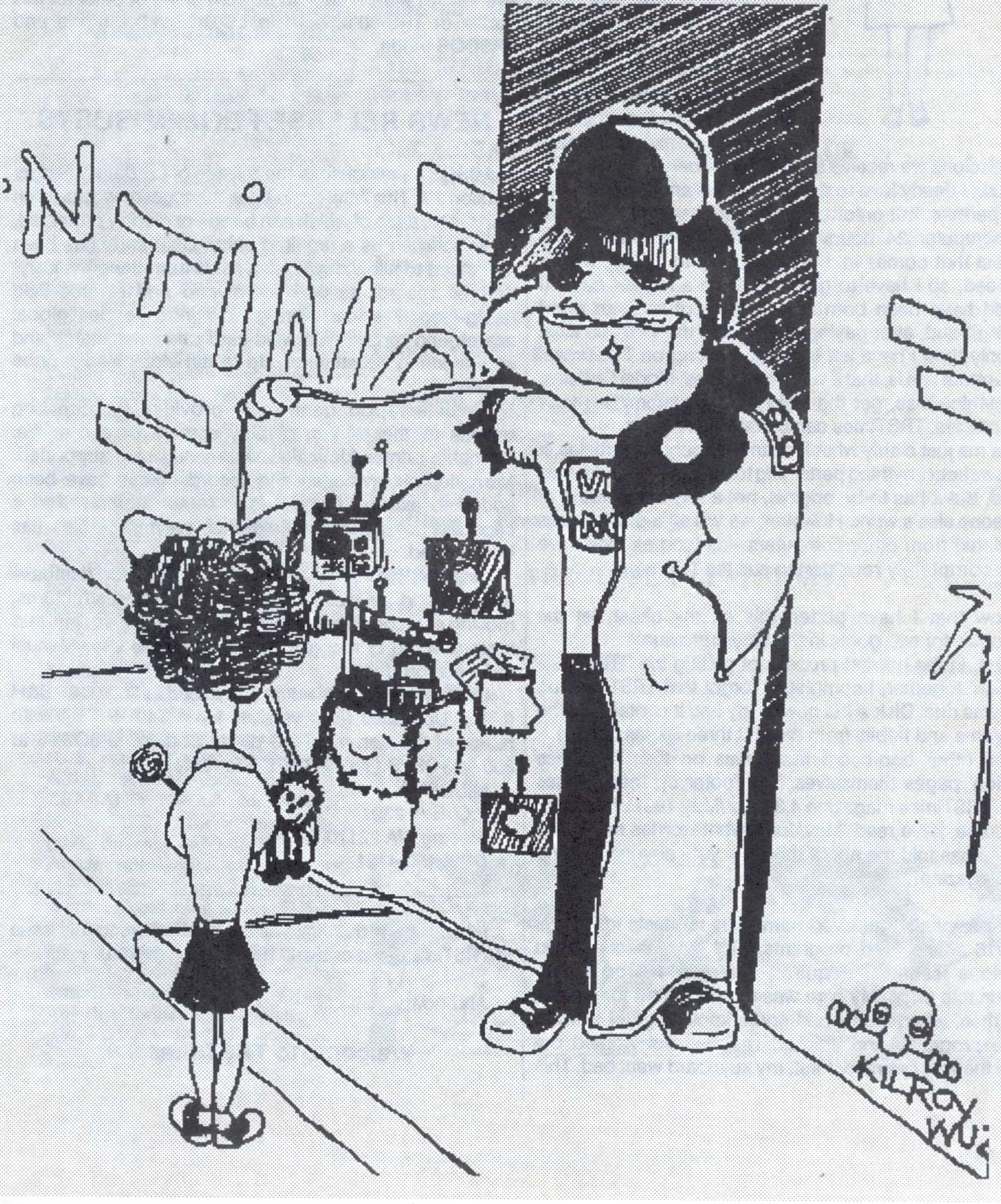
TRSTimes

Volume 4. No. 4. - Jul/Aug 1991 - $4.00



LITTLE ORPHAN EIGHTY

TRSTimes has received a few letters accusing us of making the last couple of issues too serious. It has also been said that we have been neglecting the Model I.

To be sure, both accusations are true. The reason for the neglect is simple. Though the interest seems to there, we are not receiving submissions in these two categories. As is to be expected, most submissions we receive are for the Model 4.

Now, I dearly love to write programs and I will do what time permits, but unfortunately, I just can't sit in front of my computer 24 hours a day, filling each and every request that comes in. I have a family to clothe, shelter and feed, so I have to go to work on a regular basis (I should have been born rich, instead of so darn good looking), and, after paying attention to the wife and kids, the only time I have left is spent working on TRSTimes. The bottom line is, that if you want games, Model I articles, or anything else, get the ball rolling by submitting your own articles. TRSTimes can certainly use them.

Let me just clarify what I mean by 'your own' articles. It can be about anything pertaining to the TRS-80 world, new or old, but it has to be original, not a slight rewording of someone else's work. However, we will accept programs translated from other computers - as long as they have been **completely rewritten** to suit the TRS-80.

Now that I have gotten this off my chest, let me apologize for two goofs in the May/Jun issue.

First, in the hurried process of getting the 'TRSTimes Auction' together, I completely forgot that TRSTimes on Disk was due. Disk #7 is now ready and it contains all the programs and tidbits from the first three issues of 1991.

The other boo-boo I made was on the 'TRSTimes Auction' pages themselves. The footer on these pages read 'TRSTimes magazine 4.3 - Apr/May 1991'. It should, of course, have read 'May/Jun'. What worries me is that no one has told me about this goof yet - anyone reading the magazine?

We have also received numerous requests for Model II/12/16 articles and programs. As I have mentioned in previous issues, I acquired a Model 16 for a very reasonable price. My plan was to play with it for a while and then, when I felt I had some knowledge of it, write some programs and articles. This has not worked out quite the way I wanted. First, my keyboard went bad. This caused my newly purchased copy of LS-DOS 6.3.1 to hang up occasionally, making it difficult to keep my interest. After acquiring another keyboard, and installing a second drive, my drive :0 ceased to function, making it impossible to do anything with the machine. Also, my printer port was not working correctly. Maybe the price was not quite as reasonable as I first thought!

However, Roy Beck has promised to fix it up for me, so eventually I will get back to do something or other for this series of TRS-80's, both in LS-DOS 6.3.1 mode and TRSDOS mode.

NEWS RELEASE FROM MISOSYS

MISOSYS announces the release of LB Data Manager Version 2.1. This flatfile data base manager now sports ten field types, each of which can be edit or display protected, or established as a required entry. Data entry has been enhanced to allow duplicating fields from the previously-entered record, as well as forward and reverse field navigation. Flexible editing now incorporates global search and replace with wild-card character match and source string substitution. Up to ten index files may be created.

Flexible report generation provides for directing reports to the printer, display screen, or a disk file. Definable printer (de)initialization strings are supported. Numeric field averages and record counts have been added to existing (sub)totals in footer support. And a capability for generating external mail/merge files has been added.

Included maintenance utilities provide for database restructuring, as well as duplicating existing structures, moving and copying records between database files, and mass purging of records. A new 200-page User Manual has been produced.

LB Data Manager Version 2 is priced at $99 + $5 S&H (U.S.). Upgrades from version 1 are $40 + S&H with return of the Ver. 1.0 TOC page. Contact MISOSYS at 800-MISOSYS (800-647-6797) for details.

MISOSYS
P.O. Box 239
Sterling, VA 22170-0239
(703) 450-4181

Finally, many thanks to all the contibutors to this issue of TRSTimes - we couldn't have done it without you.

And now.................

Welcome to TRSTimes 4.4

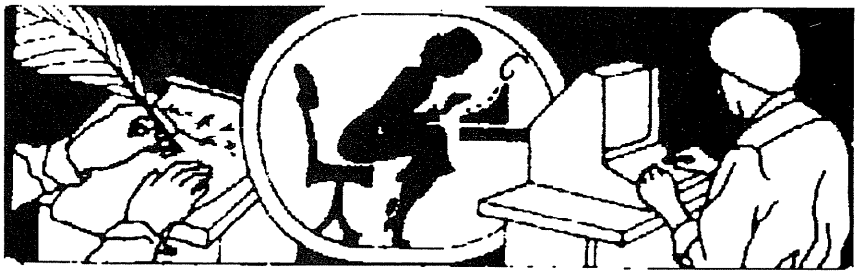
TRSTimes magazine

Volume 4. No. 4. - Jul/Aug 1991

PUBLISHER EDITOR
Lance Wolstrup
CONTRIBUTING EDITORS
Roy T. Beck
Dr. Allen Jacobs
Dr. Michael W. Ecker
TECHNICAL ASSISTANCE
Members of:
San Gabriel Tandy Users Group
Valley TRS-80 Users Group
Valley Hackers' TRS-80 Users Group

TRSTimes magazine is published bi-monthly by TRSTimes Publications. 5721 Topanga Canyon Blvd, Suite 4. Woodland Hills, CA. 91367. (818) 716-7154.
Publication months are January, March, May, July, September and November.

1991 subscription rates (6 issues):
UNITED STATES & CANADA: $18.00 (U.S. currency)
EUROPE, CENTRAL & SOUTH. AMERICA: $23.00 for surface mail or $29.00 for air mail. (U.S. currency only)
ASIA, AUSTRALIA & NEW ZEALAND: $25.00 for surface mail or $$32.00 for air mail. (U.S. currency only)
Article submissions from readers are welcomed and encouraged. Anything pertaining to the TRS-80 will be evaluated for possible publication. Please send hardcopy and, if at all possible, a disk with the material saved in ASCII format. Any disk format is acceptable, but please note on label which is used.



THE MAIL ROOM

MODEL II/12/16

I want to congratulate Mrs. Welcomb on her article about Model 4 and Model 12. Very good. Maybe she can help me with the following: I recently acquired a Model II with two expansion bays (8 inch SS/DD drives - 509K per diskette). I have no problems using Mod II software on it, but I also have a Model 6000 HD and I cannot seem to get the drives in the expansion bay recognize Mod 12 software disks - even when the disk is SS/DD. Is there a patch that I am missing?

I would also appreciate it if Mrs. Welcomb could explain the different DOSes on Mod II up to Mod 6000 in single users mode, and the rationale of THIN, UNTHIN, etc. I also need to know what to do about adapting Mod II Basic to Mod 12 and 16, as I will be upgrading the Mod II to Mod 16 shortly. Will the expansion bays still work on that souped-up Model II?

I understand that it is possible to connect a Mod I/III/4 to an expansion bay if you have the right interface. How do I get/make one? It could serve as a cheap way to store data/programs if one cannot afford a hard drive. Double sided/double density expansion bays can hold up to 1 meg per drive and the maximum is 3, so that's a 3-meg floppy drive.

I am still trying to make a go at XENIX System III v3.2.0 on my Model 6000. This version of XENIX is close to the BSD (Berkeley System) system V release 2 with Xenix features. Most books at bookshops deal with the UNIX System V release 4.3, which is different from ours, so go to used book stores and make sure the XENIX/UNIX Systrem III books are dated before 1989.

R. Yves Breton
P.O. Box 95
Stn Place d'armes
Montreal, Quebec H2Y 3E9

WHAT ABOUT SUPERSCRIPSIT, PROFILE AND VISICALC

I am a new subscriber to your magazine, and I am happy to have found a source of information for my computer. I have enjoyed my first three issues and look forward next one.

I do have a complaint, though. Why do you not have articles about the three most popular software packages for the Model 4: SuperScripsit, Profile and Visicalc. I think many of your readers would enjoy reading and learning about them.

Frank Melchior
Green Lake, WI

Glad you like the magazine. We would be happy to publish information about the three programs you mentioned. As you have figured out by now, TRSTimes exists only because the readers share information with us in the form of articles, programs, or both, and lately we have not had any submissions dealing with SuperScripsit, Profile or Visicalc. Readers are hereby encouraged to send us something about these programs.

For the record, may I say that Profile and Visicalc are fine programs that do the job they were intended for. SuperScripsit, I understand, has it share of serious problems. However, all three are DEAD programs; that is, they are no longer available and they are no longer supported. It may be smart to move on to software that is both available and supported by the author. Instead of SuperScripsit, you can get the latest version of LeScript (complete with spelling checker) from ANITEK SOFTWARE, P.O. Box 361136, Melbourne, FL. 32936. Attn: Peter Ray.

Profile can be replaced with a much more flexible and faster database manager called LITTLE BROTHER. An upgraded version has just been released by MISOSYS Inc. P.O. Box 239, Sterling, VA. 22170. Attn: Roy Soltoff.

Finally, VISICALC can be replaced with either BUSY-CALC or BCX. Both are brandnew, very powerful spreadsheets. They are available from COMPUTER NEWS 80, P.O. Box 680, Casper, WY. 82602-0680.

Ed.

COLLECTION HELP

I am need the following old Radio Shack Catalogs to complete my collection: RSC-1, RSC-13, RSC-18E and RSC 19E.

Also, I need a copy of page 116 from the PC-1 pocket computer manual. Any help will be much appreciated.

Roy Beck
2153 Cedarhurst Dr.
Los Angeles, CA 90027

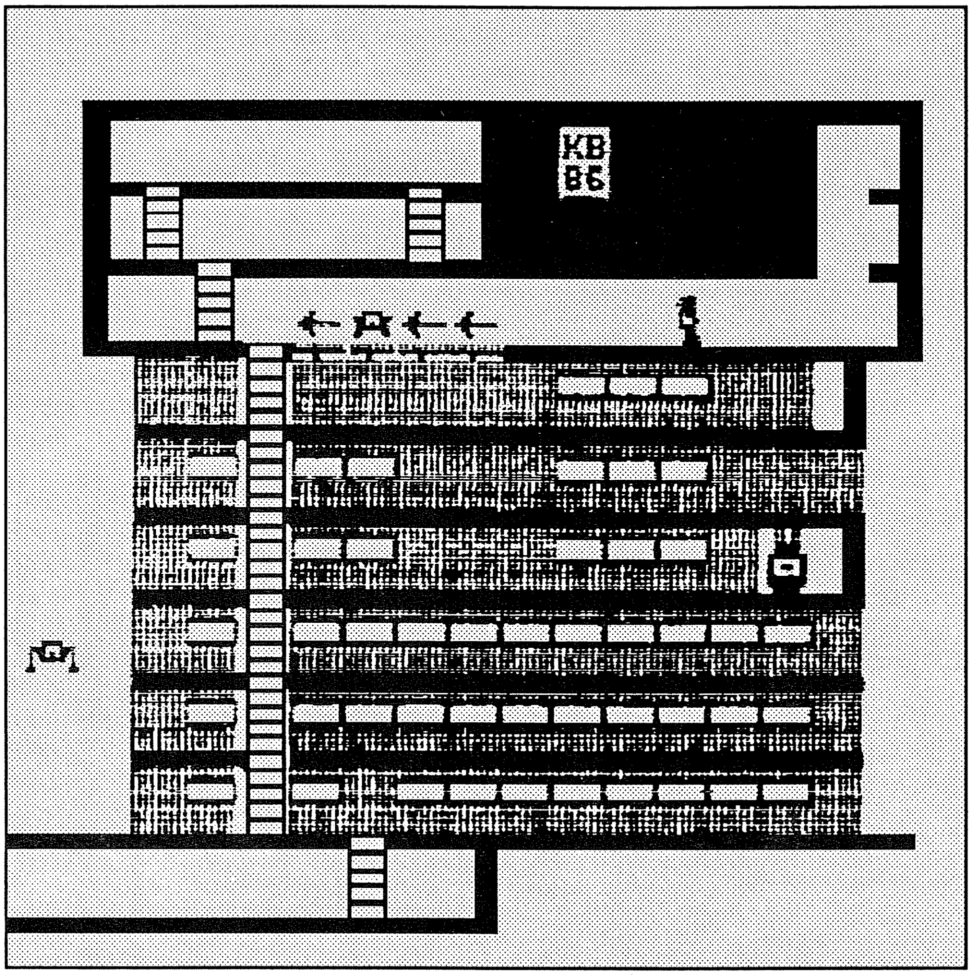
MODEL III & 4 FOR SALE

I have a Model 4 with 128K that I am willing to part with for $90. I would like to get $40 for the Model III. If you buy both, the price will be $100. Contact me any evening before 11pm (PCT).

Matt Monaster (213) 653-2904

THE GREAT VOLCANO HUNTER HACK

By Gary W. Shanafelt



There are a lot of definitions of "hacker." I suppose in general a hacker is someone who tries to figure out how a computer program works by analyzing all the code modules in it. My wife's definition is that I'm "hacking" when my Model 4 is on and I'm doing anything except word processing on it.

I was originally sucked into this form of computeritis because of various programs I had which used misspelled words, which I, as a teacher, couldn't stand seeing on my screen. Being compulsive, I didn't like having some things on disk and others on tape, nor did I like protected programs since I wanted everything I had to run under the same operating system (which ended up being LDOS in Model III mode and LS-DOS in Model 4 mode).

Gary Shanafelt can be reached at:
Dept. of History - McMurry University
Abilene, TX 79697

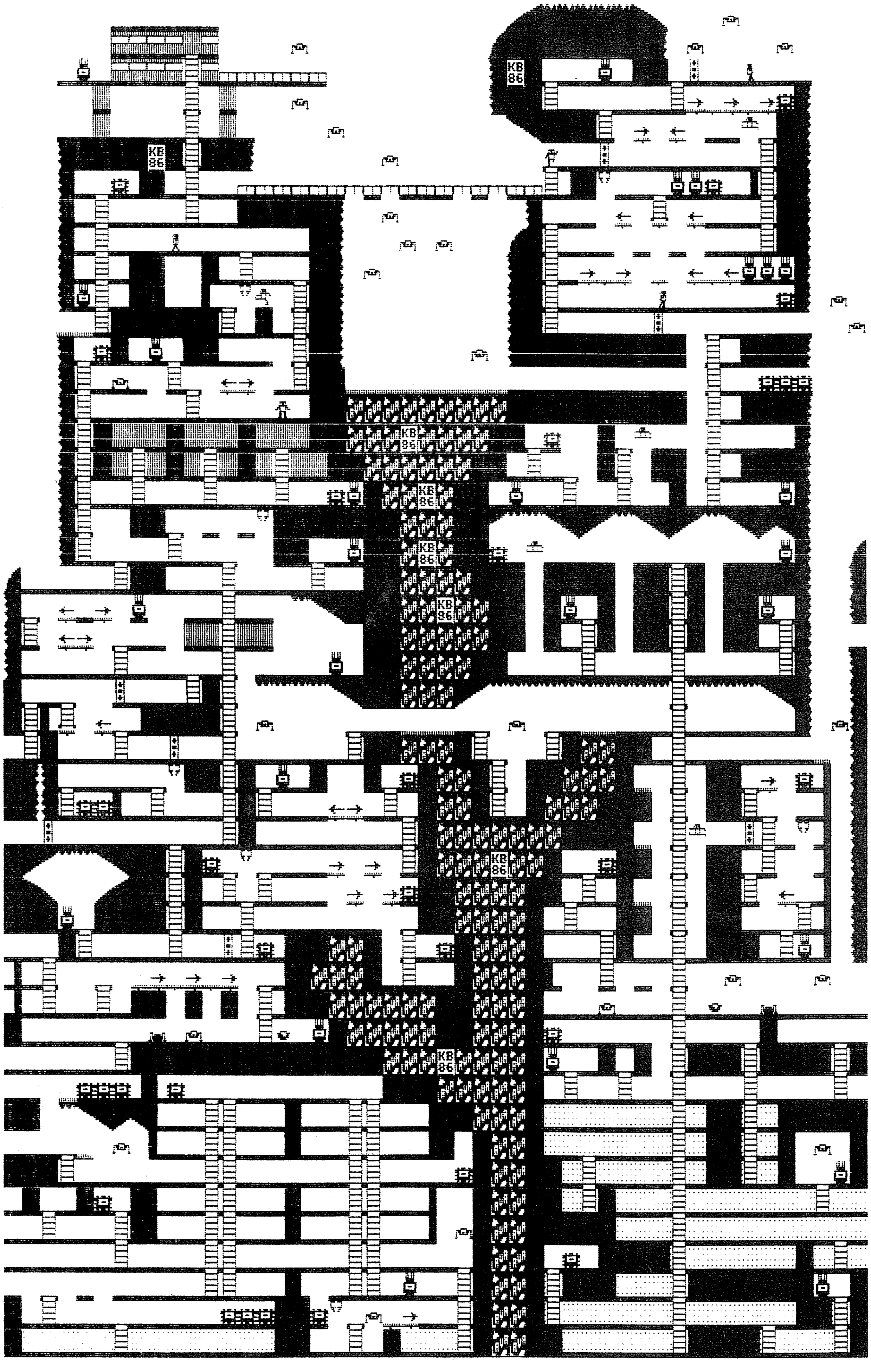
Of course, I also had to figure out how to make a number of programs which ran on my original Model I also work when I replaced the Model I with my current Model 4. I don't really know how much time and effort I spent on all this, especially when I embarked on a project of customizing all the arcade games I had to return to the DOS without the necessity of hitting the RESET button. I do know that I spent more time hacking the games than I ever spent playing them.

I say all this by way of introduction because nothing I attempted ever came close to the time and effort a fellow named Kelly Bates put into hacking a game called Volcano Hunter.

Volcano Hunter was one of the last TRS-80 arcade games. It was written by David Smith of Mississippi State University. He was apparently a student there, for when I tried to write him my letters got returned with no forwarding address (which I assume means that he graduated). I have no idea what he is doing now. Volcano Hunter was never one of the big name games like Galaxy Invasion or Sea Dragon or Scarfman, perhaps because it appeared at the end of the TRS-80 game era, in 1984, though it got a four star review in the September 1984 issue of 80 Micro.

Most arcade games of the time followed one of several formats. Aliens descended from the top of the screen while you shot at them from the bottom (like Missile Attack or Defense Command); or you scrolled sideways across an obstacle-filled landscape while aliens came at you (like Zaxxon or Penetrator or Eliminator); or you moved in four directions on the screen among obstacles and aliens (like Scarfman or Apple Panic).

Volcano Hunter fit none of these molds. It was a sort of graphic adventure, like Zork might have been if it had


KB 86
LAVA

somehow been converted into an arcade game from a text adventure. There were some other TRS-80 adventure games with graphics, like MED Systems' Asylum or Warriors of Ras series, but nothing with the graphics of Volcano Hunter. In it, you explored the innards of a Volcano to retrieve fuel canisters and gold. Those innards were replete with lava, water-filled chambers, ladders, dead ends, moving conveyer belts, drop-offs, and, of course, the Drut monsters. Your screen would show you one small part of this underground maze; you saw other parts as you advanced in different directions. The trick was to figure out how each part you saw fit together like the pieces of a puzzle, so you learned the best ways to penetrate the world inside the volcano as well as how to get back out with your fuel canisters. Since you only got eight men, most people probably never realized just how complex the whole Volcano world was because they got zapped before they ever managed to explore very far. Advertisements for Volcano Hunter said that it comprised over 200 screens -- a real tour de force for a program of just 16K -- but most of those screens were probably never seen except by the program's author, David Smith... and Kelly Bates.

Who was Kelly Bates? I never met him. Though he is now on his second MS-DOS machine, he still tinkers on a Model 4P and attends a user group in Oklahoma City, where he resides. I got to know him five years ago through the mail after I acquired a tape version of Volcano Hunter (which I transferred to disk) and began asking around to locate someone with a disk version. I wanted to know how the two were different, since the disk version was supposed to have various enhancements absent on the tape one (as it turns out, it just had a few more screens). Kelly also had the tape version, and what he had done with it was truly amazing.

What he had done was to determine to find out just what all 200-plus screens of Volcano Hunter looked like or die in the attempt. Arcade games back then often aroused strange passions in people: I remember staying up all night once trying to discover what lay at the end of Sea Dragon, and exhausting myself trying to get to 50,000 points on Galaxy Invasion Plus to see the screen flip around. It didn't take Kelly long to realize that he could never get through the entire volcano with just eight men. So, he went into the program code with the Newdos diskzapper SUPERZAP, found where the A register was loaded with the number of men and then decremented, and nulled the decrement instruction. That meant the men counter was always set at eight no matter how many times he was destroyed by the Druts or the heat from the lava or drowning in the underground pools. Slowly, he explored and mapped out the whole grid. He discovered several fuel cells which were totally sealed off, as well as how the whole program worked.

But that was just the beginning. To make the game easier to play, he began studying how the various screens were mapped in the program code, so that he could modify the grid. Each screen, he found, consisted of three rows of eight hex characters each, for a total of 24 bytes. Each byte was a special code for a portion of a wall, a hot spot, a fuel cell, a ladder, etc. The program read the 24 bytes for each screen, then matched them with their equivalent graphics characters to form a new picture on the screen. Armed with this information, he could basically create any pattern of screens that he wanted.

But like any TRS-80 devotee, Kelly was not content to rest on his laurels. He wanted to see what the entire grid looked like as a whole, not in 200 separate pieces. With Karl Hessinger's graphics editor ZGRAPH, he replicated all the graphics blocks used in Volcano Hunter to create the individual screens. He then converted those designs into characters for a Dotwriter font which he called, appropriately, VOLCANO/PR. Finally, with the new font and Dotwriter, he printed out the entire 200-screen grid on his Gemini printer. It took several pages which he then taped together... and he had the only map in existence of all the byways of Volcano Hunter.

He says it all took him about two years, and he made his final printout on February 18, 1986. When he sent me a copy, I was totally amazed. I was amazed first of all at the sheer work involved, but I was also amazed at just how incredible a program Volcano Hunter actually was. Like most people, I had never realized how much was there. Armed with Kelly's map, I went into that volcano to explore new worlds where no one (except Kelly) had ever gone before.

The glory days of the TRS-80 are over, but I hope that some of the deeds it inspired will be remembered even when no one any longer remembers things like expansion interfaces or supervisor calls.

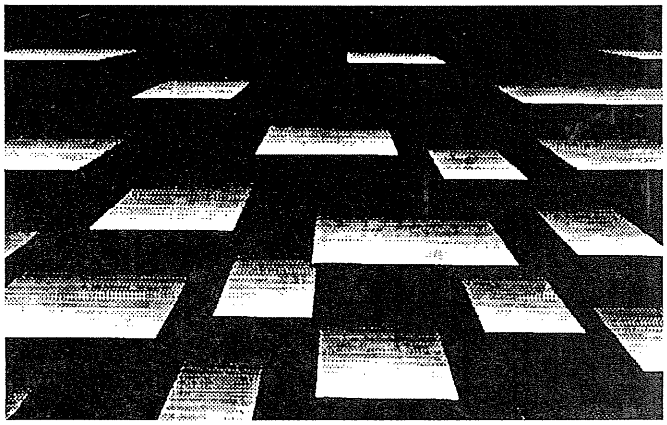
Software & manuals
for TRS-80
CoCo 1, 2, 3
Models I, II, III, 4
1000, 2000, 100

Low prices.
Write for what you need.
Will ship UPS/COD.

Werner L. Jordan
10 N. Morley St.
Baltimore, MD 21229

SHELL in BASIC for MultiDOS

By Jim E. King



A SHELL is intended to make DOS commands and general operation easier, one-key operations at best. When I saw the excellent shell by Stephen Milliken for TRSDOS 6.2, I decided that I wanted something like it in MultiDOS.

My Shell is not identical but it does facilitate faster 'one-key' operation of many functions. It takes advantage of the MultiDOS capability of executing any DOS command from BASIC. The 1.71 Model 3 version is a little different from the 2.1 Model 4 version, and the 1.71 version is the one I am looking at while I write this article.

I moved all my operations over to MultiDOS 1.6 some years ago because it was so user friendly, and this program is possible because ANY DOS command can be executed from BASIC.

The one-key operation is based on my standard Inkey$ routine in line 8, and the use of many IF tests on what you choose in line 8 (lines 102-199).

I always put the menu in line 99 in my programs (no particular reason for it - it's just the way I do it), and my menu does not appear and disappear as you use the program, but scrolls with everything else. You can still see what you were working on when you return to the menu. The reason the UPPER & lower cases look funny, is that you press the Upper to choose that function.

The programs begins at line 0 with some necessary setup, definitions, title, and then GOTOs line 80, which holds the string value of the number of the date developed by Rosenfelder's (BF&B) date routines in lines 81-86.

Line 90 tests whether you are entering the program for the first time, or returning. IF the first time, it routes to line 200 where it displays a day and date, and instructs you to press arrows to change the date, usually the right arrow (Lines 200-280). This part is my attempt to duplicate the excellent program 'SETDATE' by Jack Decker. When you have displayed today's day and date, pressing <ENTER> saves that date number back into the program (lines 290 to the end). The pokes in line 290 put the date into Time$.

After saving the new date, or there is a value in Time$ in line 90, lines 92-- process the date to display the day & date in the menu, line 99.

Menu (all with 1 keystroke):
Press a single number for a short directory of any disk drive (106).
If you want a long directory, press the shifted numbers and the * for 0 (110).
caT, the special CATalog, will read 1 or 2 sided disks automatically and configure that drive.
dAte allows you to redo the date.
confiG allows you to quickly configure drives 1-3, 1 or 2 side.
Format....
Map: displays a map of the disk.
List a file and see if you can make sense of it.
copY a file from 1 disk to another with the same name.
rEname a file.
Kill a file.
reStore a killed file.
VFU: Hester's Very Fine Utility for copying, etc.
Zap (go ahead, make my day)

Some commands available that are not shown on the menu are:
I - type in a FileName to be made invisible.
& - the semi-automatic /BAK routine.
6 - commands S4 to a 64 column screen
8 - commands it to an 80 column screen.

The 2.1 version has added routines to input the time in 24 hour clock for inclusion in Time$.

In general, most of the time, you can recover from an error, by entering '<' to back up or return to the menu.

Lines 3-79 are subroutines, etc. Note that if you leave the write protect tab on, line 70 will goto 40 and flash a message at you. Remove the tab and press any key.

If you want to drop into the READY> mode, press CLEAR, line 8.
Line 7 changes lower case to UPPER.
Line 9 jumps up a line.

I save the program under the letter 'S' for speed. Line 1 contains comments to me, things to do. The program code is 2 pages long, and while it is fairly difficult to type in, it is worth it. I have been using one version or another of this program for several years, and It has made my computing time easier and much more fun.

Finally, credit should go to Lance Wolstrup who gave me invaluable assistance on saving the date back to the program, and also the time input routines.

The TRSTimes readers are invited to modify the program to suit any particular needs.

SHELL/BAS

```
0 CLEAR999:DEFINTD-R:DEFSTRS-Z:
U="SHELL- Model 3 MultiDOS 1.71":GOSUB 3:
PRINT"Written by Jim E. King with BF&B, Public
Domain, NOT for Sale":
 U="":POKE 16409,0:
ON ERROR GOTO 70:GOTO 80 'PokeLowerCase;-S
1  ' AUTO BASIC 1,64746,RUN"START3;
 Make Variable table;
Add: Verify File Copy ??,
LList of a File;
Conventional hour #s line300 & Inkey$ input
3 PRINT TAB(30-LEN(U)/2)U:RETURN
4 FOR J=0 TO 555:NEXT:RETURN
6 IF Z="<"OR LEN(Z)>12 THEN JU=2:GOSUB 9:
GOTO 99
7 IF LEN(Z) THEN FOR L=1 TO LEN(Z):
O=ASC(MID$(Z,L,1)):
MID$(Z,L,1)=CHR$(O+32*(O>96)):
NEXT:RETURN:ELSE RETURN
8 Z=INKEY$:IF Z=""THEN 8 ELSE
IF Z=CHR$(31)THEN END
ELSE GOSUB 7:RETURN
9 PRINT CHR$(29)STRING$(JU+1,27)CHR$(31);:
JU=0:RETURN
10 PRINT"DIR "V"(A,I)":CMD"DIR "+V+"(A,I)":
GOSUB 8:GOTO 102
11 PRINT" FMAP "V:CMD"FMAP "+V:X=" ":GOTO 99
12 PRINT TAB(48)CHR$(27)"/PCL?":GOSUB 8:
IFZ="P"ORZ="Y"ORZ=""ORZ="@"THEN
X=X+"/PCL":RETURN ELSE RETURN
13 IF LEN(Z) THEN GOSUB 7:X=Z:GOSUB 12
14 GOSUB9:PRINT"LISTING "X:CMD"LIST "+X:
GOSUB 8:GOTO 102
16 GOSUB 9:
PRINT"Are you SURE you want to KILL: "X", <Y>/N?"
:GOSUB 8:IF Z="Y"OR Z="K"THEN GOSUB9:
PRINT"KILLING "X;:
CMD"KILL "+X:
Z=V:PRINT TAB(48)"KILLED
17 GOTO 102
18 GOSUB 9:PRINT"Copy "X" from Drive#? ";:
GOSUB 8:Y=Z:
PRINT STRING$(2,8)" "Y" to Drive#? ";:
GOSUB 8:
IF Z="<"THEN 99
ELSE:PRINT TAB(56)"COPYING "X":"Z:
CMD"VERIFY":
CMD"COPY "+X+":"+Y+" :"+Z:V=Z:
GOSUB 8:GOTO 102
20 PRINT TAB(8)CHR$(27)CHR$(31)X" to ";:
INPUT Z:IF Z="<"THEN 99 ELSE GOSUB 7:
PRINT TAB(48)CHR$(27)Z:
CMD"RENAME "+X+" "+Z:
GOSUB 8:GOTO 102
22 PRINT TAB(47)CHR$(27)" Drive #? ";:
GOSUB 8:PRINT Z:
IF ASC(Z)>47 AND ASC(Z)<52 THEN V=Z:
CMD"RESTOR "+X+":"+V:
GOSUB8
23 GOTO 102
24 IF ASC(Z)<49 OR ASC(Z)>51 THEN ZC="":
GOTO 28 ELSE
PRINT TAB(40)CHR$(27)"<1> or <2> Sides? ";:
GOSUB 8:PRINT Z:
IF VAL(Z)>0 AND VAL(Z)<3 THEN ZS=Z ELSE
JU=2:GOSUB 9:GOTO 99
26 ZC=ZD+"(SI="+ZS:PRINT"CMD'CONFIG "ZC
28 CMD"CONFIG "+ZC:
JU=3:GOSUB 9:GOTO 99
40 U="REMOVE WRITE PROTECTION"
42 FORI=0TO2:
GOSUB9:GOSUB4:GOSUB3:GOSUB4:
NEXT:
GOSUB8:GOTO360
70 IF ERR=120 OR ERR=136 THEN 40
72 PRINT ERR"Error-Line"ERL;:CMD"E":V=""
74 RESUME 99
79 WD=FNDY$(FNDN!(IR,MO,DY)):RETURN
'day of week
80 A$="727272":A=VAL(A$)
81 DEF FNDN!(Y%,M%,D%)=Y%*365+INT((Y%-1)/4)
 +  (M%-1)*28
 +VAL(MID$("000303060811131619212426",(M%-1)*2
 +1,2))-((M%>2)AND((Y%ANDNOT-4)=0))+D%
'-Day# p110
82 DEF FNRY%(N!)=INT((N!-N!/1461)/365)
' - Year p111
83 DEF FNRJ%(N!)=N!-(FNRY%(N!)*365
 +INT((FNRY%(N!)-1)/4))
' - p111
84 DEF FNRM%(J%,Y%)=-((Y%ANDNOT-4)< >0)*
(1-(J%>31)-(J%>59)-(J%>90)-(J%>120)-(J%>151)-
(J%>181)-(J%>212)-(J%>243)-(J%>273)-(J%>304)-
(J%>334))-((Y%ANDNOT-4)=0)*(1-(J%>31)-(J%>60)-
(J%>91)-(J%>121)-(J%>152)-(J%>182)-(J%>213)-
(J%>244)-(J%>274)-(J%>305)-(J%>335))
85 DEF FNRD%(Y%,M%,J%)=(J%-((M%-1)*28
 +VAL(MID$("000303060811131619212426",(M%-1)*2
 +1,2))))+((M%>2)AND((Y%ANDNOT-4)=0))
' - p111
86 DEF FNDY$(N!)=MID$("Friday   Saturday Sunday
Monday   Tuesday  WednesdayThursday ",(N!-INT(N!/7)
*7)*9+1,9)
' - p110
87 DEF FNN$(N)=RIGHT$(STR$(N),LEN(STR$(N))-1)
' #-->$
88 DEF FNZ2(N)=RIGHT$(STR$(N),2)
```

```
89 V=" "
90 IF VAL(TIME$)=0 THEN 200
92 IR=VAL(MID$(TIME$,7,2))+1900:
MO=VAL(LEFT$(TIME$,2)):
DY=VAL(MID$(TIME$,4,2)):GOSUB 79
99 PRINTWD" "LEFT$(TIME$,8)
" DIR<0,1,2> dAte confiG Format caT Map List
copY rEname Kill reStore Vfu Zap   elecPencil
Other?":GOSUB 8
102 IF Z="A"THEN 200'date
104 IF Z="G"THEN
PRINT"Configure Sides of Drive # (1-3)? ";:
GOSUB 8:PRINTZ:GOSUB6:
IFASC(Z)>48 AND ASC(Z)<52 THEN ZD=Z:GOTO 24
ELSE 24
106 IF ASC(Z)>47 AND ASC(Z)<52 THEN V=Z:
PRINT"DIR "V:
CMD"DIR "+V:GOSUB8:GOTO102
108 IF Z="*"    THEN V="0":GOTO 10 ELSE IF Z="!"
THEN V="1":GOTO 10
ELSE IF Z=CHR$(34) THEN V="2":GOTO 10
ELSE IF Z="#"THEN V="3":GOTO 10'(A,I)
110 IF Z="T"THEN
PRINT"CAT of a Foreign Disk (L)<1,2,3>? ";:
GOSUB8:PRINT Z:
IF ASC(Z)>48 AND ASC(Z)<52 THEN V=Z:
CMD"CAT "+V+"(L):GOSUB 8:GOTO 99
112 IF Z="M"THEN IF ASC(V)>47 AND ASC(V)<51
THEN GOTO 11 ELSE
PRINT"Display a MAP of Drive # <0,1,2,3>?";:
GOSUB 8:GOSUB 6:V=Z:GOTO 11
114 IF Z="F"THEN PRINT"Format":CMD"FORMAT
116 IF Z="B"THEN PRINT"Backup":CMD"BACKUP"
ELSEIF Z="V"THEN PRINT"VFU":CMD"VFU"
ELSEIF Z="Z"THEN PRINT"ZAP!":CMD"ZAP":GOTO 99
118 IF Z="L"THEN
PRINT"List FileName ("X"):Drive#? ";:LINEINPUT Z:
GOSUB 6:GOTO 13
120 IF Z="Y"THEN
PRINT"Copy FileName ("X") Without Drive #? ";:
LINEINPUT Z:GOSUB6
IF LEN(Z)THEN GOSUB 7:X=Z:GOSUB12:GOTO 18
ELSE 18
122 IF Z="E"THEN
PRINT"Rename: OldName ONLY ("X")? ";:
LINEINPUT Z:GOSUB6:
IF LEN(Z)THEN GOSUB 7:X=Z:GOSUB12:GOTO 20
ELSE 20
124 IF Z="K"THEN PRINT"KILL FileName ("X")? ";:
LINEINPUT Z:GOSUB6:IF LEN(Z)THEN GOSUB 7:
X=Z:GOSUB12:GOTO 16 ELSE 16
126 IF Z="S"THEN PRINT"Resurrect FileName ("X")?";:
LINEINPUT Z:GOSUB6:IF LEN(Z)THEN X=Z:
GOSUB12:GOTO 22 ELSE 22
128 IF Z="$"PRINT"S/BAK ";:
CMD"COPY S S/BAK":
PRINT"Saving S":SAVE"S"
130 IF Z="I"THEN PRINT"Make File Invisible? ";:
LINEINPUT Z: GOSUB6:
CMD"ATTRIB "+Z+"(I)":
GOSUB8:GOTO102
132 IF Z="P"PRINT"Electric Pencil":
CMD"PENCIL"
170 'IF Z="C"THEN PRINT"Run Account AZC":
RUN"AZC
175 'IF Z="4"THEN PRINT"Run #,$,$,$ Database, D4":
RUN"D4" ELSE IF Z="D"
THEN PRINT"Run 2 Dimensional $ Database, DBZ":
RUN"DBZ
185 'IF Z="W"THEN PRINT"Run WIRE":RUN"WIRE"
195 IF Z="O"THEN PRINT"Run Name/FileSpec ("X"): ";:
LINEINPUT Z:
IF Z="<"THEN 99 ELSE IF LEN(Z) THEN GOSUB 7:
X=Z:RUN X ELSE RUN X
199 GOTO 99
200 K=0:PRINT"PRESS: Right Arrow to Advance 1 Day,
Left Arrow to Back Up 1 Day, > or < to increment by
10 days.  <Enter> when Date is Correct"
210 IR=FNRY%(A):
J=FNRJ%(A):
MO=FNRM%(J,IR):
DY=FNRD%(IR,MO,J)
220 GOSUB 79:
PRINTWD" "FNZ2(MO)"/"FNZ2(DY)"/19"FNZ2(IR):
GOSUB 8
230 IF PEEK(14400)=64 THEN A=A+1:K=1:
GOSUB 9:GOTO 210
240 IF PEEK(14400)=32 THEN A=A-1:K=1:
GOSUB 9:GOTO 210
250 IF Z=">"THEN A=A+10:K=1:GOSUB 9:
GOTO 210
260 IF Z="<"THEN A=A-10:K=1:GOSUB 9:
GOTO 210
270 IF Z=CHR$(13)OR Z="1"THEN 290
280 GOSUB 9:GOTO 220
290 IR=IR-1900:
POKE&421C,MO:
POKE&421B,DY:
POKE&421A,IR
'date
300 IF K=0 THEN CLS:GOTO 92
ELSE AA$=STR$(A):
PRINTTAB(33)CHR$(27)"Saving
330 B=PEEK(VARPTR(A$)+2)*256
+PEEK(VARPTR(A$)+1):
B=B+65536*(B>32767)
340 BB=PEEK(VARPTR(AA$)+2)*256
+PEEK(VARPTR(AA$)+1):
BB=BB+65536*(BB>32767)
350 FOR I=0 TO 5:
POKE B+I,PEEK(BB+I+1):
NEXT
360 SAVE"S":
CLS:GOTO92
```

A FAST SYSTEM DRIVE pt. 2

HOW ABOUT A 'GRAFDISK'?

128K Model 4 - LS-DOS 6.3.0 & 6.3.1

By Lance Wolstrup

In the last issue, part 1 of this mini-tutorial covered the installation procedure and usage of a normal 128K Model 4 Memdisk. In essence, the system files were copied up to the Memdisk, which was then configured to be the system drive with the SYSTEM (SYSTEM = d) command.

Now, I have a 15 meg hard disk hooked up to my three-floppy Mod 4, partitioned to three logical drives of 5 megs each. For my own personal reasons, the hard drive partitions are used to store data only; thus none have the LS-DOS system files. Whenever I boot, it is done completely from the floppy in drive :0. Upon completion of the boot, the drives are as follows:

```
drive :0 (first floppy)
drive :1 (second floppy)
drive :2 (third floppy)
drive :3 (first hard drive partition)
drive :4 (second hard drive partition)
drive :5 (third hard drive partition)
```

At this point I set up the Memdisk as drive :6, copy the system files to it, and then switch the drives with the SYSTEM (SYSTEM = 6) command. The drives are now:

```
drive :0 (Memdisk)
drive :1 (second floppy)
drive :2 (third floppy)
drive :3 (first hard drive partition)
drive :4 (second hard drive partition)
drive :5 (third hard drive partition)
drive :6 (first floppy)
```

The problem with this setup is that all the system utilities, such as FORMAT and BACKUP, are now located on drive :6. Trying to FORMAT a fresh disk in drive :1 will cause a hang-up because the system will try to find FORMAT/CMD on the unformatted diskette.

To solve this minor problem I needed to make room on the Memdisk for the FORMAT/CMD utility file, so I removed SYS0/SYS and SYS13/SYS.

```
REMOVE SYS0/SYS.SYSTEM6:0
REMOVE SYS13/SYS.SYSTEM6:0
```

SYS0 is not needed, as it is at this point stored in memory. SYS13 is the Extended Command Interpreter (ECI). It serves no purpose, unless a user program has been copied to it. Since I never use SYS13/SYS, it was expendable.

```
COPY FORMAT/CMD.UTILITY:6 :0
```

It would have been handy to also have BACKUP/CMD available from the Memdisk, but I was fresh out of Memdisk-space, so I had to do without this utility.

It is possible to make room for BACKUP/CMD on the Memdisk by removing SYS5/SYS and SYS9/SYS. However, I chose not to do that as I need these system files when I do assembly language programming.

SYS5 and SYS9 contain the code for DEBUG and the extended DEBUG commands. If you do not plan to use DEBUG, by all means, get rid of these two files.

```
REMOVE SYS5/SYS.SYSTEM6:0
REMOVE SYS9/SYS.SYSTEM6:0
```

Now there is enough room on the Memdisk for BACKUP/CMD, so:

```
COPY BACKUP/CMD.UTILITY:6 :0
```

My startup file, called SYSTEM/JCL, is as follows:

```
SYSTEM (DRIVE = 6, DRIVER = "MEMDISK")
D
D
Y
BACKUP /SYS:0 :6 (S)
SYSTEM (SYSTEM = 6)
REMOVE SYS0/SYS.SYSTEM6:0
REMOVE SYS13/SYS.SYSTEM6:0
COPY FORMAT/CMD.UTILITY:6 :0
```

With this minor annoyance now solved, I have been happily using Memdisk; that is, until Allen Jacobs asked me if I'd ever heard of a utility called GRAFDISK!

GRAFDISK is written by William R. Bowman and is available on TRSLINK #26 (December 1989) in a file called GRAFDSK/ARC.

For the readers not familiar with files having the /ARC extension, let me just mention that this type of file cannot be used directly. It is usually a collection of files saved in compressed format into one file and, before the files are usable, they have to be 'DEarced'; that is, they have to be restored to their original format. This can be done easily with another program available from TRSLINK, called DEARC4V2/CMD (#21, July 1989).

If you only have two drives, you will have to make room on your boot disk for both DEARC4V2/CMD and GRAFDSK/ARC, and then copy these files from the TRSLINK disks in drive :1 to the boot disk in drive :0.

You can now issue the command:

```
DEARC4V2 GRAFDSK :1
```

This will copy the following eleven files to the data disk in drive :1.

```
GDINFO/TXT, GDLDSV/DOC, GDLOAD/CMD
GDLOAD/FIX, GDSAVE/CMD, GDSAVE/FIX
GRAFDISK/DCT, GRAFSYSB/JCL, SHRTBOOT/CMD
SWAP/CMD, SWAP/DOC
```

Of these files, GRAFDISK/DCT, GDLOAD/CMD, GDSAVE/CMD and SWAP/CMD will be of immediate interest to us.

GRAFDISK/DCT is the all important one here. It is a driver that completely replaces MEMDISK/DCT when setting up a memory disk drive. This new driver does everything that MEMDISK/DCT does. What makes it more powerful is, that if you have a hi-res board installed in your Model 4, it lets you use the hi-res board memory in addition to the 64K in banks 1 and 2. In the case of a Radio Shack hi-res board, it is an extra 32K, giving you a 96K memory disk drive. The Micro Labs board adds only 24K, but that still gives you an 88K memory drive.

If you don't have a hi-res board, don't stop reading now, you can still use this driver. Of course, you won't get the added memory, but you can benefit greatly from the two utilities, GDSAVE/CMD and GDLOAD/CMD.

Copy GRAFDISK/DCT to the boot disk in drive :0 and then issue the command:

```
SYSTEM (DRIVE=2, DRIVER="MEMDISK")
```

Your screen will now display the following prompt:

```
[A] Banks 1, 2
[B] Banks 1, 2 and Grafmem
[C] Disable GrafDISK

Your choice ?
```

If you have a graphics board installed, choose B, otherwise choose option A.

You will now be asked:

```
Do you wish to format it <Y,N> ?
```

Answer 'Y' to format the Memdisk (now called GrafDISK). After the format has been verified, you will be told that 'GrafDISK Successfully Installed'.

If you are working with LS-DOS 6.3.0 you are now in business. Depending on the presence of a hi-res board, you will have a memory disk drive of either 96K or 64K.

However, if you are using LS-DOS 6.3.1, you will have noticed that the driver does not work. The problem is GRAFDISK/DCT was written to work with LS-DOS 6.3.0. When LS-DOS was upgraded from 6.3.0 to 6.3.1 the system code became slightly larger and, unfortunately, it GRAFDISK/DCT now overwrites part of SYS8/SYS.

As LS-DOS 6.3.1 is now the standard DOS for Model 4, a fix was needed, and this fix can be found on TRSLINK #31 (May 1990). It is a BASIC program, written by L. E. Evans, called GDMOVE/BAS. Running this program will patch GRAFDISK/DCT to load at 3000H, and it will now work with both LS-DOS 6.3.0 and 6.3.1.

I encountered a problem when I tried to run GDMOVE/BAS. BASIC stopped and told me there was a syntax error in line 20. Indeed there was, and a very deceptive one at that.

At the end of the first line, the command 'PRINT' is wrapped around to the next line. Somehow, the TRSLINK text editor inserted a blank between the PRI and the NT.

This also occurs in line 30. Same problem. At the end of the first line, the command 'PRINT' is wrapped around to the next line. The blank is inserted between P and RINT.

Edit out the two blanks and the program will now correctly patch GRAFDISK/DCT. The syntax errors may be unique to my copy of the TRSLINK #31disk, but just in case it is not, you now know what to do.

It is obvious why you should go through all of this trouble if you have a hi-res board installed - getting 32K (or 24K) of extra memdisk space allows more utilities or programs to reside there. In my case, I now have EDAS, TED and a disassembler up there and, boy, execution is fast.

What is not so obvious, however, is why you should go through the above trouble in order to use GRAFDISK/DCT if you do not have a hi-res board installed.

The key to why you should use GRAFDISK/DCT, even if you do not have a graphics board installed, is the two utilities, GDSAVE/CMD and GDLOAD/CMD. Read on!

The easiest way to set up a MEMDISK or GRAFDISK is to write a /JCL file to send all the commands to DOS, rather than typing them yourself each time you boot.

My STARTUP/JCL file to initialize the GRAFDISK in my 2-drive Model 4P with a Radio Shack hi-res board and a 15 meg hard drive (partitioned into 4 logical drives) is:

```
SYSTEM (DRIVE=2,DRIVER="GRAFDISK")
B
Y
BACKUP /SYS:0 :2 (S)
REMOVE SYS0/SYS.SYSTEM6:2
REMOVE SYS13/SYS.SYSTEM6:2
COPY FORMAT/CMD.UTILITY:0 :2
COPY BACKUP/CMD.UTILITY:0 :2
COPY EDAS/CMD:3 :2
COPY TED/CMD:3 :2
COPY DD/CMD:3 :2
COPY DDFORM/CMD:3 :2
COPY TRSLABL4/CMD:3 :2
SYSTEM (SYSTEM=2)
SYSTEM (DRIVE=1, SWAP=2)
```

The STARTUP/JCL file to initialize the GRAFDISK in my desktop 3-drive Model 4 with a 15 meg hard drive (partitioned into 3 logical drives), but without a hi-res board is:

```
SYSTEM (DRIVE=6,DRIVER="GRAFDISK")
A
Y
BACKUP /SYS:0 :6 (S)
REMOVE SYS0/SYS.SYSTEM6:6
REMOVE SYS13/SYS.SYSTEM6:6
COPY FORMAT/CMD.UTILITY:0 :6
COPY TED/CMD:4 :6
SYSTEM (SYSTEM=6)
SYSTEM (DRIVE=1,SWAP=6)
SYSTEM (DRIVE=2,SWAP=6)
```

Running any /JCL file is very slow. Executing the GrafDISK/JCL with 'DO STARTUP' upon boot-up takes better than a minute, which is almost as slow as my PC. This is where GDSAVE/CMD and GDLOAD/CMD come to the rescue.

After initializing and copying all the programs that will fit on the GrafDISK, you can use GDSAVE/CMD to write the contents of the high banks of memory, the contents of the graphic memory, all 8 DCT's, and the GrafDISK driver to a disk file.

If you have a hi-res board and chose option B when initializing the GrafDISK, this file will be 96K in length. Without a hi-res board, and you chose option A, the file will be 64.5K in length. I have chosen to call the file GRAFDISK/INI and to store it on my first hard drive partition (:3), so the command syntax if a hi-res board is present (option B) is:

```
GDSAVE GRAFDISK/INI:3
```

The syntax if the hi-res board is not present (option A) is:

```
GDSAVE GRAFDISK/INI:3 (A)
```

Notice that you must specify the (A) parameter if a hi-res board is not present.

Now we get to the good part. From now on, whenever you boot, you need not execute the slow STARTUP/JCL file. Rather, you can use GDLOAD/CMD to initialize the GrafDISK and restore its contents to what it was when you saved it. It is much faster than the /JCL method, approximately 15 seconds, as opposed to over a minute.

The command syntax if a hi-res board is present (option B) is:

```
GDLOAD GRAFDISK/INI:3
```

If the hi-res board is not present (option A), the syntax is:

```
GDLOAD GRAFDISK/INI:3 (A)
```

If you are running LS-DOS 6.3.0, you need not patch GRAFDISK/DCT with GDMOVE/BAS. I recommend that you do, however, as the patched version will run on both 6.3.0 and 6.3.1, and you really should be using 6.3.1 anyway.

Users of LS-DOS 6.3.0 will have problems with portions of my STARTUP/JCL files. The lines with the SYSTEM (DRIVE=n,SWAP=m) will not work with 6.3.0. These parameters to the SYSTEM command were not implemented until 6.3.1.

The good news is that the SWAP/CMD file, written by Franklin Veaux, can completely replace the SYSTEM (DRIVE=n,SWAP=m) command. SWAP/CMD will work on both 6.3.0 and 6.3.1 and, frankly, it is more convenient to use. The syntax for SWAP/CMD is:

```
SWAP d=d,d=d....,d=d
```

For example, in the STARTUP/JCL for my desktop Model 4, the lines

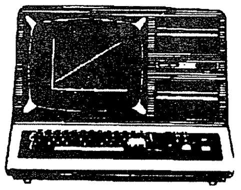
```
SYSTEM (DRIVE=1,SWAP=6)
SYSTEM (DRIVE=2,SWAP=6)
```

could be replaced with

```
SWAP 1=6,2=6
```

In conclusion, let me say that setting up the desktop 4 and the 4P with GrafDISK, while being a bit of trouble to start with, has proved to be well worth the effort. The added convenience and speed make the machines even more pleasurable to work with.

The Model 4 is alive and kicking, still being supported with good software. Thanks to William Bowman and Franklin Veaux for these public domain programs that makes a great computer even greater.



Faster Sequential Disk Input on the Model 4

by J.F.R. Slinkman

The usual method of reading a sequential disk file under TRSDOS/LS-DOS 6 is to read it one byte at a time by pointing the DE register to the File Control Block (FCB) and invoking the @GET SVC.

However, in certain applications there are much better and faster methods. One such method is "disk buffering."

Obviously, RAM limitations are why disk buffering is so rare as to be almost non-existent in TRS-80 software. However, the benefits of disk buffering are such that the extra effort and the extra RAM required will often pay for themselves many times over in program performance.

Disk I/O buffers, at least in the MS-DOS world, are usually 4K or 8K in size. I don't know why they choose those particular values. It may just be that programmers are in the habit of thinking in terms of even powers of two.

For BUFTRACK/ASM and BUFTRAK2/ASM, two of the three demonstration listings for this article, I used a buffer size of 4.5K. I chose this value because it is the amount of data on one track of a standard Model III or Model 4 180K disk drive. The other listing, BUFSECTR/ASM, merely makes better use of the 256-byte buffer any open file must have under the operating system anyway.

In all three routines, the next byte in the file will be returned in the A register by simply making a CALL to GETBYTE. Also, to only the AF and DE registers are affected by any of them, the same registers used by the @GET SVC.

Since BUFSECTR/ASM is the easiest and simplest, and takes the least RAM of the three, let's look at it first:

First, be aware that this routine can only be used on files of a known size, (e.g., /HR graphics files) or files which contain end-of-data markers or other information, including known size after decompression, within them to tell the program when to stop reading, such as /CHR, /MAC, /GIF and VisiCalc files. Fortunately, this is rather common with bitstream (e.g., /GIF and other LZW-compressed files), bytestream (e.g., /CHR and /MAC files), and meta files.

To reduce the amount of code, the normal 256-byte I/O buffer is placed on a memory page boundary. The FCB can be anywhere that FCB's normally can be.

Upon entry, at GETBYTE, DE is loaded with a pointer to the last byte which was read from the file. The initial value of this pointer, which is the address of the last byte in the buffer, causes the routine to read the first sector the first time the routine is CALLed.

The LSB of this pointer is incremented to point to the byte to be read, which will set the Z flag if all the data in the buffer has been read. Thus, on an NZ condition, the branch is made to GTB010, where the updated pointer is stored for the next invocation, the data byte is loaded into the A register, and the RETurn made to the calling routine.

If the INC E instruction sets the Z flag, then the next sector is read into the buffer. Since DE already points to the first byte of the buffer, control passes to the instruction at GTB010, described above.

It would be easy to modify this routine to replace the JR NZ,ERROR instruction with RET NZ, and add a CP A instruction just before the RETurn, which would have the effect of returning with the Z flag set if there is a valid value in A, and an error number in A if the routine returns with NZ.

The code starting at PUTBYTE illustrates how the same concept is applied to writing data to disk. Output to disk also requires additional code in the main program to handle writing a partially filled buffer before closing the file. If you don't need to maintain the EOF byte in the FCB, this can be done simply by calling LASTSCT.

This method gives a slight improvement in speed over the @GET method in many cases, but, surprisingly, gives a substantial improvement when reading from an ERAM-DISK using PEXMEM.

The next routine is BUFTRACK/ASM, which has the same file type limitations as BUFSECTR/ASM, namely that the program must have some way of knowing, from the contents of the file, when to stop reading from the file.

Again, to reduce the amount of code, the buffer must begin on a RAM page boundary.

On entry, at GETBYTE, DE is loaded with a pointer to the last byte previously read. This pointer is immediately incremented to point to the byte to be read.

Then the msb of the pointer is tested to make sure the pointer still points to unread data in the buffer. If so, control passes to GBT040, where the updated pointer is stored, the data byte loaded into A, and RETurn made to the calling routine.

If all data in the buffer has been "used up," then the HL and BC registers are saved, and B loaded with the number of sectors which must be read to refill the buffer. HL is pointed to the msb of the buffer address in the FCB, which is then loaded with the msb of the address of BUFFER.

The sectors are then read into the buffer. If an error occurs, a branch is made to GBT020, where limited error checking is done. If no error, the A register is loaded with 28, the code for an end-of-file error. This is done only to save code, since an EOF error is not fatal, and is not even acted upon. At GBT020, BC and HL are restored, and the check made for errors.

If the error involves anything other than an attempt to read past the end of the file, it is treated as fatal. Otherwise it is simply ignored, since it is assumed the program will know when to stop CALLing this routine.

Assuming the error is not fatal, DE is loaded with the address of BUFFER, the location of the first byte of new data, BUFPTR is updated, the data byte loaded into A, and the RETurn made to the CALLing routine.

This routine would require substantial alteration to perform more sophisticated error handling. If you need it, it would probably be easier just to use the BUFTRAK2/ASM routine described below.

The speed increases produced by this routine are substantial, as you will see from the results chart.

The final routine, BUFTRAK2/ASM, is more general in that can be used to read any type of file sequentially. Upon return, the Z flag indicates valid data in the A register, and NZ indicates that A contains an error number. This routine does not require the 4.5K buffer to begin on a RAM page boundary.

On entry, at GETBYTE, DE is loaded with a counter of the number of remaining unread bytes in the buffer, which is immediately checked for zero. If non-zero, control goes to GBT060, where the counter is decremented and stored. Then DE is loaded with a pointer to the next available byte, which is loaded into A. This pointer is incremented and stored for the next invocation. The CP A instruction sets the Z flag for the RETurn to the calling routine.

If there is no unread data in the buffer, a check is made to see if the disk file has been fully read. This is done by comparing the ERN (Ending Record Number) and NRN (Next Record Number) fields of the FCB. If these two values are equal, then the disk file has been fully read; so A is loaded with the code for an EOF error and the Z flag is reset for return to the calling routine.

If the two values are not equal, there is still data to be read from disk; so control passes to GBT010. Here the disk file is read into the buffer in the same manner as within BUFTRACK/ASM. Assuming no errors, DE is loaded with the number of bytes in the buffer, which is the new counter. HL is used to update the pointer to the next data byte. Then the BC and HL registers are restored before branching to GBT060, which is described above.

If an error does occur on one of the disk reads, control goes to GBT040, where error checking is done. Errors related to reaching the end of the disk file cause control to pass to GBT050. Otherwise, the stack is cleared and a RETurn-with-error is made to the calling routine with the Z flag reset.

If the end of the disk file has been reached, the number of valid bytes in the buffer must be calculated. At GBT050, the value in B is subtracted from the number of sectors worth of data the buffer can hold. The result, which is the number of sectors actually read, is stored in D, and becomes the tentative msb of the new counter value.

Then the end-of-file offset byte from the FCB is loaded into E, and then checked for zero. If zero (which means 100H in this case) then DE contains the actual number of valid data bytes in the buffer, and control goes to GBT030. If non-zero, then the value is less than 100H; so the msb in D must be decremented to arrive at the actual byte count before going to GBT030.

While this routine has been tested, it was not timed. However, it should produce speeds nearly identical to those of BUFTRACK/ASM.

The time tests listed in the chart were made as follows: four MacPaint files were run through MAC2HR/CMD, a program which resizes 576 x 720 MacPaint images to 384 x 240 TRS-80 hi-res format images, dithers them, and displays them on the screen via a hi-res graphics board (see "Image Processing on the Model 4," TMQ V.ii).

Each of four MacPaint files were read from each of three types of drives on my Model 4D, equipped with an XLR8er running at 6.144 MHz, with settings of 0,1,80 and a 2T refresh cycle duration. The three types of drives were an ERAMDISK, a MISOSYS 40 MB hard drive, and a 360K floppy.

After the original program, which uses the @GET SVC, was run to establish benchmark times, the program was modified to replace the invocations of @GET, first with the code in BUFSECTR/ASM, and then with the code in BUFTRACK/ASM. In the results table, the lines labeled "sector" refer to the BUFSECTR/ASM version, and the lines labeled "track" refer to the BUFTRACK/ASM version.

All times are the average of three runs timed with a stopwatch, rounded to the nearest 0.05 second. They are still stopwatch times, and are therefore subject to human reflex time error. However, the trends shown in the chart are quite clear.

First, look at the "ERAMDISK" column. Frankly, I was astonished with these results when I first noted them. I have not delved deeply into all the code of the component parts of the ERAMDISK driver and the system code which invokes them, but it appears the relative slowness of @GET is due to a considerable amount of system overhead needed to manage single byte I/O from disk, which is bypassed by the much simpler BUFSECTR/ASM, which performs only input, and has no control or output functions, and only works when the logical record length (LRL) equals the sector size of 256 bytes.

Another surprise can be found in the "40 MB Hard" column, where single sector buffering is slightly faster, in all cases, than 4.5K buffering. The differences, however, are not major, the largest being approximately 5% for the 46K file.

The most dramatic differences -- and the only ones I did **not** find surprising (impressive, yes, surprising, no) -- are in the "360K Floppy" column.

The important thing to note here, at least in my opinion, is that in every case (taking into account the effect of the floppy drive coming to a complete stop not once, but twice, with the smallest file) is that "track" buffering turned in better times than @GET from the floppy, the hard drive and, as incredible as it sounds, even @GET from the ERAMDISK!

Therefore, if you're writing software involving sequential disk input, and if there's any way you can spare roughly 4.5K of RAM in your application, you should seriously consider incorporating the buffering techniques demonstrated in BUFTRACK/ASM and BUFTRAK2/ASM.

The result will be significantly faster disk I/O speed no matter what kind of drives the user has.

But if your application is such you absolutely can't spare the extra 4.5K, you should at least think about implementing the techniques in BUFSECTR/ASM, which require only a very few extra bytes of RAM.

Time Comparisons

Process MacPaint Files With MAC2HR/CMD
Program running time in seconds

File	Size in bytes	Type of Access	ERAM-DISK	40 MB Hard	360K Floppy
SEP/MAC	16,203	@get	20.25	21.60	27.15
		sector	17.30	18.10	23.95
		track	16.90	18.40	20.50*
NOV/MAC	22,607	@get	22.40	23.80	31.25
		sector	17.50	18.50	25.15
		track	17.60	19.10	20.90
AUG/MAC	32,105	@get	25.25	27.20	34.90
		sector	18.30	19.70	32.90
		track	18.30	20.30	22.85
INQUIRY/MAC	46,827	@get	29.05	31.90	42.90
		sector	18.75	20.70	42.75
		track	18.60	21.85	24.85

* The highly compressed nature of this file resulted in the floppy drive coming to a full stop between the first and second, and second and third, 4.5K reads, which added substantially to the time.

Frank Slinkman can be reached at:
1511 Old Compton Road, Richmond, Va. 23233
804/741-0205 - CompuServe 72411,650

```
;               BUFSECTR/ASM --        03-May-91
;               by J.F.R. "Frank" Slinkman, 1511 Old Compton Rd.,
;               Richmond, Va. 23233    CompuServe 72411,650
;               Released to the public domain
;
@ERROR   EQU  26
@POSN    EQU  66
@READ    EQU  67
@WRITE   EQU  75
;
;        org  0xx00H          lsb of buffer address MUST
                              ;  be 00H for this code)
;
BUFFER   DS   100H            ;one sector
FCB      DS   20H
;
; Use only for files of known size or which have end-of-
; data information within them. File must have been
; opened with an LRL of 0 (256) and a buffer address of
; BUFFER.
;
; Entry conditions:           None
; Exit conditions:            A contains data byte
;                             DE altered
;
GETBYTE  LD   DE,BUFFER+255  ;initial value forces read
BUFPTR   EQU  $-2             ;storage for data pointer
         INC  E               ;- next data byte
         JR   NZ,GBT010       ;go if still in BUFFER
         LD   DE,FCB          ;  else read a new sector
         LD   A,@READ         ;  from disk
         RST  28H
         JR   NZ,ERROR
         LD   DE,BUFFER       ;- start of new data
GTB010   LD   (BUFPTR),DE     ;save new pointer value
         LD   A,(DE)          ;get the data byte
         RET                  ;and exit
;
ERROR    OR   40H             ;can be as simple as this,
         LD   C,A             ; or a sophisticated as
         LD   A,@ERROR        ; you want to make it
         RST  28H
;
; Entry conditions:           byte to write in A
; Exit conditions:            AF, DE altered

PUTBYTE  LD   DE,BUFFER
BUFPTR2  EQU  $-2
         LD   (DE),A          ;write byte in A to BUFFER
         INC  E               ;advance pointer
         LD   (BUFPTR2),DE ;  and store
         RET  NZ              ;if BUFFER not full
```

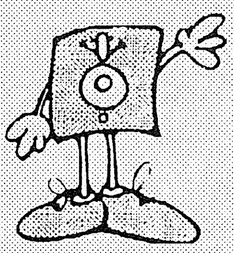
```
LASTSCT  LD    DE,FCB        ;if full, write sector
         PUSH BC
         LD    BC,0          ;p/u sector number
RECNUM   EQU   $-2
         LD    A,@POSN       ;prepare for write
         RST   28H
         INC   BC            ;update sector counter
         LD    (RECNUM),BC ;  for next write
         POP   BC
         LD    A,@WRITE      ;write the sector
         RST   28H
         JR    NZ,ERROR
         RET
;
         END
```

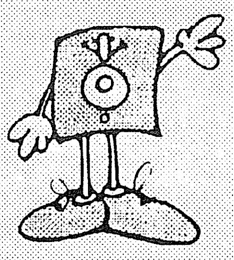
```
; BUFTRACK/ASM          -- 03-May-91
;by J.F.R. "Frank" Slinkman, 1511 Old Compton Rd.,
;Richmond, Va. 23233      CompuServe 72411,650
;Released to the public domain
;
@ERROR   EQU  26
@READ    EQU  67
;
;        org   0xx00H  lsb of buffer address MUST
;                   ;  be 00H for this code)
;
BUFFER   DS    9*1024/2      ;4.5K
ENDBUF   EQU   $
FCB      DS    20H
;
; Use only for files of known size or which have end-of-
; data information within them. File must have been
; opened with an LRL of 0 (256) with a buffer address of
; BUFFER.
;
; Entry conditions:               None
; Exit conditions:                data byte in A
;                                 DE altered
;
GETBYTE  LD    DE,ENDBUF     ;initial value forces read
BUFPTR   EQU   $-2           ;storage for data pointr
         INC   DE            ;--> next data byte
         LD    A,D           ;p/u pointer msb
         CP    .HIGH.ENDBUF ;buffer need loading?
         JR    NZ,GBT040     ;go if not
         PUSH HL             ;   else save regs
         PUSH BC
         LD    B,ENDBUF-BUFFER<-8  ;# sectors
                                   ;to read
         LD    DE,FCB
         LD    HL,FCB+4      ;--> msb of
                             ; buffer address
         LD    (HL),.HIGH.BUFFER
                             ;FCB --> BUFFER
GBT010   LD    A,@READ       ;read a sector
         RST   28H
         JR    NZ,GBT020     ;go if error
         INC   (HL)          ;point FCB to next page
         DJNZ GBT010
         LD    A,28          ;phony EOF err saves code
GBT020   POP   BC            ;restore regs
         POP   HL
         CP    28            ;EOF error?
         JR    Z,GBT030      ;OK if so
         CP    29            ;Out of range error?
         JR    NZ,ERROR      ;OK if so otherwise handle
GBT030   LD    DE,BUFFER     ;--> start of new data
GTB040   LD    (BUFPTR),DE   ;save new pointer value
         LD    A,(DE)        ;get the data byte
         RET                 ;and exit
;-----------------------------------------------------------------
ERROR    OR    40H           ;can be as simple as this,
         LD    C,A           ; or a sophisticated as
         LD    A,@ERROR      ; you want to make it
         RST   28H
;
         END
```

```
; BUFTRAK2/ASM          -- 03-May-91
;by J.F.R. "Frank" Slinkman, 1511 Old Compton Rd.,
;Richmond, Va. 23233      CompuServe 72411,650
;Released to the public domain
;
@READ    EQU  67
;
BUFFER   DS    9*1024/2      ;4.5K
ENDBUF   EQU   $
FCB      DS    20H
OFFSET   EQU   FCB+8
NRN      EQU   FCB+10
ERN      EQU   FCB+12
;
; Use for files of unknown size which do not have end-of-
; data information within them. File must have been
; opened with an LRL of 0 (256) and a buffer address of
; BUFFER.
;
; Entry conditions:               None
; Exit conditions:                DE altered
;                                 if Z, data byte in A
;                                 if NZ, error number in A
;
GETBYTE  LD    DE,0          ;initial value forces read
BUFCTR   EQU   $-2           ;stores # remaining bytes
         LD    A,D           ;is there still unused
         OR    E             ; data in BUFFER?
         JR    NZ,GBT060     ;go if so
         PUSH HL             ;here if must read
         LD    HL,(NRN)
         LD    DE,(ERN)
         SBC   HL,DE         ;are we at end of file?
```

```
        JR      NZ,GBT010       ;go if not
        POP     HL              ;  else restore HL
        LD      A,28            ;End of File error
        OR      A               ;reset Z
        RET                     ;return w/NZ
;
GBT010  PUSH BC                 ;save regs
        LD      B,ENDBUF-BUFFER<-8
                                ;# sectors to read
        LD      DE,FCB
        LD      HL,FCB+4        ;--> msb of buffer
                                ;pointer
        LD      (HL),.HIGH.BUFFER  ;FCB - BUFFER
GBT020  LD      A,@READ         ;read a sector
        RST     28H
        JR      NZ,GBT040       ;go if error
        INC     (HL)            ;F-->CB - next
                                ;memory page
        DJNZ GBT020             ;falls thru if buffer full
        LD      DE,ENDBUF-BUFFER ;new counter
                                 ; value
GBT030  LD      HL,BUFFER
        LD      (BUFPTR),HL     ;address of new data
        POP     BC              ;restore regs
        POP     HL
        JR      GBT060          ;go get data byte
;
GBT040  CP      28              ;was it an EOF error?
        JR      Z,GBT050        ;OK if so
        CP      29              ;'out of range' error?
        JR      Z,GBT050        ;OK if so
        POP     BC
        POP     HL              ;restore regs
        RET                     ;ret w/error # and NZ
;
GBT050  LD      A,ENDBUF-BUFFER<-8
        SUB     B               ;calc # of sectors read
        LD      D,A             ;new counter msb
        LD      A,(OFFSET)      ;p/u EOF offset ptr
        LD      E,A             ;new counter lsb
        OR      A               ;full sector of data?
        JR      Z,GBT030        ;counter OK if so
        DEC     D               ;else adjust msb
        JR      GBT030
;
GBT060  DEC     DE
        LD      (BUFCTR),DE     ;update counter
        LD      DE,BUFFER       ;p/u data pointer
BUFPTR  EQU     $-2
        LD      A,(DE)          ;p/u data byte
        INC     DE
        LD      (BUFPTR),DE     ;update pointer
        CP      A               ;set Z for return
        RET
;
        END
```

TIRED OF SLOPPY DISK LABELS?
TIRED OF NOT KNOWING WHAT'S ON YOUR DISK?
YOU NEED 'DL'





- 'DL' will automatically read your TRSDOS6/LDOS compatible disk and then print a neat label, listing the visible files (maximum 16).
- You may use the 'change' feature to select/reject the filenames to print. You may even change the diskname and diskdate.
- 'DL' is written in 100% pure Z-80 machine code for efficiency & speed.

'DL' is available for TRS-80 Model 4/4P/4D using TRSDOS 6.2/LS-DOS 6.3.0 & 6.3.1. with either an Epson compatible or DMP series printer.

'DL' for Model 4 only $9.95
TRSTimes magazine - Dept. 'DL'
5721 Topanga Canyon Blvd. #4
Woodland Hills, CA 91367

RECOVERY OF LOST SCRIPSIT FILES

Model 4

By M. C. Matthews

This is intended as a guide to saving the text of a file after a reset while writing a Scripsit document.

There are two basic cases. One where part or all of the document is on disk and part in memory, and the other where it is all in memory.

Scripsit stores text on disk after about three or four pages, or after pressing CTRL W. This means that the average letter is entirely in memory. This is the simple case. The document is normally in memory after a reset, and the first step is to secure it by dumping the memory to a disk file.

The first part of SAVEMEM/BAS (see the program listing) does this. It dumps the memory from the start of the buffer at A5C0H to F3FFH to a disk file with the extension /MEM. This file can be read with list, and TED/CMD will also load it, but it is of considerable length and much of it is rubbish.

The second part of the program therefore, recovers the text from the file and filters out a lot of the rubbish, and refiles it in a file called RECOVER/ASC. There will still be some rubbish left which will have to be edited out.

This file can now be converted into a Scripsit document by using the command A from the Scripsit menu. Note that RECOVER/ASC may not be closed in a manner that Scripsit likes. If you get the error message 'File not open', exit from Scripsit, call TED, load it into TED by using CTRL L, and immediately refile it under the same name with CTRL F. Now try Scripsit again, and all should be well.

SAVEMEM/BAS

```
10 CLS:GOTO 60:
REM Savemem 30.11.90 V4 . Saves memory from
A5C0H to FF3FH used by Scripsit, also saves ASCII
from a Scripsit file. M.C.Matthews
20 Z$=""
30 R$=INKEY$:
IF R$="" THEN 30
ELSE R=ASC(R$):
IF R>96 THEN R=R AND 223:
R$=CHR$(R)
40 IF R=13 THEN RETURN
ELSE IF LEN(Z$)>0 THEN IF R=8 THEN
Z$=LEFT$(Z$,LEN(Z$)-1):
PRINT R$;:
GOTO 30
50 Z$=Z$+R$:
PRINT R$;:
GOTO 30
60 PRINT TAB(10)"Do you wish to save the memory
from A5C0H up? ";:
GOSUB 20:
IF LEFT$(Z$,1)="N" THEN 90
70 PRINT:
SYSTEM"DUMP SAVEMEMO/HI
(START=X'A5C0',END=X'FF3F')
80 PRINT:PRINT"Memory now recorded in file
SAVEMEMO/HI. Do you want to recover the ASCII
text?";:
GOSUB 20:
IF LEFT$(Z$,1)="N" THEN CLOSE:
END:
ELSE GOTO 100
90 PRINT "If you want the file just written then
press ENTER, else enter the name of the  file to be
recovered:";:
GOSUB 20:
IF Z$="" THEN F$="SAVEMEMO/HI"
ELSE F$=Z$:
PRINT:
PRINT
100 OPEN"R",1,F$:
FIELD 1,128 AS A$,128 AS B$
110 OPEN"O",2,"RECOVER/ASC":
Y=1
120 GET 1,Y:
FOR I=1 TO 128:
R=ASC(MID$(A$,I,1)):
IF R=13 THEN 130
ELSE IF R<32 OR R>127 THEN 150
130 D$=CHR$(R):PRINT#2,D$;:PRINT D$;
140 IF EOF(1) THEN CLOSE:
PRINT:
PRINT TAB(10)"Text now in file RECOVER/ASC":END
150 NEXT:
FOR I=1 TO 128:
R=ASC(MID$(B$,I,1)):
IF R=13 THEN 130
ELSE IF R<32 OR R>127 THEN 180
160 D$=CHR$(R):
PRINT#2,D$;:
PRINT D$;
170 IF EOF(1) THEN CLOSE:
PRINT:
PRINT TAB(10)"Text now in file RECOVER/ASC":END
180 NEXT:Y=Y+1:GOTO 120
```

CELEBRATE

Model I & III

By Stacy A. Brennan

This BASIC program should be run on all TRS-80 computers on the 4th of July. Type in the listing and then, Happy Birthday.

HAPPY4/BAS

```
10 CLS
20 FOR X=0 TO 127:FOR J=0 TO 2:Y=J+3
30 SET(X,Y+40):SET(X,Y+33):SET(X,Y+27):SET(X,6)
40 IF X=63 THEN 60
50 SET(X,Y+21):SET(X,Y+15):SET(X,Y+9):
SET(X,Y+3)
60 NEXT:NEXT
70 FOR X=6 TO 56 STEP 10
80 SET(X,9):SET(X,13):SET(X,17):SET(X,21):SET(X,25)
90 IF X=50 THEN 120
100 X1=X+5
110 SET(X1,11):SET(X1,15):SET(X1,23):SET(X1,27)
120 NEXT
130 FOR Y=6 TO 32:
SET(62,Y):SET(0,Y):SET(63,Y):SET(1,Y):NEXT
140 IF INKEY$="" THEN 140 ELSE 10
```

PATCH FOR LITTLE BROTHER 2.1.0

Model 4

From Misosys, Inc

Dear folks:

Another small problem has surfaced with LB 2.1.0 affecting only the TRS-80 Model 4 version. The problem relates to record data corruption when the second sequential update or add is performed and the data definition includes the "date-last-updated" field type. The following two command line patches will correct the problem in both the EDIT and ADD modules.

```
PATCH LB/OV5 (D1B,CB=21 38:F1B=2A 3F)

PATCH LB/OV8 (D35,88=21 6C:F35,88=2A 73
```

I apologize for any inconvenience this may have caused you.

Roy Soltoff

CRAFT-80 GROUP SPECIAL '90 DISKETTE

End your PDRIVE-problems with our Automatic Pdrive Recognizer AGCAP/CMD version 5.3. AGCAP automatically detects the Pdrive-settings on any TRS-80 NEWDOS v2.0 diskette. Particularly useful for figuring out other people's diskettes, both on 40 and 80 track disk drives. Works on Model 1, Model 3 and Model 4(p) (in Model 3 mode) without modifications. Manual on disk.
Bonus program on this diskette:
The TRS-80 version of the PC game Sokoban.
Send a letter with your name and address and a twice signed American Express Travelers Check worth $10.00 to:
THE CRAFT-80 GROUP
POSTBUS 73
4854 ZH BAVEL
THE NETHERLANDS
Plerase allow 5-6 weeks for delivery.
Programs supported by original authors.

TRSTimes on DISK #7

Issue #7 of TRSTimes on DISK is now available, featuring the programs from the Jan/Feb, Mar/Apr and May/Jun 1991 issues:

TRSTimes on DISK is reasonably priced:

U.S. & Canada: $5.00 (U.S.)
Other countries:$7.00 (U.S.)

Send check or money order to:

TRSTimes on DISK
5721 Topanga Canyon Blvd. #4
Woodland Hills, CA 91467
U.S.A.

TRSTimes on DISK #1, 2, 3, 4, 5 & 6 are still available at the above prices

TIPS FROM THE 'UPPER LEFT COAST'

By Eric Bagai

I've been here for about a month now, and wherever this is, it is **not** North Hollywood. It is green, and quiet, and wet, and quiet, and cool --sometimes even cold-- but mostly it's quiet. None of these things happen in North Hollywood. They happen here. A lot. It hails twice a week. It rains three times a week. Nobody notices. Yesterday, my neighbor didn't even stop mowing his lawn when it hailed. Joggers keep jogging. Shoppers keep shopping. Street basketball (the local mania) just gets faster: when it rains you can **slide** into the key. And the vegetation grows like it's out of a science-fiction movie. You don't plan where you want a lawn, you plan where you **don't** want a lawn, and then cover that area with something like motor oil or plutonium. And quiet? It's so quiet you can hear the moss growing on your roof. It's so quiet that, well, did you know that the TRS-80 Model 3 has an internal fan?

Computing hasn't changed. The only problem so far is the curse of irregularity. Electrical irregularity. First there is static electricity, which is surprising because I'd expect the humidity to be fairly high. Well, it is when it rains (or hails), but then it dries out. And then it rains. And then it dries out. And then it. . . . The irregularity in the weather seems to act as an electrostatic pump. A static-dissipation strip mounted on the front of your keyboard is a good idea. It is commonly sold for for under $10 by mail order.

Eric Bagai can be reached at
Box 82289
Portland OR 97282 (503) 653-2614

Eric also has a book of his essays for sale. It received one wonderful and one terrible review. So, what do you expect for $6.00, including shipping?

Contrary to popular wisdom, static electricity does **not** zap the data on disks. The amount of static electricity needed to alter a magnetic charge on floppy media is enough to generate ball lightning. But a disk can still **hold** and **transmit** a static charge. If it picks up a charge from you, and you then put it into a drive, that charge goes right into your computer. Then, the charge may be saved back to that disk as a corrupt file. (Static discharge, cosmic rays, and air pollution account for most of the unaccountable errors in your files.)

Power glitches are another story. There are lots of ways to deal with power glitches. Most of them are insufficient or incomplete. The usual remedy is a surge/spike/noise filter that you stick on the end of the power cord. Good ones (with ratings over 120 joules) can cost well over a hundred dollars. Alternatively, hardware hackers like to stick metal oxide varistors (MOVs) all over the inside of their machines (and delight in explaining why you have to use **three** MOVs on your power line). Filters and MOVs will keep minor noise and spikes from spoiling your day, but a brownout, or a one-second outage, and you've got trouble. A nearby lightning strike and you've got burnt trouble.

If your data are what is important, then one approach, advocated by Jim King, is to compute with your drive doors open. He puts "close drive doors" notices into his disk-saving routines.

And of course, the traditional reason for making periodic disk backups is to minimize data loss. But if your hardware is fried, what are you going to run your data **on**? This is especially a problem if you have the only LNW, Tano Dragon, or TLS-8E in town.

If your data are not important but you fear for your machine's safety (like, if you are a heavy gamer, or a Wellbeing, or a GEniac), you can just apply a dose of insurance, like Safeware. It's reasonably inexpensive, and they will promptly pay for the replacement of any hardware that gets smoked.

The only way to avoid the nasty things that come out of your wall socket is to separate your machine from the power grid. There are several ways of doing this. You can use a solar panel, a gel cell, and a converter. (All for under $300 from the Real Goods catalog.) You can use a gas-powered generator. (From $200 to $500 at Sears or your local Honda dealer.) You can even get "broadcast power" from a commercial radio station by using a three-foot copper grid antenna, filtering the received DC output through a battery and converter, and feeding it to your computer. You just have to be within 500 feet of the station's antenna and be ready to deal with the FCC. But the most common solution is an Uninteruptable Power Supply, or UPS.

A UPS will cost you between a hundred and a thousand dollars, but it works. Outages, brown-outs, major spikes, sustained surges, ridiculous line noise, and nearby lightning strikes can all be tamed with a UPS. In fact, you won't even be aware of anything less than a total, sustained blackout. In that event, a small UPS will give you sufficient time to shut down in an orderly fashion, saving any files that are open. A big UPS will allow you to continue working for several hours --which is longer than most blackouts. Just be sure to read the ads in **Computer Shopper** and send for manufacturers' info: you **can** buy a UPS that is worthless if you don't read the fine print.

But a UPS is only part of the answer. In the words of the immortal Keye Luke, "A chain is only as strong as its weakest link." To protect your computer from major power snarfs you must examine **everything** connected to it. What about your printer? If you have a printer plugged into the wall, and its data cable is plugged into your computer, then if your printer gets zapped, your computer gets zapped, too! Unplug the data cable or get a UPS big enough to handle both machines. What about your modem? A lightning strike on the phone lines is as deadly as one on the power line. Either unplug the modem or get a UPS with a phone-line filter. This may add considerably to the UPS cost, and even then, the weakest point in the system will be the phone filter. Best to just unplug the modem when not in use. The same precautions apply to everything that you hang on your computer: monitors, LANs, outboard drives, waxed string; everything.

The last link in the great chain of being zapped is yourself. (You knew that, didn't you?) As you walk across the room toward your computer you can generate enough static electricity to illuminate swamp gas. So, make sure the first thing you touch is the static pad. If you are really concerned about all this, then think redundancy: UPS, insurance, static sink, line filter, data backup system, and duplicate hardware. Of course, the safest procedure is to leave everything unplugged, all the time, and to insulate each piece of equipment with its original packing carton.

But what's life without a little risk? You've lasted this long without losing it all, so what's the big deal? Just figure your own odds, and act accordingly. But as the summer gets hotter, and more demand is made on the Western states power grid, think twice before you turn on that air conditioner. In the long run, the cheapest solution probably **is** the solar panel from Real Goods.

Well, that's all for now from the upper left coast.

Real Goods Trading Co.
(Also water and wind turbine systems!)
966 Mazzoni Street
Ukiah CA 95482
(707) 468-0301

National Computer Accessories
(Stat-Mat, Stat-Touch, Stat-Kybd)
1510 McCormack St., Sacramento CA 95814
(916) 441-1568

Safeware Insurance Agency
Box 02211
Columbus OH 43202
(800) 848-3469

TRS-80 NOSTALGIA

Thinking guilty little thoughts abbout getting a 386?

Then you need:

WHAT I DID WITH MY TRASH

(Ten years with a TRS-80)

A new book by Eric Bagai, bound in mercedes silver an filled with essays, parodies, weird rumors, mythic hacks, and TRS-graphics. The way it was when love was a warm Z-80 - the way it should have been.

Send $5.95 (non US: $6.95) now to:

Flaming Sparrow Press
Box 82289
Portland, OR 97282

Add five cents for autograph.

Order yours today!

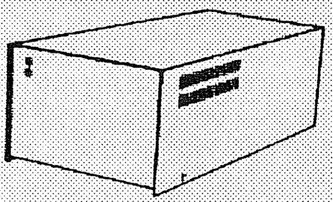
See? You're feeling better already.

HARD DRIVES FOR SALE

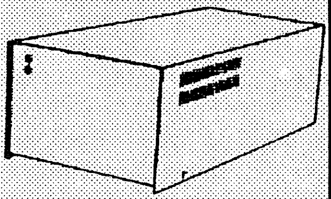
Genuine Radio Shack Drive Boxes with Controller, Power Supply, and Cables. Formatted for TRS 6.3, installation JCL included. Hardware write protect operational. Documentation and new copy of MISOSYS RSHARD5/6 included. 90 day warranty.

5 Meg $175 - 10 Meg $225 - 15 Meg $275 - 35 Meg $445

Shipping costs add to all prices



Roy T. Beck
2153 Cedarhurst Dr.
Los Angeles, CA 90027
(213) 664-5059



GIF4MOD4 AND HR2GIF

for the Model 4/4P/4D

They say a picture is worth 1,000 words. This picture was converted from GIF to TRS-80 format using GIF4MOD4. Until now, Model 4 users had no way to view GIF images or to send their own hi-res graphics creations to other types of computers. GIF4MOD4 will decode any GIF image up to 640 x 480 x 256 (VGA)

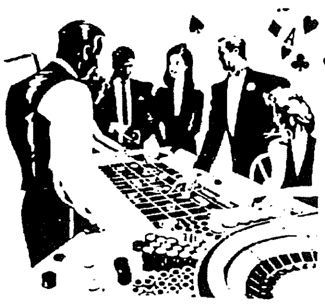
and put it on your hi-res screen. If you have no hi-res board, GIF4MOD4 puts it in an /HR disk file so you can dump it to your dot-matrix printer. HR2GIF converts /HR, /CHR and /BLK files to GIF format.

ORDER FORM -- Mail to
J.F.R. Slinkman, 1511 Old Compton Rd., Richmond, VA 23233
Please send me GIF4MOD4 & HR2GIF for my TRS-80 Model 4. I am enclosing $37.95 + $2.00 S&H (add $2.00 outside N. America. Virginia residents please add 4-1/2% [$1.71]).
Name____________________
Address____________________
City____________ State________ Zip________

THROW THEM DICE

A game for Model I/III & 4

By Lance Wolstrup



I live only a few driving hours from Las Vegas. This convenience, coupled with the fact that I love glitter and gambling, can be tough on the wallet. So, rather than putting my will power to a strenuous test, I simply stay in boring old Los Angeles and play gambling games on my TRS-80. It is not quite as much fun as being in Vegas, but I feel a lot better when I bet $10,000 - and lose!

The very first gambling game I ever played on a computer was called CHUKLUCK. A friend of mine had put together his own computer from a kit. Not only did it work, but he had also managed to get Basic installed and running. I know this doesn't sound like much, but this was in was 1975 or early '76 - before the Model I existed. The computer had no monitor, so all output went to an old converted teletype machine. The Basic language was one of the weird dialects floating around back then - and certainly not from Microsoft.

Anyway, I was invited to his house to view his new treasure. To show it off, he loaded a program (yes, it also had a cassette recorder - very impressive) and after several tries it finally worked. The program was CHUKLUCK and we proceeded to spend the entire evening and half of the night playing that silly game (his wife, I'm sure, had plenty to say the next day). This was my introduction to a 'real' computer and I have never forgotten the fun we had.

To relive that night some 16 years ago, I wrote a version of CHUKLUCK for my TRS-80's and it is presented in the program listing at the end of this article. It will work on Model I and III, as well as on the Model 4.

The game is very simple. You are given $500, and your goal is to convert that money into $100,000 by betting on three dice. You may bet any or all of the money (in whole dollars). After you have placed your bet, you select the side of the die you think will come up. This is called 'your point'. Since you'll throw three dice, you have three chances to win. Further, if your point comes up twice, you double your winnings - and if all three dice matches your point, your winnings are tripled. Sounds as if you'll win $100,000 in no time at all, doesn't it? Believe me, it is much more difficult than that! I have yet to reach $25,000 - and I have logged many hours trying. But maybe your gamesmanship (or is that gamespersonship?) is better than mine. Try it - and have fun.

For anyone interested in the program code, here is a blow by blow description:

- Lines 2 through 6 set up data for the fancy character set used for the program title. A special thanks to the Craft-80 Group in Holland for permission to use this screen font.
- Line 10 defines the integer variables.
- Line 11 checks if PEEK(42)=64. If 64 is found there it is fairly safe to assume that the machine is either a Model I or III, and thus variable SW (screen width) is set to 64. Variable H (horizontal position of the cursor) is set to 2, and 5000 bytes are cleared for string space. If 64 is not found at PEEK(42) the machine is assumed to be a Model 4. The screen width (SW) is set to 80 and, adjusting for the wider display, the horizontal position of the cursor (H) is set to 10. Finally, since Model 4 does not automatically turn off the cursor, it is now turned off with a CHR$(15).
- Lines 13 through 15 read the data forming the program name into the HD$(1), HD$(2), HD$(3) array. The reason that three strings are needed is that the font is three lines high. Thus, HD$(1) holds the top portion of the characters, HD$(2) hold the middle portion, and HD$(3) stores the bottom part.
- Lines 16 through 18 set up the the top of a screen-wide graphic box in BX$(1), the bottom of a screen-wide graphic box in X$(2), the top of the die outline is stored in BX$(3), and the bottom the die outline is is stored in BX$(4). Also, in line 18, DI$ is formed to smoothly move the image of the die across the screen later in the program.
- Line 19 jumps over the subroutines in lines 20 through 70 to the main body of the program, which begins in line 100.
- Lines 20 through 23 are the print to screen subroutines. A GOSUB to line 20 will display whatever is stored in A$ left justified. A GOSUB to line 21 will display whatever is stored in A$ centered, and a GOSUB to line 22 will display the contents of A$ right justified. Variable A$ (the displayable text) and variable V (vertical position of the cursor) must be defined before entering each of these routines. All end up in line 23, which moves the cursor to the desired position and the text in A$ is displayed. If the text needs to be positioned other than left, centered or right justified, a GOSUB to line 23 will

do the job. Entering at line 23 requires that variable H (horizontal position of the cursor), variable V (vertical position of the cursor) and Variable A$(text to be displayed) are all defined prior to entry. To be sure, these routines use a lot of string space, but they make screen formatting extremely easy.

- Lines 30 through 38 is the multiple keystroke INKEY$ routine. I have always disliked the INPUT command because, in some Basics, there is no way to control the user input - and I hate the question mark prompt. This routine solves both problems. Before entry, variable L must be set for the maximum keystroke allowed. In the case where the user is asked to press <ENTER>, L should be set to 0.
- Line 30 stores the current cursor position in variable PO. A$ is 'nilled', the length of the user answer (LE) is set to 0, and the cursor is turned on with CHR$(14).
- Line 31 loops until a key is pressed. While looping, random numbers are constantly stored in variable X. This is simply to generate true random numbers when we need them later in the program. The numbers stored in X here are throwaways. They are never used. The key pressed is stored in I$.
- Line 32 is reached only if a key was pressed. If it was <ENTER>, the cursor is turned back off, and the subroutine is exited. Our keystrokes are stored in A$ or, if just <ENTER> was pressed, A$="".
- Line 33 checks for backspace (left arrow). If LE=0 (no characters entered yet) the routine goes back to line 31 for different keystroke.
- Line 34 also check for backspace (left arrow). If it is a backspace, since we fell through line 33, it means that there is at least one character in the A$ buffer. Thus, we need to erase the previous character by backspacing the cursor with CHR$(8). The length of the string (LE) is now one less than it was (LE=LE-1). Make A$ one character shorter with A$=LEFT$(A$,LE). Decrement the cursor position with PO=PO-1 and go back to line 31 for another keystroke.
- Line 35 locks out control characters (less than 32) and characters higher than lower case z. If any of these undesirable characters are pressed, the routine goes back to line 31 for another keystroke.
- Line 36 checks if we have already reached the maximum length allowed for the user answer. If so, back to line 31 for another keystroke (backspace or ENTER would be the only keystrokes that would now work).
- Line 37 is reached only if a legal character is pressed. The character is added to the A$ buffer. The length of the input is incremented by one, and the keystroke is displayed on the screen. Finally, the cursor position is incremented by one.
- Line 38 sends the routine back to line 31 for another keystroke.
- Lines 40 and 41 contain the subroutine to display the 'press <ENTER> to continue' prompt.
- Line 40 displays the prompt centered.
- Line 41 sets variable L=0 for use with the inkey$ routine, so that only <ENTER> can be pressed. Note that this subroutine does not terminate with a RETURN; instead it uses GOTO 30 to enter the inkey$ routine. There it eventually RETURNs back to the main body of the program in line 32.
- Lines 50 and 51 hold the subroutine that will erase incorrect user input.
- Line 50 saves the horizontal position of the cursor (H) in variable H1. Then the horizontal position is recalculated to point to the first character of the user input.
- Line 51 erases the screen from the cursor to the end of the screen with CHR$(31). The original horizontal cursor position is then copied back to variable H.
- Line 60 is the subroutine that moves the image of the of the die accross the screen.
- Line 70 contain the subroutine that erases the screen from line 7 to the end and then displays how much money the user has. Again, this subroutine does not terminate with a RETURN; instead it uses the RETURN in line 23.
- Line 100 is the actual beginning of the program. The screen is erased and the program title is displayed using the fancy screen font. This is done by using a loop to display each of the three lines.
- Line 120 displays centered a short description of the program on the next line. Note that, since V was used as the loop counter in line 100, it is automatically incremented upon exit. Thus, we don't need to increment it ourselves.
- Line 130 increments the vertical position of the cursor, and the copyright notice is displayed centered.
- Line 140 increments the vertical position of the cursor, and the acknowledgement of the fancy font authorship is displayed centered.
- Line 145 checks the value in variable SW to see if the program is running on a Mod I/III or a Mod 4. If it is a Mod 4, we will make the screen display more pleasing by taking advantage of the longer vertical screen.
- Lines 150 through 170 draws the screen-wide graphic box on the screen.
- Line 175 determines which vertical position of the cursor to use, depending on the program running on Mod I/III or Mod 4.
- Lines 180 through 230 displays the rules of the game inside the graphic box.
- Line 235 stores the user's startup money in variable M.
- Lines 240 and 250 displays the prompt to press <ENTER> and go to the inkey subroutine to process the keyboard response.

- Line 300 erases the screen from line 6 to the end of the display.
- Line 310 draws a graphic line under the program headings.
- Line 320 uses the subroutine in line 70 to display how much money the player has.
- Line 330 prompts the user to place a bet.
- Line 340 erases any possible leftover incorrect bet (gosub 50), and the maximum number of keystrokes allowed for the bet is calculated and stored in variable L. The inkey$ subroutine is then accessed for the user response.
- Line 350 converts the user response (stored as a string in A$) to a numeric and stores it in variable B. If the value in B is a whole number, obviously the player bet a whole dollar amount, and we skip over the 'dollars and cents' error routine to line 400.
- Lines 360 and 370 hold the 'dollars and cents' error routine. It ends by going back to line 330 to prompt for another bet.
- Line 400 check to make sure that the bet is not larger than the available money. If the bet is legal, the program skips over the error routine to line 450.
- Lines 410 and 420 holds the 'bet larger than money' error routine. It ends by going back to line 330 for another bet.
- Line 450 checks if the bet is larger than 0. If so, it jumps over the error routine to line 500
- Lines 460 and 470 contain the 'less than 1' error routine. It ends by going back to line 330 for another bet.
- Line 500 erases the screen from line 9 to the end of the display.
- Line 510 displays the bet the user just made.
- Line 600 prompts the player to enter the side of the die he/she thinks will come up.
- Line 605 accesses the subroutine in line 50, which will erase any erroneous leftover user input.
- Line 610 sets the maximum character input to one (L=1) and then goes to the inkey$ subroutine in line 30 for processing.
- Line 620 converts the user answer in A$ to a numeric in variable P.
- Line 630 checks if variable P contains a number corresponding to the side of a die (1-6). If so, we skip the error routine by jumping to line 700.
- Line 640 and 650 hold the error routine. Here we simply display a message telling the player what the legal input is. Then we go back to line 700 for another try at entering the point.
- Line 700 erases the screen from line 10 to the end of the display.
- Line 710 display the point just chosen.
- Line 720 prompts the player to press <ENTER> to roll the dice.
- Line 730 sets variable L to only accept <ENTER> (L=0) in the inkey$ routine, which is then accessed.
- Line 740 erases the 'press <ENTER> prompt.
- Line 750 selects the three winning numbers using the random number function.
- Lines 760 and 770 simulate the three dice being thrown (one at a time) across the screen. At the end of each die-throw the side number is displayed inside the image of the die.
- Line 800 sets the vertical position of the cursor to line 15. Also, variable MA (number of times your point matches the dice) is set to 0.
- Lines 810 and 820 check to see how many times the point matches the dice. Each time a match is found, variable MA is incremented by 1.
- Line 830 checks if variable MA is zero or non-zero. If MA contains zero, no match was found. If MA is non-zero the point has matched at least one die (the value of MA determines how many times a match was found) and we go to the win routine in line 900
- Line 840 is the losing routine. No match was found so the bet is deducted from the money (M=M-B) and we store a message in A$ telling the player how much money he/she lost.
- Line 850 checks if the player has any money left. If variable M contains a non-zero value , the player still has money left and we branch to the routine in line 1000.
- Line 860 is the 'no money left' routine'. It displays a message to that effect and then prompts the player to press <ENTER>.
- Line 870 sets variable L=0 so the inkey$ routine will only accept <ENTER> and then accesses that routine in line 30. After <ENTER> has been pressed the program visits the subroutine in line 70 to erase the screen from line 7 to the end of the display, and display how much money is left. Then the program jumps to line 1115.
- Lines 900 to 950 hold the win routine.
- Line 900 pays the winnings (M=M+B*MA). Then we begin building the appropriate win message in A$.
- Line 910 checks to see how many times the point matched the dice (MA) and accordingly the program is sent either to line 920, 930 or 940 to continue to build the win message in A$.
- Line 950 continues to build A$ by including a message telling the player how much he/she won.
- Line 1000 is accessed from both the lose routine (if the player still has money left) in line 850 and the win routine in line 950. The building of A$ is finished by adding the 'Press <ENTER>' message and it is then displayed centered on screen line 15.
- Line 1010 sets variable L=0 so only <ENTER> can be pressed in the inkey$ routine in line 30.
- Line 1020 checks if the player has reached the goal of $100,000 or more. If so, the program jumps to the 'break bank' routine in line 1100.
- Line 1030 is reached if the $100,000 or more goal is not reached yet, and the program recycles to line

320 for another round of betting and choosing a point.

- Line 1100 is the 'break bank' routine. Here we access the subroutine in line 70, which erases the screen from line 7 to the end of the display and then displays how much money the player has accumulated.
- Lines 1110 stores the 'congratulations' message in A$.
- Line 1115 is reached from either line 870 or line 1110. Depending on which line we come from, the message in A$ reflects whether we have lost all the money, or we have broken the bank. In either case, the message is displayed on line 9 centered.
- Line 1120 prompts the player if he/she wishes to play again. This message is displayed on line 12 centered.
- Line 1125 calculates the vertical cursor position immediately following the prompt and stores it in variable H1.
- Line 1130 sets variable L=1 to accept only one keystroke in the inkey$ routine in line 30, which is then accessed.
- Line 1135 checks the player input returned in A$. There are three possibilities. The player can have pressed just the <ENTER> key in response to the prompt. This is an incorrect response and if this is the case, it is trapped here and program flow is sent back to line 1120 for another try.
- Line 1140 extracts the ascii value of the user input and stores it in variable A. Then, by performing a boolean AND operation, it makes sure that bit 5 of the value in variable A is stripped (reset), thus ensuring that the value is now the ascii value of the key pressed in upper case.
- Line 1150 checks for the ascii value of upper case Y. If found, the money (M) is reset to $500 and the game starts over by going back to line 320.
- Line 1160 checks for the ascii value of upper case N. If found, the screen is erased and the game ends.
- Line 1170 is reached if the player typed something besides Y, y, N, or n, which is an incorrect response. Thus, the horizontal position of the cursor (immediately following the prompt) is copied from variable H1 back to variable H. Then the erroneous answer is erased from the screen and program flow goes back to line 1120 for another chance at answering the prompt.

CHUKLUC/BAS

```
0 'CHUKLUC/BAS for Model I/III & 4
1 '(c) Copyright 1991 by Lance Wolstrup
  all rights reserved
```

fancy character set data

```
2 DATA 150,190,135,139,155,189,128,128,169,171,151,
       128, 169,171,151,128,128,169,171,151,128,128,
       169,171,151,128,128,169,171,151,160,166,158,
       129,128,128
3 DATA 169,171,151,128,128,128,128,128,169,171,151,
       128,128,169,171,151,128,128,150,190,135,139,
       155,189,0
4 DATA 149,191,128,128,128,128,128,128,170,170,157,
       140,174,170,149,128,128,170,170,149,128,128,
       170,170,149,128,128,170,170,157,149,183,128,
       128,128,128
5 DATA 170,170,149,128,128,128,128,128,170,170,149,
       128,128,170,170,149,128,128,149,191,128,128,
       128,128,0
6 DATA 165,175,180,184,185,159,128,128,154,186,181,
       128,154,186,181,128,128,138,154,189,176,176,
       182,190,133,128,128,154,186,181,130,166,173,
       144,128,128
7 DATA 154,186,181,176,176,184,128,128,138,154,189,
       176,176,182,190,133,128,128,165,175,180,184,
       185,159,0
```

initialization

```
10 DEFINT B-L,N-Z
11 IF PEEK(42)=64 THEN CLEAR 5000:SW=64:H=2
ELSE PRINT CHR$(15);:SW=80:H=10
13 FOR X=1 TO 3:HD$(X)=""
14 READ A:IF A=0 THEN 15 ELSE HD$(X)=HD$(X)
+CHR$(A):GOTO 14
15 NEXT
16 BX$(1)=CHR$(156)+STRING$(SW-2,140)+
CHR$(172)
17 BX$(2)=CHR$(141)+STRING$(SW-2,140)
+CHR$(142)
18 BX$(3)=CHR$(156)+CHR$(140)+CHR$(172):
BX$(4)=CHR$(141)+CHR$(140)+CHR$(142):
DI$=CHR$(191)+CHR$(128)
19 GOTO 100
```

subroutines

```
20 H=0:GOTO 23
21 H=INT((SW-LEN(A$))/2):GOTO 23
22 H=SW-LEN(A$)
23 PRINT@SW*V+H,A$;:RETURN
30 PO=SW*V+H+LEN(A$):A$="":LE=0:
PRINT CHR$(14);
31 I$=INKEY$:IF I$="" THEN X=RND(6):GOTO 31
32 IF I$=CHR$(13) THEN PRINT CHR$(15);:RETURN
33 IF I$=CHR$(8) AND LE=0 THEN 31
34 IF I$=CHR$(8) THEN PRINT@PO,CHR$(8);:
LE=LE-1:A$=LEFT$(A$,LE):PO=PO-1:GOTO 31
35 IF I$<CHR$(32) OR I$>CHR$(122) THEN 31
36 IF L=LE THEN 31
37 A$=A$+I$:LE=LE+1:PRINT@PO,I$;:PO=PO+1
```

```
38 GOTO 31
40 A$="Press <ENTER> to continue ":GOSUB 21
41 L=0:GOTO 30
50 H1=H:H=LEN(A$)
51 PRINT@SW*V+H,CHR$(31);:H=H1:RETURN
60 FOR H=ST TO E STEP-1:GOSUB 23:NEXT:RETURN
70 V=7:A$=CHR$(31):GOSUB 20:
A$="You have $"+MID$(STR$(M),2):GOTO 21
```

beginning of main program

```
100 CLS:FOR V=0 TO
2:PRINT@SW*V+H,HD$(V+1):NEXT
110 '
120 A$="A game of chance for Model I/III & 4":
GOSUB 21
130 V=V+1:
A$="(c) Copyright 1991 by Lance Wolstrup - all rights
reserved":GOSUB 21
140 V=V+1:
A$="Fancy character set is courtesy of the Craft-80
Group":GOSUB 21
145 IF SW=80 THEN V=V+5
150 V=V+1:A$=BX$(1):GOSUB 20
160 Y=V:FOR X=1 TO 7:
V=Y+X:A$=CHR$(149):GOSUB 20:
A$=CHR$(170):GOSUB 22:NEXT
170 V=V+1:A$=BX$(2):GOSUB 20
175 IF SW=80 THEN V=12 ELSE V=7
180 A$="You start the game with $500.":GOSUB 21
190 V=V+1:A$="Use any or all of this money to bet
on which sides will come":GOSUB 21
200 V=V+1:A$="up when you throw 3 dice. Matching
one die will return even":GOSUB 21
210 V=V+1:A$="money. Matching two dice returns
double money, and matching":GOSUB 21
220 V=V+1:A$="all three dice returns triple money.
Only whole dollar bets":GOSUB 21
230 V=V+1:A$="are allowed.  You win the game if
you can collect $100,000.":GOSUB 21
235 M=500
240 V=V+1:A$="Press <ENTER> to play ":
GOSUB 21
250 L=0:GOSUB 30
300 V=6:A$=CHR$(31):GOSUB 20
310 A$=STRING$(SW,140):GOSUB 20
320 GOSUB 70
330 V=9:A$="Make your bet, please: ":GOSUB 20
340 GOSUB 50:L=LEN(STR$(M))-1:GOSUB 30
350 B=VAL(A$):IF B=INT(B) THEN 400
360 V=11:A$="We don't want your spare change":
GOSUB 21
370 V=13:GOSUB 40:GOTO 330
400 IF B<=M THEN 450
410 V=11:A$="You can't bet more than you have":
GOSUB 21
420 V=13:GOSUB 40:GOTO 330
450 IF B>0 THEN 500
460 V=11:A$="Minimum bet is $1.00":GOSUB 21
470 V=13:GOSUB 40:GOTO 330
500 V=9:A$=CHR$(31):GOSUB 20
510 H=9:A$="Your bet: $"+MID$(STR$(B),2):
GOSUB 23
600 V=10:A$="Choose your point: ":GOSUB 20
605 GOSUB 50
610 L=1:GOSUB 30
620 P=VAL(A$)
630 IF P>0 AND P<7 THEN 700
640 V=12:A$="Choose 1, 2, 3, 4, 5, or 6":GOSUB 21
650 V=14:GOSUB 40:GOTO 600
700 V=10:A$=CHR$(31):GOSUB 20
710 H=7:A$="Your point: "+STR$(P):GOSUB 23
720 V=12:A$="Press <ENTER> to roll the dice ":
GOSUB 21
730 L=0:GOSUB 30
740 A$=CHR$(31):GOSUB 20
750 FOR X=1 TO 3:DI(X)=RND(6):NEXT
760 ST=SW-2:E=10
770 FOR X=1 TO 3:V=11:A$=DI$:GOSUB 60:
A$=BX$(3):GOSUB 23:V=V+1:A$=BX$(4):
GOSUB 23:V=V+1:A$=STR$(DI(X)):GOSUB 23:
E=E+15:NEXT
800 V=15:MA=0
810 FOR X=1 TO 3:IF P=DI(X) THEN MA=MA+1
820 NEXT
830 IF MA THEN 900
840 M=M-B:A$="You lost $"+MID$(STR$(B),2)+". "
850 IF M THEN 1000
860 A$=A$+"You lost all your money. Press
<ENTER> ":GOSUB 21
870 L=0:GOSUB 30:GOSUB 70:A$="You had a streak
of bad luck - but maybe it'll get better.":GOTO 1115
900 M=M+B*MA:A$="You matched "
910 ON MA GOTO 920,930,940
920 A$=A$+"once. ":GOTO 950
930 A$=A$+"twice. ":GOTO 950
940 A$=A$+"three times. "
950 A$=A$+"You win $"+MID$(STR$(B*MA),2)+". "
1000 A$=A$+"Press <ENTER> ":GOSUB 21
1010 L=0:GOSUB 30
1020 IF M=>100000! THEN 1100
1030 GOTO 320
1100 GOSUB 70
1110 A$="Congratulations. You broke the bank."
1115 V=9:GOSUB 21
1120 V=12:A$="Would you like to try your luck again
(Y/N) ":GOSUB 21
1125 H1=H+LEN(A$)
1130 L=1:GOSUB 30
1135 IF A$="" THEN 1120
1140 A=ASC(A$):A=A AND 223
1150 IF A=89 THEN M=500:GOTO 320
1160 IF A=78 THEN CLS:END
1170 H=H1:A$=CHR$(31):GOSUB 23:GOTO 1120
```

EXPLORING CONFIG/SYS

By Roy T. Beck

The other day it occurred to me to wonder how the CONFIG/SYS file worked its magic, which allows my 4P's to boot up with the hard drives configured and running. In case you don't already know, the CONFIG/SYS file is created when you run the SYSGEN function. SYSGEN creates and saves CONFIG/SYS, but as an invisible file, which is why you may not have become aware of its presence.

Its purpose is to save a record on the boot floppy of the configuration of the machine at the time SYSGEN is run. On the next bootup, the BOOT ROM loads enough of the system files in order to get started, and then looks for the CONFIG/SYS file. If it finds a CONFIG/SYS file, it shows the word SYSGEN where you expect DOS READY to appear, and installs its contents at the appropriate locations in your RAM. Among the critical parameters are, in my case, the driver and DCT's for my hard disk. (Aha, you just knew I would get around to hard disks eventually, didn't you!)

When these parameters are installed correctly, the machine immediately knows it has hard drives, etc, installed and working. It then tries to verify the correct operation of all the drives. If all is well, you just get DOS READY. However, if a drive is missing, not turned on, busted or otherwise geflummoxed, you will get, (usually) a diagnostic of some sort. If the hard drive controller (HDC) is dead, asleep or missing, an "H" will appear on the screen in the upper right corner. If the HDC is working, but cannot get a sensible response from the hard drive (the bubble), then some error message will show on the screen. ERROR 08H can mean the bubble is not up to speed, for example. I have a 12 Meg bubble which sometimes produces this. Just give it a little time, and then it comes to full speed and I get DOS READY.

Despite my best intentions, (you all know about the paving on the road to Hell, I'm sure), I don't always document things as I should. There being so many possible themes and variations on partitioning of hard drives, I decided recently to explore the CONFIG/SYS file, as I knew all the necessary data was stored in it somewhere, even if I had failed to document the configuration on paper when I set it up.

But, how would I recognize and interpret the data for the hard disks if I found it? From other adventures and snooping, I have learned of the existence and purpose of the Drive Control Tables (DCT's) which the DOS keeps in RAM so that it in turn knows how to operate its various drives.

At this point I will briefly explain the DCT and its construction. To begin with, each of the eight drives permitted under TRSDOS 6.X and LDOS 5.X is allocated a DCT, whether the drive is enabled or not. The DCT consists of ten bytes, the meaning of some of which vary, depending upon whether the drive is a floppy, a hard disk, or a MEMDISK. I will only discuss the DCT as it applies to my hard drives. For interpretation for the other drives, see page 37 of the Programmer's Guide to LDOS/TRSDOS Version 6, by Roy Soltoff.

TRSDOS HARD DRIVE DCT

NAME	Byte	Value	Function
DCT VECTOR	00	C3	Drive enabled flag; C3 in use, C9 not in use
	01	F4	LSB of driver address
	02	04	MSB of driver address
FLAG1	03	0A	Bit mapped as follows:
		Bit	Meaning
		7	1 = Software Write Protected
		6	Not used
		5	0 = 5" drive, 1 = 8" drive
		4	Not used
		3	1 = Hard disk
		2	0 = Removable hard disk 1 = Fixed hard disk
		1&0	Prefix for byte 05
FLAG2	04	00	Bit mapped as follows:
		7	Reserved for future use
		6	Not used
		5	DBLBIT
		4	Not used
		3-1	Starting head # of partition
		0	Hard disk address
CURCYL	05	80	Prefix with bits 0&1 of byte 03 to form starting cylinder # of partition (subject to DBLBIT)
MAXCYL	06	98	Highest numbered logical cylinder on drive (subject to DBLBIT)
CONF1	07	FF	Bit Mapped as follows:
		7-5	Number of heads assigned to partition
		4-0	Highest numbered sector on track
CONF2	08	7F	Bit mapped as follows:
		7-5	Granules per cylinder (Subject to DBLBIT)
		4-0	Sectors per granule
DIRCYL	09	40	Directory logical cylinder (Subject to DBLBIT)

If the drive is not enabled, the first byte is a C9, and the remaining 9 bytes are garbage. If the drive is available, then CALLing the first byte gets you to the driver whose address is in the next two bytes following the C3. If an unenabled DCT is CALLed, the C9 gives a return and no harm is done.

The DBLBIT, if set, means the partition has more than 203 cylinders, and all references to logical cylinders must be doubled.

By the way, some details of the above are at variance with the printed word in the Programmer's Guide. The probable reason is that the guide truly was current at DOS Version 6.2.0, but some changes have been made since then. Also, since MISOSYS supports several different hard disks, it appears that some of the original definitions have been altered to suit actual hardware. Where such deviations occur, what you see is my best interpretation, but I'm not perfect.

ACCESSING CONFIG/SYS

How to get into the CONFIG/SYS file? My first step was to realize the file was present on the BOOT floppy of my machine. Since I have SuperUtility available, I called it up and told it to go get the file. I immediately saw the first sector, and discovered the file was 4 sectors in length. (This can vary, and may be different in your machine.

ANALYSIS

By dumping each successive sector to the printer, and then studying the printout with a sharp pencil, I immediately recognized the familiar pattern of TRS machine code files. An initial 05, then 06, then the name CONFIG, then 01, n, ssss, and n-2 bytes of data. The 01 identifies a block of machine code, n bytes long, to be loaded at ss in RAM. This latter pattern was repeated many times. Finally the familiar closing sequence 02, 02, mm appeared, where the first 02 means "this is the end", and 2 address bytes will follow. The mm is the transfer address, TRA, meaningless in this particular case.

And now the question of how to recognize the DCT in the CONFIG/SYS file. First, I knew the eight DCT's occupy 80D or 50H bytes. By carefully examining the introduction to each block of code, I found there were only three blocks of 50H or more bytes. The first one looked like machine code. (This turned out to be the hard disk driver). The second contained many ASCII names and appeared to be a listing of DEVICES in the machine. The third one was paydirt! 52H bytes, and 52H-2 = 80D bytes, exactly what I was looking for. Next I looked for the pattern of 10 bytes repeated 8 times, which showed up immediately. By listing the 80 bytes in columns on paper, I had the DCT's ready for study. Note the 8 DCT's are stored in the sequence :0, :1, :2, etc.

Having found the ten bytes for each partition of a hard drive, it is a fairly straightforward task to reconstruct the configuration and reset the partitions of the drive.

CAUTION! This essay is true for TRSDOS, but NOT for DOSPLUS, even though it is similar. I don't use DOSPLUS, I don't know how its equivalent of CONFIG/SYS is constructed, and I am sure there are differences between its DCT's and TRSDOS's DCT's. For further info on DOSPLUS, study the appendices at the back of their DOS manual. I think there is sufficient info there to repeat what I have done for TRSDOS, but I won't swear to it, and I'm not going to do it, as I use TRSDOS (and LDOS) exclusively.

There is obviously much more info in the CONFIG/SYS file; I have barely scratched the surface. Whether or not I dig deeper really depends upon my needs. Since my immediate needs were related to my hard drive adventures, that became the focus of my digging. Perhaps in the future, I or some one of you clever people out there may further pursue this public documentation of CONFIG/SYS. Any takers?

COMMENTARY

As an aside, Roy Soltoff, the Sage of Sterling, VA (MISOSYS) once remarked that it would be a simple matter to create a utility which would pull out and neatly present all the information in the CONFIG/SYS file. That's easy for him to say, he having been involved at the conception of LDOS and TRSDOS, but for me it has been somewhat of a chore. Now if Roy could find time to put that utility together......

Seriously, Roy has done a marvelous job, and I never cease to wonder at all he has accomplished in what must be little more than a labor of love, all to our benefit. Thanks again, Roy.

REFERENCE

The Programmer's Guide to LDOS/TRSDOS VERSION 6, Roy Soltoff, BSEE, MISOSYS, Alexandria, Virginia 1983. Reprinted by DiskCount Data, Plano, Texas.

SOME ERRATA

My sins have caught up with me! I have to admit to two errors in my "Speeding Up Your Hard Drive" article in the May/June 1991 TRSTimes. I must have slipped some bits in my memory.

Look at the third paragraph of page 8. I will repeat the paragraph as it should have been:

By the way, MISOSYS' RSHARD5/6 series asks for a speed in the range of 10us to 7.5 ms. The values 0.5 to 4.5 ms are for slower drives. The default "10" value corresponds to 10 microsecond buffered seek. Try 10 first. If you have to settle for 3 ms stepping, the value to be entered is 3.0.

The second error is in the fourth paragraph of page 9. The erroneous sentence presently reads "....300 RPM or 5 revolutions per sector." It should read "....300 RPM or 5 revolutions per second." This will make better sense.

All I can say in my defense is, I'm only human. (My wife sometimes argues that should be subhuman, but ....). In any event mea culpa. My apologies to you all.

-Roy-

RECREATIONAL & EDUCATIONAL COMPUTING

REC is the only publication devoted to the playful interaction of computers and 'mathemagic' - from digital delights to strange attractors, from special number classes to computer graphics and fractals. Edited amd published by computer columnist and math professor Dr. Michael W. Ecker, REC features programs, challenges, puzzles, program teasers, art, editorial, humor, and much more, all laser-printed. REC supports many computer brands as it has done since inception Jan. 1986. Back issues are available.

To subscribe for one year of 8 issues, send $27 US or $36 outside North America to REC, Att: Dr. M. Ecker, 909 Violet Terrace, Clarks Summit, PA 18411, USA or send $10 ($13 non-US) for 3 sample issues, creditable.

ATTENTION MODEL II/12/16 USERS

SOFTWARE BLOWOUT

TRSTimes has recently bought several collections of the difficult-to-get Model II/12/16 software and manuals. We are making this goldmine available to interested readers on an auction basis. Bid on any or all the items listed below and in the next column and highest bid gets the item(s). Auction ends on August 1, 1991.
Please send bid to:

TRSTimes II/12/16 Auction
5721 Topanga Canyon Blvd. #4
Woodland Hills, CA 91367

SCRIPSIT 3.0 Model II/12 for Thinline and hard disk (disk, 2 manuals & 4 training cassettes) RS 26-4835
SCRIPSIT UTILITIES PACK Model II (2 disks & manual) RS 26-4532
SCRIPSIT 2.0 Model II (disk & manual) RS 26-4531
SCRIPSIT UPGRADE 2.1.0 Model II (disk only) RS 700-2036
MULTIPLAN 1.05.00 Model II/12 (disk & manual) RS 26-4580
TRS-XENIX 1.03.04 Model 16 (4 disks & manual) RS 26-6040
TRS-XENIX 1.01.00 Model 16 (3 disks & manual) RS 700-2052
TRS-XENIX SYSTEM REFERENCE MANUAL Model 16 RS 26-6401
TRS-XENIX COURSE Model 16 FC-1014
15-MEG HARD DISK OWNER'S MANUAL RS 26-4157
TRS-80 MODEL 16 SERVICE MANUAL RS 6004/5/6
COMPILER BASIC DEVELOPMENT SYSTEM Model II (disk & manual) RS 26-4705
PROFILE II Model II (disk & manual) RS 26-4512
PROFILE PROSORT Model II (disk & manual) RS 26-4558)
PROFILE FORMS Model II (disk & manual) RS 26-4556
PROFILE PLUS Model II (disk & manual) RS 26-4515
TRSDOS-II Model 12/16B HD (2 disks & manual) RS 26-4912
MODEL 16B OWNER'S MANUAL Model 16B RS 26-6041
BASIC REFERENCE MANUAL Model 16B RS 26-6041
ASSEMBLER-16 Model 16B (manual only) RS 26-6041
TRSDOS 2.0b thinline Model 12 (disk & manual) RS 26-4911
TRSDOS 2.0 Model II (disk & manual) RS 26-4910
TRSDOS 2.0b thinline Model 12 (disk only) RS 26-4911
TRSDOS-II 4.2 Model 12 (disk only) RS-264912
BASIC REFERENCE MANUAL Model 12 (manual only) RS 26-4922
SERIES-I EDITOR ASSEMBLER Model II (manual only) RS 26-4713
VISICALC Model II (disk & manual) RS 26-4511
COBOL RUN-TIME Model 16 (disk only) RS 26-6101
FASTBACK 3.0 Model II from Racet Computes (disk & manual)
MOD II UTILITY PACKAGE from Racet Computes (disk & manual)
SPEED-UP KIT Model II/16 from Racet Computes (disk & manual)
HARD DRIVE SUPERZAP Model II from Racet Computes (disk & manualy)
SNAPP-WARE TRIAL PACKAGE Model II (disk & manual)
SNAPP-III Model II (disk & manual)
PICKLES & TROUT CP/M 2 v2.2m Model II (disk & manuals)
WORDSTAR 3.30 from MicroPro (disk only)
VIDTEX EXECUTIVE 2.1 Model II (disk only)
ISAM 2000 Model II (disk & manual)

Let our LB Data Manager solve your data storage problems

LB Version 2: A Flat File Data Manager with more powerful and easy to use features in this latest enhancement of Little Brother!

We've added many features asked for over the past few years by LB users; yet LB is still just about the easiest, most flexible data manager you can use for managing your data. It's menu driven for ease of use; absolutely no programming is needed to create a database with numerous fields, construct input screens for adding and editing data, and creating your own customized report. Quickly you define your data fields in response to LB's prompts, and then draw your data input screen using simple keystrokes. In no time at all, you're entering data. Customize your printed reports with user-definable print screen definitions. LB is just what you need in a data manager!

Data capacity per database:
LB supports up to 65,534 records per data base; 1,024 characters (64 fields) per record; and up to 254 characters per field.

Field types supported:
LB allows ten field types for flexibility: *alphabetic* {A-Z, a-z}, *calculated* {operations on "numeric" fields using +, -, *, /}, *date last modified* {YYYY/MM/DD automatically maintained}, *dollar* {±dddddddd.dd}, *floating point* {±dddddddd.ddddddd, *literal* {any ASCII character}, *numeric* {0-9, -, .}, *right-justified numeric* {flush right numeric}, *upper case alphabetic* {A-Z, automatic conversion of a-z}, and *upper case literal* { literal with automatic conversion of a-z}. All field types utilize input editing verification so invalid data cannot be added to a record. Field name strings can be up to 19 characters long.

Data entry and editing:
LB allows you to design up to ten different input/update screens to provide extreme flexibility for selectively viewing your database fields. Using a database password provides the capability of selectively protecting fields from being displayed or printed without entry of the correct database password, or they can be protected from being altered. This is quite useful in a work-group environment. Fields may selectively be established to require a data entry before a record being added or edited is saved. You can enable a special index file to keep track of records being added. This can be subsequently used, for example, for a special mailing to newly added *customers*. Flexible editing includes global search and replace with wild-card character match and source string substitution. Search and replace can be performed on all records, or on records referenced in an unsorted or sorted index file.

Record selection and sorting:
You can maintain up to ten different index files to keep your data organized per your multiple specifications; keep one alphabetic, another by address. Records may be selected for reference in an index file by search criteria using six different field comparisons: EQ, NE, GT, GE, LT, and LE. You can select on up to eight different fields with AND and OR connectives. Index files can be left unsorted, or you can sort in ascending or descending order. By attaching a sorted index file, any record may be found within seconds - even in a very large database.

Report generation:
Report generation incorporates a great deal of flexibility. Your report presentation is totally customized through print definition formats which you define on the screen as easily as you define the input/update screens. You can truncate field data or strip trailing spaces. You control exactly where you want each field to appear. LB provides for a report header complete with database statistics: database name, date, time, and page numbers. A report footer provides subtotaling, totaling, and averaging for dollar, floating point, and calculated fields; print number of records printed per page and per report. Attach any of the ten index files and you control exactly what records get printed; even a subset of indexed records can be selected for printing to give you a means of recovering from that printer jam halfway through your 30-page printout. You can even force a new page when the key field of an index file changes value. Up to ten different printout definition formats can be maintained for each database. Reports may be sent easily to a printer, the console display screen, or to a disk file - useful for subsequent printing or downstream data export to other programs. Report formatting allows for multiple across mailing labels, multiple copies of the same record, or even printing one record per page for sales books. You can easily generate mail/merge files of address or other data for your word processor. Or you can use LB's built-in form letter capability.

Automatic operation:
For automating your processing needs, LB can be run in an *automatic* mode, without operator intervention. Frequently used procedures can be saved by LB's built-in macro recorder for future use. Entire job streams may be produced, so that LB operations may be intermixed with literally any DOS function that can be *batch* processed.

Maintenance utilities:
To make it easy for you to grow your database as your data needs grow, we provide two utility programs for managing your database. One allows you to construct a new database with an altered data structure and populate it with data from your existing database Another allows you to duplicate your database structure, copy or move records from one to another, or automatically purge un-needed records.

Help is on the way:
The main menu even provides a shell to DOS so you can temporarily exit LB to perform other DOS commands. LB provides extensive on-line help available from almost every sub-command. A 200-page User Manual documents every facet of LB's operation.

Specify MS-DOS or TRS-80 (M4) version. LB is priced at $99 + $5 S&H.
To upgrade from version 1.0, send Table of Contents page and $40 + S&H.
Remit to:

MISOSYS, Inc.
PO Box 239
Sterling, VA 22170
703-450-4181 or orders to 800-MISOSYS