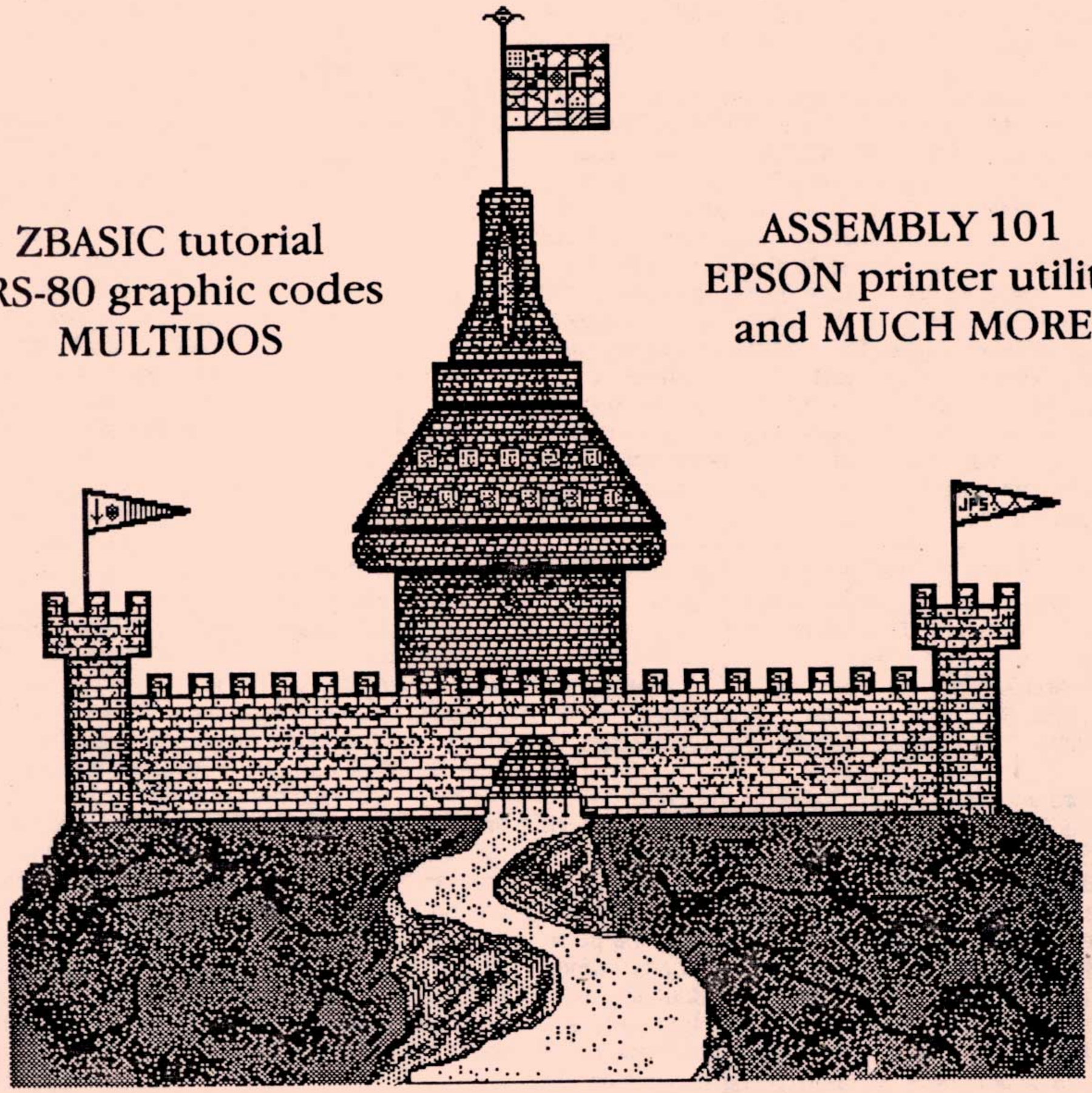
TRSTIMES

Volume 2. No. 5. - Sep/Oct 1989 - $4.00

ZBASIC tutorial
TRS-80 graphic codes
MULTIDOS

ASSEMBLY 101
EPSON printer utility
and MUCH MORE



The TRS-80 Safe Haven

LITTLE ORPHAN EIGHTY

I have several annoucements this issue. First, of course, is the continuation of TRSTimes in 1990 where we will publish 6 brandnew bi-monthly issues. The price for U.S. and Canada will remain at $18.00 and, as usual, each issue will be shipped by first class mail. The one change from 1989 affects our overseas subscribers to whom we have shipped air-mail for $23.00 per year. We simply cannot hold that price, as it does not cover reproduction and shipping costs. The 1990 overseas subscription price will remain at $23.00, but each issue will be shipped by surface mail. Should you wish to continue receiving issues by airmail, the airmail subscription price will be $29.00. I do not like to do this but, like all businesses, the continued existence of TRSTimes is dependent upon taking in more money than is going out. I do hope you understand.

Second, a couple of issues ago, I mentioned that we were looking into the possibility of a TRSTimes BBS. This did not work out satisfactorily, so the idea was dropped. However, a few weeks ago I ran into Don Roudebush, who runs one of the finest BBS' in the Los Angeles area. To be sure, his board is primarily oriented towards MS-DOS, but after some thought Don agreed to also support the TRS-80. He promised to set aside a limited amount of his hard-disk space for us. Well, that 'limited' amount turned out to be 40 megabytes for messages and up- and downloads. There is free access to all TRS-80 users. Quite generous, I'd say. So give the board a call; leave messages for TRSTimes (we will check on a weekly basis) and download some files. If you can, please upload something also, so we can build this into something special for the TRS-80 crowd.

DATA SHACK (818) 773-0372
N-8-1 300/1200/2400 baud 24 hours
Sysop: Don Roudebush

To up- or download, choose area 'T' (for TRS-80) from the file menu. Also, for any of you with a Model 100/102, Don will also support these machines; choose area 'M'.

The third announcement is very exciting for us here at TRSTimes. We have made a special arrangement with GRL Software and, as of August 1, 1989, we will be the exclusive distributor of SYSTEM 1.5. This is the ultimate upgrade to TRSDOS 1.3. for Model III, giving this DOS many new unheard of powerful features. This fine package, written by Gary Campbell, received a very high rating in Bob Rose's review from issue 2.3. It is something every TRSDOS 1.3. user should have, and now you can get it directly from TRSTimes. The price for this fantastic piece of software is $39.95 + $4.00 shipping & handling. As TRSTimes is set up for mass mailing, we will waive the $4.00 S&H, thus making the total price $39.95. (*see our ad elsewhere in this issue.*)

The last announcement is a sad one. It seems we have a 'sickie' in the TRS-80 world. Yes, we now have a documented case of a 'virus'. To the best of our knowledge, so far, it is not the kind that will infect or destroy programs or equipment. Nevertheless, it is a nasty and really dumb thing. Someone has tampered with DEARC31/CMD (which can be found on most BBS') and then proceeded to spread the damaged file as widely as possible. The altered file will, at random intervals, display very obscene messages on the screen. If you already have DEARC31/CMD, or you download it from a BBS, you can tell the two versions apart by the size of the file. The original (good) version is around 5K, while the corrupted version is approximately 27K. So watch out! It is certainly not the type of program you will want in your collection.

Incidentally, a reliable source informs us that the BBS community knows who he is, and is taking appropriate measures to isolate both the file and the demented individual. Good!

We have received several letters indicating a desire for more Model 4 coverage, as It seems the last few issues have been favoring Model I and III. We do try to balance the content of each issue as much as possible, but, as you know, we are at the mercy of the readers. If you do not send us articles and programs pertaining to the Model 4, the ratio will remain somewhat out of balance. While we do have several in-house Model 4 projects in the works, we prefer the pages of TRSTimes to reflect the works and discoveries of the readers. We want you to be involved, so send your Model 4 articles to us. This does not mean that we are not accepting Model I and III articles. Quite the contrary, send them in; we are always looking for good information for our machines. As a matter of fact, Model 100/102 articles and programs will also be considered. We are not planning extensive coverage of Tandy's portables but, as several readers own one or the other of these machines, we feel that an accasional article or two might be appropriate.

Please note that when submitting articles and/or programs for any of the machines, make sure that your name and address is written in the article text, on the print-out, and on all disks. We want you to get due credit for your work.

and now.....welcome to.....TRSTimes 2.5.

Lance W.

TRSTimes - Volume 2. No. 5. Sep/Oct 1989

LITTLE ORPHAN EIGHTY 2
Editorial

THE MAIL ROOM 4
Reader mail

AN INTRODUCTION TO ZBASIC 7
Mark Speer

THE NEW PEOPLES' LITERATURE 8
Eric Bagai

ASSEMBLY 101 9
Lance Wolstrup

TRS-80 GRAPHIC CODES 18
Fred Blechman

A GUIDED TOUR THROUGH MULTIDOS 21
Andrew Bruns

HINTS & TIPS 23
Bruns - Showalter - Parsons - Burkholz

SET YOUR PRINTER WITH EPSONSET 25
Robert Doerr

SETTING UP A HARDISK ON THE MOD 4 UNDER CP/M 28
Roy Beck

THE SWAP MEET 31
Classifieds

CLOSE #5 34
Editorial

TRSTimes magazine is published bi-monthly by TRSTimes publications.
20311 Sherman Way suite 221. Canoga Park, CA. 91306. U.S.A.

1989 subscription rates (6 issues):
United States & Canada: $18.00 (U.S.)
All other countries: $23.00 (U.S.)

THE MAIL ROOM

A RAVE REVIEW

I would like to recommend a BBS. I **AM BLESSED** to be within local calling distance of one of the **PREMIER** TRS-80 BBS's in the country! I refer to 8/N/1 BBS of Gainesville, Florida, sysoped by Guy Omer. It is loaded to the gills (30+ megabytes of gills) with all kinds of I/III/4 software, plus extensive support for the CoCo, Model 100 and CP/M, with a moderate amount of art & Hi-Rez files and GOBS of ORC-90 files. It is 300/1200/2400 baud, online 24 hours a day. If you have called 8/N1 #4 in Philadelphia (home of TRSLINK), you know how it works - no message base - just plenty of files! It deserves plenty of calls.

Michael E. Webb
Gainesville, FL.

UPGRADING TO DOUBLE-SIDED DRIVES

I am presently upgrading my Model IV from single-sided to double-sided ones. Hardware-wise it is only necessary to connect the drive, adding the side select line which Radio Shack designates as a drive select, but does not use.

I do, however, have a software problem. How do you make a double-sided boot disk? I tried BACKUP from a single-sided to the double-sided, but it won't function as a boot disk. I have access to POWERTOOL AND HYPER-ZAP if needed. (My own, of course, not copies).

John Greenland
Kelligrews, New Foundland

Congratulations on getting double-sided drives. I am sure you'll be very satisfied with your investment.

Assuming that you are using TRSDOS 6.x. or LS-DOS 6.3., here are the necessary steps required to make your double-sided boot disk.

1. Boot up your Mod 4 with the single-sided boot disk.

2. FORMAT a disk in drive :1
a: Answer '2' to the sides prompt.
b: Answer '40' to the tracks prompt.

3. When the format is complete, type:
BACKUP SYS0/SYS:0 :1 (S) <ENTER>

4. Now perform a complete backup, type:
BACKUP :0 :1 (S,I) <ENTER>

Step 3 is necessary because it insures that SYS0/SYS is copied to the front side of the double-sided disk. The backup in step 4, since the the two disks are of different size, forces a copy-by-file backup. By specifying S,I, all files are copied over to the double-sided disks, thus making it bootable.

Ed.

TRACING YOUR ROOTS

I would like to know if there is some type of GENEOLOGY program available for the Model 4 or III that you might be able to either put in TRSTimes, or give me information so I may purchase it.

William R. Salisbury
Columbia, SC.

I know of a program called HERITAGE IV. It is a Model 4 package written in Basic by Jerry Truhill. Unfortunately, I have not been successful in obtaining Mr. Truhill's address, nor do I have information on whether this program is commercial or in the public domain. Sorry!

Ed.

THE MEMORY COMMAND, NEWDOS/80 & SUPER U.

In the LS-DOS 6.3. update information there is a reference to a use of the MEMORY command which sets the lowest cylinder on a disk which will be written to. It is of the form MEMORY (A="A", B="xx"). I can't find any other reference to this syntax anywhere else in either the update info or the 6.2. manual. Does anyone know the significance of the A="A"?

Also, when formatting a disk in single density under NEWDOS/80 v2. on my Model 4, the format fails at track 20. However, there is no problem when using LDOS for the same job. I feel it is related to switching of write pre-comp which happens here, but can't quite see the connection. I've had people tell me that they've heard of this before, but can't remember the solution.

Finally, under Super Utility 4/4P, what are the permissable DOS specifiers and combinations for the configuration table?

I would be most grateful for any answers you are able to publish.

Terry Murphy
Auckland, New Zealand

Let's take 'em one at a time. First, the MEMORY command is a good example of improper documentation. Though the manual uses two pages to explain this command, it leaves many questions unanswered.

The syntax for MEMORY IS: MEMORY (parameters)

The parameters are: CLEAR=value, HIGH=address, LOW=address, ADD=address, WORD=word,

BYTE = byte, GO = address.

Now to answer your question: If you type:

MEMORY (A = "A") <ENTER> DOS will answer with:

X'006A = 106 (X'0100) High = X'FFFF' Low = X'2FFF

The A in (A =) is an undocumented abbreviation for the ADD parameter. The "A" in (= "A") is the address of the A flags memory location. DOS is telling you that A flags are located at 6AH, which is 106 in decimal. Further, it tells you that the byte at 6AH is 01H, and the next byte (at 6BH) is 00H. Also, you are notified that HIGH$ is FFFFH and LOW$ is set to 2FFFH.

Now, A flags is the allocation flag, containing the starting cylinder number to be used when searching for free space on a diskette. Thus, by using B = xx you can change the byte at 6AH to whatever you want it to be. (B is the undocumented shorthand for the BYTE parameter).

You can change the settings of any of the other flags as you see fit, but be careful, you must know what you are doing as most of the flags are 'bit flags'. For example, you can change K flags to set the keyboard to uppercase by:

MEMORY (A = "K",B = 32)

or to lowercase by:

MEMORY (A = "K",B = 0)

Even better, MEMORY (A = "N",B = 128) will disable all password protection. This bit (7) of the N flags is not documented in the Model 4 Technical Reference Guide; it is simply explained as 'reserved for system use'.

On to NEWDOS/80. I formatted 5 single-sided, single-density 35 track disks using this PDRIVE setting:

PDRIVE,0,1,TI = A,TD = A,TC = 35,SPT = 10,TSR = 3, GPL = 2,DDSL = 17,DDGA = 2

I then changed the track count to: TC = 40 and formatted the same 5 disk to single-sided, single-density 40 track disks. Resetting TC to 35 and changing to TD = C (double-sided and single-density), I repeated the process. I had no problems whatsoever. My guess is that either your PDRIVE setting is incorrect, the disk is flaky, or your drive is out of alignment. Incidentally, here is an interesting PDRIVE setting:

TI = A,TD = E,TC = 40,SPT = 18,TSR = 3,GPL = 6, DDSL = 12,DDGA = 2

Stick this setting on drive 1 and you'll be able to read the directories of TRSDOS/LS-DOS 6.x., and LDOS disks.

My knowledge of SU4 leaves much to be desired, so I passed the question to Roy Beck who instantly prepared the following chart:

		I	III	4
TRSDOS	SD	T1	---	TS
	DD	T1D	T3	T4
LDOS	SD	L1	L	TS
	DD	L1D	L3	T4
DOS PLUS	SD	D1	DS	TS
	DD	D1D	D3	T4
MULTIDOS	SD	M1	MS	---
	DD	M1D	M3	---
ND80v2	SD	N1	NS	---
	DD	N1DR	N3R	---
DBLDOS	DD	B1R	---	---

Ed.

PATCHING BLUES

In Gary Campbell's article "BRAND NEW PATCH FOR TRSDOS 1.3", there is an error in one of the patches. The one at hex address 54E7 produces the error "String Not Found", and will not apply the CHG code. All the other patches work properly. I attempted the published patches on plain, unpatched, vanilla original copies of TRSDOS 1.3, versions "May 1, 1981", "May 2, 1981", "July 1, 1981", "June 1982", and "Feb 20, 1984". The results were the same on all Tandy produced versions of TRSDOS 1.3. The FIND string given for ADD = 54E7 of CD2044C22352 simply is not there! I brought out my copy of SUPER UTILITY and took a look at the system file *09. What is there at that location is CD2044C3955F! Note that the first three bytes in Gary's article are correct (CD2044), but the last three are different. The ones Gary gives as a FIND string are the same as part of his CHANGE string for ADDRESS 5F95H, which is the very next PATCH. Did Gary maybe misread part of the line as he was copying his work for final publication?

The following is a corrected set of patches that produce the results that Gary talks about.

(Editors note: Do NOT apply this patch fix until you have read the answer below.)

PATCH *09 (ADD = 59A8,FIND = 32E054,CHG = 000000)

PATCH *09 (ADD = 5358,FIND = 15,CHG = 11)

PATCH *09 (ADD = 54EA,FIND = C22352,CHG = C3955F)

PATCH *09 (ADD = 540E,FIND = 3AA6549047C5,
CHG = 2A6561C3DB54)

PATCH *09 (ADD = 54D0,FIND = CDF35221FFFF,
CHG = 3EC9322A53CD)

PATCH *09 (ADD = 54D6,FIND = 22BA52CD9B,
CHG = F352C35753)

PATCH *09 (ADD = 54DB,FIND = 59C2235206,
CHG = 110063B700)

PATCH *09 (ADD = 54E1,FIND = 119461210064,
CHG = ED527C47B7CA)

PATCH *09 (ADD = 54E7,FIND = CD2044**C3955F** ,
CHG = 3854C5C31454)

PATCH *09 (ADD = 5F95,FIND = 43616E277420,
CHG = C22352ED4BD7)

PATCH *09 (ADD = 5F9B,FIND = 416374697661,
CHG = 56CD4244CD39)

PATCH *09 (ADD = 5FA1,FIND = 746520447561,
CHG = 44CAAD5F0000)

PATCH *09 (ADD = 5FA7,FIND = 6C207768696C,
CHG = 000000C30944)

PATCH *09 (ADD=5FAD,FIND=6520524F5554, CHG=CD3F443AA453)

PATCH *09 (ADD=5FB3,FIND=4520697320, CHG=47AFC3B153)

If you have attempted the set of patches in Gary's article, and the one patch for ADD 54E7 failed, you might desire to apply only the following patch that I call COPYFIX3/BLD.

PATCH *09 (ADD=54E7,FIND=CD2044C3955F, CHG=3854C5C31454)

Arthur N. McAninch, Jr.
Borger, TX.

Michael Webb of Gainesville, FL. also reports that the patch to 54E7H failed. This is very mysterious because the patch, as listed originally in issue 2.4., is CORRECT. I personally applied the patches to a fresh copy of my virgin master disk dated July 1, 1981. I had no problem with the patch at 54E7H. It, as well as all the others, went in perfectly, and the COPY command now works exactly as Gary described. Checking the TRSDOS 1.3. 'patchers bible', TRSDOS COMMENTED, 1982 Soft Sector Marketing, Inc., I found that the op-code at 54EAH to be:

JR NZ,5223H which is: C2 23 52

You report the code at 54EAH to be:

C3 95 5F which translates to: JP 5F95H

According to my disk, and TRSDOS COMMENTED, 5F95H is the beginning of the error message: "CAN'T ACTIVATE DUAL WHILE ROUTE IS IN OPERATION."

Now, since ROUTE is not implemented in TRSDOS 1.3., it is very possible that someone, maybe even TANDY, for some unknown reason patched 54AEH to JumP to 5F95H where they overwrote the error message(s) with new code. I am extremely interested in what your July 1, 1981 version has stored there.

Ed.

BUGS IN TRSLABEL/BAS

I found two typos in the TRSLABEL/BAS in the Jul/Aug issue of TRSTimes.

1. the end of line 510 needs to say to press <SHIFT><CLEAR>.

2. Line 650 should read: IF I<1 OR I>5 THEN 530

You'll get plenty of mail, I'm sure, but I had to write as this is the first time I really understood what was wrong with a non-working program.

Jim Savage
Clinton, MS.

The error in line 650 is the another of Ventura Publishings minor, but irritating quirks. As mentioned in prior issues, it will sometimes (but not always) leave out the < and > symbols when importing text. To make matters worse, at the time of leaving out the < or >, it may also leave out the character immediately following. This, unfortunately, happened in line 910, which should read:

910 IF I$<CHR$(31) THEN 840

(Jim, you didn't catch that one; maybe we both need thicker glasses!)

I am not using the Ventura Publishing quirk as an excuse; though this is what happens, it is my job as editor to make sure the listings are correct. I shall do a better job of checking in the future. Sorry.

Now, the <SHIFT><CLEAR> addition is not necessary on the Model III. The <CLEAR> key, by itself, does invoke the preset label. I tested this on LDOS 5.1.3. & 5.1.4., TRSDOS 1.3., DOS PLUS 3.4. & 3.5., MULTIDOS 1.6. & 1.7., and NEWDOS/80 v2. As my Model I is at this time not working properly, I did not test the program there, so It is possible that the <SHIFT><CLEAR> combination is needed to invoke the preset label on Model I.

Ed.

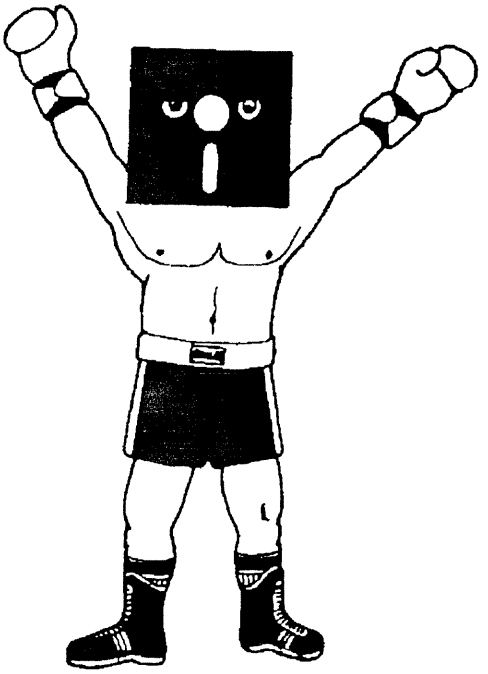
ROSES

I am a new subscriber to your magazine and was surprised at the quality of the information. After 80 Micro left us, I really didn't think there would be much left for us. Thanks for the continued support. I was especially pleased to see regular coverage of the CP/M side of my Model 4. Roy Beck is doing a great job. Another thing that caught my attention was the program called ROTATE/BAS (in issue 2.3. Ed.) with the reference to David Ahl's 'More Basic Computer Games'. This book has been sitting on my shelf for many years and, being curious as to the differences between your program and the original, I typed in both programs. What a pleasant surprise. Your version is far superior to the original. Basic sure has come a long way.

Keep up the good work; I'll be with you next year also.
T. A. Bilecki
Houma, LA.

AN INTRODUCTION TO ZBASIC

by Mark D. Speer



THE BASIC CHAMP?

Introduction

Compiled programming languages have been around for years on large computer systems, but got off to a rocky start in the early days of micro computing. Now they are becoming very inexpensive, easy to use, and practical on MS/DOS computers. I wonder how many TRS-80 owners realize such a thing exists for us also? It is the ZBASIC compiler from Zedcor. I have been using it on my Model 4 (in both Model III and Model 4 modes!) for over 3 years now with great success and enjoyment. In this article I will cover some of the basics of this fine language, tell how it is used, and give some of it's advantages and disadvantages. I hope to write a series of articles so let us know what you want to know about ZBASIC.

What is a Compiler?

To understand the advantage of a language "compiler" you must understand what a language "interpreter" does. The BASIC that comes with your TRS-80 computer is a BASIC "interpreter". Suppose your spouse was from another country and his or her father didn't speak English. Every time you visited him you would have to have your spouse by your side to "interpret" what you said in English and translate that to your father-in-law's native language and vice versa when he said something to you. This can get very time consuming! Likewise your TRS-80 does not really understand "*450 IF LN = 54 THEN GOSUB 5000*". It only understands machine language which is a stream of zero's and one's. So every time you write a BASIC program line like the one above the interpreter has to jump in and translate that into something that your computer understands just like your spouse would have to translate your English into another language for your father-in-law. Again, this can get time consuming. A compiler must interpret your program and translate it to machine language also but the big difference is that it only has to do it ONCE!! After that you save it and that translation never has to occur again (unless of course you modify the program and have to recompile it). The interpreter on the other hand has to keep doing the same slow translation EVERY time you run the program. If you have a simple 20 line program an interpreter does the job just fine. But if you have a large program or a program that has portions where speed is very important (a complex input routine, sorting data, displaying large graphics screens, etc.) then skipping that translation step can make a noticeable difference.

Compatibility

A major concern when moving from one language to another is the complexity of the move. How much will you have to learn about the new language? How much will you have to modify any programs you convert to the new language? How hard is it to learn to work in the environment of the new language (e.g. the program editor, executing the program, debugging tools, etc.)? In my opinion the transition to ZBASIC should be pretty easy. Any pain you experience in moving to ZBASIC will be more than offset by the new and exciting features you will be gaining. The line editor is the same as that found in native TRS-80 BASIC. The reason for this is simple. The author started out on a TRS-80 and knew what to keep and what to throw out. The vast majority of statements are identical or very similar. Some areas where things get a little different are in screen handling and file handling. There are major enhancements in both of these areas, but of course that makes it different. String allocation is different (they totally eliminated the garbage collection problem!) as is number crunching (no more floating point math errors!).

The good news is once you have your favorite program converted to ZBASIC then converting it to another

machine can be simple because ZBASIC is available on many of the popular micro computers. If written with just a slight bit of forethought a ZBASIC program written on a TRS-80 can be taken to an MS/DOS computer, recompiled and it will run with no changes. Of course you have to avoid some obvious things like not POKEing some TRS-80 specific memory address. This compatability between computers in my opinion is a very important feature!

Sample ZBASIC Session

Let me show you how to start using ZBASIC. At the DOS Ready prompt type in "ZBASIC" and press <ENTER>. Soon a banner and copywrite notice will appear along with a list of options. Don't panic, just press <E> to put you in the Edit mode. You may have occasion to use some of the other options (especially the configuration option) when you get more experienced. However the defaults will handle many if not most of your programming needs. Now just start writing a program much the way you would with regular BASIC. Your favorite command are all there like List, Delete, Edit, LList, Save, etc. Within lines you can search, delete, change, extend, etc. just like you are used to doing. Okay, this is pretty easy! The compiling is the hard part, right? Nope! To compile and run your program you simply type "RUN". It compiles the program in seconds and then runs the program for you without further delay. If it sounds simple and nearly what you are used to with regular BASIC then you are correct. Now this is for a small program and there can be a few complications for extremely large programs but those are in the minority.

How to Get ZBASIC

ZBASIC for the TRS-80 is available in two versions. The first works on Model I, Model III, and Model 4 in Model III mode. The second version works on all Model 4's in Model 4 mode. The cost is $89.95, which includes the ZBASIC compiler, some sample programs, the help file, and a very large (but easy to use) manual. To order write or call Zedcor, Inc., 4500 East Speedway Blvd., Suite 22, Tuscon, Arizona 85712-5305. Telephone (800) 482-4567.

What's Next?

I hope I have given you a brief introduction to ZBASIC. In my next article I will explain some of it's neat features, explain some conversion problems and show you some code from one of my own ZBASIC programs to give you an idea of how the code looks.

The New Peoples' Literature

by Eric Bagai

Although not a frequent user of electronic bulletin boards, I have noticed that some callers use little cartoon-like constructions of alphanumeric characters for various purposes. Sometimes these icons serve as personal signatures, and sometimes they serve more to indicate the affect they wish to convey when the sender is not sure that his hastily typed words are sufficiently clear. The simplest construction consists of a colon, a dash, and a left parenthesis, which when viewed sideways becomes the common happy face; **:-)** see?
They look best on the computer screen, and some do not print on all printers.

Perhaps they are compensating mechanisms for the lack of affect that is supposed to be typical of computer "nerds." Perhaps they express the same mindless aesthetic that popularized the Happy Face, Keene paintings, and "Have a nice day." More likely, they are another means of personal expression by people who have learned that looking at the world sideways is not necessarily a handicap. In any case, here are some of the ones I've seen.

:-)	happy face, not serious, well meant
:-] :- 3-) B-) 8-D	varient smiles
;-)	pun or joke (wink)
%-}	cubist smile
$-)	prospect of profit smile
O-)	frogman's smile
0-)	dead frogman's smile
phemic smile	
8-O :O :o	"WOW" "wow" "oh"
:-(	sad - not approving
> -(	anger, disagreement
> >B-:O	wrathful native deity
B-I	Batman (personal signature)
:-&	tongue-tied (personal signature)
(:-	Kilroy (impersonal signature)
BD BO	Comedy/Drama theatrical masks
3-Q	raspberry, . . .

With thanks to Phillip Cohen, who carries a grapefruit.

Eric Bagai can be reached at Box 9747.
North Hollywood, CA. 91609

ASSEMBLY 101
Z-80 without tears

by Lance Wolstrup

In our last installment I presented the Basic program called 'TRSLABEL/BAS, promising to translate it line-by-line into Assembly language as much as possible. Well, let's not procrastinate any further; get out the Basic listing and let's get down to work. I will present the Assembly language source code and include explanations and comparisons to the Basic listing.

```
00010  ;TRSLABEL/SRC
00020 ;assembles into TRSLABEL/CMD
00030  ;(c) 1989 TRSTimes & Lance Wolstrup
00040  ;written for Sep/Oct Assembly 101
00050  ;the Assembly language version
00060 ;
00070 ;
00080 ;
00090 ;
```

;Basic line 100 - CLEAR 500 - Is not needed; Instead, we must tell the assembler where to assemble our program in memory.

```
00100                ORG       7000H
```

;Basic line 110 - CLS - we will simply call the ROM routine that performs the screen clear function.

```
 00110  START        CALL      1C9H      ;cls
00120 ;
```

;Basic line 120 - PRINT@0,"TRSTimes Presents:"
all PRINT statements need to be broken down into two segments. 1. position the cursor, and 2. print the message. Though we really don't need to position the cursor, as we are printing @0 and have just performed a CLS, we will position it anyway, just for good measure.

```
00130                LD        HL,15360  ;screen pos 0
00140                LD        (4020H),HL;position  cursor
00150                LD        HL,MSG1
00160                CALL      21BH      ;display MSG1
00170                                      ;M1- CALL 4467H
```

;Basic line 130 - PRINT@28,"TRSLABEL"
same as above. Position cursor, then proceed to print the message at MSG2.

```
00180                LD        HL,15388  ;screen pos 28
00190                LD        (4020H),HL;position  cursor
00200                LD        HL,MSG2
00210                CALL      21BH      ;display MSG2
00220                                      ;M1-CALL 4467H
```

;Basic line 140 - PRINT@46,"(c) Lance Wolstrup"
same as above. Position cursor & print the message at MSG3

```
00230                LD        HL,15406  ;screen pos 46
00240                LD        (4020H),HL;position  cursor
00250                LD        HL,MSG3
00260                CALL      21BH      ;display MSG3
00270                                      ;M1-CALL 4467H
```

;Basic line 150 - PRINT@80,"a quickie mailing label program"
Again, same as the previous statements. Position the cursor and print the message at MSG4.

```
00280                LD        HL,15440  ;screen pos 80
00290                LD        (4020H),HL;position  cursor
00300                LD        HL,MSG4
00310                CALL      21BH      ;display MSG4
00320                                      ;M1-CALL 4467H
```

;Basic line 160 - PRINT@128,STRING$(64,131)
This, of course, draws a line of CHR$(131) characters from the left side to the right side of the screen. We will set up a DJNZ loop to accomplish this in Assembly language. Register B will contain the loop counter, while register HL will point to the screen position. In essence, this will act just like a FOR-NEXT loop that will POKE 131 into each screen position.

```
00330                LD        HL,15488  ;screen pos 128
00340                LD        B,64      ;B=loop counter
00350  LOOP          LD        (HL),131  ;POKE screen with 131
00360                INC       HL        ;next screen pos
00370                DJNZ      LOOP      ;continue until B=0
00380 ;
```

;Basic line 170 - PRINT@192,CHR$(31)
We need to erase from the start of the third line to the end of screen, so we position the cursor at 15552, then we load register A with the value 31; finally we call the ROM routine that will send one byte (rather than a string) to the screen.

```
00390  NEWLBL     LD        HL,15552  ;screen pos 192
00400                LD        (4020H),HL;pos  cursor
00410                LD        A,31      ;A=chr$(31)
00420                CALL      33H       ;print chr$(31)
00430 ;
```

;Basic line 180 - PRINT@269,"Name: "
position cursor and display message.

```
00440                LD        HL,15629  ;screen pos 269
00450                LD        (4020H),HL;position  cursor
00460                LD        HL,NAME
00470                CALL      21BH      ;display Name
00480                                      ;M1-CALL 4467H
```

;Basic line 190 - PRINT@330,"Address: "
position cursor and display message.

```
00490                LD        HL,15690  ;screen pos 330
00500                LD        (4020H),HL;position  cursor
00510                LD        HL,ADDRS
00520                CALL      21BH      ;display Address
00530                                      ;M1-CALL 4467H
```

;Basic line 200 - PRINT@384,"City, State & Zip: "
position cursor and print message.

```
00540                LD        HL,15744  ;screen pos 384
00550                LD        (4020H),HL;position  cursor
00560                LD        HL,CITY   ;display
00570                CALL      21BH      ;City, State & Zip
00580                                      ;M1-CALL 4467H
```

;Basic line 210 - PRINT@458,"Country: "
position cursor and print message.

```
00590                LD        HL,15818  ;screen pos 458
00600                LD        (4020H),HL;position  cursor
00610                LD        HL,CNTRY
00620                CALL      21BH      ;display Country
00630                                      ;M1-CALL 4467H
```

```
;Basic line 220 - PRINT@524,"Other: "
position cursor and print message.
00640                LD          HL,15884  ;screen pos 524
00650                LD          (4020H),HL;position cursor
00660                LD          HL,OTHER
00670                CALL        21BH      ;display Other
00680                                      ;M1-CALL 4467H

;Basic line 230 - LO=275
this is the start of the first input field (Name:)
00690                LD          HL,15635  ;screen pos 275

;Basic line 240 - GOSUB 810
line 810 is the INKEY$ subroutine that will draw a line of
CHR$(95) to indicate the length of the input field. Then it will
allow input and normal line editing.
00700                CALL        INKEY     ;gosub INKEY

;Basic line 250 - IF I$=CHR$(31) then 580
this line is not needed in our AL program. In Basic you MUST
always return from a subroutine. This is not necessary in As-
sembly language, as we have the power to control the stack.
Therefore, if the <CLEAR> key was pressed during input, the
AL subroutine will send us directly to the appropriate routine.

;Basic line 260 - LB$(1)=A$
transfer the input stored in A$ into LB$(1). Assembly language
does not deal with variables; instead, the AL INKEY routine
stored the NAME input in a series of memory locations. These
memory locations just happened to be the screen locations
where the input was typed. We now need to copy it into
another series of memory locations (usually referred to as a
buffer) that we will designate as NABUF, so we point register
HL to the beginning of the screen input; we then point register
DE to where we wish to place a copy of the input. Finally, we
call the routine that will perform that action.
00710                LD          HL,15635  ;point HL to input
00720                LD          DE,NABUF;point DE to buffer
00730                CALL        MOVEIN    ;move input
00740                                      ;to buffer

;Basic line 270 - LO=339
start of the second input field. (Address:)
00750                LD          HL,15699  ;screen pos 339

;Basic line 280 - GOSUB 810
draw input line - allow input and editing
00760                CALL        INKEY     ;gosub INKEY

;Basic line 290 - IF I$=CHR$(31) THEN 580
is not needed for the same reasons as listed above.

;Basic line 300 - LB$(2)=A$
transfer the address information into LB$(2). In AL we transfer
the information into another buffer. This one we call ADBUF.
00770                LD          HL,15699  ;point HL to input
00780                LD          DE,ADBUF  ;point DE to buffer
00790                CALL        MOVEIN    ;move input
00800                                      ; to buffer

;Basic line 310 - LO=403
start of the third input field. (City, State & Zip)
00810                LD          HL,15763  ;screen pos 403

;Basic line 320 - GOSUB 810
draw input line - allow input and editing
00820                CALL        INKEY     ;gosub INKEY

;Basic line 330 - IF I$=CHR$(31) THEN 580
this line is not needed for the reasons listed previously.

;Basic line 340 - LB$(3)=A$
 transfer the City, State & Zip information into LB$(3). We copy
the info from the screen into a buffer we call CIBUF.
00830                LD          H,15763   ;point HL to input
00840                LD          DE,CIBUF  ;point DE to buffer
00850                CALL        MOVEIN    ;move input
00860                                      ;to buffer

;Basic line 350 - LO=467
start of the fourth input field. (Country:)
00870                LD          HL,15827  ;screen pos 467

;Basic line 360 - GOSUB 810
draw input line - allow input and editing.
00880                CALL        INKEY     ;gosub INKEY routine

;Basic line 370 - IF I$=CHR$(31) THEN 580
is not needed for the reasons listed previously.

;Basic line 380 - LB$(4)=A$
transfer the country information into LB$(4). We copy the info
from the screen into the buffer we call COBUF.
00890                POP         HL        ;point HL to input
00900                LD          DE,COBUF;point DE to buffer
00910                CALL        MOVEIN    ;move input
00920                                      ;to buffer

;Basic line 390 - LO=531
start of the fifth input field. (Other:)
00930                LD          HL,15891  ;screen pos 531

;Basic line 400 - GOSUB 810
draw input line - allow input end editing
00940                CALL        INKEY     ;gosub INKEY routine

;Basic line 410 - IF I$=CHR$(31) THEN 580
line is not needed - reasons listed previously.

;Basic line 420 - LB$(5)=A$
transfer the 'other' information into LB$(4). We copy the info
from the screen in a buffer we call OTBUF.
00950                LD          HL,15891  ;point HL to input
00960                LD          DE,OTBUF;point DE to buffer
00970                CALL        MOVEIN    ;move input
00980                                      ;to buffer

;Basic line 430 - PRINT@192,CHR$(31)
we erase from the beginning of line 3 to the end of screen.
position cursor, load register A with 31 and then call the ROM
routine that sends one byte to the screen. This is the begin-
ning of the edit routine.
00990  SHOWIT       LD          HL,15552  ;screen pos 192
01000                LD          (4020H),HL;position cursor
01010                LD          A,31      ;A=chr$(31)
01020                CALL        33H       ;PRINT chr$(31)
01030 ;

;Basic line 440 - PRINT@271,"1. - ";LB$(1)
01040                LD          HL,15631  ;screen pos 271
```

```
01050                LD          (4020H),HL;position  cursor
01060                LD          HL,NA1      ;point HL to input
01070                CALL        21BH        ;display line
01080                                        ;M1-CALL 4467H

;Basic line 450 - PRINT@335,"2. - ";LB$(2)
01090                LD          HL,15695    ;screen pos 335
01100                LD          (4020H),HL;position  cursor
01110                LD          HL,AD1      ;point HL to input
01120                CALL        21BH        ;display line
01130                                        ;M1-CALL 4467H

;Basic line 460 - PRINT@399,"3 - ";LB$(3)
01140                LD          HL,15759    ;screen pos 399
01150                LD          (4020H),HL;position  cursor
01160                LD          HL,CI1      ;point HL to input
01170                CALL        21BH        ;display input
01180                                        ;M1-CALL 4467H

;Basic line 470 - PRINT@463,"4 - ";LB$(4)
01190                LD          HL,15823    ;screen pos 463
01200                LD          (4020H),HL;position  cursor
01210                LD          HL,CO1      ;point HL to input
01220                CALL        21BH        ;display input
01230                                        ;M1-CALL 4467H

;Basic line 480 - PRINT@527,"5 - ";LB$(5)
01240                LD          HL,15887    ;screen pos 527
01250                LD          (4020H),HL;position  cursor
01260                LD          HL,OT1      ;point HL to input
01270                CALL        21BH        ;display input
01280                                        ;M1-CALL 4467H
```

;Note that in the above 5 routines we pointed register HL to buffers NA1, AD1, CI1, CO1 and OT1 instead of the buffers where the inputs are stored (NABUF, ADBUF, CIBUF, COBUF and OTBUF). Look at the end of the listing and you'll see that each of the regular buffers is preceded by a smaller buffer that, if addressed, will continue right into the normal buffer. For example, the small buffer NA1 contains the values for "1 - " and, since it has no termination byte, it continues right into NABUF where the NAME input is stored.

;Basic line 490 - PRINT@640,CHR$(31)
we erase from the beginning of the 10th line to the end of screen. Position cursor and load register A with 31. Then call the ROM routine that sends one byte to the display.

```
01290  PROMPT        LD          HL,16000    ;screen pos 640
01300                LD          (4020H),HL;position  cursor
01310                LD          A,31        ;A=chr$(31)
01320                CALL        33H         ;PRINT chr$(31)
01330  ;
```

;Basic line 500 - PRINT@646,"Press number of field to change - or press 0 to print"
position cursor and print message.

```
01340                LD          HL,16006    ;screen pos 646
01350                LD          (4020H),HL;position  cursor
01360                LD          HL,PMSG1
01370                CALL        21BH        ;display PMSG1
01380                                        ;M1-CALL 4467H
```

;Basic line 510 - PRINT@708,"Press (ENTER) for new label - press (CLEAR) for YOUR label"
position cursor and print message.

```
01390                LD          HL,16068    ;screen pos 708
01400                LD          (4020H),HL;position  cursor
01410                LD          HL,PMSG2
01420                CALL        21BH        ;display PMSG2
01430                                        ;M1-CALL 4467H
```

;Basic line 520 - PRINT@783,"Press (SHIFT)(UP-ARROW) to quit"
position cursor and print message.

```
01440                LD          HL,16143    ;screen pos 783
01450                LD          (4020H),HL;position  cursor
01460                LD          HL,PMSG3
01470                CALL        21BH        ;display PMSG3
01480                                        ;M1-CALL 4467H
```

;Basic line 530 - I$=INKEY$:IF I$="" THEN 530
The ROM call to 49H returns ONLY if a key has been pressed. The value of the key pressed is stored in register A.

```
01490  GETKEY        CALL        49H         ;get selection
```

;Basic line 540 - IF I$=CHR$(27) THEN CLS:END
compare contents of register A to 27. If they are equal then jump to the EXIT routine which erases the screen and returns to DOS.

```
01500                CP          27          ;is it (SHIFT)
01510                                        ;(UP-ARROW)
01520                JP          Z,EXIT      ;yes-exit
```

;Basic line 550 - IF I$=CHR$(13) THEN 170
if the <ENTER> key is pressed we go back to NEWLBL.

```
01530                CP          13          ;is it (ENTER)
01540                JP          Z,NEWLBL    ;yes-go do
01550                                        ;a new label
```

;Basic line 560 - IF I$="0" THEN 730
if the "0" key is pressed we go to the PRINT routine.

```
01560                CP          48          ;is it "0"
01570                JP          Z,PRINT     ;yes-print label
```

;Basic line 570 - IF I$<>CHR$(31) THEN 640
this line is changed from comparing NOT EQUAL to comparing EQUAL. If it is found to be CHR$(31) we jump to the routine called YOURS.

```
01580                CP          31          ;is it (CLEAR)
01590                JP          Z,,YOURS    ;yes-pick up
01600                                        ;YOUR label
```

;Basic lines 570 - 630 will be found in the routine called YOURS later in the listing.

;Basic line 640 - I=VAL(I$)
is not needed as we will check for the ASCII value of the keypress.

;Basic line 650 - IF I<1 OR I>5 THEN 530
if we get this far, obviously neither (SHIFT)(UP-ARROW), (ENTER), or (CLEAR) was pressed. Therefore it must be some other key. We must allow only keys "1", "2", "3", "4", or "5".

```
01600                CP          49          ;is it "1"
01610                JR          C,GETKEY    ;don't allow
01620                                        ;if smaller
01630                CP          54          ;is it "6"
01640                JR          NC,GETKEY;don't allow
01650                                        ;if equal
01660                                        ;or larger
```

```
;Basic line 660 - PRINT@640,CHR$(31)
we have come to the edit routine. Position cursor, but, since
our choice of edit line is stored in register A, we need to save it
temporarily. We do that by PUSHing AF onto the stack. Then
we continue to put 31 in register A (that is why we had to save
the previous contents) and call the routine to send one byte to
the screen.
01670 EDIT    LD      HL,16000  ;screen pos 640
01680         LD      (4020H),HL;position cursor
01690         PUSH    AF        ;save edit choice
01700         LD      A,31      ;A=chr$(31)
01710         CALL    33H       ;PRINT chr$(31)

;Basic line 670 - PRINT@651,"Change: ";LB$(I)
position cursor and print the message followed by the con-
tents of the chosen field.
01720         LD      HL,16011  ; screen pos 651
01730         LD      (4020H),HL;position cursor
01740         LD      HL,CHANGE;point HL to
01750         CALL    21BH      ;CHANGE &dsply
                                ;M1-CALL 4467H
;Note: we have displayed the prompt 'Change: ' but we
haven't yet displayed the contents of the chosen buffer. This
will happen in just a second.

;Basic line 680 - PRINT@719,"To: ";
position cursor and display message.
01760         LD      HL,16079  ;screen pos 719
01770         LD      (4020H),HL;position cursor
01780         LD      HL,TO     ;point HL to TO
01790         CALL    21BH      ;display it
                                ;M1-CALL 4467H
;we will now take care of the unfinished business from Basic
line 670. We need to display the current contents of the line to
edit. First position the cursor, then restore the edit choice by
POPping the stack into AF.
01800         LD      HL,16019  ;screen pos 659
01810         LD      (4020H),HL;position cursor
01820         POP     AF        ;restore choice

;continuing, we point register HL to screen position 723. For
now, however, we don't set the cursor. We will do that in just a
moment. Instead, we push this value onto the stack, not once,
but twice.
01830         LD      HL,16083  ;screen pos 723
01840         PUSH    HL        ;save it
01850         PUSH    HL        ;save it again

;having done this slight bit of house-keeping, we can continue
by finding out which key was pressed for the edit choice. If we
pressed "1" we go to the EDIT1 routine. This is pretty much like
the Basic statement: ON A GOTO 100,200,300,400,500.
01860         CP      49        ;is it "1"
01870         JR      Z,EDIT1   ;yes-edit name

;if "2" was pressed we go to the EDIT2 routine.
01880         CP      50        ;is it "2"
01890         JR      Z,EDIT2   ;yes-edit address

;if "3" was pressed we go to the EDIT3 routine.
01900         CP      51        ;is it "3"
01910         JR      Z,EDIT3   ;yes-edit city, etc

;if "4" was pressed we go to the EDIT4 routine.
01920         CP      52        ;is it "4"
01930         JR      Z,EDIT4   ;yes-edit country

;if we get here, neither 1,2,3, or 4 was pressed. Obviously then,
the keypress was "5". Here we finally display the missing part
from BASIC line 670. We POP off the screen position for the
edit and enter the INKEY routine. Coming out of the INKEY
routine, we POP the screen position of the start of the edit,
point register DE to the appropriate buffer and jump to the
COMMON routine which copies the edited line into the buffer.
01940         LD      HL,OTBUF  ;must be "5"
01950         CALL    21BH      ;display OTBUF
01960                           ;M1-CALL 4467H
01970         POP     HL        ;restore screen pos
01980         CALL    INKEY     ;go get new input
01990         POP     HL        ;restore screen pos
02000         LD      DE,OTBUF;point DE to OTBUF
02010         JR      COMMON    ;jump to common

;EDIT4 behaves exactly like the above routine, with the excep-
tion that COBUF is used as the buffer instead of OTBUF.
02020 EDIT4   LD      HL,COBUF;editing field "4"
02030         CALL    21BH      ;display COBUF
02040                           ;M1-CALL 4467H
02050         POP     HL        ;restore screen pos
02060         CALL    INKEY     ;go get new input
02070         POP     HL        ;restore screen pos
02080         LD      DE,COBUF;point DE to COBUF
02090         JR      COMMON    ;jump to common

;same as EDIT4 except we use CIBUF as the buffer instead of
COBUF.
02100 EDIT3   LD      HL,CIBUF  ;editing field "3"
02110         CALL    21BH      ;display CIBUF
02120                           ;M1-CALL 4467H
02130         POP     HL        ;restore screen pos
02140         CALL    INKEY     ;go get new input
02150         POP     HL        ;restore screen pos
02160         LD      DE,CIBUF  ;point DE to CIBUF
02170         JR      COMMON    ;jump to common

;same as EDIT3 except we use ADBUF as the buffer instead of
CIBUF.
02180 EDIT2   LD      HL,ADBUF  ;editing field "3"
```

```
02190                CALL     21BH      ;display ADBUF
02200                                   ;M1-CALL 4467H
02210                POP      HL        ;restore curs pos
02220                CALL     INKEY     ;get new input
02230                POP      HL        ;restore curs pos
02240                LD       DE,ADBUF  ;DE =ADBUF
02250                JR       COMMON    ;jump to common

;same as EDIT 2 except we use NABUF as the buffer instead
of ADBUF.
02260  EDIT1         LD       HL,NABUF  ;editing field "1"
02270                CALL     21BH      ;display NABUF
02280                                   ;M1-CALL 4467H
02290                POP      HL        ;restore curs pos
02300                CALL     INKEY     ;get new input
02310                POP      HL        ;restore curs pos
02320                LD       DE,NABUF;DE = NABUF
02330 ;

;this code is common to EDIT1,2,3,4, &5, so rather than write
it 5 individual times, we write it once here. This routine moves
the new input into the appropriate buffer and then proceeds
to jump to the routine that will display the edited label.
02340  COMMON        CALL     MOVEIN    ;move it into buffer
02350                JP       SHOWIT    ;show edited label
```

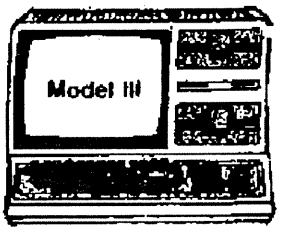
;Basic line 570 IF I$<>CHR$(31) THEN 640
is not needed here, as our ML inkey$ routine will send us to this routine only if the <CLEAR> key was pressed.

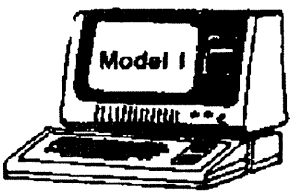
Basic line 580 LB$(1)="TRSTimes magazine"
position cursor, erase whatever text was stored there previously and point HL to where YOUR name is stored; then save the pointer to YOUR name, display YOUR name, restore the pointer, point register DE to NABUF and copy YOUR name to that buffer. NABUF is where the lprint routine will pick up the name.

Do note the very first instruction: POP HL. Here we are doing something we cannot do in most Basics. We are in effect eliminatiing the need to RETURN from a GOSUB. Keep in mind that we got here by pressing the <CLEAR> key from one of the prompts in the INKEY routine. That routine is accessed by CALL INKEY from many places in the NEWLBL routine. Since we want our preset label, we do not want to return to the Name, Address, City, State & Zip. etc. prompts; instead we want to proceed directly to the lprint routine. Thus, we strip off the RETurn address, which is stored at the top of the stack, by POPping it into HL. As we immediately put another value into HL, we are simply throwing the RETurn value away. Sure do wish I could do this as easily in Microsoft Basic!

```
02360  YOURS0        POP      HL        ;strip off RET
02370  YOURS         LD       HL,15635  ;screen pos 275
02380                LD       (4020H),HL;position cursor
02390                CALL     ERASE     ;erase prev chrs
02400                LD       HL,YNAME;HL=YOUR name
02410                PUSH     HL        ;save pointer
02420                CALL     21BH      ;display it
02430                                   ;M1-CALL 4467H
02440                POP      HL        ;restore pointer
02450                LD       DE,NABUF;DE = NABUF
02460                CALL     MOVEY     ;move YNAME
```

;Basic line 590 LB$(2)="20311 Sherman Way. Suite 221"





position cursor, erase whatever was there previously and point register HL to YOUR address. Save this pointer before displaying the address. After the display, restore the pointer, point register DE to ADBUF, and copy the address there for future use by the lprint routine.

```
02470                LD       HL,15699  ;screen pos 339
02480                LD       (4020H),HL;position cursor
02490                CALL     ERASE     ;erase prev chrs
02500                LD       HL,YADDRS;HL=YOUR addr
02510                PUSH     HL        ;save pointer
02520                CALL     21BH      ;display it
02530                                   ;M1-CALL 4467H
02540                POP      HL        ;restore pointer
02550                LD       DE,ADBUF  ;DE=ADBUF
02560                CALL     MOVEY     ;YADDRS to buffer
```

;Basic line 600 LB$(3)="Canoga Park, CA. 91306"
position cursor, erase whatever text might be here already and point register HL to the text at YCITY. Save HL on the stack, display the city, state and zip, restore the pointer to HL, point register DE to CIBUF and copy the text there for the lprint routine.

```
02570                LD       HL,15763  ;screen pos 403
02580                LD       (4020H),HL;position cursor
02590                CALL     ERASE     ;erase prev chrs
02600                LD       HL,YCITY  ;HL =YOUR city
02610                PUSH     HL        ;save pointer
02620                CALL     21BH      ;display it
02630                                   ;M1-CALL4467H
02640                POP      HL        ;restore pointer
02650                LD       DE,CIBUF  ;DE=CIBUF
02660                CALL     MOVEY     ;YCITY to buffer
```

;Basic line 610 LB$(4)="U.S.A"
position cursor, erase whatever text might already be here and point register HL to the text at YCNTRY. Save HL, display the country and restore the pointer to HL. Point register DE to COBUF and copy the text here for use by the lprint routine.

```
02670                LD       HL,15827  ;screen pos 467
02680                LD       (4020H),HL;position cursor
02690                CALL     ERASE     ;erase prev chrs
02700                LD       HL,YCNTRY;HL=YOUR cntry
02710                PUSH     HL        ;save pointer
02720                CALL     21BH      ;display it
02730                                   ;M1-CALL 4467H
02740                POP      HL        ;restore pointer
02750                LD       DE,COBUF;DE=COBUF
02760                CALL     MOVEY     ;YCNTRY to buffer
```

;Basic line 620 LB$=""
position cursor, erase whatever text might already be here, point register HL to the text at YOTHER and display it. Then restore the pointer to HL, point register DE to OTBUF and copy the text there for future use by the lprint routine.

```
02770                LD       HL,15891  ;screen pos 531
```

```
02780               LD          (4020H),HL;pos  cursor
02790               CALL        ERASE      ;erase prev chrs
02800               LD          HL,YOTHER;HL=YOUR  othr
02810               PUSH        HL         ;save pointer
02820               CALL        21BH       ;display it
02830                                      ;M1-CALL  4467H
02840               POP         HL         ;restore pointer
02850               LD          DE,OTBUF;DE=OTBUF
02860               CALL        MOVEY      ;YOTHER to buff
```

;Basic line 630 GOTO 430
the basic routine at lines 430 - 530 displays the label, followed by the EDIT/PRINT, NEW LABEL, or QUIT prompts. Our ML routine does not display the label (this is already done), but instead we JumP directly to the prompt routine at PROMPT.

```
02870               JP          PROMPT     ;go get prompt
```

;Basic line 730 LPRINT LB$(1)
we point register HL to the text stored in the buffer called NABUF and then CALL the subroutine LPRINT, which will do the actual work of sending the name top the printer

```
02880 ;
02890 PRINT         LD          HL,NABUF   ;point HL to name
02900               CALL        LPRINT     ;print  name
```

;Basic line 740 LPRINT LB$(2)
point register HL to the text stored in the buffer called ADBUF and call the LPRINT routine.

```
02910 ;
02920               LD          HL,ADBUF   ;point HL to addrs
02930               CALL        LPRINT     ;print  address
```

;Basic line 750 LPRINT LB$(3)
point register HL to the text stored at CIBUF and then call the LPRINT routine.

```
02940 ;
02950               LD          HL,CIBUF   ;point HL to city
02960               CALL        LPRINT     ;print  city,st  &Zip
```

;Basic line 760 LPRINT LB$(4)
point register HL to the text at COBUF and then call the LPRINT routine.

```
02970 ;
02980               LD          HL,COBUF;point HL to cntry
02990               CALL        LPRINT     ;print  country
```

;Basic line 770 LPRINT LB$(5)
point register HL to the text at OTBUF and then call the LPRINT routine.

```
03000 ;
03010               LD          HL,OTBUF   ;point HL to other
03020               CALL        LPRINT     ;print  other
```

;Basic line 780 LPRINT
since a mailing label is 6 lines long, we need to issue another carriage return to get to the top of the next label. We do this in the subroutine called CR.

```
03030               CALL        CR         ;do one more CR
```

;Basic line 790 GOTO 430
the Basic program goes back and re-displays the label, followed by the prompts. We go directly to the portion that waits for the keystroke for your next selection.

```
03040               JP          GETKEY     ;go get prompt
```

;Basic line 800 END
first we erase the screen with the ROM call to 1C9H, then, since out stack has been maintained properly, we RETurn to DOS with a simple RET. Had our stack not been maintained, we could have exited to DOS by using the routine at 402DH with the instruction: JP 402DH.

```
03050 ;
03060 EXIT CALL     1C9H                   ;cls
03070               RET                    ;back to DOS
```

;this is the routine that erases the field before we enter new information there. The cursor position (start of the field to erase) has been stored in register HL before this routine is called. We use the DJNZ loop to put spaces on top of any previous characters that might be there. Register B is the loop counter. It is set to 33, and we will loop that many times from ERASE1 to DJNZ ERASE1. The loop puts a space into the location pointed to by HL; register HL is then incremented (now points to the next location), and register B is decremented by the DJNZ instruction. When register B contains 0 the loop is exited and we RET to the caller.

```
03080 ;
03090 ERASE         LD          B,35       ;# of chrs to erase
03100 ERASE1        LD          (HL),32    ;erase chr
03110               INC         HL         ;next chr
03120               DJNZ        ERASE1     ;cntnu until B=0
03130               RET
```

;Basic line 810 A$=""
this line is useless in Assembly language. We don't need it.

;Basic line 820 PRINT@LO,STRING$(34.95)
the cursor location is already set in register HL by the caller. We save it on the stack and proceed to draw a line of chr$(95)'s with the same type of DJNZ loop we used to erase the fields above. The difference, of course, is that now we store 95 in the location pointed to by HL. Finally, we restore the cursor position by POPping it back into HL.

```
03140 ;
03150 INKEY         PUSH        HL         ;save cursor pos
03160               LD          B,34       ;B is chr counter
03170       LOOP2   LD          (HL),95    ;POKE scrn with 95
03180               INC         HL         ;next screen pos
03190               DJNZ        LOOP2      ;cntnu until B=0
03200 ;
03210               POP         HL         ;restore cursor pos
```

;Basic line 830 L=0
this is the character count. We will use register B to store this.

```
03220               LD          B,0        ;B will count chrs
```

;Basic line 840 I$=INKEY$
;Basic line 850 IF I$="" THEN 840
we will use the ROM routine at 49H. It returns only if a key has been pressed. The ASCII value (in hex) of the pressed key is stored in register A on return.

```
03230 INKEY1        CALL        49H        ;ROM inkey$ routine
```

;Basic line 860 IF I$=CHR$(13) THEN PRINT@LO,CHR$(30):RETURN
we ComPare the contents of register A with 13 (ENTER). If the comparison is found NOT EQUAL (NZ flag set) we jump up to the next comparison at INKEY2. However, if equal, we place the cursor at the position of the input line where <ENTER> was pressed. We then load register A with 30 and call the ROM

routine at 33H that sends one byte to the screen, thus erasing the line from the cursor position to the end of the line. Then we return to the caller.

```
03240                CP      13          ;is it (ENTER)
03250                JR      NZ,INKEY2;no-cntnu
03260                LD      (4020H),HL;cursor  position
03270                LD      A,30        ;A=chr$(30)
03280                CALL    33H         ;Print chr$(30)
03290                RET                 ;return to caller
```

;Basic line 870 IF I$=CHR$(8) AND L=0 THEN 840
first we check if the character is the <LEFT ARROW> (8). If not, the comparison will set the NZ flag and we go on to the INKEY3 routine. Now, if the character is chr$(8), we need to find out if we have any characters to erase. Since the character count is stored in register B, and register A is the only register that is capable of comparing anything, we must copy the the character count into register A. By using the instruction OR A, we effective check if the contents of A is 0, as the Z flag is set if so. Therefore, we disallow a backspace by jumping back to INKEY1 if we have no characters.

```
03300  INKEY2        CP      8           ;is it (left arrow)
03310                JR      NZ,INKEY3;no-cntnu
03320                LD      A,B         ;get chr count
03330                OR      A           ;is chr count 0
03340                JR      Z,INKEY1    ;if 0-do not allow
```

;Basic line 900 IF I$=CHR$(8) THEN L=L-1:A$=LEFT$(A$,L): PRINT@LO+L,CHR$(95);:GOTO 840
since we didn't jump to either INKEY3 or INKEY1, the <LEFT ARROW> was pressed and we have already input characters, so we decrement the cursor position and overwrite whatever character was there with the underline character (95). Finally, we decrement the character counter (B) and go back for the next keypress at INKEY1.

```
03350                DEC     HL          ;dec screen pos
03360                LD      (HL),95     ;POKE 95 to scrn
03370                DEC     B           ;one less chr
03380                JR      INKEY1      ;next key press
```

;Basic line 880 IF I$=CHR$(9) THEN 840
check if <RIGHT ARROW> is pressed. If so, lock it out by jumping back to INKEY1.

```
03390  INKEY3        CP      9           ;is it (right arrow)
03400                JR      Z,INKEY1    ;yes-do not allow
```

;Basic line 890 IF I$=CHR$(10) THEN 840
check if <DOWN ARROW> is pressed. If so, lock it out by jumping back to INKEY1.

```
03410                CP      10          ;is it (down arrow)
03420                JR      Z,INKEY1    ;yes-do not allow
```

;Basic line 920 IF I$=CHR$(31) THEN RETURN
now we check if the <CLEAR> key is pressed. If the contents of register A is 31, then the Z flag is set and we jump to the routine YOURS0 and get the preset label.

```
03430                CP      31          ;is it (CLEAR)
03440                JP      Z,YOURS0    ;yes-get YOUR lbl
```

;Basic line 910 IF I$<CHR$(31) THEN 840
since we have already taken care of <ENTER> (13), <LEFT ARROW> (8), <RIGHT ARROW> (9) and <DOWN ARROW> (10), we will lock out any other potential keystrokes producing a value less than 31. (yes, I know, we could have used this line earlier to lock out 9 & 10!) Note that when the comparison sets the C flag, it means that the value in A is less than the value it is being compared to.

```
03450                JR      C,INKEY1    ;don't allow <31
```

;Basic line 930 IF L=34 THEN 840
if the character count is already 34 we must not allow another character. In Basic we just go back to line 840. In AL it takes a little more housekeeping.
As register A is the only register capable of doing comparisons, we need to copy the contents of register B over to A. But wait... register A holds the value of the pressed key - we can't lose this value; so before doing anything else, we copy the keypress value from register A over to register C. (don't get confused, the C register - not the C flag). At this point in our program the C register is unused, so it is a good place to store a temporary value. Now we can copy the character count over to register A and compare it to 34. If the comparison is equal, it sets the Z flag and we jump to INKEY1, thus not allowing the keystroke.

```
03460                LD      C,A         ;store chr in C
03470                LD      A,B         ;get chr count
03480                CP      34          ;is it 34
03490                JR      Z,INKEY1    ;yes-no more
```

;Basic line 940 PRINT@LO+L,I$;:A$=A$+I$:L=L+1: GOTO 840
getting here means that the keypress produced a valid character and the character count has not yet reached the maximum of 34. We now copy the character value, temporarily stored in register C, back to register A, then putting the character value into the memory location pointed to by register HL (this is the screen location). Register B (the character counter) is incremented and we go back to INKEY1 to allow another keypress.

```
03500                LD      A,C         ;get chr back in A
03510                LD      (HL),A      ;display it
03520                INC     HL          ;next screen pos
03530                INC     B           ;inc chr count
03540                JR      INKEY1      ;back for next chr
```

;the following subroutines are not used in the Basic program, but make our AL program easier to manage.
this routine takes the typed input of a field and copies it into a buffer. Before entering this routine, register HL points to the screen location of the start of the input field, and register DE points to the buffer.
First we need to check if the field is blank (that is,was <ENTER> pressed as the first character). We do that by copying the character count from register B into register A. Now we check if the contents of A is 0 by ORing A with itself. (we could have done CP A,0, but OR A is one byte shorter and both sets the Z flag if the comparison is 0. If so, we jump down to the ENDBYT portion of the routine.
If the character count is larger than 0 we copy the character from the screen location pointed to HL into A, and copy that character into the buffer location pointed to by register DE. The screen location (HL) and the buffer location (DE) are then incremented. Finally, using the DJNZ loop, we continue to copy all the characters from the input field into the buffer. When completed, we return to the caller.

```
03550  ;
03560  MOVEIN       LD      A,B         ;chr count to A
03570                OR      A           ;is count 0
03580                JR      Z,ENDBYT    ;jump if field is blank
03590                LD      A,(HL)      ;put chr into A
03600                LD      (DE),A      ;put end byte in buffer
```

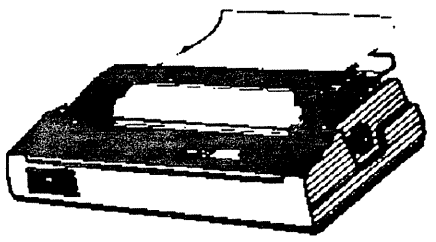
```
03610                INC     HL
03620                INC     DE
03630                DJNZ    MOVEIN
03640  ENDBYT        LD      A,3
03650                LD      (DE),A
03660                RET                     ;return to caller
```

;this is the routine that actually prints the label. Before entry the caller has pointed register HL to the correct buffer. The first character found there is copied into register A and compared to 3 (this ending byte (3) acts just like the semi-colon on the Basic PRINT and LPRINT statements). If it is 3 we need to convert it to a CR (we don't want the name & address on the same line) and we jump to the routine called CONVCR. If not 3, we must check if the character is the other ending byte (13). If so, we have come to the end of the label line in the buffer and need to jump to the routine (CR) where we issue the carriage return to the printer and return to the caller.
If neither 3 or 13, we have a regular character which must be printed, using the ROM LPRINT routine at 3BH. This routine prints one character at a time. We then increment register HL to point to the next character in the buffer, and we go back and repeat the whole procedure.

```
03670 ;
03680  LPRINT        LD      A,(HL)    ;move chr into A
03690                CP      3         ;is it end byte
03700                JR      Z,CONVCR;convert to CR
03710                CP      13        ;is it the end byte
03720                JR      Z,CR      ;yes-go do CR
03730                CALL    3BH       ;byte to printer
03740                INC     HL
03750                JR      LPRINT    ;go get next chr
03760  CONVCR        LD      A,13      ;make it a CR
03770  CR            CALL    3BH       ;CR to printer
03780                RET                     ;return to caller
```

;we need this routine when the <CLEAR> key was pressed, indicating we want our preset label. This moves the preset label from its buffers to the print buffers. Again registers HL and DE are loaded with the location of the preset label buffers and the print buffers respectively, by the caller.
The first character from the preset label buffer, pointed to by HL, is copied into register A. If this character is the end byte, we jump to the ENDBYT routine where a '3' is put in the print buffer. If not the end byte, we put the character in the buffer location pointed to by DE. Then registers HL and DE are incremented, and we go back to the MOVEY routine, repeating until we encounter the end byte.

```
03790 ;
03800  MOVEY        LD      A,(HL)    ;move chr into A
03810                CP      3         ;is it the end byte
03820                JR      Z,ENDBYT  ;yes-jump
```



```
03830                LD      (DE),A    ;put chr in buffer
03840                INC     HL        ;next chr
03850                INC     DE        ;next buffer loc
03860                JR      MOVEY     ;go do it again

;this is the message area
03870 ;
03880  MSG1          DEFM    'TRSTimes Presents:'
03890                DEFB    13
03900 ;
03910  MSG2          DEFM    'TRSLABEL'
03920                DEFB    13
03930 ;
03940  MSG3          DEFM    '(c) Lance Wolstrup'
03950                DEFB    13
03960 ;
03970  MSG4          DEFM    'a quickie mailing label
                             program'
03980                DEFB    13
03990 ;
04000  NAME          DEFM    'Name: '
04010                DEFB    13
04020 ;
04030  ADDRS         DEFM    'Address: '
04040                DEFB    13
04050 ;
04060  CITY          DEFM    'City, State & Zip: '
04070                DEFB    13
04080 ;
04090  CNTRY         DEFM    'Country: '
04100                DEFB    13
04110 ;
04120  OTHER         DEFM    'Other: '
04130                DEFB    13
04140 ;
04150  YNAME         DEFM    'TRSTimes magazine'
04160                DEFB    3
04170  YADDRS        DEFM    '20311 Sherman Way.
                             Suite 221'
04180                DEFB    3
04190  YCITY         DEFM    'Canoga Park, CA. 91306'
04200                DEFB    3
04210  YCNTRY        DEFM    'U.S.A.'
04220                DEFB    3
04230  YOTHER        DEFM    ' '
04240                DEFB    3
04250 ;
04260  PMSG1         DEFM    'Press number of field to change'
04270                DEFM    ' - or press 0 to print'
04280                DEFB    13
04290 ;
04300  PMSG2         DEFM    'Press (ENTER) for new label'
04310                DEFM    ' - press (CLEAR) for YOUR label'
04320                DEFB    13
04330 ;
04340  PMSG3         DEFM    'Press (SHIFT)(UP ARROW) to quit'
04350                DEFB    13
04360 ;
04370  CHANGE        DEFM    'Change: '
04380                DEFB    3
04390  TO            DEFM    'To: '
04400                DEFB    3

;this is the buffer storage area
04410  NA1           DEFB    49
```

```
04420                   DEFB    32
04430                   DEFB    45
04440                   DEFB    32
04450  NABUF            DEFS    35
04460  ;
04470  AD1              DEFB    50
04480                   DEFB    32
04490                   DEFB    45
04500                   DEFB    32
04510  ADBUF            DEFS    35
04520  ;
04530  CI1              DEFB    51
04540                   DEFB    32
04550                   DEFB    45
04560                   DEFB    32
04570  CIBUF            DEFS    35
04580  ;
04590  CO1              DEFB    52
04600                   DEFB    32
04610                   DEFB    45
04620                   DEFB    32
04630  COBUF            DEFS    35
04640          ;
04650  OT1              DEFB    53
04660                   DEFB    32
04670                   DEFB    45
04680                   DEFB    32
04690  OTBUF            DEFS    35
04700  ;
04710                   END     START
```

Believe me, I did not intend this installment to be 9 pages long. However, the material is such that I could not justify splitting it over two issues. I do hope that you found the concept, taking a Basic program, translating it to Assembly language almost line by line, of value. It is certainly something I would have liked when I first started with the mysteries of Z-80 code.

In our next installment we will switch over to look at Assembly language for Model 4. Again, we will be using EDTASM. "Wait a minute", you say, "there never was a version of EDTASM for Model 4." Very true, but 80 Micro published an article and program in their August 1984 issue on page 42. It was called 'MODEL 4 EDTASM FOR FREE', written by Douglas Payne. The program, reasonably short, loads EDTASM into Model 4 memory, modifies it, and then executes a working Mod 4 version of EDTASM.

My recommendation is to read the article and type in the program. However, if you send me a disk and $1.00 for s&h (North America only) **AND** a photo copy of the cover of the August 1984 issue of 80-Micro, I will copy my working version of the program to your disk and return it to you. For obvious reasons, there will be no exceptions to the photo copy rule.

Until next time....think Z-80.

TRS-80 GRAPHIC CODES

tutorial by Fred Blechman

The TRS-80 Models I & III display screen is divided into 1024 printing locations, 0 to 1023 (64 across by 16 down). Model 4, employing 80 columns by 24 rows, has 1920 printing locations, 0 to 1919. Normally, each of these locations is occupied by a letter, number, symbol or blank, i.e., a character. Each of these locations is a rectangular area divided into six segments (two columns of three rows each), as shown in Fig. 1. By proper use of the CHR$ command, you can light any single segment or combination of segments on your display. By putting these combinations together, you can form symbols, shapes, large letters, simulated playing fields, etc.

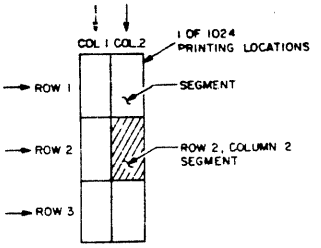


Fig. 1. How a printing location is divided into segments.

Fig. 2 shows the graphic code for each of the possible 64 segment "on" combinations, from "all segments off" (128) to "all segments on" (191). These are used in a program as CHR$(number). For example, if you used CHR$(157) in a program after a PRINT instruction, you'd light all column 1 segments as well as the column 2 segment if row 2 at the current printing location. As another example, CHR$(140) lights both columns of row 2 at the current printing location.

Actually, this graphic code is based on a binary code and is easy to remember if you crack the code. Look at Fig. 3. Notice that each of the six segments in a printing location is assigned a decimal number representing the powers of 2. Going from left to right, and from top to bottom, starting with 1, each number is exactly twice the previous number. This is the basis of binary counting.

To determine the TRS-80 graphic code number, just add the numbers of each lighted segment, and then add 128! Fig.3 shows some examples. Now you won't have to have Fig. 2 handy all the time, since you'll be able to quickly determine the number you want for every one of the possible 64 combinations. Remember, 128 is a total

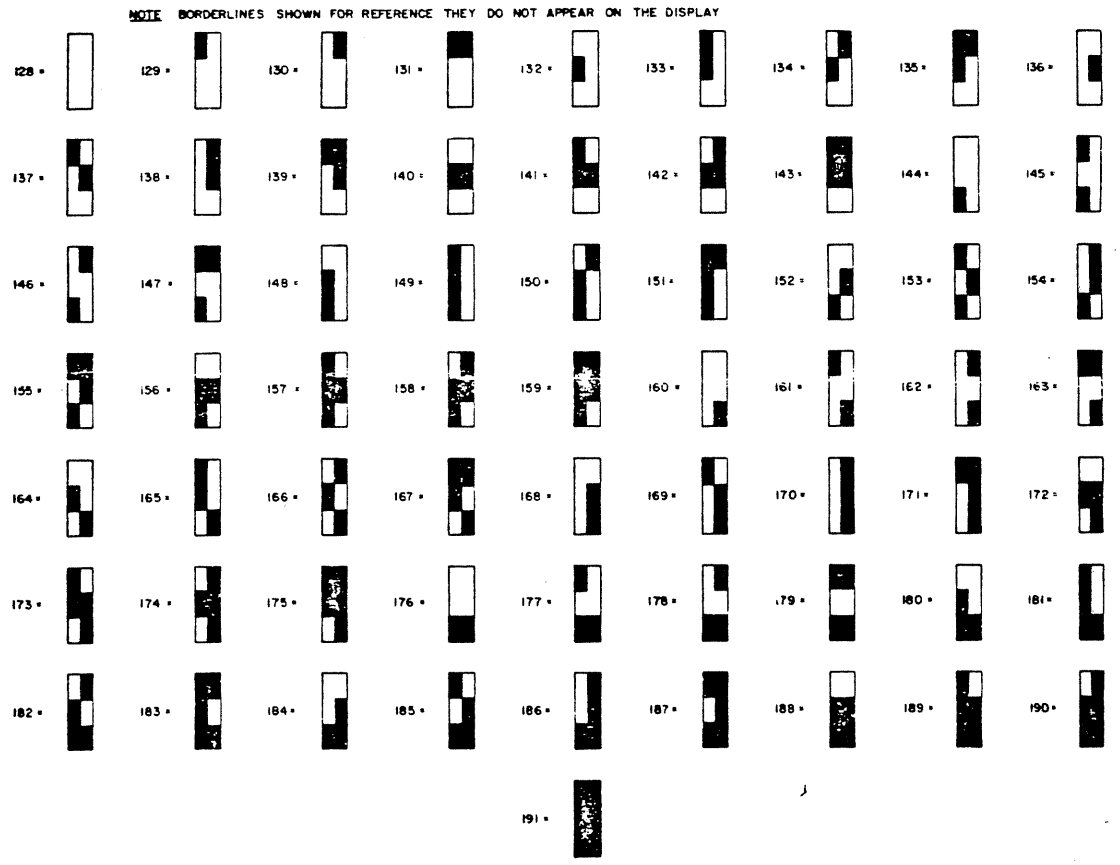


Fig. 2. TRS-80 graphic code. (■ = lighted segment)

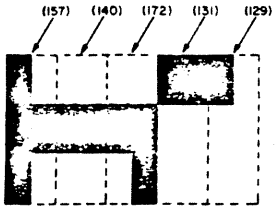


Fig. 4. Graphic horse or dog using five printing locations.

blank space, and 63 (total of all segments) plus 128 is equal to 191, a fully lighted soace.

To design your own symbol or large letters, use the "TRS-80 Video Display Worksheet" in your manual. Simply draw lightly in pencil whatever shape you want, noting that the heavier lines on the worksheet form the 1024 (Model I & III) or 1920 (Model 4) printing location rectangles, and the lighter lines subdivide these rectangles into six segments each. Now, convert your design to the proper combination of CHR$ numbers to "draw" this shape on your display screen.

An example will make this more understandable. Look at Fig. 4. This is a symbolic horse or dog composed of five CHR$ number on a single display line.You can place this "dog" anywhere on your screen with PRINT@ instructions. How about a racing dog? Try the short program, which I call "RUN, SPOT, RUN", in Example 1.

Line 40 creates enough pause in the program for "Spot" to remain at one location long enough to be visually stable. Line 50 blanks out the space for the next image (try running without line 50 and watch what happens)! Line 70 (and the semicolons at the ends of lines 30 and 50) keeps "Spot" from being chopped into small pieces that scroll up the screen.

Perhaps you'd like to see your name in huge letters on the screen. Fine. Just follow the same procedure, but remember that you'll need several screen lines. Since you'll be commanding a relatively large number of locations on the screen, READ/DATA statements make sense. If you'd like to see *my* name in big letters (over 2 inches high), try the program in Example 2.

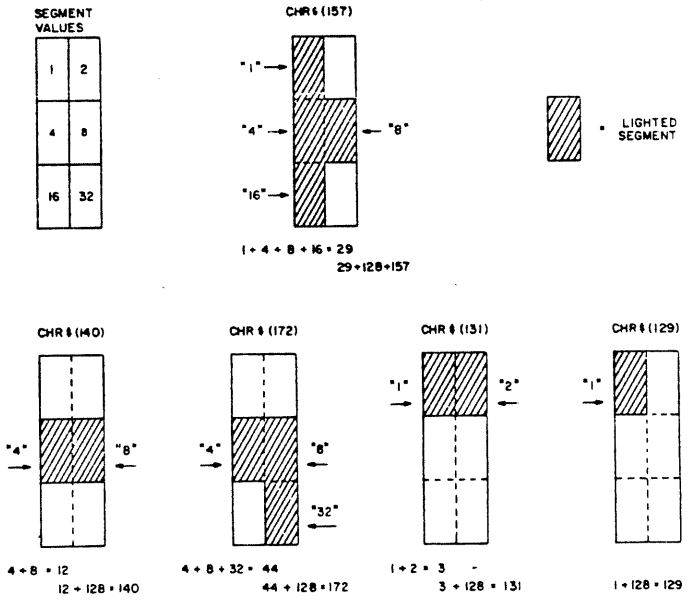


Fig. 3 Cracking the code.

This by no means exhausts the possibilities of using the graphic code. You are only really limited by your imagination, patience and the size of the computer memory. Example 3 shows a listing of a 5-dog race, with graphic dogs, a finish line and winner announcement.

```
10 CLS
20 X=0
30      PRINT@X,CHR$(157);CHR$(140);CHR$(172);
CHR$(131);CHR$(129)
40 FOR Y=1 TO 40:NEXT Y
50 PRINT@X,"   " REM 5 spaces
60 X=X+1
70 IF X=1018 THEN CLS:X=0
80 GOTO 30
```

Example 1 (Model I & III only)

```
10 CLS
20   PRINT@266,CHR$(191):GOSUB   500
30   PRINT@330,CHR$(191):GOSUB   500
40   PRINT@394,CHR$(191):GOSUB   500
50   PRINT@458,CHR$(191):GOSUB   500
60   PRINT@522,CHR$(191):GOSUB   500
65 GOTO 65
70    DATA    191,191,191,191,191,191,191,191,128,128
75    DATA    191,191,191,191,191,191,191,191,180
80    DATA    128,128,128,191,191,191,191,191,191,191

85    DATA    191,191,128,128,191,191,191,191,191,191,191,
189,144
90    DATA    191,191,128,128,128,128,128,128,128,128
95    DATA    191,191,191,128,128,128,179,191,191,157,128,128
100   DATA    191,191,191,128,128,128,128,128,128
105   DATA    128,128,191,191,191,128,128,139,191,191,191
110   DATA    191,191,191,191,191,128,128,128,128
115   DATA    128,191,191,191,191,191,191,191,191,135,128,
128,128
120   DATA    191,191,191,191,191,191,128,128,128
125   DATA    128,128,191,191,191,128,128,128,191,191,191
130   DATA    191,191,128,128,128,128,128,128,128
135   DATA    128,191,191,191,128,139,191,191,180,128,128,
128,128
140   DATA    191,191,191,128,128,128,128,128,128
145   DATA    128,128,191,191,191,128,128,184,191,191,191
150   DATA    191,191,128,128,128,128,128,128,128
155   DATA    128,191,191,191,128,128,130,191,191,189,176,
128,128
160   DATA    191,191,191,191,191,191,191,191,191
165   DATA    128,128,191,191,191,191,191,191,191,159,129
170 FOR R=1 TO 42
510 READ X
520 PRINT CHR$(X);
530 NEXT R
540 RETURN
```

Example 2 (Model I & III only)

```
5 REM * DOGRACE WITH GRAPHIC DOGS *
6 REM * FRED BLECHMAN *
```

```
7 REM * SET SPEED AT LINES 60-100 (A=A+2, ETC) *
10 CLS
15 FOR Y=3 TO 29:SET(123,Y):NEXT
20  A=64:B=192:C=320:D=448:E=576
21      PRINT@A,CHR$(157);"1";CHR$(172);CHR$(131;
CHR$(129);
22      PRINT@B,CHR$(157);"2";CHR$(172);CHR$(131);
CHR$(129);
23      PRINT@C,CHR$(157);"3";CHR$(172);CHR$(131);
CHR$(129);
24      PRINT@D,CHR$(157);"4"'CHR$(172);CHR$(131);
CHR$(129);
25      PRINT@E,CHR$(157);"5";CHR$(172);CHR$(131);
CHR$(129);
26 GOTO 55
30      PRINT@A,CHR$(157);"1";CHR$(172);CHR$(131);
CHR$(129);:GOTO  50
31      PRINT@B,CHR$(157);"2";CHR$(172);CHR$(131);
CHR$(129);:GOTO  50
32      PRINT@C,CHR$(157);"3";CHR$(172);CHR$(131);
CHR$(129);:GOTO  50
33      PRINT@D,CHR$(157);"4";CHR$(172);CHR$(131);
CHR$(129);:GOTO  50
34      PRINT@E,CHR$(157);"5";CHR$(172);CHR$(131);
CHR$(129);
50 IF A>120 THEN PRINT@720,"#1 WINS!!!":END
51 IF B>248 THEN PRINT@720,"#2 WINS!!!":END
52 IF C>376 THEN PRINT@720,"#3 WINS!!!":END
53 IF D>504 THEN PRINT@720,"#4 WINS!!!":END
54 IF E>632 THEN PRINT@720,"#5 WINS!!!":END
55 X=RND(5)
56 ON X GOTO 60,70,80,90,100
60 PRINT@A,"   ";:A=A+1:GOTO 30 REM 5 spaces
70 PRINT@B,"   ";:B=B+1:GOTO 31 REM 5 spaces
80 PRINT@C,"   ";:C=C+1:GOTO 32 REM 5 spaces
90 PRINT@D,"   ";:D=D+1:GOTO 33 REM 5 spaces
100 PRINT@E,"   ";:E=E+1:GOTO 34 REM 5 spaces
```

Example 3 (Model I & III only)

MORE GOODIES FOR YOUR TRS-80

Get the latest issue of TRSLINK

TRSLINK is the new disk-based magazine dedicated to providing continuing information for the TRS-80. A new issue is published monthly, featuring Public Domain programs, "Shareware", articles, hints & tips, nationwide ads, letters, and more. TRSLINK can be obtained from your local TRS-80 BBS, or download it directly from:

8/n/1 #4
(215) 848-5728
(Philadelphia, PA.)
Sysop: Luis Garcia-Barrio

Believe it or not:
TRSLINK is FREE

NEW PROGRAMS

from the Valley TRS-80 Hackers' Group
public domain library
for Model I, III & 4

Send SASE for annotated list
Sample disk $5.00 (US)

VTHG
BOX 9747
N. HOLLYWOOD, CA. 91609

* NEW *

Recreational & Educational Computing

Have you been missing out on the only publication devoted to the playful connection of computers and math?

The REC Newsletter features programming challenges and recreational math, such as:

the Magic of Schram's 123 String - the probability of an N game at Bingo - time to complete a collection - 6174 - Next Number in Sequence - Locate the Bomb - perfect numbers - Fibonacci numbers - prime number generation and contest - self-reference and paradoxes - self-listing program challenge and solution - pi - mystery programs explained - probability - Monte Carlo simulations.

Also:
Fractal art - the world's best card trick (based on algebra) - reviews of best software and books - editorial - humor - cartoons - art - reader solutions and more!

Programs supported for: TRS-80, Tandy, MS-DOS and others.

REC is available for $24.00 per calendar year of 8 issues

REC Newsletter
129 Carol Drive
Clarks Summit, PA. 18411
(717) 586-2784

A guided tour through MULTIDOS

by Andrew J. Bruns

A taste of MULTIDOS history

I first heard about MULTIDOS back in '81. At the time, I had a TRS-80 model I and had just bought a bare expansion interface, that extra black box with all the goodies. Being an experimenter, I decided that, in addition to adding memory chips, I should buy a couple of disk drives and so I wound up looking for a friendly DOS to operate them.

Back then, there were quite a few DOSes on the market and I started reading any system reviews I could find. What I discovered was that MULTIDOS had three things going for it: it was able to read other DOS formatted disks, so swapping would be no problem, it had a very powerful Basic language and it was relatively inexpensive - about $79 compared to the $129 of most other systems. So I bought MULTIDOS 1.6 in single density.

> Two specific features of MULTIDOS made it uniquely suited to the novice, and those were the Versatile File Utility (VFU) and SuperBasic

Right off, I realized that MULTIDOS was written with the user in mind and not the hacker. Booting up was simple, you got a banner and a request for the date. If you didn't want to input a date, just hit <ENTER> and all was well. It was loaded with Hot keys and sported a MiniDos that could be accessed from within an active program, almost like multi-tasking, and directories were available with ONE keystroke.

Two specific features of MULTIDOS, however, made it uniquely suited to the novice and those were the Versatile File Utility (VFU) and SuperBasic.

VFU was unique in that it handled large scale file manipulation in a simple and direct way for the user. It allowed the user to copy any number of files, visible or invisible, system or non-system, and on almost any DOS formatted disk. Transfer from one disk to another was accomplished by letting the user mark the files in the directory display that he/she wanted to have moved. Then the files would all be copied automatically. VFU could also create a hard copy of any disk directory that would fit in the 5 1/4" sleeve or purge out any unwanted files from a disk. Additionally, VFU could act as a program execution menu, causing programs on disk to be run simply by marking them on a menu created from the disk's directory and typing " Y" .

The VFU also was powerful in that it could bypass password protection. If you forgot that access password, VFU would copy the file anyway and remove the password in the process giving you an easily accessible copy again (for forgetful folk like me, this was almost paradise.)

There were several other DOS functions that were created to simplify user interactions too, like the CAT, BACKUP and RESTOR functions. The CAT command would create a read of any DOS directory and set the DCT to allow access, if possible. If DIR didn't work, CAT always did. The BACKUP command combined formatting a new disk with an image copy of all the files. This prevented problems from forgetting to use a clean freshly formatted diskette to backup a disk. Just put the source in one drive and a blank disk in the destination drive, type in BACKUP with the drive extensions, hit <ENTER> and you're ready to go grab a cup of coffee. Everything would be handled automatically. And for those of us who might accidentally purge or kill a hot file..(we know who we are) there was the RESTOR command which brought the file back from the dead. All in all, this was a really nice DOS and I enjoyed using it.

For users comfortable using Level 2 Basic, SuperBasic was like a brave new world. It had all the L2 Basic functions, but then went on to create several more ideally suited to the committed Basic program writer. I had originally been tutored in Fortran and took some habits into writing Basic with me. I always put spaces into statements to separate functions for easy reading and filled my programs with comments so that it was easier to figure out which variables did what and just what was going on. I also started out using a lot of subroutines to minimize my writing, but the programs ran like snails in the Basic interpreter. Then there was also the problem of debugging those programs efficiently. To the rescue came SuperBasic!

SuperBasic had a function to delete spaces in runtime basic programs and it was fast. Another function wiped comments from the code to speed the interpreter on its way. There was a special feature that "packed" Basic programs too. This seemed strange to me, but it actually forced the Basic interpreter to look at larger segments of code at one time. Translations were less frequent and so execution was quicker. These three features alone made my Basic programs more compact and much faster than before.

For troublesome debugging, there was BossBasic, the Kingpin of Basics. This Basic interpreter took a little more memory, but allowed a step-by-step execution with checking of the variables on the way. For major changes, there were several Global commands and you could renumber a program's statements in any fashion you pleased. BossBasic had the unique ability to push small Basic routines into high memory,separate from the normal Basic pro-

gram storage area, and hold them in line like having a RAM disk. This allowed me to load a "little treasure" from my disk and push it into high memory, out of the way. Then I could build a larger program and merge this gem in high memory into the existing code. This was a lot of power in the hands of a new disk drive owner and made MULTIDOS 1.6 a winner for me. It was not without its short comings, however.

MULTIDOS 1.6 didn't allow the user to allocate disk space to an empty file and chaining Basic programs that were larger than the memory into smaller segments without having to store and recover variable data was not possible. Version 1.6 also had a real nifty keyboard driver that did all kinds of things except when it was "conflicting" with other DOS code, which seemed like all the time ! So I never used that keyboard driver much.

When Model I disk driver doublers became available, I decided to get one and found that I needed a DOS upgrade to take advantage of double density. The upgrade that I got was MULTIDOS 1.71. This version handled double density without a hitch and allowed access to two sides of a drive (a big deal back then). It also let you see both sides of a diskette as two separate sides with their own directories or as a single disk with a common directory for both sides. This was needed for reading some other DOS formats and it did allow you to access each side of a drive as a separate drive. The keyboard driver had been significantly simplified and everything worked well together. There were enhancements to SuperBasic as well.

I neglected to mention that MULTIDOS 1.6 had a Basic program packer that was confused easily. If it was given a very complex program to pack, it often would create a program file that would bomb when executed. This meant using the packer required some art and prayer. Version 1.71 had a more polite and tolerant packer. It was also supposed to allow the support of a hard disk, but creating the driver was a user problem. At the time I got MULTIDOS 1.71, the upgrade cost me $10, so I felt I certainly got my money's worth.

Now with my data doubler and three disk drives (two one sided and one two sided) I generally got on with my life. This lasted until around the end of 1985. That was when I "discovered" telecommunications. I purchased Les Mikesell's MODEM80 terminal program, which was aging at the time, but it met my needs at 300 baud and it got me hooked. Fortunately, MODEM80 worked with MULTIDOS flawlessly. There were clouds on the horizon, however.

By 1987, we were all seeing libraries and archives in the telecommunications computers and BBS's. Some of the utilities worked in all DOSes, but the ones destined for fame weren't wholly compatible with MULTIDOS. I also was seeing references to a great public domain terminal program called"XTERM" and it wouldn't run under MULTIDOS. At this point, I went out and bought another DOS...*GASP!* I thought that my problems were over when I bought LDOS 5.1.4. I would transfer all my programs into this DOS and that would be that! With my years experience in software I was still foolishly optimistic!!

First,I discovered that my favorite word processor, Ed Levy's Word Machine 3.0 w/ Spell checker and Corrector, wouldn't work with LDOS. This meant that I would be using both DOSes unless I wanted to learn another word processor and spend the money. I was cheap, so I became a two DOS user from then on. Then the fun began, when on Jan 1, 1988, LDOS 5.1.4 wouldn't boot up, and I joined many other users who got very frustrated with LDOS. Of course, MULTIDOS kept on booting and running after the kill date for LDOS5.1.4. and so I was back to MULTIDOS again full-time.

Realizing that I was now hooked on XTERM and ARC31 for archiving and telecommunications, I had to find an LDOS fix, and I did. A developer had provided patches to make LDOS5.1.4 into 5.1.4+ with dates that worked again. After registering my copy of the shareware (the cost was minor to get back my LDOS for BBSing), all those boxes of diskettes would now have to be converted to LDOS 5.1.4+ and I was still using two DOSes.

The infamous dating problem wasn't escaped by MULTIDOS, either. While it would boot up and run fine, the dates stored in directories no longer reflected reality. This was frustrating and soon after, I upgraded from MULTIDOS 1.71 to the latest version, MULTIDOS 2.1. As of this moment, MULTIDOS appears to be the only DOS that supports a uniform interface between all the TRS-80 models, I/III/IV. You'll remember that LDOS is now in three versions, depending on the machine, 5.1.4+ or 5.1.5 for the model I (patches provided by non-LDOS sources), 5.3 for the model III and LS-DOS 6.X is for the model IV. There is a conversion for 5.3 to the model I in the public domain, but I have not been successful in getting it to work.

So I am now a committed user of two disk operating systems, each for it's own merits. Each also has its own rewards and frustrations.

While MULTIDOS is clearly user friendly and supports the Basic programmer fully, LDOS 5.1.4 has a hacker environment which is good for applications and probably the most advanced Job Control Language available for these machines today. Problem is, with different incompatible versions of LDOS available for each machine, the LDOS environment has now become almost unfriendly for universal applications. Hopefully, currently active writers will take heed and have another look at MULTIDOS for their applications.

Next issue will feature a review of MULTIDOS 2.1., which will focus on three areas, Library commands, System Utilities and that very excellent SuperBasic with it's enhancements.

Andrew Bruns can be reached at: RR#1, Box 226, Marble Hill Rd. Great Meadows, NJ. 07833.

HINTS & TIPS

More disk storage for Model I users

by Andrew J. Bruns

We've been told that the model I will not access more than 4 disk drives and no more than a total of 4 sides. This was certainly true using TRSDOS, but recently, I have discovered that it is possible to trick your system into seeing up to 7 disk sides. This means 3 double sided drives and one single sided drive. I know this works with LDOS 5.1.4 and with MULTIDOS.

When you set up your drives, you set the dip switches inside each drive to allow them to be accessed as drives 0,1,2,3. When you access a drive, you have probably noticed that while one drive access light goes on, all the drives are spinning.

Here is what appears to be possible! When a drive with two sides is accessed as two sides, the controller looks for data from drive 0 on lines 0 and 1. Drive 1 two sided is looked for on lines 1 and 2. Drive 2 is looked for on lines 2 and 3. Drive 3 can only be seen as one sided.

By putting a two sided drive in locations 1 and 2, for instance, and setting your device table to access each as a two sided drive, lines 2 & 3 do double duty. It should be noisy, but it appears to work flawlessly. I have a model I with drive 0 as a single side, but drive 1 & 2 are both run as two sided drives in both MULTIDOS and LDOS 5.1.4 and I get a lot of online storage this way. With 40 track drives,it is possible to access 900 K in double density in this configuration. 80 track drives would double that.

My setup is SSDD 40 track in drive 0, DSDD 40 track in drive 1, and DSDD 80 track in drive 2 for a total of 1.3 meg of floppy storage.

CHEAPO PRINTER ACCESSORIES & INKING

by Bruce Showalter

When applied to me, the letters TRS mean Tightwad Reaps Savings. Here are some tightwad tips on customizing your printer operation.

The first idea is a cheapo do-it-yourself printer stand. Actually, I got this idea from another computer user. And I'm sure that others have thought of it. The building material is PVC pipe, available from your hardware store or lumber yard. This might be called the adult version of "Tinker Toys". A durable stand can be built using 1/2" PVC sections, elbows and tees. Most joints fit tight enough that they don't even require cement (more $$$ saved).

A paper catcher behind the printer can also be built with PVC pipe, but I've found an even cheaper material: trouser hangers. My dry-cleaners use a wire hanger that has a cardboard tube for the horizontal member. By cutting this tube to any desired length and reconnecting to the wire portion, a variety of useful shapes can be had. Two or more hangers can be cut up and reassembled into paper guides and trays.

The price of commercial reinkink gadgets makes me shy away. But the savings of do-it-yourself reinking is most appealing. So, if you've got the time and the patience, you might try using a toothpick dipped in stamp pad ink and applied to your fabric ribbon. The ink is available from office-supply stores or in the stationary department of some discount stores. With some imagination, you could probably come up with a manual ribbon winder to speed up the process.

A word of caution: I've read that some kinds of ink will decrease the life of dot-matrix print heads. So you might wish to investigate this matter before reinking. Daisy-wheel printers, of course, are not affected.

Don't be too hasty in discarding obsolete or faulty printouts. If you've only printed on one side, save the back side for proof-runs or draft copies. Some people even cut up old printouts and staple the pieces together for memo pads.

Do you sometimes wish you had a rubber stamp, but can't justify the price because of a one-time or infrequent need? Well, your computer and printer can make good substitutes. If the documents you want to stamp will fit in the printer, just write a short BASIC program to print the message on the documents. If the documents won't fit the printer, buy some self-stick labels and print your message on them. Then you can attach the printed label to your documents. I print up a batch of address labels from time to time, for bills I pay where no pre-addressed envelope is provided.

So much for MY tightwad tips. How about sending YOURS to TRSTimes?

MODEL 4 HASHCODES

by Dale Parsons

I am running LS-DOS 6.3. on a Cat 1069-A Model 4 and would, of course, like to see expanded Mod 4 coverage. To get the ball rolling, here is a short program that will calculate filespec hash codes on the Model 4.

```
10 CLS:PRINT"HASHER, a routine for finding TRS80 file
spec hash codes"
20 PRINT"Enter file name with '/' and extension, if any,
as it appears"
30 PRINT"in the disk directory":PRINT
40 L1=0:R1=0
50 LINE INPUT"file name ";A$
60 A=LEN(A$):A1=INSTR(A$,"/"):IF A1=0 THEN 90
70 M$="":FOR I=1 TO 12-A:M$=M$+" ":NEXT
80 N$=LEFT$(A$,A1-1)+M$+RIGHT$(A$,A-A1):
GOTO 100
90 M$="":FOR I=1 TO 11-A:M$=M$+" ":NEXT:
N$=A$+M$
100 FOR I=1 TO 11:B=ASC(MID$(N$,I,1))
110 L=INT(B/16):R=B MOD 16
120 L=L XOR L1:R=R XOR R1:L=2*L:R=2*R
130 IF L15 THEN L=L-16:R=R+1
140 IF R15 THEN R=R-16:L=L+1
150 L1=L:R1=R
160 NEXT
170 IF R=0 AND L=0 THEN R=1
180 PRINT" the code is ";HEX$(L);HEX$(R):PRINT
190 INPUT"1 for more, else quit ";C:IF C=1 THEN 40
ELSE END
```

COUNT WORDS & CHARACTERS IN YOUR ASCII FILES

Model I, III & 4

by Dennis Burkholz

The following program should work on Model I, III & 4. It will read an ASCII file, do some internal math, and then tell you how many words and characters it found in the document. The program is bare-bones, no error checking has been written in, so be careful when entering the filename. If the file is not found, the program aborts to Basic with the infamous 'FILE NOT FOUND' message.

```
90 REM WDCOUNT/BAS
95 REM for I, III & 4 - 03/17/89
100 INPUT"FILENAME: ";F$
110 OPEN"I",1,F$:CH=0:WD=0
120 IF EOF(1) THEN 190 ELSE LINE INPUT#1,CH$:
Z=1
130 V=INSTR(Z,CH$," ")
140 IF V=0 OR V=LEN(CH$) THEN WD=WD+1:
CH=CH+LEN(CH$):GOTO 180
150 Z=V+1
160 IF MID$(CH$,Z,1)" " THEN WD=WD+1
170 GOTO 130
180 GOTO 120
190 PRINT"    Words = ";:PRINT USING"#####";WD
200 PRINT"Characters = ";:PRINT USING"#####";CH
```

EXTRA TRACKS ON YOUR FLOPPIES

by Andrew J. Bruns

Don't forget that 40 track drives can format 42 tracks and 80 track drives can format 84 tracks. The extra tracks mean more storage if you use them and both LDOS and MULTIDOS let you choose the number of tracks when formatting a floppy.

ANOTHER MASTER PASSWORD FOR LDOS AND OTHER GOODIES

by Michael E. Webb

Regarding the **R0S0LT0FF** ultimate password on LDOS, Guy Omer of */N/1 #1 BBS (904)-377-1200), who is my mentor in TRS-80, has figured out that a four digit equivalent is **P3UF**. I use P3UF routinely in my LDOS work, and it is superhandy, easier than using other passwords, or even remembering the others! It's handy in dealing with TRSDOS/LS-DOS 6.x disks as well. (*the master passwords will only access TRSDOS/LS-DOS 6.x. disks if used as data disks under LDOS. -Ed*)

I need to alert TRSTimes readers to a possible fatal bug in SCRPATCH/BAS (TRSTimes 1.5). At first, I thought that a part of the code had not been printed correctly in the issue, or that I had misread the code as printed. However, I had the same problem when I ran the version in TRSTimes on Disk. The problem is the error handling routines are fritzed in the finished program. I got the MODSCRIP program and SCRIPSIT 1.0 and it worked perfectly. I also tried it with the patches from the LDOS FIX DISK, and it worked fine. However, under SCRPATCHed SCRIPSIT 1.0., if a command is wrongly entered, instead of 'INVALID COMMAND' flashing at the bottom of the screen, a whole row of oddball characters flash across the bottom, and when following this I try to do something like PRINT or END, either the program will freeze, or it will freeze and flash more of those oddball characters over the screen. I have played with it some, and have been able to use it, but it has "Zero Tolerance" as far as mistakes in commands are concerned.

I would also like to share a patch with the readers. It is a patch to SCRIPSIT 3.2. which I culled from SCRIPT32/FIX on the LDOS FIX DISK and adapted for TRSDOS 1.3.-based SCRIPSIT.

The one line that changes the END routine from REBOOT to EXIT TO DOS, rewritten for TRSDOS 1.3. context, is:

```
PATCH SCRIPSIT/CMD
(ADD=6602,FIND=C30000,CHG=C32D40)
```

SET YOUR PRINTER CODES WITH EPSONSET

Model 4
EPSON FX & COMPATIBLES

by Robert M. Doerr

The Epson series of printers are probably the most powerful dot-matrix machines on the market today. Such is the power of these printers that other companies have for years copied their capabilities. Most printers today, be it STAR, CITIZEN, or whatever, will most likely have an "*Epson emulation mode*"; that is, the printer will respond to Epson printer codes, which have now become the standard of the dot-matrix industry.

These codes, though they allow the user access the vast power of Epson printing, are difficult to remember, and generally do not make any sense whatsoever. Below is a partial list of the codes and their functions:

CHR$(15)	compressed mode on
CHR$(18)	compressed mode off
CHR$(27)"M"	elite mode on
CHR$(27)"p"	elite mode off
CHR$(27)"W1"	expanded mode (continuous) on
CHR$(27)"W0"	expanded mode off
CHR$(14)	expanded mode (one line) on
CHR$(20)	expanded mode (one line) off
Default	pica mode
CHR$(27)"G"	double-strike mode on
CHR$(27)"H"	double-strike mode off
CHR$(27)"E"	emphasized mode on
CHR$(27)"F"	emphasized mode off
CHR$(27)"p1"	proportional mode on
CHR$(27)"p0"	proportional mode off
CHR$(27)"@"	master reset
CHR$(27)"4"	italics on
CHR$(27)"4"	italics off
CHR$(27)"S1"	subscript on
CHR$(27)"T"	subscript off
CHR$(27)"S0"	superscript on
CHR$(27)"T"	superscript off
CHR$(27)"-1"	underline on
CHR$(27)"-0"	undeline off
CHR$(8)	backspace
CHR$(7)	sound bell
CHR$(24)	cancel text in buffer
CHR$(127)	delete last text character
CHR$(27)"mx"	graphic characters. x=character
CHR$(27)"s1"	half speed mode on
CHR$(27)"s0"	half speed mode off
CHR$(27)">"	high order bit on
CHR$(27)"="	high order bit off
CHR$(27)"#"	high order bit normal
CHR$(27)"i1"	immediate mode on
CHR$(27)"i0"	immediate mode off
CHR$(17)	enable printer to receive data (default)
CHR$(19)	disable printer from receiving data
CHR$(10)	line feed
CHR$(13)	carriage return
CHR$(27)"2"	set 1/6 inch line spacing (default)
CHR$(27)"0"	set 1/8 inch line spacing
CHR$(27)"1"	set 7/72 inch line spacing
CHR$(27)"A"CHR$(x)	set x/72 inch line spacing
CHR$(27)"3"CHR$(x)	set x/216 inch line spacing
CHR$(12)	form feed
CHR$(27)"C"CHR$(0)CHR$(x)	set form length in inches. x=length in inches
CHR$(27)"C"chr$(x)	set form length in lines x=length in lines
CHR$(27)"8"	paper out sensor on
CHR$(27)"9"	paper out sensor off
CHR$(27)"N"CHR$(x)	skip over perforation on x=number of lines to skip
CHR$(27)"O"	skip over perforation off

To demonstrate some of these abilities, and to make life easier for myself, I wrote EPSONSET/BAS. The program, which acts as a menu from where one or more printer setting choices can be selected, also employs some of the powerful, but often overlooked, screen-control commands available for the Model 4 Basic programmer. These commands give the programmer access to a variety of useful built-in machine routines by using the easy PRINT CHR$(code) syntax.

The screen codes (in decimal) are as follows:

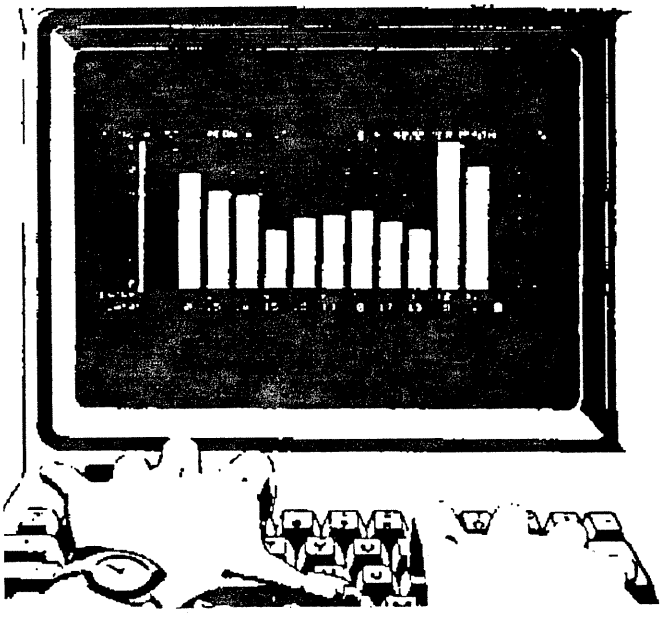
CHR$(8)	backspace with erase
CHR$(9)	tab to next tab position (tabs at fixed interval of 8)
CHR$(10)	move cursor to start of next line
CHR$(11)	move cursor to start of next line
CHR$(14)	turn cursor on
CHR$(15)	turn cursor off
CHR$(16)	reverse video on (high bit routine on; that is, 128 decimal is added to code of each subsequent key and display character.)
CHR$(17)	reverse video off (high bit routine off)
CHR$(21)	swap space compression & special characters
CHR$(22)	swap special characters & alternate characters
CHR$(23)	double-wide characters on (limit: 40 chrs per line)
CHR$(24)	backspace without erase
CHR$(25)	cursor right
CHR$(26)	cursor down
CHR$(27)	cursor up
CHR$(28)	cursor home (hight bit routine off - reverse video off - double-wide characters off. note: as CLS is nothing than the machine code version of 'PRINT CHR$(28);CHR$(31), it also performs the above functions.
CHR$(29)	erase line and place cursor at start of same line
CHR$(30)	erase from cursor to end of line
CHR$(31)	erase from cursor to end of screen

While I am listing things, I might as well give the codes generated by the function keys:

CHR$(129)	<F1>
CHR$(130)	<F2>
CHR$(131)	<F3>
CHR$(145)	<SHIFT><F1>
CHR$(146)	<SHIFT><F2>
CHR$(147)	<SHIFT><F3>

The function keys are not used in the program, but could easily have been employed to, for example, move the cursor instead of the arrow keys.

So, power up your Model 4, get into Basic, type in EPSONSET/BAS and gain some control over that monster-printer without always having to first consult the manual.



EPSONSET/BAS

```
1 GOTO 100 'Program to set up the Epson FX printer; R. M. Doerr
2 SAVE"EPSONSET/BAS:1"
3 SAVE"EPSONSET/ASC:1",A:
STOP
4 LPRINT CHR$(27) CHR$(64) CHR$(15) CHR$(7) CHR$(27) CHR$(78) CHR$(12);:
RETURN
5 GOSUB 4:
LLIST:
STOP
100 PRINT CHR$(15);:
OPTION BASE 1:
TR=4:
BR=18:
DIM C(BR-TR+1,2),H(2):
H(1)=10:
H(2)=67:
N=192:
ESC$=CHR$(27):
GOSUB 1000:
PRINT:
PRINT"For printer setup, don't forget the valuable FORMS comand. (See DOS manual.)"
110 PRINT"Invocations via EPSONSET interact with those invoked via FORMS;":
PRINT"for example, the physical width of a margin set as N characters via FORMS":
PRINT"will depend on the character width set via EPSONSET"
120 PRINT:
PRINT"Often it is best to switch the printer off, then on, before using EPSONSET.":
PRINT"This cancels all user- or software-invoked setings.":
PRINT"However, EPSONSET includes cancellation (OFF) options so that it can be used"
130 PRINT"while the printer is printing from its buffer.":
PRINT:
PRINT"On the selections screen, use the arrow keys to position the cursor.":
PRINT"Type any character to make a selection; to rescind one, hit the spacebar."
140 PRINT"Do not confuse cancellation of an old setting with rescinding one not yet sent.":
PRINT:
PRINT"Then, when all selections have been made, press <ENTER>.":
PRINT CHR$(14);
150 I$=INKEY$:
IF I$="" THEN 150
ELSE PRINT CHR$(15);
200 PRINT@(1,0),CHR$(31);:
PRINT"(Position by arrows, character to select, spacebar to rescind, <ENTER> to send.)":
PRINT TAB(H(1)+1)"ON" TAB(H(2)+1)"OFF"
210 PRINT TAB(H(1)+7)"Proportional pitch (correspon dence mode)":
PRINT TAB(H(1)+7)"10 pitch (fast draft mode)":
PRINT TAB(H(1)+7)"12 pitch (elite)":
PRINT TAB(H(1)+7)"17 pitch (condensed"
220 PRINT TAB(H(1)+7)"Micro-pitch (vertically offset)":
PRINT TAB(H(1)+7)"Emphasized type for quality":
PRINT TAB(H(1)+7)"Double-strike to darken":
PRINT TAB(H(1)+7)"Horizontally expand printing (2x)"
230 PRINT TAB(H(1)+7)"Italic type":
PRINT TAB(H(1)+7)"Linespace normal 6 per inch":
PRINT TAB(H(1)+7)"Linespace 4 per inch":
PRINT TAB(H(1)+7)"NO print; process without printing"
240 PRINT TAB(H(1)+7)"Print unidirectionally":
PRINT TAB(H(1)+7)"Enable printing on single cut sheets":
```

```
PRINT TAB(H(1)+7)"Skip perforations for top and bot
tom margins"
250 FOR R=TR TO BR:
PRINT@(R,H(1)+2),"";:
PRINT USING"##";R-TR+1;:
PRINT@(R,H(2)-3),"";:
PRINT USING"##";R-TR+1;:
NEXT R:
PRINT
290 R=TR:
C=1:
GOSUB 2000
300 X$=INKEY$:
IF X$="" THEN 300
310 IF ASC(X$)14 THEN 340
ELSE IF X$=" " THEN PRINT" ";:
C(R-TR+1,C)=0:
GOTO 320
ELSE PRINT"*";:
C(R-TR+1,C)=-1:
ON C GOTO 313,316
313 IF C(R-TR+1,2) THEN C(R-TR+1,2)=0:
PRINT@(R,H(2))," ";:
C=2:
GOTO 320
ELSE 320
316 IF C(R-TR+1,1) THEN C(R-TR+1,1)=0:
PRINT@(R,H(1))" ";:
C=1
320 GOSUB 2000:
GOTO 300
339 REM Arrow keys - to move cursor - and <ENTER> key
to execute selected commands
340 IF ASC(X$)8 THEN 345
ELSE ON (ASC(X$)-7) GOTO 350,360,370,380,345,400
345 X$="*":
GOTO 310 'Enables <ctrl>-Characters, to avoid exceptions
349 REM Left arrow
350 IF R=TR AND C=1 THEN SOUND 0,0:
GOTO 300
ELSE IF C=2 THEN C=1
ELSE R=R-1:
C=2
355 GOSUB 2000:
GOTO 300
359 REM Right arrow
360 IF R=BR AND C=2 THEN SOUND 0,0:
GOTO 300
ELSE IF C=2 THEN R=R+1:
C=1
ELSE C=2
366 GOSUB 2000:
GOTO 300
369 REM Down arrow
370 IF R=BR THEN SOUND 0,0:
GOTO 300
ELSE R=R+1:
GOSUB 2000:
GOTO 300
379 REM Up arrow
380 IF R=TR THEN SOUND 0,0:
GOTO 300
ELSE R=R-1:
GOSUB 2000:
GOTO 300
399 REM <ENTER> - to send selections to printer; printer
rejects any incompatible selections
400 IF C(1,1) THEN LPRINT ESC$ "p1";:
IF C(7,1) THEN LPRINT ESC$ "G";:
GOTO 490 'Correspondence quality
420 IF C(2,1) THEN 460 'Draft mode
430 IF C(3,1) THEN LPRINT ESC$ "M";:
GOTO 460 'Elite
440 IF C(4,1) THEN LPRINT ESC$ CHR$(15);:
GOTO 460 'Condensed
450 IF C(5,1) THEN LPRINT ESC$ "S1";:
GOTO 490 'Superscript
460 IF C(6,1) THEN LPRINT ESC$ "E"; 'Emphasized
470 IF C(7,1) THEN LPRINT ESC$ "G"; 'Double-strike
480 IF C(8,1) THEN LPRINT ESC$ "W1"; 'Expanded
490 IF C(9,1) THEN LPRINT ESC$ "4"; 'Italic
500 IF C(10,1) THEN LPRINT ESC$ "2"; '6 lines per inch
vertically
510 IF C(11,1) THEN LPRINT ESC$ "A" CHR$(18); '4
lines per inch vertically
520 IF C(12,1) THEN LPRINT ESC$ CHR$(19);
'Suppress printing
530 IF C(13,1) THEN LPRINT ESC$ "U1"; 'Print unidirec-
tionally
540 IF C(14,1) THEN LPRINT ESC$ "9"; 'Suppress paper-
out switch
550 IF C(15,1) THEN LPRINT ESC$ CHR$(78)
CHR$(12); 'Skip perforations
610 IF C(1,2) THEN LPRINT ESC$ "p0";
630 IF C(3,2) THEN LPRINT ESC$ "P";
640 IF C(4,2) THEN LPRINT ESC$ CHR$(18);
650 IF C(5,2) THEN LPRINT ESC$ "T";
660 IF C(6,2) THEN LPRINT ESC$ "F";
670 IF C(7,2) THEN LPRINT ESC$ "H";
680 IF C(8,2) THEN LPRINT ESC$ "W0";
690 IF C(9,2) THEN LPRINT ESC$ "5";
710 IF C(11,2) THEN LPRINT ESC$ "2";
720 IF C(12,2) THEN LPRINT ESC$ CHR$(17);
730 IF C(13,2) THEN LPRINT ESC$ "U0";
740 IF C(14,2) THEN LPRINT ESC$ "8";
750 IF C(15,2) THEN LPRINT ESC$ "D";
800 LPRINT CHR$(7);:
CLS 'Signal completion
999 END
1000 CLS:
PRINT CHR$(16) CHR$(30):
PRINT@(0,25)"EPSONSET - Epson FX printer setup":
PRINT CHR$(17):
RETURN
2000 PRINT@(R,H(C)),CHR$(14);:
RETURN
```

Setting Up a Hard Disk on the Mod 4 under CP/M

(Or, a Little Hardware and Software Hacking)

by Roy T. Beck

Recently I have set up a new (to me) hard disk on my Mod 4/4P, and I thought it might be interesting to some of you to see how I did it.

First of all, I will review the hardware I used. The second aspect will be the choice of partitions for the drive. The third portion will be a discussion of the actual partitioning, formatting, testing, etc.

HARDWARE

The drive I am using began life as a universal package by Xebec with a SCSI (Small Computer System) Interface. The bubble is a Quantum Q540 with 8 heads, 512 tracks and an ST506 interface. The power supply is a linear unit. The original hard disk controller (HDC) was a Xebec S1410A. All of the above was contained in a neat package about 6-1/2" wide, 5" high, and about 15" long. An internal fan takes care of the cooling. A power cable and a 50 line ribbon cable come out the rear, and that's it. Since the package was designed with a SCSI interface, Xebec then provided a host adaptor designed for use with the customer's non-SCSI computer, whatever it was. The package I acquired in a trade included a host adaptor for use in an Apple IIG or IIE.

Since I don't own an Apple, that didn't do me any good. Also Xebec doesn't make this equipment or HDC any more and doesn't want to talk about it. Casting about, I considered several alternatives. One consisted of wire wrapping a TRS to SCSI host adapter copied from a VRDATA unit which I have in storage. With that host adapter, I could probably use the software for the VRDATA. All kind of iffy. The second possibility was to order Roy Soltoff's software and host adapter which he is promising for delivery in the near future. Again, some degree of uncertainty. Finally, Dave Dalager offered me a brand new Radio Shack 26-1138 host adapter, complete with owner's instruction sheet. I wasn't familiar with the package, but he assured me it was a good one. I bought it, and I am delighted! In brief, the 26-1138 unit includes the Western Digital WD1000-TB1 HDC board which was used in the RS 26-4155 15 Meg primary HD unit, among other things. This board and a small power supply were housed in a neat box. I promptly took the box apart to see what I had, and found the WD1000-TB1 board is the same footprint as the HD bubble. (It even has screw slots to match the ones on the bubble). With a minimum of tinsmithing I was able to mount the WD1000-TB1 HDC in the Xebec box after removing the Xebec S1410A board. I had to swap one of the cables because one connector differed, and finally I had to reconfigure the power cable connector, because, wouldn't you know, the WD board had different pin assignments for ground and +12V than the Xebec board. Luckily I checked for any discrepancy here, as failure to do so might have been expensive. Fortunately the female cable pins could be withdrawn from the connector and reinserted in the correct position.

Because I am now using the RS WD1000-TB1 HDC, the host adapter problem is solved as this HDC includes the necessary host adaptor for a Mod 3 or 4 on the board, just like the corresponding HDC in the 5 Meg unit. All that is needed is a 50 line ribbon cable with edge card connector to mate with the Mod III or 4. For now I ignored the "3 wires", and am operating without the WP feature.

As a result of my manipulations, I now have the functional equivalent of Radio Shack's 35 Meg hard disk package, and the available RS drivers can be used. Note the 35 Meg designation was based on the use of 34 sectors per track on the Quantum drive. 256 bytes/sector X 34 sectors/track X 512 tracks/head X 8 heads = 35,651,584 bytes. This drive was originally intended for the Model II family, and I suppose they did format 34 sectors per track. However, our Mod 3/4 DOSes format only 32 sectors/track, so our real capacity is 33,554,432 bytes. Hence this bubble ought to be identified as a 33 or 34 Meg bubble when used on the Mods 3/4. But the front of the case is marked 35 Megs, so that's the official name. I explain this for those of you who might do the arithmetic and wonder "How Come?". Incidentally, in a later conversation Dave told me the Quantum Q540 bubble which I have was the bubble RS actually supplied in their 35 Meg box, so by good fortune, my bubble is the one RS intended to be used in this setup. Later I obtained the docs for the RS 35 Meg HD (Cat #26-4172) and discovered the Quantum bubble docs are included. Among other things, this told me the Quantum auto-parks on an inner track, which protects the data tracks. Sometimes you get lucky!

CHOICE OF PARTITIONS

With the hardware problems out of the way, I next turned to the software. There is only one source of hard drivers for use with Montezuma Micro CP/M, that I know of, and that is Monte's package, courtesy of Jesse Bob Overholt, software wizard for Monte. He provides extensive documentation, (some 28 pages of text), which explains what is required to operate CP/M on a hard disk.

His hard driver disk costs $30 + $5 S&H. Some time ago I purchased the driver for the 5 Meg RS drive, which worked for that unit. Something I had either forgotten or didn't know has now bitten me with regard to the 35 Meg unit. Monte's present practice is to put a list on his disk of all the drivers he currently offers for sale. BUT, he only puts on the disk the one or two drivers which fit the drive specified. So, when I dug out the disk, I found the listing for the 35 Meg RS drive, among many others, but when I tried to PIP it to my new boot disk, no dice. It isn't there. Telephone contact with Dallas revealed their present practice. I should buy the 35 Meg driver if that's what I wanted to run. *Grrr.*

But I said present practice. When I bought a CP/M library from another fellow a year or two ago there was a Montezuma Micro hard driver in the collection, still sealed in its original "shrinkwrap". I now opened it up, and found a gift from on high! In addition to two 5 Meg drivers, the disk also contained drivers for 12, 15, and 35 Megs plus several versions for Percom packages for a total of eight drivers on the disk. The serial number of the disk was long before the one I recently bought, so evidently Monte's past practice was to supply everything he had, until they outgrew one disk, and he began considering the economics of the matter. For whatever reason, I now have the requisite 35 Meg CP/M hard disk driver.

Since I want to put other DOSes on the 35 Meg drive along with CP/M, I wanted to partition the DOS by head, and then partition the logical drives within the DOS by cylinder offsets. That's what I wanted to do, but Monte threw me another curve. When I read the documentation very carefully, it became apparent that Monte allows only partitioning by head, and (with one trivial exception), does not allow partitioning by cylinder. Further, the maximum number of logical drives which can be assigned to hard disks is limited to four, and these collectively insist on taking 100% of the hard disk. In the case of my 8 head drive, if only one logical drive of the permitted four is assigned, the driver insists on taking 25% of the total drive, which in this case means 25% of 33 + Megs. I did not really like this partition, as I want to put other DOSes on the same drive, but I did not immediately see any other way out.

As a sidebar to the partitioning effort, you should know that CP/M limits every logical drive to about 8 Meg maximum due to constraints in the directory structure. TRSDOS' corresponding limit is 13.3 Meg.

OPERATIONS

The next concern was that the 64K version of CP/M received from Monte has no room for the hard disk drivers. Consequently, I had to prepare a 63K version of CP/M which has 1K less TPA than the standard version, but will accommodate the HD drivers in high memory above BDOS. In exchange for reduction of TPA, Monte has added a couple of features to CP/M via a file named FIXCPM which comes on the disk with the drivers. One of the features is implementation of an item from ZCPR, which is the display of the User # at DOS READY. The form is A0. This allows you to know what user area you are in and encourages the use of alternate user areas. He also modifies the DOS so that if a called-for file is not found in the signed-on user area, the DOS will then look into User 0 area to see if the file is there, and will execute it if it is indeed there. That doesn't give you quite the flexibility of TRSDOS, but it is an improvement.

The actual steps followed will be shown next, with comments interpolated where it seems appropriate.

I backed up the original driver disk received from Monte. This disk is configured as a 170K system disk, SS DD, with no system on the system track. I added a system to this backup, but not to the original disk. That remains virgin with a write protect tab on it. I also backed up the original CP/M disk that I received from Monte. Next I connected but did not yet turn on my hard drive subsystem. The next step was to boot up CP/M from the fresh backup copy of the system disk.

Next I put a blank disk in drive B and formatted it to match the system and driver disks; 170K SS DD system disk. I didn't put a system on it yet. This third new disk was to become my CP/M HD bootup disk. The following files were then copied from the system disk in A to the HD bootup disk in B:

PIP.COM SUBMIT.COM XSUB.COM STAT.COM
DUP.COM KEYDEF.COM DISK.FDF CONFIG.COM

Using PIP, I copied three files from the backup of the driver disk to my new Bootup disk in B. The files I need are BUILD.COM, CPMFIX.COM and driver file HDRS35M.COM.

The next step was to create a 63K system on my new HD boot disk. To do this I used the command:
MOVCPM 63 * <CR>

Next I used SYSGEN to place the new 63K system on my HD bootup disk. When the program asked for the SOURCE of the image, I just hit <CR>. When it asked for the DESTINATION, I typed B, which is where my new HD boot disk was.

I then pulled my system disk out, put the new HD boot disk in A and pushed RESET. I verified that the signon banner now said "63K", and not "64K".

Next I called CONFIG to configure the system as I want it to be. I enabled all four floppies, configuring the first two as 170K SS DD drives, and the last two as 800K DS DD for use as backup devices. *(See my comments on CP/M formats in the Jul/Aug issue of TRSTimes).*

While I was in CONFIG, I ran KEYDEF to set up the

function keys. I saved all of this to drive A, which was still my first floppy at this point. I answered the AUTO command question with SUBMIT HDBOOT. This will call up the hard disk at boot time when I am all done.

Now it was time to turn on power to the hard disk. The next step was to actually set up the partitions, and for this I called:
HDRS35M H=***A F=BCDE INI T<CR>

This obviously requires some explanation. The first string is the name of the HD driver I selected. The H=***A sequence causes the driver to ignore the first 3/4 of the HD, which I intend for TRSDOS, LDOS, etc and assigns CP/M logical drive A to the last 1/4 of the HD, which in this case is heads 7 & 8. I avoided head 1 because I want to make the 4P autoboot feature call TRSDOS, not CP/M. (Whatever is on head 1 is a candidate for autobooting on the 4P). The sequence F=BCDE will assign logical drives B:, C:, D:, and E: to physical floppies :0, :1, :2, and :3, respectively. The INIT sequence triggers the actual formatting of heads 7 & 8. This command sequence will set up the partitioning and format the HD, all in one shot. The driver reminds me this will destroy everything already on the disk. I replied "Y" to this.

At this point the driver paused for me to enter any known bad tracks. I took a chance on this and simply entered a <CR> meaning there were none. If there are any I would catch them later with the verify program. The driver then proceeded to format the heads, followed by verification of the surfaces to detect any bad sectors. (It found none). Incidentally, if any bad tracks are found, they will be assigned to a file named BADTRACK.TBL in USER AREA #15, made invisible with the SYSTEM attribute, and tagged READ ONLY. With all of this protection, it is very unlikely they will ever be written to.

As a part of its operation, the driver program copies the operating system from the first floppy to the HD, and reassigns the drive numbers as dictated by the command string discussed above. All was well up to this point, and I breathed a prayer of thanksgiving to Monte!

I was running on the hard disk at this point, , but I still had to create a SUBMIT file to recreate this setup each time I reboot. The BUILD program allowed me to create the required SUBMIT file:

```
BUILD HDBOOT <CR>
EXBIOS <CR>
CPMFIX <CR>
HDRS35M H=***A F=BCDE <CR>
<CR>
```

Note that "INIT" is not included in this sequence. I don't want to run FORMAT at every boot time. This created a file named HDBOOT.SUB on drive A:. I was running on the HD, so this file was written on the HD; I then copied it over to my first floppy, which was now drive B:. The EXBIOS file is included because Monte indicated it should be there. With the ample space available on the HD, there is no penalty to having it loaded at boot time. Some of Monte's floppy configurations require the file, so I guess he just put it there as recommended good practice. Next I made several backups of my boot disk. Remember, if you are operating from the HD, the first floppy is now drive B:, and the second floppy is now drive C:. So backup from B: to C:.

Now I RESET and proved out the operation of the HD system. My sequence of operation on my Mod 4 is to turn on the HD power switch, turn on the Mod 4 power switch, push in the boot floppy in the first floppy drive, and sit back and enjoy the performance. As execution begins, the first thing I see is the loading of EXBIOS, followed by CPMFIX, followed by the HDRS.. string. After this comes A0. Executing the STAT command will show a large number of K's of free space on the A: drive. The F1 key will give the directory of the HD and F2 will show the first floppy. By the way, don't put a write protect tab on the boot disk. Execution of the SUBMIT function requires writing to the system disk, and a tab will prevent this. I did all of the above, and was extremely happy to have the whole system operating properly, with 8 Megs of space on drive A:. But then came that nagging thought, can't I do something to reduce the size of drive A:, as it is grossly too large for my purposes?

IMPROVISATION

Up to this point, everything I have described is "by the book" and defensible; just carefully read Monte's documentation for the system and for the hard drivers, and you will find the source of what I have done. And it works. But I was still troubled by the excess space dedicated to CP/M in this arrangement. Back to the books, or more specifically to the list of hard drivers I actually possessed, and especially to Table 4 on page 8 of the HD driver documentation. This table explains the head assignments for 6 heads. Since Monte only partitions by head, he had a problem with head arrangements not evenly divisible by 4. Table 4 shows that for 6 heads and 4 Logical drives, he divides the total drive as follows for a 15 Meg drive with 306 cylinders:

Phys Head	Logical Drive		% of Meg HD of HD
1	A	17	2.5
2	A	17	2.5
3	A	17	2.5
4	B	17	2.5
5	C	17	2.5
6	D	17	2.5
7	-	0	0
8	-	0	0

My attention was drawn to the fact his logical drives were unequal in size with 6 heads. What would happen if I used the 15 Meg driver on my 35 Meg drive? In the table I added the last two heads to show the expected effect. If this scheme worked, then I could assign Drive A to head 6, giving me a CP/M HD with one logical drive of 2.5 Meg. Would it work? Try it. YAHOO! IT WORKS!

To accomplish this I had only to change HDRS35M to HDRS15M wherever it appeared. Now I have assigned only a part of head #6 to CP/M, the first 306 cylinders. The remainder of that head, 512 - 306 = 206 cylinders is available for assignment to something else. I don't know whether the Quantum bubble is truly "happy" with this setup, but it WORKS. Actually, the worst effect that I can see is that write precompensation is probably occuring at cylinder 153 instead of 256. Write current reduction is controlled in the bubble itself and therefore is correct anyway. But I know these factors are not critical, and in this case it seems to do the job.

Later, two other ideas occurred to me. The first one is, I could try the 12 Meg driver. The second is, at format time we have the option of locking out bad tracks. If I specified all tracks "bad" above some specific track, The verifying function of the formatter would assign these bad tracks to the "BADTRACK.TBL" file and never use them again, or at least not until I did another format. This should allow later use of the "bad" tracks by another DOS. Both of these ideas should work, but I haven't tried them. Maybe later. As you can see, a little imagination and daring can accomplish quite a bit.

Until Next Time, ADIOS, AMIGOS!

Roy

The Swap Meet

Wanted: I am looking for back issues of the Radio Shack Computer Catalog to complete my collection. I specifically need the following issues: RSC-1, 3, 10, 11, 12, 13, 15, 16, 18, and 19. Any assistance would be greatly appreciated.
Roy Beck. 2153 Cedarhurst Dr. Los Angeles, CA. 90027

Wanted for Model I: Peripherals: CP/M boards, HI-RES boards, Orchestra 85, EPROM Programmer, super mem boards (above 48k RAM), and documentation for these. Operating Systems (DOS): VTOS, SUPERDOS, MULTIDOS, DOSPLUS 3.0, 3.1, 3.2, 3.3, 4.0, TRSDOS 2.0, and any other old, exotic, rare DOS. Utilities: DOUBLE ZAP II, ALE assembler, any other disk based utility.
Art McAninch, 122 Pecan, Borger, TX. 79007

Wanted: Used Model 100 or 102 in good condition with at least 24K. Also need single/double-sided disk drive for above.
Lance Wolstrup. 20311 Sherman Way. Suite 221.
Canoga Park. CA. 91306.

Wanted: Original Copy of Maxi Manager II Data Base Manager for the TRS-80 Model 3/4 (in 3 mode). Manufacturer out of business. To discuss, call collect / write:
Practical Programs. 1104 Aspen Drive.
Toms River, NJ. 08753 (201) 349-6070

Wanted: TRSDOS 1.3. (disk), Scripsit & Newdos/80v2 (disks & manuals), Electric Pencil operators manual (by M. Schrayer), Basic Faster and Better (by L. Rosenfelder), How to use your Tandy/Radio Shack printer (by Wm. Barden).
R. Yves Breton. C.P.95, STN. Place D'Armes
Montreal, P.Q. Canada H2Y 3E9

FOR SALE: TRS-80 Model III, Dual drive (one flippy), Fan, 2/4MHz switch, Goldplugs on ports, and Manuals, Includes shipping - $300.00. Call/write:
Practical Programs, 1104 Aspen Dr.
Toms River, NJ. 08753 (201) 349-6070

FOR SALE: PUBLIC DOMAIN GOOD GAMES for Model I/III.
GAMEDISK#1: amazin/bas (maze), blazer/cmd (arcade), breakout/cmd (break down the walls), centiped/cmd (arcade), elect/bas (a simulation of the 1980 election), madhouse/bas (adventure), othello/cmd (board), poker/bas (almost better than going to las vegas), solitr1/bas (great solitaire card game), towers/cmd (puzzle game).
GAMEDISK#2: crams2/cmd (chase game), falien (arcade), frankadv/bas (adventure), iceworld/bas (adventure), mini-golf/bas (play putt-putt on the trs-80), pingpong/cmd (1 or 2 player arcade game), reactor/bas (simulation), solitr2/bas (another good solitaire card game), stars/cmd (2 player race game), trak/cmd (maze game).
GAMEDISK#3: ashka/cmd (d&d), asteroid/cmd (arcade), crazy8/bas (card game), french/cmd (space invaders in french), hexapawn (board game), hobbit/bas (adventure), memalpha/bas (adventure), pyramid/bas (good solitaire card game), rescue/bas (arcade), swarm/cmd (arcade game).
Price per disk: $5.00 (U.S.) or get all 3 disks for $12.00 (U.S.)
TRSTimes - PD DISKS. 20311 Sherman Way. Suite 221
Canoga Park, CA. 91306

FOR SALE: Printer Buffer, Centronics Port compatible (IBM PC + others), 64KBytes (25 pages), Reset / ByPass / Copy buttons, 8 LED Indicators (status + memory fullness), 5x7x2 inch metal case, 2 pound, AC/DC with Power supply, builtin Self-check, 1 year Guarantee, includes shipping - $119. Call/write: Practical Programs. 1104 Aspen Drive.
Toms River, NJ. 08753 (201) 349-6070

FOR SALE: TRS-80 SOFTWARE, Models 1/3/4/4P/4D. Many useful programs, Economical prices, Send $3 for listing. Practical Programs. 1104 Aspen Drive.
Toms River, NJ. 08753 (201) 349-6070

WORD PROCESSOR

Full-featured with Mail Merge.
In Basic for Models I, III, 4 (III mode) w/ 16K-48K.
Justify, underline, set fonts, graphics.
20 page manual.
Specify your system.
$12.00 tape/disk.
Tandy 1000 fast compiled $25.00

Delmer Hinrichs
2116 S.E. 377th
Washougal, WA. 98671-9732

AT LAST! A full-featured

GIF DECODER FOR THE MODEL 4

Unlock the wonderful world of GIF graphics on your Mod 4! **GIF4MOD4** decodes images up to 640 x 480. Excellent graphics quality. Has 4 user-selected dithering methods for optimum rendering of each image. **$37.95**

EXITING MOD 4 HI-RES CASINO GAMES!

VIDPOKR4 is a 100% accurate video poker machine simulation. Winning system included! 128K required. **$19.95**

SLOTMOD4 is a 100% accurate, fully animated slot machine simulation w/sound. See what your graphics board can do! **$14.95**

Special offer: Order any 2 of the above on the same disk and subtract $5 from the order. Get all 3 on same disk for only: **$62.95**

Hi-res graphics board required for all programs. Add $2 S&H to total order. Va. residents please add 4-1/2% sales tax.

J.F.R. "Frank" Slinkman
4108-C Fairlake Ln., Glen Allen, VA. 23060

Join the TRS-80 fun in 1990 with

TRSTimes magazine

Our third year of publication continues exclusive coverage and support for Model I, III, 4/4P/4D.

The 1990 editions of TRSTimes features:
Type-in TRS-80 programs in Basic & Assembly language, Hands-on tutorials, Hints & Tips, TRS-80 tricks, TRSDOS 1.3. column, TRSDOS 6.x., LDOS, DOS PLUS, MULTIDOS, NEWDOS/80, CP/M column, Humor, Letters, and much more.

Subscribe now!

1990 calendar year subscription rates (6 issues):

U.S. & Canada - $18.00 (U.S.) first class mail
Other countries - $23.00 (U.S.) surface mail
Other countries - $29.00 (U.S.) air mail

TRSTimes magazine
20311 Sherman Way, suite 221
Canoga Park, CA. 91306
U.S.A.

TRSTimes on DISK #3

LISTER/BAS	I/ III/4	All
CPY/CMD	III	TRSDOS 1.3./1.4./1.5.
VCXREF4	4	TRSDOS 6.2/6.3.
A/CMD	I	NEWDOS/80 V2.
LPRINT/CMD	I	All
SBASIC/BAS	I/ III/4	All
NX/CMD	4	TRSDOS 6.2./6.3.
LIBDVR/BAS	III	TRSDOS 1.3./1.4/1.5.
MENUDEM/BAS	4(III)	All
ROTATE/BAS	III	All
MAC2PWR/CMD	I/ III	NEWDOS/80 V2.
EDITOR/BAS	III	TRSDOS 1.3./1.4./1.5.
WATRTURN/BAS	4	All
WATRTRN3/BAS	III	All

U.S. & Canada: $5.00 (U.S.)
Anywhere else: $7.00 (U.S.)

Send check or money order to:
TRSTimes on DISK
20311 Sherman Way, Suite 221
Canoga Park, CA. 91306
U.S.A.

TRSTimes on DISK #1 & 2
are still available at the above prices.

ATTENTION TRSDOS 1.3. USERS!

ANNOUNCING "SYSTEM 1.5.", THE MOST COMPREHENSIVE 1.3. UPGRADE EVER OFFERED!

MORE SPEED!! MORE POWER!! MORE PUNCH!!

While maintaining 100% compatibility to TRSDOS 1.3., this DOS upgrade advances TRSDOS 1.3. into the 90's!
SYSTEM 1.5. supports 16k-32k bank data storage and 4 MGHZ clock speed (4/4P/4D).
DOUBLE SIDED DRIVES ARE NOW 100% UTILIZED! (all models).

CONFIG = Y/N	CREATES CONFIG BOOT UP FILE	DATE = Y/N	DATE BOOT UP PROMPT ON or OFF
TIME = Y/N	TIME BOOT UP PROMPT ON or OFF	CURSOR = 'XX'	DEFINE BOOT UP CURSOR CHAR
BLINK = Y/N	SET CURSOR BOOT UP DEFAULT	CAPS = Y/N	SET KEY CAPS BOOT UP DEFAULT
LINES = 'XX'	SET *PR LINES BOOT UP DEFAULT	WP = d.Y/N (WP)	WRITE PROTECT ANY or ALL DRIVES
ALIVE = Y/N	GRAPHIC MONITOR ON or OFF	TRACE = Y/N	TURN (SP) MONITOR ON or OFF
TRON = Y/N	ADDS an IMPROVED BASIC TRON	MEMORY = Y/N	BASIC FREE MEMORY DISPLAY MONITOR
TYPE = B/H/Y/N	HIGH/BANK TYPE AHEAD ON or OFF	FAST	4 MGHZ SPEED (MODEL 4's)
SLOW	2 MGHZ SPEED (MODEL III's)	BASIC2	ENTER ROM BASIC (NON-DISK)
CPY (parm.parm)	COPY/LIST/CAT LDOS TYPE DISKS	SYSRES = H/B.'XX'	MOVE /SYS OVERLAY(s) to HI/BANK MEM
SYSRES = Y/N	DISABLE/ENABLE SYSRES OPTION	MACRO	DEFINE ANY KEY TO MACRO
SPOOL = H/B.SIZE	SPOOL is HIGH or BANK MEMORY	SPOOL = D.SIZE = 'XX'	LINK MEM SPOOLING to DISK FILE
SPOOL = N	TEMPORARILY DISABLE SPOOLER	SPOOL = Y	REACTIVATE DISABLED SPOOLER
SPOOL = RESET	RESET (NIL) SPOOL BUFFER	SPOOL = OPEN	OPENS, REACTIVATES DISK SPOOLING
SPOOL = CLOSE	CLOSES SPOOL DISK FILE	FILTER *PR.ADLF = Y/N	ADD LINE FEEDS BEFORE PRINTING 0DH
FILTER *PR.IGLF	IGNORES 'EXTRA' LINE FEEDS	FILTER *PR.HARD = Y/N	SEND 0CH to PRINTER (FASTEST TOF)
FILTER *PR.FILTER	ADDS 256 BYTE PRINTER FILTER	FILTER *PR.ORIG	TRANSLATE PRINTER BYTE to CHNG
FILTER *PR.FIND	TRANSLATE PRINTER BYTE to CHNG	FILTER *PR.RESET	RESET PRINTER FILTER TABLE
FILTER *PR.LINES	DEFINE NUMBER LINES PER PAGE	FILTER *PR.WIDTH	DEFINE PRINTER LINE WIDTH
FILTER *PR.TMARG	ADDS TOP MARGIN to PRINTOUTS	FILTER *PR.BMARG	ADDS BOTTOM MARGIN to PRINTOUTS
FILTER *PR.PAGE	NUMBER PAGES, SET PAGE NUMBER	FILTER *PR.ROUTE	SETS PRINTER ROUTING ON or OFF
FILTER *PR.TOF	MOVES PAPER to TOP OF FORM	FILTER *PR.NEWPG	SET DCB LINE COUNT to 1
FILTER *KI.ECHO	ECHO KEYS to the PRINTER	FILTER *KI.MACRO	TURN MACRO KEYS ON or OFF
ATTRIB:d.PASSWORD	CHANGE MASTER PASSWORD	DEVICE	DISPLAYS CURRENT CONFIG INFO

All parms above are installed using a new LIBRARY command SYSTEM (parm,parm). Other new LIB options include DBSIDE (enables double sided drive by treating the "other side" as new independent drive, drives 0-7 supported) and SWAP (swap drive code table #s). Dump (CONFIG) all current high and/or bank memory data/routines and other current config data to a disk data file. If your type ahead is active, you can (optional) store text in the type buffer, which is saved. During a boot, the config file is loaded back into high/bank memory and interrupts are recognized. After executing any active auto command, any stored type ahead data will be output. FANTASTIC!. Convert your QWERTY keyboard to a DVORAK! Route printer output to the screen or your RS-232. Macro any key, even F1, F2 or F3. Load *01-*15 overlay(s) into high/bank memory for a memory only DOS!. Enter data faster with the 256 byte type ahead option. Run 4 MGHZ error free as clock, disk I/O routines are properly corrected! Spool printing to high/bank memory. Link spooling to disk (spooling updates DCB upon entering storage). Install up to 4 different debugging monitors. Print MS-DOS text files ignoring those unwanted line feeds. Copy, Lprint, List or CATalog DOSPLUS, LS-DOS, LDOS or TRSDOS 6.x.x. files & disks. Add top/bottom margins and/or page numbers to your hard copy. Rename/Redate disks. Use special printer codes eg:LPRINT CHR$(3); toggles printer output to the ROUTE device. Special keyboard codes add even more versatility. This upgrade improves date file stamping MM/DD/YY instead of just MM/YY. Adds optional verify on/off formatting, enables users to examine *01-*15, DIR, and BOOT sectors using DEBUG, and corrects all known TRSDOS 1.3. DOS errors. Upgrade includes LIB/DVR, a /CMD driver that enables LIBRARY commands, such as DIR, COPY, DEBUG, FREE, PURGE, or even small /CMD programs to be used within a running Basic program, without variable or data loss.

By special arrangement with GRL Software,
SYSTEM 1.5. is now distributed exclusively by TRSTimes magazine.

ORDER YOUR COPY TODAY!

Send $39.95 (U.S. funds) to:
TRSTimes - SYSTEM 1.5.
20311 Sherman Way. Suite 221
Canoga Park, CA. 91306

CLOSE #5

Here we are again, at the last page of an issue. This time it closes issue 5 of our second year, making it the eleventh issue we have produced. Time is moving on, and we are all getting older by the day. Our favorite machine is also getting older; as a matter of fact, according to the original authors of LDOS and TRSDOS 6.x., it should have died of old age almost 2 years ago. They coded the portion of the operating system that prompts for the date at boot-up to accept dates only to 12/31/87. Any date later than that will be rejected, and access to the machine is denied. This, of course, is not a problem to the old-timers of the TRS-80 world, but to a newcomer it is a major problem and several have asked me if their machine is broken. So, for the benefit of new readers, let's go through the solutions to this annoyance once again.

There are basically three ways to solve the LDOS and TRSDOS 6.x. date problem:

1. Buy LDOS 5.3. (model III) or TRSDOS 6.3. (model 4) from MISOSYS, INC. P.O. Box 239, Sterling, VA. 22170-0239, (703) 450-4181. Both DOSes support dates to 12/31/99.

2. When prompted for the boot-up date, enter a date between 01/01/80 and 12/31/87 and don't worry about it. The only thing that will happen is that new disk files will be created with an incorrect date. No big deal.

3. Enter a date between 01/01/80 and 12/31/87 at boot-up, and then enter the following command from DOS ready:

SYSTEM (DATE=N) <ENTER>

This will turn off the dreaded date prompt once and for all and, as in solution 2, the only thing that happens is that new disk files will be given an incorrect date. I work extensively with LDOS and TRSDOS (other than 5.3. and 6.3.) and I frankly don't care what date my files carry in the directory.

The majority of users will never need the date to use their machine. However, the few that are running business programs that make use of the date (for dating checks, payroll entries, etc.) will have to write their own date routine, or bite the bullet and buy LDOS 5.3. or TRSDOS 6.3.

If you are working with any version of MULTIDOS or NEWDOS/80, you will have no date problem. Both operating systems allow dates past 12/31/87. The same goes for TRSDOS 1.3., 1.4., and 1.5.

Besides supporting LDOS 5.3. and TRSDOS 6.3., MISOSYS also publishes 'THE MISOSYS QUARTERLY', a fine magazine devoted these DOSes. It is published by Roy Soltoff, and as the name indicates, it comes out four times a year. The information provided by this magazine is top drawer and it is geared toward the experienced user. Newcomers, without previous computer experience, will be in over their heads here. The subscription rate is $25.00 per year and is well worth it. MISOSYS also offers a companion disk to each issue, called 'DISK NOTES'. Each disk is $10.00 plus S&H of $2.00. This is a good magazine, so give them a try.

Before bringing this to a close, I want to thank Roy Beck, Robert Doerr, Mark Speer, Eric Bagai, Fred Blechman, Andrew Bruns, Bruce Showalter, Dale Parsons, Dennis Burkholz and Michael Webb, for making this issue possible. Thanks guys, #2.5. would have been mighty thin without you.

Next issue will bring a word-puzzle generator for Model 4, more on ZBasic and Multidos, the TRSDOS 1.3. column will return with some more patches, Assembly 101 begins coverage of Model 4, Roy Beck will be back with more on CP/M, and much more.

So until November....THINK TRS-80.

Lance W.

ADVERTISING RATES FOR THE 'SWAP MEET' PAGE

The 'SWAP MEET' page is devoted to classified ads.
If you are looking to buy something, TRSTimes will provide a 6 line WANTED ad free of charge.
FOR SALE ads costs $5.00 per 6 lines.
Send to:

TRSTimes - swap meet
20311 Sherman Way, Suite 221
Canoga Park, CA. 91306. U.S.A.