIR
C030 010045     00420         LD    BC,0A000H-5800H ;clear mem length
C033 110058     00430         LD    DE,5800H       ;after program
C036 21FF5A     00440         LD    HL,5AFFH       ;byte = 0
C039 EDB0       00450         LDIR
C03B F1         00460         POP   AF             ;restore regs
C03C E1         00470         POP   HL
C03D D1         00480         POP   DE
C03E C1         00490         POP   BC
C03F C9         00500         RET                  ;start program
                00510 ;
C040            00520         ORG   0C040H         ;start of save sequence
C040            00530 SAVSEQ  EQU   $
C040 47         00540         LD    B,A            ;replaced instr
C041 C5         00550         PUSH  BC             ;replaced instr
C042 C5         00560         PUSH  BC             ;save regs
C043 D5         00570         PUSH  DE
C044 E5         00580         PUSH  HL
C045 F5         00590         PUSH  AF
C046 F3         00600         DI
C047 ED7302C0   00610         LD    (STKPTR),SP    ;save stack pointer
C04B 010000     00620         LD    BC,SAVSEQ-RESTOR ;len of move
C04E 1100A0     00630         LD    DE,RSTSAV      ;dest for move
C051 2100C0     00640         LD    HL,RESTOR      ;source
C054 EDB0       00650         LDIR
C056 010001     00660         LD    BC,256         ;save low memory
C059 110098     00670         LD    DE,ROMSAV
C05C 210000     00680         LD    HL,0
C05F EDB0       00690         LDIR
C061 010004     00700         LD    BC,1024        ;screen size
C064 11009C     00710         LD    DE,SCRSAV      ;save area.
C067 21003C     00720         LD    HL,3C00H       ;screen start
C06A EDB0       00730         LDIR
C06C 01001B     00740         LD    BC,ROMSAV-PRGSAV ;program length
C06F 110080     00750         LD    DE,PRGSAV      ;addr to save
C072 210040     00760         LD    HL,4000H       ;source addr
C075 EDB0       00770         LDIR
C077 21003C     00780         LD    HL,3C00H       ;screen
C07A 11013C     00790         LD    DE,3C01H
C07D 3620       00800         LD    (HL),20H       ;blank
C07F 01FF03     00810         LD    BC,1023
C082 EDB0       00820         LDIR                 ;clear screen
C084 2191C0     00830         LD    HL,DISTXT      ;text to display
C087 11003C     00840         LD    DE,3C00H
C08A 016400     00850         LD    BC,DISTXE-DISTXT
C08D EDB0       00860         LDIR
C08F 18FE       00870         JR    $              ;wait for reset
C091 52         00880 DISTXT  DEFM  'Reboot DOS system and issue
command:
    65 62 6F 6F 74 20 44 4F 53 20 73 79 73 74 65 6D
    20 61 6E 64 20 69 73 73 75 65 20 63 6F 6D 6D 61
    6E 64 3A 20 20 20 20 20 20 20 20 20 20 20 20 20
    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
C0D1 44         00890         DEFM  'DUMP program/CMD:8000H,0A03FH,0A000H'
    55 4D 50 20 70 72 6F 67 72 61 6D 2F 43 4D 44 2C
    38 30 30 30 48 2C 30 41 30 33 46 48 2C 30 41 30
    30 30 48
C0F5            00900 DISTXE  EQU   $
0000            00910         END
00000 TOTAL ERRORS

DISTXE  C0F5    DISTXT  C091    PRGSAV  8000    RESTOR  C000    ROMSAV  9B00
RSTSAV  A000    SAVSEQ  C040    SCRSAV  9C00    STKPTR  C002
```

FASTER STRING SORTS/SWAPS IN MODEL I/III BASIC

If you don't want to use a machine language string sorting routine, but can't wait ten hours for a BASIC string sort, try this: Instead of switching two string variables, switch the variable pointers instead. For example, instead of T$=A$(N): A$(N)=A$(M): A$(M)=T$ try using the following:

```
FOR L=0 TO 2: T=PEEK(VARPTR(A$(N))+L): POKE VARPTR(A$(N))+L,
    PEEK(VARPTR(A$(M))+L): POKE VARPTR(A$(M))+L,T: NEXT
```

The code is longer, BUT you avoid the dreaded "garbage collection" because you are not creating any new strings in memory - only reassigning existing strings to different variable names. It's sort of a Model I/III BASIC version of the "SWAP" command found in the Model 4 version of BASIC.

UNDERUSED BASIC PROGRAMMING STATEMENTS
by Jack Decker

This article is intended for the experienced BASIC programmer. In particular, I'm going to assume that you have some idea of how BASIC manipulates string variables, and that you understand the implications and the cause of "garbage collection" of string variables (that mysterious process that can cause your computer to "lock up" for as much as several MINUTES at a time). If you need more information on either of these points, may I respectfully refer you to Chapter 3 of my book "TRS-80 ROM Routines Documented" ($19.95 + $3.00 shipping from The Alternate Source).

There are several BASIC language statements that are available to Disk BASIC users that are underutilized by 99.5% of all BASIC language programmers. The reason is simple - these statements are usually presented in the section of the Disk BASIC manual that deals with file input and output, leading one to believe that these statements cannot be used unless file manipulation is being performed. Nothing could be further from the truth. The fact is that the proper use of some of these statements could considerably speed execution time of many programs and/or save a whole lot of BASIC code that might otherwise be needed to accomplish the same function.

The particular statements that I want to focus on in this article are FIELD, LSET, and RSET. I will also be mentioning MID$= (the MID$ function used on the left side of an equals sign) since it goes along with LSET and RSET in this application.

First, let's take the FIELD statement. FIELD is used to assign string pointers to one of the BASIC input/output buffers that is created when BASIC is first initialized. Normally, the default is three file buffers, but you can specify as few as zero or as many as 15 (or even more if you patch BASIC/CMD) when you first call up BASIC.

But suppose you aren't doing file I/O? Well, unless you bothered to tell BASIC to use zero file buffers when you first entered BASIC, those three file buffers are just sitting there, unused. By the same token, if you use less than three buffers for file I/O, the remaining buffers are unused. Each buffer has 256 bytes of free memory just sitting there, ready for you to use for whatever purpose you wish. Those buffers will stay in the same locations until you exit BASIC - unlike string variables, they can't move around in memory. So, you can store non-relocatable machine language code segments or any other combination of bytes that suits your fancy in these buffers.

The only problem is finding where the buffers are located in the first place, and that's where FIELD comes in. Simply execute a statement such as FIELD 1, 1 AS X$ and then use the VARPTR function to find the location of X$ (and thus the location of I/O buffer number 1) in memory.

Of course, the normal use of the FIELD statement is to assign string variables to the buffer. But, when you stop and think about it, all you are doing is pre-defining a string variable to reside in a specified portion of FIXED memory (the I/O buffer) rather than the normal string storage area of memory. Not only do you pre-define the location, you also pre-define the string length. So, for example, a statement such as:

FIELD 1, 30 AS A$, 20 AS B$, 10 AS C$

simply pre-defines A$ as a string 30 characters long, starting at the beginning of file I/O buffer number 1. B$ is a string 20 characters long and immediately follows string A$, and C$ is a string 10 characters long that immediately follows string B$ in memory. Note that the FIELD statement does not actually assign a "value" to any of these variables - if we were to PRINT A$ at this point, we'd get 30 "garbage" characters (whatever happened to be in those 30 bytes of memory).

Now we can't just assign a string to A$ in the normal manner, using a statement such as A$="SOME TEXT", because BASIC would create a new A$ in high memory and forget all about the "old" A$ variable pointer that we have established to the I/O buffer. So, we are forced to use the statements LSET and RSET to assign a new value to A$ without changing its location in memory. Now, please note that the use of LSET and RSET will never change the length of the variable that they are used on - in other words, once A$ has been specified as being 30 characters long, the use of LSET and RSET will never change that length. More on this later.

Let's try something that's NOT in the manual. Suppose we enter a BASIC line that looks like this:

FIELD 1, 8 AS A$: FIELD 1, 3 AS E$, 1 AS H$, 4 AS N$

Now you might think that the second FIELD statement would cancel out the first, but that's not the case. What happens is that both A$ and E$ will point to the SAME place in memory (the start of file I/O buffer number 1), and the eight characters of E$, H$, and N$ will be the same as the eight characters of A$. Suppose that we then issue the command LSET A$="555-1212" (or any other telephone number). At that point, A$ will contain the full eight digit telephone number, E$ will contain "555" (the exchange), H$ will contain "-" (the hyphen), and N$ will contain "1212" (the last four digits). Any of these variables can be printed or assigned to other variables. If E$, H$, or N$ is changed (using an LSET or RSET statement), then the corresponding portion of A$ will also be changed. This may not be very useful as it stands, but the principle can be used to extract other substrings from strings where the data to be extracted remains in the same relative position. Of course, there are other ways to break a string into substrings, but this method would be considerably faster. Furthermore, no new strings are created in the normal string storage area of memory by the statements shown above, thus forestalling the dreaded "garbage collection"!

Conceivably you could do some really fancy things by dividing up the file I/O buffers in this manner. For example, suppose that you stored a segment of machine language code in that buffer. By FIELDing the buffer properly, you could actually insert BASIC variables (after conversion by MKI$, MKS$, or MKD$ if the variable is numeric) into the proper position of the machine language segment. Thus, each time you would call the machine code segment (via the USR function), you could have already "pre-inserted" one or more variables (string or numeric) into that code segment. And, you could just as easily get one or more results back from another portion of that code segment (probably a pre-defined storage location within the code) by pointing a string variable to the proper location, then using CVI, CVS, or CVD to convert the value back to a number if necessary. No more "one variable in, one variable out" limitation.

Yet another possible use for the buffer would be as a storage location for "temporary" string variables, particularly when you want to avoid garbage collections. Actually, the uses are limited only by the imagination. Just think of the file I/O buffers as 256 byte blocks of memory that are available for your use. You can use them for the intended purpose if you choose, but you can also use them for all sorts of other purposes.

On to LSET and RSET. Basically, what these statements do is to assign a new value to a string variable, without changing either the length or the location in memory of the original variable. The only difference between the two is that if the new string being assigned to the variable is shorter than the original variable, LSET will pad the string with spaces on the right (keeping the actual text starting at the left side of the variable), while RSET does the opposite, padding with spaces on the left and keeping the actual text as far to the right as possible. If the new string is the same length or longer than the old string, it will make no difference whether LSET or RSET is used (if the new string is longer than the old, the additional characters will be chopped off the right-hand side of the string).

Normally, these statements are used on variables in the file I/O buffer, but there is no law that says that they can't be used on other variables. In some cases, it may be much more efficient to use a statement such as LSET A$=B$ as opposed to a straight A$=B$ assignment statement. The reason for this is that the latter statement will always create a new string in high memory, abandoning the "old" value of A$ (and thus hastening a "garbage collection"). However, the use of LSET or RSET does not change the length of the original string (as stated above), so BASIC just overwrites the "old" string at its present location in memory, wherever that may be. No "new" string is created.

You should use LSET or RSET whenever you are replacing strings that always have the same length. For example, if you have a database program that uses, say, eight character code strings, and you always replace one such code string with another as you get new records from the file, use LSET or RSET to make the replacement so that you don't leave a "garbage" string behind in memory. In addition, you can use LSET or RSET in any application where you want a string to remain a predetermined length. For example, suppose you're using a printer to create a table. You might use a statement such as:

A$=STRING$(20,0):B$=A$:C$=A$:D$=A$

This would create four strings, each 20 bytes in length, in the string storage area of memory. Now, if we were printing four strings on each line, we could use LSET or RSET (depending on the type of data being printed) to assign the four strings to be

printed to A$, B$, C$, and D$, knowing that the length of each of these variables would not change (they would be padded with spaces to keep the length constant, if necessary). Then we would LPRINT A$;B$;C$;D$ to print the entire line at once. Again, this may not be the best example (perhaps the TAB function or PRINT USING might serve the purpose better in a given situation), but in certain situations this technique could be invaluable.

And now, a brief cameo appearance from MID$=. This function, when used on the LEFT side of an equation, replaces a portion of a string with another string. But it has much in common with LSET and RSET - when it is used, the length of the original string is not changed, the string is not moved in memory, and no new strings are created in the string storage area. As a matter of fact, MID$ should be thought of as an alternative to LSET and RSET in certain situations. For example, here are two ways to accomplish the same thing, namely, replacing the middle three characters of the eight-character string A$ (which is assumed to be the first eight characters in file I/O buffer number 1) with "XYZ":

```
FIELD 1, 8 AS A$: FIELD 1, 2 AS X$, 3 AS X$: LSET X$="XYZ"
FIELD 1, 8 AS A$: MID$(A$,3)="XYZ"
```

Note that in both cases, the initial FIELD statement would have probably been executed earlier in an actual program, it is just here for illustrative purposes. Also note that neither of these statements creates an additional string in high memory.

You might be wondering about the use of X$ twice within one FIELD statement in the first example line above. What happens in a case like this is that the second assignment of X$ replaces the first. Thus, the first assignment of X$ (2 AS A$) "counts off" the two characters that we are not interested in. Then, we re-assign X$ to the following three characters (3 AS A$), which happens to be the middle three characters that we are interested in. We could have used a "dummy" variable in place of the first assignment of X$, but that would have created an unnecessary three-byte string variable pointer in memory (which could conceivably slow down the program a bit).

Anyway, read up on MID$ in your Disk BASIC manual, then keep in mind that it can be used just like LSET or RSET. In fact, you should think of LSET, RSET, and MID$= as three statements that perform very similar functions. Most Disk BASIC manuals put MID$= in their "BASIC enhancements" section and then put LSET and RSET in the "file I/O" section. This doesn't make sense, since the three statements are all essentially variations on the same theme.

One more note about the use of LSET/RSET/MID$=. They operate much faster than a normal string assignment statement, since no new string needs to be created in memory. And, because they do not change the location of the string on which they operate, they can be useful in putting strings (of text or graphics) on the video display of a Model I or III (or 4 in the Model III mode). How? By first redefining a string pointer to point to the video display - that is, the string variable is actually stored in the video display memory itself! To do this, you have to create a dummy string variable, then go in and change the VARPTR to point to the video memory location you desire.

Here's an example: I wrote a program that read lines from a file, processed each line, and wrote the processed line to a second file. I wanted a visual indication of how many lines had been processed at any given time, so I included the statements

```
CLS: V$="LINE 00000": POKE VARPTR(V$)+1,54: POKE VARPTR(V$)+2,60
```

at the start of the program. V$ was first defined to be a string of ten characters in length (the actual contents of V$ were irrelevant at that point, I could have just as easily used V$="1234567890" or even V$=STRING$(10,0)). Next, the POKEs were used to redefine the storage location of the string variable as the last ten characters of the top line of the video display (starting at location 15414 decimal, 3C36 hexadecimal). Then, every time I wished to update the line count (which was stored in the numeric variable N), I simply executed the statements

```
N=N+1: RSET V$="LINE"+STR$(N)
```

The RSET statement would dutifully store the new contents of string V$ (the word "LINE" and the line number, preceded by initial spaces to make a ten-character-long variable) in its "old" location in memory - which just happened to be the upper right hand corner of the video display. Furthermore, this did not upset the current cursor location or overwrite any other text on the screen (a requirement of the program since certain error messages could be printed on the video display during a program run, and I didn't want to lose any of these).

I have heard of BASIC programmers using similar techniques to draw graphics on the video display at nearly machine-language speed. The thing to remember is that LSET, RSET, and MID$= can be used on strings anywhere in memory, even if they are not in the "normal" string storage areas. You could define a string literal within a BASIC program line, then use LSET, RSET, or MID$= to change the string within the program itself (self-modifying programs, anyone?). Your string could be in "protected" memory, or in a file I/O buffer, or somewhere in your memory-resident DOS code (how about changing DOS text strings "on the fly"), or just about anywhere else in memory, provided you know what you're doing. The mind boggles at the possibilities.

In closing, I will just mention that the MKI$, MKS$, MKD$, CVI, CVS, and CVD functions are also normally associated with file management, yet they could potentially be useful in non-file applications (I can't think of any offhand, but that doesn't mean that there aren't any). Just because a BASIC statement is found in the "file manipulation" section of your Disk BASIC manual doesn't necessarily mean that it can only be used in file-handling applications. Indeed, the advanced BASIC programmer can often make programs run faster and use less memory by utilizing Disk BASIC statements in unconventional ways!

ZAP FOR ALLWRITE! TEXT EDITOR

by Jack Decker

Allwrite is a great word processing program, but it has one design "feature" that I have always found irritating. That is that when you are entering text and place two spaces after a word, and then the next word you type exceeds the end of video line limit and is "wrapped around" to the start of the next line, one of your two spaces will be dropped. The only exception is when the two spaces follow a period, in which case both will be retained, presumably because a period ends a sentence and many people like to leave two spaces between sentences.

Unfortunately, some of we riters (?!) use other punctuation marks to end a sentence, such as question marks, exclamation points, etc. I like to use two spaces after each. Besides, if I type in two spaces it's because I want two spaces, and I don't appreciate Allwrite second-guessing me and removing one of them.

Here are some zaps that make Allwrite treat all characters as it formerly treated periods for word wrap purposes - that is, if you type two spaces after a character, you'll get two spaces. This won't work properly for more than two spaces (you may wind up with only one space if you insist on trying to use three or more spaces in a row, especially if you save the text to disk and then reload it), but if you need more than two spaces in a row, you can always use "hard spaces". PLEASE NOTE that these zaps should be considered experimental. I do not guarantee that they won't cause any undesirable side effects, which is another way of saying that if you install them and they eat portions of your text files, don't blame me!

The zaps shown below insert relative jump instructions that skip the test for a period and the code that is normally executed when something other than a period is encountered. The Model I/III and Model 4 zaps are the same, but the location of the zaps differs. Note that the locations given are for version 1.12 of Allwrite, if you have another version you may need to search for the "old" byte sequence before you can make the replacement. Here are the zaps to AL/CMD:

```
MI/III FRS 20,82 - M4 FRS 25,9A: change 04 FE 2E 20    to 04 18 07 20
MI/III FRS 21,84 - M4 FRS 26,CC: change 28 7E FE 2E 20 to 28 08 18 F3 20
MI/III FRS 23,64 - M4 FRS 28,7C: change 17 28 3E 2E    to 17 18 05 2E
MI/III FRS 28,0F - M4 FRS 33,27: change 28 3E 2E BE    to 28 18 03 BE
```

HELP WANTED - CAN YOU HELP?

Has anyone ever attempted to add more than two external floppy disk drives (more than four drives total) to a Model 4 or If so, George Matyaszek (1718 North Long Avenue, Chicago, Illinois 60639) would like to hear from you. George runs a BBS and has some extra floppy drives that he wants to attach to his Model 4, but has the normal maximum of two external drives connected already. Surely there must be some way to attach more external drives???

USE OF CMD"J" - NEWDOS/80 VERSION 2.

by Alf West

[This is reprinted from BITS & BYTES, issue #25, which appears in the TRS-80 SYSTEM 80 Computer Group (Queensland, Australia) club newsletter. Note that the references to "our language" in this article refer to the King's (Queen's?) English as is spoken in Australia.]

Converting dates to the number of days in the year for, say, interest calculations may be old hat to our accountant members, but for those not so well qualified, the use of CMD"J" in NEWDOS/80 is very handy in certain programs.

Page 7-10 [of the NEWDOS/80 manual] briefly explains this facility, but glosses over how it may be used. It gives the command as:

CMD"J",date1,date2

where date1 & date2 are strings. It might be better understood if we re-wrote the command as:

CMD"J",Input$,Answer$

in which you provide the "Input$" and the computer comes up with the "Answer$".

In simple terms, if you make the "Input$" a date, then the computer will give "Answer$" as the number of days that date is from the beginning of the year. Conversely if you make the "Input$" a number of days, then the computer will give you an "Answer$" which is the date counting those number of days from the beginning of the year.

BUT you must obey certain rules on the syntax of "Input$". If the "Input$" is a date, it must conform to the American convention of putting the month before the day, i.e. the "Input$" must be in the format of "MM/DD/YY".

If the "Input$" is the number of days, you must refer to the year you are talking about and in this case the format (rather odd) is "-YY/DDD". That is, it must be a 7 character string starting with a minus sign, the last two digits of the year, a "/" followed by 3 digits for the number of days. The following examples will illustrate these points:

```
10 INPUT$ = "06/30/84" - 30th. June '84 in our language
20 CMD"J",INPUT$,ANSWER$
30 PRINT ANSWER$
```

You'll get the answer of "182" being the number of days from 1st. January to 30th. June. Now put a different date for "Input$" and subtract the two "Answer$" and you have the number of days interval between the two dates.

If you input the number of days, the example would be:

```
10 INPUT$ = "-84/182"
20 CMD"J",INPUT$,ANSWER$
30 PRINT ANSWER$
```

You will get the answer "06/30/84" or in our language 30th. June '84.

The command CMD"J" is very simple to use in a program if the dates or the number of days fall within the one year from 1st. January to 31st. December. The programming becomes a little more involved if the time intervals span a number of years. The following program illustrates the use of CMD"J" in a little utility to:

a) Give the number of days between any two given dates

b) Give the end date after a given number of days from a starting date.

The program is "padded" a bit to ensure correct inputs and also illustrates the use of DEFFN, used to put the "Input$" in the proper format where the conversion of a number to a string can be a trap because of the leading blank when using STR$ and concantenating. Although this program is for Disk BASIC only, the use of the space compression codes CHR$(192) to (255) for ease in fiddling around with screen layouts could be of interest to Level II (and Disk) owners.

[NORTHERN BYTES editor's note: Line 360 in the listing below started with "IF N<J3 ..." in the original article, however, I found that this would cause errors if, for example, a starting date of January 1, 1984 (or 1 January 1984, to use the program's syntax) was given and then the program was asked to calculate the date when the exact number of days remaining in the year (365) was added. The change to "IF N<= J3 ..." seems to cure this. I do not represent the program listing below as being bug free, since we had to retype it to get it into NORTHERN BYTES, and who knows what unknown typos may still be lurking...!]

```
10 CLS: CLEAR 500
15 DEFFNF1$(A$)="0"+A$: DEFFNF2$(A$)="00"+A$
16 DEFFNF3$(A)=MID$(STR$(A),2,2)
17 DEFFNF4$(A)=MID$(STR$(A),3,2)
18 E$="12/31/"
20 Z$="* A DEMONSTRATION OF THE USE OF
CMD"+CHR$(34)+"J"+CHR$(34)+" *":PRINTCHR$(200)Z$
30 PRINT: PRINT"THE NUMBER OF DAYS BETWEEN TWO DATES":
PRINT: PRINT
40 PRINT"ENTER THE FIRST DATE...": GOSUB 50: GOTO 80
50 V=350: GOSUB 900: IF K=1 THEN K=0: GOTO 50
60 GOSUB 910: IF K=1 THEN K=0: GOTO 60
70 GOSUB 920: IF K=1 THEN K=0: GOTO 70
75 RETURN
80 Y1=Y: GOSUB 1150: D1$=D$: PRINT@V,CHR$(30)CHR$(210)B$
90 PRINT: PRINT"ENTER THE SECOND DATE ..."
100 V=478: GOSUB 900: IF K=1 THEN K=0: GOTO 100
110 GOSUB 910: IF K=1 THEN K=0: GOTO 110
120 GOSUB 920: IF K=1 THEN K=0: GOTO 120
130 Y2=Y: GOSUB 1150: D2$=D$: PRINT@V,CHR$(30)CHR$(210)B$
140 IF Y1=Y2 THEN 220
150 IF Y2>Y1 THEN DG$=D2$: DL$=D1$: YG=Y2: YL=Y1: GOTO 170
160 DG$=D1$: DL$=D2$: YG=Y1: YL=Y2
170 CMD"J",DL$,J$: J1=VAL(J$): CMD"J",DG$,J$: J2=VAL(J$):
J3=0
190 FOR T=YL TO YG-1: T$=FNF3$(T): DM$=E$+T$
200 CMD"J",DM$,J$: J=VAL(J$): J3=J3+J: NEXT
210 N=J3-J1+J2: GOTO 250
220 CMD"J",D1$,J$: J1=VAL(J$): CMD"J",D2$,J$: J2=VAL(J$):
N=ABS(J1-J2)
250 PRINT: PRINT"THE NUMBER OF DAYS BETWEEN THESE
DATES IS ..." CHR$(198)N
260 PRINT@960,"ENTER < A > FOR ANOTHER CALCULATION  < B >
FOR SECOND DEMO.";
270 GOSUB 500: IF A$="A" THEN PRINT@320,CHR$(31);: GOTO 40
280 IF A$="B" THEN 300 ELSE 270
300 CLS: PRINTCHR$(200)Z$: PRINT: PRINT"NEW DATE AFTER A
NUMBER OF DAYS FROM STARTING DATE": PRINT: PRINT
310 PRINT"ENTER STARTING DATE ...": GOSUB 50
320 GOSUB 1150: PRINT@V,CHR$(30)CHR$(210)B$
330 PRINT: INPUT"ENTER THE NO. OF DAYS FROM THIS DATE :-
";N
335 IF N>36500 THEN GOSUB 2000: PRINT@384,;: GOTO 330
340 CMD"J",D$,J$: J1=VAL(J$)
350 DM$=E$+Y$: CMD"J",DM$,J$: J2=VAL(J$): J3=J2-J1
360 IF N<=J3 THEN JF=J1+N: GOTO 400
365 N=N-J3
370 Y=Y+1: IF Y>99 THEN P=1: Y$=FNF4$(Y) ELSE Y$=FNF3$(Y)
380 DM$=E$+Y$: CMD"J",DM$,J$: J2=VAL(J$): IFJ2<N THEN N=N-
J2: GOTO370
390 JF=N
400 J$=STR$(JF): J$=MID$(J$,2)
410 IF LEN(J$)=1 THEN J$=FNF2$(J$) ELSE IF LEN(J$)=2 THEN
J$=FNF1$(J$)
420 J$="-"+Y$+"/"+J$: CMD"J",J$,DE$
430 DE$=MID$(DE$,4,3)+LEFT$(DE$,2)+RIGHT$(DE$,3)
440 PRINT: PRINT"THE END DATE IS ....."CHR$(216)DE$
445 IF P=1 THEN PRINT: PRINT"THIS ANSWER IS IN THE NEXT
CENTURY & IS SLIGHTLY SUSPECT BEING": PRINT"OUTSIDE THE
RANGE 1900 TO 1999":P=0
450 PRINT@960,"ENTER < A > FOR ANOTHER CALCULATION  < Q >
TO QUIT PROGRAM";
460 GOSUB 500: IF A$="A" THEN PRINT@320,CHR$(31);: GOTO 310
470 IFA$="Q" THEN CLS: END ELSE 460
500 A$=INKEY$: IF A$="" THEN 500 ELSE RETURN
530 PRINTDE$: END
900 PRINT@V,CHR$(30)"THE DAY"CHR$(195);: INPUTD$: GOSUB
1000: RETURN
910 PRINT@V,CHR$(30)"THE MONTH"CHR$(193);: INPUTM$: GOSUB
1100: RETURN
920 PRINT@V,CHR$(30)"THE YEAR"CHR$(194);: INPUTY$: GOSUB
1130:RETURN
1000 IF LEN(D$)>2 THEN K=1: GOTO 2000
1005 D=VAL(D$):IF D<1 OR D>31 THEN K=1: GOTO 2000
1010 IF LEN(D$)<2 THEN D$=FNF1$(D$)
1020 RETURN
1100 IF LEN(M$)>2 THEN K=1: GOTO 2000
1105 M=VAL(M$): IF M<1 OR M>12 THEN K=1: GOTO 2000
1110 IF LEN(M$)<2 THEN M$=FNF1$(M$)
1120 RETURN
1130 Y$=RIGHT$(Y$,2): Y=VAL(Y$)
1140 Y=VAL(Y$): IF Y<0 OR Y>99 THEN K=1: GOTO 2000 ELSE
RETURN
```

```
1150 B$=D$+"/"+M$+"/"+Y$: D$=M$+"/"+D$+"/"+Y$: RETURN
2000 PRINT@960,CHR$(205)"**** THAT ENTRY IS INVALID ****";:
FOR T=1 TO 1500: NEXT: PRINT@960,CHR$(30);: RETURN
```

* * * * * * * * * * *

ZAPS FOR CMD"J"
by Kevin O'Hare

[This article is a follow-up to the above article and is also reprinted from the TRS-80 SYSTEM 80 Computer Group (Queensland, Australia) club newsletter. Kevin O'Hare is the author of the TRS-80 HACKER'S HANDBOOK ($24.95 plus $3.00 shipping from The Alternate Source).]

In issue 25, on the use of CMD"J", Alf West referred to the necessity of having the required syntax of putting the Month before the Day and his program had quite a bit of string manipulation so that the input or the answer could be given in "our language". I have yet to find a rational reason for this American abberation. This is a Zap to BASIC/CMD of NEWDOS/80 version 2 which would eliminate the need for string manipulation in a program and reduces the hassle of remembering to put the month before the day in Date entry.

The conversion of the month of the year to the day of the month requires only 5 bytes changed but the reverse is more complicated. The following are the Zaps to BASIC/CMD Sector 3 for the Month/Day conversion.

Mod I	Mod III	Change	To	Comments
A3	7C	0C	1F	; change Month limit to Day limit
A8	81	F5	D5	; store 'E', not 'A'
AD	86	1F	0C	; change Day/Month
B3	8C	3D	0D	; decr C (days) not A
B4	8D	4F	47	; store Months in B

Now the complicated one. The Day/Month conversion requires nearly the same number of byte changes but the order of some bytes have to be relocated to allow extra bytes. Where the byte value (or the address of the disassembly) of the Model III differs from that of the Model I, the Model III value follows the slash. The changes to be made in Sectors 4/3 of BASIC/CMD are as follows, the Model III value where different being underlined:

Mod I	Mod III		Changes						
4,25	3,FA	From	1B	7D	FE	AE/87	28	DB	
		To	1B	EB	7B	FE	AE/87	28	
4,28	4,00	From	4E	EB	B7	ED	42	EB	23	30
		To	DA	1A	4F	B7	ED	42	13	30
4,23	4,08	From	F2	EB	09	E5	7B	D6	A2/7B	5F
		To	F3	09	7B	D6	A2/7B	5F	54	D5
4,38	4,10	From	54	CD	32/98	5D	2A	54/2D	5D	22
		To	23	EB	CD	32/98	5D	2A	54/2D	5D
4,33	4,18	From	AE/87	(01)	(00)	(25)	(3D)	5B	D1	13
		To	22 (Mod 3 (only) load bytes)AE/87					5B	D1	13

A disassembly follows with the Sector Address and the actual address for both the Models I & III:

M1	Addr	M3	Addr	Old Code	Disassembly	New code where Different
91	5A9D	6A	5A76	CF	RST 08H	
92	5A9E	6B	5A77	2C	DEFB ','	
93	5A9F	6C	5A78	CD B7 61	CALL 61B7/6190H	
96	5AA2	6F	5A7B	20 6A	JR NZ,5B0E/5AE7H	
98	5AA4	71	5A7D	7E	LD A,(HL)	
99	5AA5	72	5A7E	FE 2D	CP '-'	; minus
9B	5AA7	74	5A80	78	LD A,B	
9C	5AA8	75	5A81	28 66	JR Z,5B10/5AE9H	
9E	5AAA	77	5A83	FE 08	CP 8	
A0	5AAC	79	5A85	38 60	JR C,5B0E/5AE7H	
A2	5AAE	7B	5A87	01 0C 01	LD BC,010CH	01 1F 01 LD BC,011FH
A5	5AB1	7E	5A8A	CD 6E 5B	CALL 5B6E/5B47H	
A8	5AB4	81	5A8D	F5	PUSH AF	D5 PUSH DE
A9	5AB5	82	5A8E	CD 66 5B	CALL 5B66/5B3FH	
AC	5AB8	85	5A91	01 1F 01	LD BC,011FH	01 0C 01 LD BC,010CH
AF	5ABB	88	5A94	CD 6E 5B	CALL 5B6E/5B47H	
B2	5ABE	8B	5A97	C1	POP BC	
B3	5ABF	8C	5A98	3D	DEC A	0D DEC C
B4	5AC0	8D	5A99	4F	LD C,A	47 LD B,A
B5	5AC1	8E	5A9A	C5	PUSH BC	
B6	5AC2	8F	5A9B	CD 66 5B	CALL 5B66/5B3FH	
B9	5AC5	92	5A9E	CD 80 5B	CALL 5B80/5B59H	
BC	5AC8	95	5AA1	21 A1 5B	LD HL,5BA1/5B7AH	
BF	5ACB	98	5AA4	11 00 00	LD DE,0000H	
C2	5ACE	9B	5AA7	C1	POP BC	
C3	5ACF	9C	5AA8	7E	LD A,(HL)	
C4	5AD0	9D	5AA9	83	ADD A,E	
C5	5AD1	9E	5AAA	5F	LD E,A	
C6	5AD2	9F	5AAB	23	INC HL	
C7	5AD3	A0	5AAC	30 01	JR NC,5AD6H	
C9	5AD5	A2	5AAE	14	INC D	
CA	5AD6	A3	5AAF	10 F7	DJNZ 5AC7H	
CC	5AD8	A5	5AB1	79	LD A,C	
CD	5AD9	A6	5AB2	BE	CP (HL)	
CE	5ADA	A7	5AB3	30 32	JR NC,5B0E/5AE7H	
D0	5ADC	A9	5AB5	EB	EX DE,HL	
D1	5ADD	AA	5AB6	03	INC BC	
D2	5ADE	AB	5AB7	09	ADD HL,BC	
D3	5ADF	AC	5AB8	EB	EX DE,HL	
D4	5AE0	AD	5AB9	CD 32 5D	CALL 5D32/5D08H	
D7	5AE3	B0	5ABC	21 53 5D	LD HL,5D53/5D2DH	
DA	5AE6	B3	5ABF	11 AE 5B	LD DE,5BAE/5B87H	
DD	5AE9	B6	5AC2	01 03 00	LD BC,3	
E0	5AEC	B9	5AC5	ED B0	LDIR	
E2	5AEE	BB	5AC7	3E 03	LD A,3	
E4	5AF0	BD	5AC9	E1	POP HL	
E5	5AF1	BE	5ACA	F5	PUSH AF	
E6	5AF2	BF	5ACB	CF	RST 08H	
E7	5AF3	C0	5ACC	2C	DEFB ','	
EB	5AF4	C1	5ACD	CD 0D 26	CALL 260DH	
EB	5AF7	C4	5AD0	CD F4 0A	CALL 0AF4H	
EE	5AFA	C7	5AD3	F1	POP AF	
EF	5AFB	C8	5AD4	E5	PUSH HL	
F0	5AFC	C9	5AD5	D5	PUSH DE	
F1	5AFD	CA	5AD6	CD BF 28	CALL 28BFH	
F4	5B00	CD	5AD9	E1	POP HL	
F5	5B01	CE	5ADA	CD 5D 28	CALL 285DH	
F8	5B04	D1	5ADD	21 AE 5B	LD HL,5BAE/5B87H	
FB	5B07	D4	5AE0	4F	LD C,A	
FC	5B08	D5	5AE1	06 00	LD B,0	
FE	5B0A	D7	5AE3	ED B0	LDIR	
00	5B0C	D9	5AE5	E1	POP HL	
01	5B0D	DA	5AE6	C9	RET	
02	5B0E	DB	5AE7	18 5B	JR 5B6B/5B44H	
04	5B10	DD	5AE9	FE 07	CP 7	
06	5B12	DF	5AEB	38 FA	JR C,5B0E/5AE7H	
08	5B14	E1	5AED	23	INC HL	
09	5B15	E2	5AEE	5E	LD E,(HL)	; ASCII of Year
0A	5B16	E3	5AEF	23	INC HL	; as entered
0B	5B17	E4	5AF0	56	LD D,(HL)	; by user into
0C	5B18	E5	5AF1	ED 53 04 5B	LD (5B04/5B0DH),DE	; storage area
10	5B1C	E9	5AF5	2B	DEC HL	
11	5B1D	EA	5AF6	CD 80 5B	CALL 5B80/5B59H	
14	5B20	ED	5AF9	CD 66 5B	CALL 5B66/5B3FH	; go check for slash
17	5B23	F0	5AFC	06 03	LD B,3	
19	5B25	F2	5AFE	CD 95 5B	CALL 5B95/5B6EH	
1C	5B28	F5	5B01	21 A2 5B	LD HL,5BA2/5B7BH	; Table of Days in Months
1F	5B2B	F8	5B04	06 00	LD B,0	
21	5B2D	FA	5B06	1B	DEC DE	
22	5B2E	FB	5B07	7D	LD A,L	EB EX DE,HL
23	5B2F	FC	5B08	FE AE	CP 0AEH	7B LD A,E
25	5B31	FE	5B0A	28 DB	JR Z,5B0E/5AE7H	FE AE CP 0AEH
27	5B33	00	5B0C	4E	LD C,(HL)	28 DA JR Z,5B0EH
28	5B34	01	5B0D	EB	EX DE,HL	1A LD A,(DE)
29	5B35	02	5B0E	B7	OR A	4F LD C,A
2A	5B36	03	5B0F	ED 42	SBC HL,BC	B7 OR A
2C	5B38	05	5B11	EB	EX DE,HL	ED 42 SBC HL,BC
2D	5B39	06	5B12	23	INC HL	13 INC DE
2E	5B3A	07	5B13	30 F2	JR NC,5B2E/5B07H	30 F3 JR NC,5B2FH
30	5B3C	09	5B15	EB	EX DE,HL	09 ADD HL,BC
31	5B3D	0A	5B16	09	ADD HL,BC	7B LD A,E
32	5B3E	0B	5B17	E5	PUSH HL	D6 A2 SUB 0A2H
33	5B3F	0C	5B18	7B	LD A,E	5F LD E,A
34	5B40	0D	5B19	D6 A2	SUB 0A2H	54 LD D,H
36	5B42	0F	5B1B	5F	LD E,A	D5 PUSH DE
37	5B43	10	5B1C	54	LD D,H	23 INC HL
38	5B44	11	5B1D	CD 32 5D	CALL 5D32/5D08H	EB EX DE,HL
3B	5B47	14	5B20	2A 54 5D	LD HL,(5D54/5D2DH)	CD 32 5D CALL 5D32/5D08H
3E	5B4A	17	5B23	22 AE 5B	LD (5BAE/5B87H),HL	2A 54 5D LD HL,(5D54/2D)
	Sector 4 bytes 19 to 1C contain control bytes in Model III (only)					
41	5B4D	1E	5B26	D1	POP DE	22 AE 5B LD (5BAE/87),HL
42	5B4E	1F	5B27	13	INC DE	D1 POP DE

```
43 5B4F 20 5B28 CD 32 5D    CALL 5D32/5D0BH
46 5B52 23 5B28 21 80 5B    LD   HL,5B80/5B89H
49 5B55 26 5B2E 3E 2F       LD   A,2FH          ; '/' separator
4B 5B57 28 5B30 77          LD   (HL),A         ; 5D32/5D0BH converts
4C 5B58 29 5B31 23          INC  HL             ; No days to ASCII
4D 5B59 2A 5B32 ED 5B 54 5D LD   DE,(5D54/5D2DH) ; and stores it
51 5B5D 2E 5B36 73          LD   (HL),E         ; then transferred to
52 5B5E 2F 5B37 23          INC  HL             ; buffer ready to
53 5B5F 30 5B38 72          LD   (HL),D         ; print.
54 5B60 31 5B39 23          INC  HL
55 5B61 32 5B3A 77          LD   (HL),A
56 5B62 33 5B3B 3E 00       LD   A,0
58 5B64 35 5B3D 18 8A       JR   5AF0/5AC9H     ; Year print
```

[NORTHERN BYTES EDITOR'S NOTE: The above listing differs slightly from the original listing in the TRS-80 System 80 Group newsletter. The reason is that I didn't want to have to re-type the entire listing, so I used a combination of techniques to enable me to generate the above listing from the (original and patched) code in memory. By doing it this way, I discovered a small number of errors in the original listing. Hopefully all of the bugs have been exterminated in the listing printed here!]

The routine at 5B66/5B3FH just checks for a slash. The 5D32/5D0BH routine converts a value in the Accumulator to ASCII and stores it in 5D54/5D2DH. 5B95/5B6EH calls the 5CFE/5CD7H routine which converts what HL is looking at to 3 characters of decimal. 5B6E/5B47H gets decimal characters entered into 'A'. 5B80/5B59H is the year routine.

* * * * * * * * * *

MODEL 4 LDOS/TRSDOS 6.2 DATE PATCH
by Larry Lewis & Rowan Evans

[This patch is excerpted (mostly for the benefit of our readers outside the U.S.A.) from the column, "The Prophet & Oracle Speak", which appears in SYDTRUG NEWS, the newsletter of the Sydney TRS-80 Users Group (P.O. Box 297, Padstow, N.S.W. 2211, Australia).]

All you Model 4 users, have you got version 6.2 from Tandy yet? No! Go get it because I have a patch to change the input of dates from MM/DD/YY to DD/MM/YY (NOTE: the internal format is unchanged).

Patch SYS0/SYS with:

```
D0D,3B=70 2B 77
D0D,6C=00
D0D,78=13
D0F,BE=44 44 2F 4D 4D
```

Patch SYS7/SYS with:

```
D05,4A=13 62 6B 4E 23 7E 12 71 1B
```

CONVERTING A CALL TO JR

[Reprinted from the TRS-80 SYSTEM 80 Computer Group newsletter (16 Laver Street, MacGregor, Queensland 4109, Australia).]

A commonly used method of interfacing a small machine routine with a Basic program is to imbed the routine in a String in the Basic program. This has the advantages that the M/L routine is part and parcel of the Basic program and does not involve the separate loading of a USR, and also that you are not worried about protecting the USR by changing Memory Size. BUT the machine routine in the String must be completely relocatable, i.e. it must contain no JUMPs or CALLs to absolute addresses in RAM, because, sure as eggs, somebody will add or delete something from the Basic program and that part of the routine containing the CALL (or JUMP) will no longer be at the place in Memory where it was originally. (JUMPs or CALLs to places in ROM are O.K. as ROM doesn't change.)

It is not too difficult to arrange the program of the routine to use Jump Relative (JR). The value of the displacement may be from -126 to +129. However, although it would be very convenient to use CALLs within the routine, it may involve quite a bit of fiddling to get around a CALL to an address within the routine.

Now there are two bytes near the start of ROM at 000BH which are E1H and E9H which in Assembly Language are:

```
E1H  =   POP   HL
E9H  =   JP    (HL)
```

This is what happens if your routine made a CALL 000BH. Firstly, the address of the instruction in your routine immediately following that CALL is placed on the Stack and control goes to 000BH. Here, the address which has just been placed on the stack is transferred to the HL Register and the following Jump to the contents of HL sends you right back to your routine at the instruction after the CALL.

You might say "So what!". Well the HL Register now holds the same value as the Program Counter - the address where you now are, in your routine. Make the instruction at this point a JR to the subroutine you would otherwise have CALLed. Preface the subroutine with an increment HL twice, followed by a PUSH HL. There are 2 bytes in a JR instruction, and now HL contains the address to where you will want to return after performing the subroutine. That address is now on the Stack and you will return there with a RET at the end of your subroutine. Figure 1 illustrates the source listing involved.

All this uses five more bytes than a regular CALL to a subroutine. It converts a CALL into a JR making the routine truly relocatable without losing the power and convenience of a CALL instruction.

Sometimes you may want to use whatever was in HL in the subroutine, but the above two increments will have changed its value. A convenient way around this is to PUSH HL before the CALL 000BH, and in the subroutine, instead of PUSH HL, use the instruction EX(SP),HL. This is illustrated in Figure 2.

With the use of CALL 000BH to convert your subroutine CALLs to JRs, you may make all your machine language routines relocatable and have no problems in imbedding the routines in a Basic program.

```
Program Main Line
        .
        .
        .
        CALL   000BH    ;Shift address of next instruction
                        ;to HL pair
        JR     SUB1     ;Relative jump to subroutine
        .               ;Return here after subroutine
        .
        .
        .
SUB1    INC    HL       ;This is the CALLed subroutine
        INC    HL       ;HL+2 ==> return address
        PUSH   HL       ;Put return address on stack
        .               ;Now perform subroutine processing
        .
        .
        RET             ;end subroutine with RETurn
```

Fig. 1 Value of HL changed

```
Program Main Line
        .
        .
        PUSH   HL       ;Save HL value on stack
        CALL   000BH    ;Shift address of next instruction
                        ;to HL pair
        JR     SUB1     ;Relative jump to subroutine
        .               ;Return here after subroutine
        .
        .
        .
SUB1    INC    HL       ;This is the CALLed subroutine
        INC    HL       ;HL+2 ==> return address
        EX     (SP),HL  ;Put return address on stack
        .               ;and recover HL value
        .               ;Now perform subroutine processing
        .
        RET             ;end subroutine with RETurn
```

Fig. 2 Retains value of HL

TWO JCL FILES FOR LDOS USERS
by Gary Bryce

[This article is reprinted from the SYDTRUG NEWS, P.O. Box 297, Padstow, New South Wales 2211, AUSTRALIA.]

The first JCL (MIRROR/JCL) presented here forms the solution to a problem which has existed for some time. Those of you using double density/sided drives on the Model I are quite familiar with the fact that BACKUP refuses to create a "MIRROR IMAGE" of the original diskette (giving a DATA SECTOR NOT FOUND DURING READ error).

When BACKUP is required to do a mirror-image copy it calculates the number of sectors per cylinder. With a double sided, double density disk there are 36 sectors per cylinder. The SOLEd disk has 36 sectors per cylinder on every cylinder EXCEPT the first (0), where there are only 10 on side 0 (0 to 9 in single density) and 18 on side 1 (0 to 17 in double density).

The SECTOR NOT FOUND error happens because BACKUP has computed that there are 36 sectors per cylinder on the disk, and when sector 10 of cylinder 0 is passed to the driver (PDUBL or RDUBL), the driver has already switched to single density and has automatically updated the Drive Control Table (DCT) to show that sector 9 is the highest numbered sector on this side. Sector 10 is readdressed to sector 0 of side 1 - BUT IN SINGLE DENSITY! When the driver tries to read this sector, it gets a record not found error because side 1 is in double density. The driver then switches density to double (part of ADR) which now updates the (DCT) to show 17 as the highest numbered sector; it now invokes the SEEK routine which calculates that sector 10 is on side 0! Therefore, the second attempt still results in a sector not found error. The driver switches to single density and repeats the process until it times out after ten retries. Sectors 10 to 17 of cylinder 0 cannot be backed up because they do not exist!

As you can see, the main cause of the problem is that BACKUP was not designed to deal with a dual density diskette. This can be avoided by locking out cylinder 0 (effectively what is done in NEWDOS/80 with T1=CK). BACKUP bypasses the locked out cylinder and proceeds with a normal "MIRROR IMAGE" backup. The only remaining problem is to update the information about the default drive types and the state of the SYSTEM (SYSGEN) configuration parameter, all of which is stored within the file BOOT/SYS, which is of course on cylinder 0. The correct method to update this information is to backup SYS0/SYS from the source to destination diskette; the required information is transferred from source to destination in the process.

Rather than patch SOLE1/CMD (as there seems to be a couple of revisions of the programme), the JCL sets all bits in the cylinder lockout table for cylinder 0, by "PATCHING" the directory. As the PATCH utility rewrites the sector with a standard data address mark, the REPAIR utility is used to correct it to the one normally used for the directory.

```
%1F.      Mirror Image Format & Backup
.          by Gary Bryce
.             10 Leyte Ave,
.             Lethbridge Park,
.             N.S.W.  2770      : Phone (02)628-5058
.
.This JCL will create a "MIRROR IMAGE" backup of a Model 1
.Double density, Double Sided, Bootable, System disk.
.Execution parameters of this JCL are :-
.    SD = Source Drive      DD = Destination Drive
.    N  = Diskname          C  = Cylinder count
.Defaults are :- SD = 0, DD = 1, N = MYDISK, C = 80
.
.Example :- DO MIRROR (SD=1,DD=2,N=LDOSYSTM,C=40)
.
//IF -SD
//ASSIGN SD=0
//END
//IF -D2
//ASSIGN DD=1
//END
//IF -N
//ASSIGN N=MYDISK
//END
//IF -C
//ASSIGN C=80
//END
//ALERT 1,0
'/PAUSE %1DMount target disk in Drive #DD# & hit <ENTER>
FORMAT :#DD# (NAME="#N#",Q=N,DDEN,CYL=#C#,SIDES=2,ABS)
//ALERT 1,0
//PAUSE %1DFormat O.K.? <ENTER> if Yes, <BREAK> if Not.
PATCH DIR/SYS.SYSTEM:#DD# (D00,60=FF)
REPAIR :#DD# (ALIEN)
BACKUP :#SD# :#DD#
BACKUP SYS0/SYS:#SD# :#DD#
SOLE2 :#DD#
//ALERT (1,0,7,0)
//EXIT
```

NOTE: The above JCL cannot be used to perform a backup where cylinder 0 of the source diskette is not "Locked Out".

The following JCL (FMBK/JCL) is an expanded version of a JCL written by Frank Marten of the Adelaide Micro Users Group. I have made additions and modifications to allow selection of Source and Destination drives and Cylinder count, as well as using the same procedure to lock out cylinder 0 as was used in the previous JCL.

The normal BACKUP (reconstruct or by class) results in a poor system disk, with LDOS 5.1.4 the system files end up clustered at the beginning of the disk, as far from the directory as is possible, increasing the time to load any system overlays.

An optimum system disk is created by this JCL by performing several patches to SYS8/SYS, and by using an organised approach to the files to be copied, resulting in a disk with the system files placed as evenly as possible on either side of the directory. After creating the disk it may be subsequently backed up using the MIRROR/JCL listed above.

```
%1F.                DOS Disk Creator
.                  by Frank Marten
.
.         additions by Gary Bryce.
.
.To override default values, the relavent parameters
.must be given on execution of the JCL.  e.g :-
.
.   DO FMBK (SD=1,DD=2,N=SYSTEM,SI=1,C=40,M3)
.
.will create a Single Sided (SI=1), Forty Track (C=40),
.Model 3 (M3) disk, named "SYSTEM" (N=SYSTEM) on drive 2
.(DD=2) using drive 1 as the Source (SD=1), any parameters
. not specified will take the default values.
.
. * * * The DOS disk must not be write protected * * *
.
.The following files must be available to the system :-
. (a) REPAIR/CMD and SOLE2 (if Model 1).
. (b) PATCH/CMD.
. (c) All normal SYSTEM files.
. (d) Utilities (FORMAT & BACKUP etc)
. (e) LOG/CMD.
.
.*** Any of the above lines may be deleted from the JCL ***
.
.* WARNING * There is no turning back from FORMAT.
//IF -N
//.No diskname specified
//.    - Default Name = MYDISK
//ASSIGN N=MYDISK
//END
//IF -SD
//.No Source Drive specified
//.    - Default Drive = 0
//ASSIGN SD=0
//END
//IF -DD
//.No Destination Drive specified
//.    - Default Drive = 1
//ASSIGN DD=1
//END
//IF -SI&-C
//.No Sides & Cylinders specified
//.    - Defaults Sides = 2, Cylinders = 80
//ASSIGN SI=2
//ASSIGN C=80
//ASSIGN T1=25
//ASSIGN T2=2A
//ELSE
//IF -SI&C
```

```
 //.No Sides specified (Cylinder count of 40 was)
 //.     - Default Sides = 2
 //ASSIGN SI=2
 //ASSIGN T1=11
 //ASSIGN T2=16
 //ELSE
 //IF -C&SI
 //.No Cylinder count specified (Side count of 1 was)
 //.     - Default Cylinders = 1
 //ASSIGN C=80
 //ASSIGN T1=23
 //ASSIGN T2=2A
 //ELSE
 //.Sides & Cylinders were specified.
 //IF SI&C
 //ASSIGN T1=0F
 //ASSIGN T2=18
 //END
 //END
 //END
 //END
 .
 //ALERT 1,0
 //PAUSE %1DPlace target disk in Drive #DD# & hit <ENTER>
 .
 FORMAT :#DD#(NAME="#N#",Q=N,DDEN,CYL=#C#,SIDES=#SI#,ABS)
 .
 //ALERT 1,0
 //PAUSE %1DFormat O.K.? <ENTER> if yes, <BREAK> if not.
 //IF -M3
 //.Lockout Track 0, grans 1, 2 & 3
 PATCH DIR/SYS.SYSTEM:#DD# (D00,60=FF)
 REPAIR :#DD# (ALIEN)
 //END
 PATCH SYS8/SYS.SYSTEM (D00,FF=#T1#)
 BACKUP SYS0/SYS:#SD# :#DD# (S,Q=N)
 BACKUP SYS6/SYS:#SD# :#DD# (S,Q=N)
 PATCH SYS8/SYS.SYSTEM (D00,FF=#T2#)
 BACKUP SYS7/SYS:#SD# :#DD# (S,Q=N)
 PATCH SYS8/SYS.SYSTEM (D00,FF=#T1#)
 BACKUP SYS:#SD# :#DD# (NEW,Q=N,S)
 PATCH SYS8/SYS.SYSTEM (D00,FF=01)
 PATCH SYS8/SYS.SYSTEM:#DD# (D00,FF=01)
 BACKUP LBASIC:#SD# :#DD# (Q=N,I)
 BACKUP /CMD:#SD# :#DD# (Q=N,I,NEW)
 BACKUP /DVR:#SD# :#DD# (Q=N)
 BACKUP /FLT:#SD# :#DD# (Q=N)
 BACKUP :#SD# :#DD# (Q=N,I,NEW)
 BACKUP :#SD# :#DD# (NEW)
 %1F.
 . Steps to complete
 .
 .(1) Type LOG :0 <ENTER> and switch disks (if double sided)
 .(2) Configure your new system disk.
 .(3) Type SYSTEM (SYSGEN) <ENTER>
 .(4) Type SOLE2 :0 <ENTER> (if Model 1)
 .
 . Press <ENTER> to exit
 //ALERT (1,0,7,0)
 //EXIT
```

References :-

LDOS Manual - BACKUP UTILITY
- Backup by class & Backup reconstruct.

LDOS Quarterly - Vol.2 No.3
- LDOS: HOW IT WORKS
- Non-Radio Shack disk drives
- by Joseph J. Kyle-DiPietropaolo

NOTES FROM MISOSYS - Iss.2 page 55
- by Roy Soltoff

ADELAIDE MICRO USER NEWS - June 1985
- A Double Sided Booting Disk under LDOS
- by Frank Marten

SQUEEZING AN EXTRA GRAN INTO A DATA DISK IN AN EMERGENCY

by Bill Allen

[Reprinted from the TRS-80 SYSTEM 80 Computer Group newsletter (16 Laver Street, MacGregor, Queensland 4109, Australia).]

If you don't know already, under NEWDOS/80 2.0 -- the first gran on the disk is dedicated to two Boot sectors, one PDRIVE table sector and two unused sectors -- and, although they are essential on your system (DOS) disk in Drive #0, this information/code can be dispensed with in an emergency on a DATA disk. A simple fudge of killing the BOOT/SYS entries in the directory and de-allocating the first gran in the GAT releases this gran for data storage. I am not proposing to go around "killing" all my BOOT/SYS's on my Data disks as a pastime -- nor am I advising you to do it yourself, but it's handy to know about it in case of an emergency. WARNING !!!! Don't EVER insert a disk with data replacing the boot sectors in Drive #0 and press Reset. Unpredictable dire results may occur.

DID I HEAR SOMEONE SAY: "WHAT'S A DATA DISK?" -- Well, for his edification, if your system has more than one disk drive, then any disks used in other than Drive #0 DO NOT need a DOS on them -- it's completely superfluous in any drive other than Drive #0!! The disks used in a drive other than Drive #0 should be merely formatted and files copied to them so they will provide a much greater total amount of storage, because of the absence of the DOS System files. They are called data disks because of the absence of a DOS. If you are putting a DOS on all of your disks and you have more than one drive, then you are certainly using your system in a most INEFFICIENT manner. I hope you have plenty of coin for the extra disks you'll have to purchase.

The FORMAT function automatically puts a BOOT/SYS on a disk whether you are doing it for a system disk or a data disk and this uses up one granule (five sectors) of storage on the data disk.

WARNING!!!!! Don't forget to slap a write protect on your SYSTEM DISK before you do any of this in case you accidentally specify the wrong drive.

Here are the details for you to kill a BOOT/SYS by hand to see exactly what happens:

1 Format a disk.

2 Using Superzap, option DFS, DIR/SYS:dn of the target disk. Sector 0. (dn = Drive Number and # = no.)

3 De-allocate the first Granule by typing MOD00 and modifying the relative byte 00 (the first one in the GAT -- granule allocation table) to be the same value as the following byte. Different drive configurations would have different byte values, so you will determine the value by reading the second byte in the Granule Allocation Table of the disk concerned and you shouldn't go wrong (this method ONLY applies to a freshly formatted data disk!). (a)Complete the MOD by <ENTER>, <Y>, <ENTER>.

4 You now move to the next sector (#1 -- the Hash Index Table) by pressing the semicolon key (;). All you have to do there is to MOD00 and make byte 00 (where the cursor is) into a pair of 0s and complete the MOD as above at (a) -- (the hash code for BOOT/SYS (A2) is always at relative byte 00 and that's the value you will have to change to 00).

5 Now press the semicolon key once more to view the next sector (#2 -- the first directory file entry sector). There you will see on the top two lines the entry for BOOT/SYS. Type MOD00 again and change the first byte to 00. The file is now completely killed. This next bit is not really necessary, but you might as well tidy the sector up by hand while you are there. Just continue overwriting the values in the top two lines with 00s so that the whole sector is 00s -- i.e. hold down the 0 key and let the auto repeat do the work till you see all 00s. Complete the MOD with <ENTER>, <Y>, <ENTER> and <EXIT>.

You may now copy a file to the disk. After copying the file across, use Superzap to look at the first gran (DTS, <ENTER>, <dn,0,0>, <ENTER>) and you'll find by paging through with the ";" key that the first file has overwritten the BOOT sectors etc. Now, this hasn't harmed the disk at all. You'll have the normal directory access etc to the files on the disk. The only thing you are prevented from doing with the disk is to change its PDRIVE table, which it no longer has (but who would want to do that anyway?). The PDRIVE table on a data disk is NEVER used normally (I'm aware that some automatic disk recognition programs use them, but (UNMODIFIED) NEWDOS/80 2.0 doesn't) and there is certainly no valid reason to change the PDRIVE table on a data

disk. Its presence or absence is completely immaterial for the purposes of a data disk.

THE LAZY METHOD

Having led you by the nose through the Directory with Superzap (so the learners will understand more about both), at this point I have to admit that there's also an extremely easy method to accomplish the addition of the extra granule (the one you would use in an actual emergency):

Step 2 could be: (from NEWDOS/80 READY) KILL BOOT/SYS:dn <ENTER> (be VERY CAREFUL to get the drive number right before entering the command) and you will use the extra gran with the first file written to the disk after this has been executed. The only difference would be that the killed directory entry of BOOT/SYS would not be cleaned up but you could use DIRCHECK/CMD to do that for you when you use the "C" option.

You might think: "I don't think I'll bother with just one gran", but this trick also affords an "out" sometimes when you may be caught in the middle of something. You might be trying to save a file (or more likely -- data) to disk AND it HAS to be this particular disk and you get the message "DISKETTE SPACE FULL". You may use DOS (or, more usually, MINIDOS <DFG>) to KILL BOOT/SYS on the disk and that may be all the extra space you need. You don't necessarily have to KILL it at the start. I took you through that method so it is easy to see with Superzap exactly what you have done, before the disk is cluttered with a lot of files.

I think it best that you understand what is actually being done when you perform a non-standard action with your DOS, so I exhort you to do exactly as I have directed (do it by the long-winded method) for the first time (if this is new to you), to further your computer education.

A SPECIAL NOTE ON COPYING: If you wish to make a backup of a modified data disk (i.e., a complete clone by FORMAT 5 COPY rather than a Copy By File -- FORMAT 6), you MUST specify the BDU parameter, otherwise NEWDOS/80 will first copy the disk faithfully, THEN REWRITE A BOOT/SYS ETC ON THE FIRST GRAN. The BDU parameter inhibits this action. The manual doesn't mention this particular action of writing the BOOT/SYS or its inhibition by specifying the BDU parameter -- it just refers to BDU as <B>ypass <D>irectory <U>pdate.

So, a Format 5 backup of a modified data disk should go thus:

COPY 1 0,,FMT,BDU

or whatever drive specs to suit your system. If you are putting the backup on a previously formatted disk, you must first KILL BOOT/SYS:dn and you then will have the choice of using CBF (Format 6) or Format 5 (with BDU). As a matter of fact, whenever you really desire to make a complete clone of any disk copyable by NEWDOS/80 2.0's COPY, you should always use FORMAT 5 as above with the BDU parameter to ensure that the disk will, in fact, be an EXACT copy. The BDU will not necessarily be mandatory, but will do no harm provided that the disk you are copying has no problems to carry over to the clone. A problem disk would have to be copied by the CBF (FORMAT 6) method to attempt to leave the woes behind.

Usually, though, one doesn't really need to clone a data disk, so it is best to copy a lot of files from a working disk onto another by the FORMAT 6 method (CBF), as a lot of file extents are usually eliminated by this mode of transfer. This gives faster access to long files on the new copy, because of the lesser amount of drive head movement necessary to read them. So, the most efficient way to backup a completely full modified data disk is to format the new disk, KILL BOOT/SYS on it, use DIRCHECK to <C>lean up the directory, then do a CBF of the files from the disk you are copying.

A FURTHER NOTE -- ON SAFEGUARDING YOUR SYSTEM DISK: I have a complicated system of mixed drives and often have to change the PDRIVE settings on my system disk (see Bits & Bytes, Feb., 85, P.2). A system disk in a multi-drive system usually has, besides the DOS, a number of utilities that are used most often. One tends to fill the system disk for this purpose, so one doesn't want any extra files to be written to this disk on default parameters, otherwise you'll get the "DISKETTE SPACE FULL" message and will have to retype the instruction to redirect the file to another drive and later use DIRCHECK to <C>lean up the system disk directory if much of this goes on. But, as I have to change the PDRIVE settings fairly frequently, I don't want the hassle of having to be taking the write protect on and off all the time -- so, we now adjourn to the SYSTEM 0 DOS command.

By specifying SYSTEM Ø,AO=1 (or whatever drive other than 0) for the default write of a new file, your system disk is protected provided you have a disk (or diskette, if you fancy that term), with available storage space, always mounted in the specified default drive (without a write protect, of course).

As my drive #2 is a double sided 80-tracker, it has a tremendous storage capacity -- so at present it is my specified default drive (AO=2) and there is always a disk there waiting to record any writes and I am able to leave the write protect tab off the system disk in drive #0 (unless I am doing a test of a machine language routine under development, when it is always advisable to protect ALL the disks in the system in case the routine goes crazy and tries to clobber everything in sight!).

NEWDOS ZAP NUMBER 89
by Alf West

[Reprinted from the TRS-80 SYSTEM 80 Computer Group newsletter (16 Laver Street, MacGregor, Queensland 4109, Australia).]

During the month I was updating some data files (random access) and was in all the strife in the world, getting partial records and in other cases, SIGN errors. On account of the sign errors, I reckoned the programs must have somehow got clobbered somewhere and spent hours in checking and then faking the GET's and PUT's to get some sort of results which were still unsatisfactory. In desperation I went to the old backup copy of the program with an old system on it and everything worked like a charm. Then the penny dropped.

Only recently I had got around to applying Newdos ZAP number 89 to my working DOS. Whereas the programs worked perfectly with the un-zapped BASIC/CMD and SYS10/SYS, they were hopeless in certain sections of the Data I/O with the Zap 89 applied. I triple checked the entries I had made, but my zapping conformed exactly to that given by Newdos. So my advice is, do NOT, repeat NOT, apply ZAP number 89 as given by Newdos. In any case, the conditions that the Zap purports to correct, would be well outside the range most of our members would have with their data files.

[NORTHERN BYTES Editor's Note: I can echo this advice, since I have heard from others who have had similar troubles after installing the infamous Zap 89!]

MODIFICATION TO CHAINBLD/BAS
by B. Gielen, Breda, Holland
Translated by Paul Fransen

If you have made a chain file, you probably would like to have a hardcopy of it. You can't use the JKL-option of NEWDOS to print out the screen (after the L-command). But the JKL routine finds a CHR$(140) after each line, and that means a Form Feed for the printer.

It is a waste of paper to print each line on another page. So here is a solution. Change CHAINBLD/BAS with the following lines:

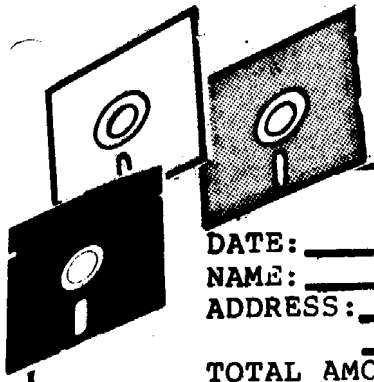
```
139 PRINT"H TOGGLE HARDCOPY AFTER L-COMMAND ON/OFF"
177 IFOC$="Q"THEN60
178 IFOC$="H"THENH=H+1:IFH>1THENH=0:GOTO158ELSE158
179 GOTO274
183 IFH=1THENINPUT"FILE NAME : ";FS$:LPRINT:LPRINTSTRING$
(80,45):LPRINT"CHAINFILE : "FS$:LPRINT
185 IF(LN>LCORY=14)ANDH=1THENLPRINTSTRING$(80,45)
197 IFH=1THENLPRINTMID$(A$,2,6);LN$(LN);:IFW<>0THENLPRINT
198 LN=LN+1:GOTO185
```

Now there is a Toggle for a Hardcopy installed.

TAS / NORTHERN BYTES ELECTRONIC MAIL ADDRESSES

MCI Mail ID - Northern Bytes:	102-7413
MCI Mail ID - The Alternate Source:	109-7407
TELEX - Northern Bytes:	6501027413
ANSWERBACK:	6501027413 MCI
TELEX - The Alternate Source:	6501097407
ANSWERBACK:	6501097407 MCI
Compuserve EasyPlex:	72167,161
Delphi Mail:	TASIO

Diskette Shopping Made Simple



Use the handy order form to order diskettes in the quantities desired. Use of the product codes to the left of the product helps insure that your order is processed properly. All diskettes are soft-sectored. "Flippies" are punched for use on both sides with a single headed disk drive. All DSDD (double sided, double density) media will work on both single and double headed disk drives. All media is fully guaranteed for five years from the date of purchase.

DATE: _____

NAME: _____

ADDRESS: _____

TOTAL AMOUNT PURCHASED: $_____

VISA _____ MASTERCARD _____ CHECK _____ OTHER _____

CARD NUMBER: _____

EXPIRATION DATE: _____

SPECIAL INSTRUCTIONS: _____

We also accept pro forma invoices and properly signed purchase Orders from most schools and governmental agencies.

Package of 10 diskettes with Boxes, Labels, Sleeves and Write Protect Tabs:

Item Code	Price	Style	QTY.
MSSDDX	$9.95	SSDD	
MDSDDX	$11.95	DSDD	
MDSFLX	$12.95	Pre-punched DSDD Flippies	
MDSMUX	$12.95	Multi-Colored DSDD	

Package of 25 BULK diskettes W/Labels, Sleeves and Write Protect Tabs (you save us the labor of putting them together; YOU do it as you need the diskettes):

Item Code	Price	Style	QTY.
MSSD25	$19.95	SSDD	
MDSD25	$22.95	DSDD	
MDSF25	$24.95	Pre-punched DSDD Flippies	
MDSM25	$24.95	Multi-Colored DSDD	

Package of 100 diskettes with Labels, Sleeves and Write Protect Tabs:

Item Code	Price First 100	Price Extra 100's	Style	FIRST	EXTRA
MSSDDC	$79.95		SSDD		
MSSDDD		$74.95	SSDD		
MDSDDC	$89.95		DSDD		
MDSDDD		$84.95	DSDD		
MDSFLC	$95.95		"FLIPPY"		
MDSFLD		$89.95	"FLIPPY"		
MDSMUC	$95.95		COLOR		
MDSMUD		$89.95	COLOR		

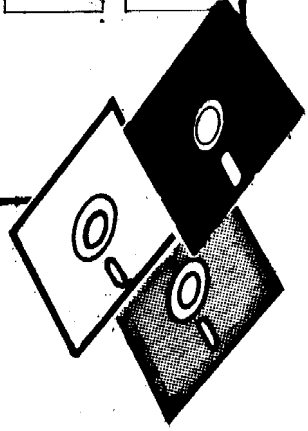
Package of 100 BULK diskettes with Labels, Sleeves and Write Protect Tabs:

Item Code	Price First 100	Price Extra 100's	Style	FIRST	EXTRA
MSSDBC	$75.95		SSDD		
MSSDBD		$69.95	SSDD		
MDSDBC	$85.95		DSDD		
MDSDBD		$79.95	DSDD		
MDSDFC	$90.95		FLIPPY		
MDSDFD		$84.95	FLIPPY		
MDSDMC	$90.95		COLOR		
MDSDMD		$84.95	COLOR		

VISA

These prices are CASH prices and reflect a 4% cash discount. You pay the first $3 in shipping; we pay the rest.

MasterCard



Phone: (517) 482-8270

The Alternate Source

704 North Pennsylvania Avenue
Lansing, Michigan 48906

NORTHERN BYTES

Subscription Information

Northern Bytes is edited by Jack Decker and published on an irregular basis by The Alternate Source Information Outlet. Back issues are available starting with Volume 5, Number 1. Issues prior to that are not available. Some of the most valuable articles from earlier issues may be reprinted in future issues of Northern Bytes. Currently there are eight back issues available for Volume 5, as well as all issues from Volume 6. All back issues are $2 each.

It is very easy to be placed on the Northern Bytes REGULAR list. Simply place your address, Visa or MasterCard number and expiration date on file with us. We will start with the issue you request. We do not bill you for ANY ISSUE until that issue has been mailed. This way, we can continue to offer you top quality information with absolutely no risk to you. There's no question of what to do about unfulfilled issues if we decide to quit publishing. Unless otherwise requested, we presume your subscription will extend through the month of your expiration date.

Don't have a charge card, huh? We understand the myriad of reasons for not having them and we feel that a "To-Be-Invoiced" policy could help increase the demand for Northern Bytes. If you'll do it for us, we'll do it for you. Would you like to be placed on a regular list TO BE BILLED for each issue? You could then send a check for the issues as they are mailed. If you didn't send a check, we would presume that your interest has died and discontinue your subscription. The only requirement for getting onto the list is to pay for the first issue up front; the next will be mailed automatically, if you request. You are insured that you will receive top of the line TRS-80 information as it is released. Ask to be placed on the NO RISK "To-Be-Invoiced Northern Bytes List".

Call or write, but SIGN UP TODAY!

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

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)

Bulk Mail
U.S. Postage Paid
Permit #815
Lansing, MI

POSTMASTER: If undeliverable return to:
The Alternate Source, 704 North Pennsylvania Avenue, Lansing, Michigan 48906

To: