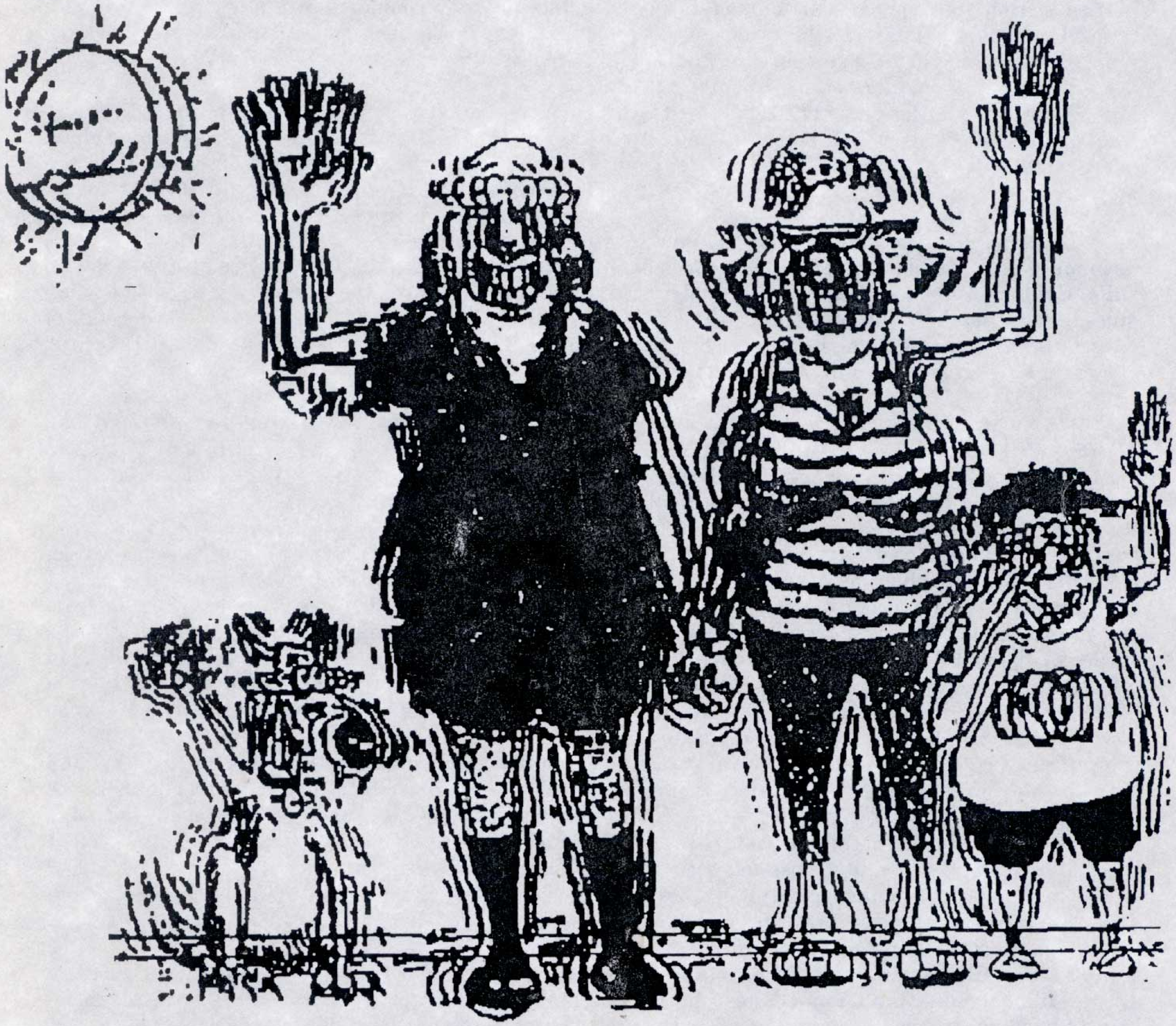
TRSTimes

Volume 7. No. 2 - Mar/Apr 1994 - $4.00



ALL SHOOK UP!!

BUT WE'RE STILL HERE

EARTHQUAKE SEASON

by Sylvia Cary Wolstrup

The joke goes: Who says we don't have four seasons here in Los Angeles? We most certainly do: Earthquake, Riot, Flood, and Fire.

Last issue of TRSTimes, Jim King wrote about the LA fire "season" in his personal account of the devastating Topanga/Malibu fire. In this issue I'm going to write about the earthquake season, also from a personal perspective. Readers might be interested in knowing what the January 17, 1994 Northridge Quake was like here at our house — since "our house" also happens to be the home of TRSTimes magazine.

I'm Lance's wife (Mrs. TRSTimes). We live in a townhouse on Topanga Canyon Blvd. in Woodland Hills, California (part of Los Angeles), only a few miles from the quake's epicenter. The townhouse has 4 levels — garage level, living room level, TRSTimes office and kitchen level, and the bedrooms level.

At 4:31 a.m. on January 17th, a Monday morning, *IT* happened. The rude awakening. The shaking started. I snapped awake. Instinctively, I leapt out of bed. By the time I reached the end of the bed, all was chaos. What I remember was the unbelievable darkness. The electricity went out immediately. Blackness. I heard Lance shouting at me to stay where I was, but it was too late. I'd rounded the end of the bed and I was suddenly stopped in my tracks by obstacles. A flying VCR (I later figured out) slammed into my shin. I fell over a TV set (also learned later), got up, tripped, fell again, got up again. The quake (we were all told days later) lifted us all up anywhere from two to nine feet and then dropped us back down again. Maybe it was the floor coming up and hitting me that brought me to me knees. I don't know. All I know is that within an instant the room was full of stuff. My computer desk fell over, along with it my computer, monitor, printer, stacks of papers. The bookcases toppled over, drawers opened, everything on every surface jumped into the center of the room. A file drawer came sailing out my four-drawer filing cabinet and landed next to where Lance's head would have been had he not sat up so quickly.

Slipping and falling over books and magazines, I made it to the dressing-room area. It was almost impossible to walk. I hung onto the wall. I expected the shaking to die down and go away, the way it had the last time we had a "temblor," but it didn't. Suddenly it got worse. Its increasing intensity felt like a kind of betrayal. This wasn't a nice earthquake. This one was violent, sneaky, vicious and mean. Our townhouse was moving so violently I didn't see how it would hold together. I felt like a little kid being shaken by the shoulders by a huge, angry monster, back and forth, back and forth. Total helplessness. We were dice in a gambler's cup in Vegas.

Aside from the darkness, the other thing I remember is the sound — the roar of it, the smashing glass, the tinkling of things falling into sinks and tubs, the slamming and banging. My mind flashed to my daughters, both in their 20s. One lives (safely) in Paris, the other lives in West Los Angeles. I wondered what was happening to her at this moment, not knowing where the epicenter was, not knowing if it was even worse for her. I had to put her out of my mind. I was powerless to do anything about it.

Our more immediate concerns were Lance's two sons down the hall — Steven, 13 and Alan, 20. When the shaking stopped, Lance and I tried to make our way to their room. We called out, and they answered. They were okay.

"The cat!" I said. "Where's the cat!" I was worried, picturing her buried under books and bookcases. I had no shoes. There was glass everywhere from falling picture frames, perfume bottles, mirrors, make up bottles. I knew I had a small flashlight in my purse, but my purse was under a pile of books, a typewriter, and splintered parts of a telephone table.

In the boys' room everything had fallen over — tall bookcase, computers, TV, stereo equipment. Lance had to pick up the bookcase so Steven and Alan could get out. All this in the dark. The four of us then crawled over yet another bookcase and pile of books in the upstairs hallway in order to get downstairs.

In the kitchen I felt my way to a drawer where candles and matches are kept. I lit a match, something you are not supposed to do in case of gas leaks. Fortunately, no explosion occurred. It was thoroughly disorienting to see everything piled in the middle of every room. In the TRSTimes office, bookcases were tipped over (one of them fell on Lance's laser printer), computers, printers, books and magazines were on the floor.

continued on page 31

TRSTimes magazine

Volume 7. No. 2 - Mar/Apr 1994

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

TRSTimes is published bi-monthly by TRSTimes Publications. 5721 Topanga Canyon Blvd., Suite 4. Woodland Hills, CA 91367. U.S.A. (818) 716-7154.

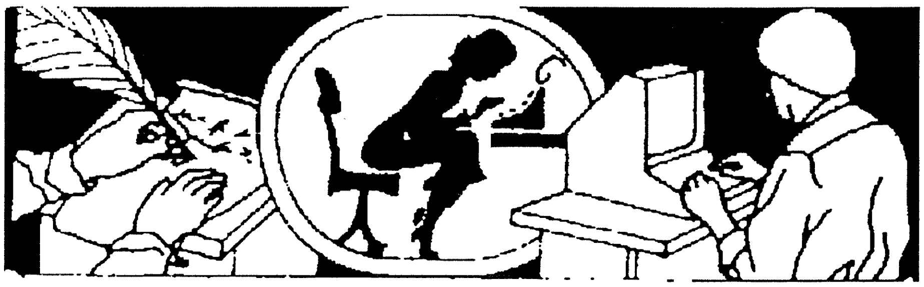
Publication months are January, March, May, July, September and November.

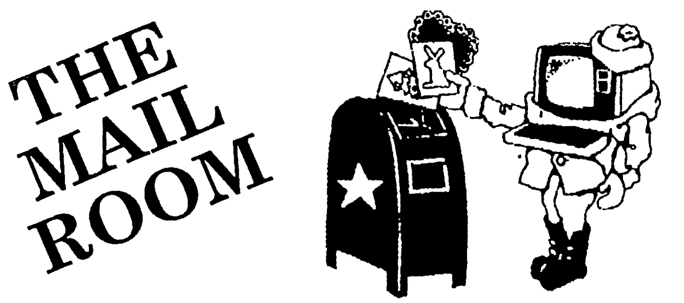
1994 subscription rates (6 issues):
UNITED STATES & CANADA:
$20.00 (U.S. currency)

EUROPE, CENTRAL & SOUTH AMERICA:
$24.00 for surface mail or $31.00 for air mail. (U.S. currency only)

ASIA, AUSTRALIA & NEW ZEALAND:
$26.00 for surface mail or $34.00 for air mail. (U.S. currency only)

Article submissions from our 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 format is used.





TRS-80 THERAPY

Thank you for the good (TRS)times in computer reading. Your articles are relaxing to read, compared to articles in PC magazines. When I read a PC article, I often feel anxious that my computer is inadequate to do the job it has been doing for years. Actually, I have more than one computer: 2 - 128K Model 4P's, 1 desktop 128K Model 4, 2 - 32K Model 100's, and 1 Model 200, as well as a, Compac 286e. My gaggle of computers have served me well - games, word-processing, data bases and spread-sheets. To get over my anxiety, I play more games.

William E. Adams
Bradford, MA

I am with you. My computers also do a variety of tasks - many of which are games. Yes, my son and I often sit side-by-side playing games - Steven on the PC, playing Dark Side of Xeen, while I struggle with a text adventure on the Model III. Good clean family fun.

Ed.

I was greatly relieved to know that you and your family had come to no harm in both the fire and earthquake. Somehow, the fire did not seem to register over here in the same way, but the 'quake' news drew my attention to your nearness to it all.

I was given a Rand McNally Road Atlas of the United States a couple of years ago and it has been most useful in knowing where so many of the TRS-80 related places are and in particular it shows Topanga Canyon Blvd., so I am able to relate places to people which is of some interest to me.

I have not had too much time to read all of the last issue (Vol. 7. No.1), but I must comment on a couple of items. I thought it a great idea to get Jim King to tell his experiences in the fire - a bit of heavy relief and something it is hard to imagine.

Roy's article on Smart Printers struck a happy note. My first printer was an MX80FT and is still working from my Model 4 and also fitted with "Finger-Print" called in Europe "Dots-Perfect". I managed to buy the last one in stock here in England and although its NLQ leaves much to be desired, it is better than the standard issue typeface! I doubt if I shall ever part with it; it is at this moment in need of a mechanical repair to one of the plastic retainers on the tractor feed, but I have no doubt it will get repaired in due course.]

Its particular use is with "Lablmaker" which I bought when I had my "Video Genie" and which converted to Model 3 mode when I bought my Model 4. The program was designed for the MX80 and it still works very well and is useful when I need LARGE type. It also is used with your TRSLABL4 and PFS file, which keeps the addresses and prints out the address label for TRSTimes subscribers here (Advert!).

I am at this moment expecting delivery of an ST157N SCSI hard drive to fit into a 4P I have just acquired, using Roy Soltoff's Host adapter and his Driver software, as well as his instructions which he published last year in TMQ. It has taken all this time to find a suitable drive as they are no longer manufactured. I have sent off an order - I tried to Fax it to him, but was told his Fax line had been disconnected - so I had to post it which is slower.

I hope to have it ready for our next week end meeting in the middle of March, but that will depend on how long it takes for the parts to arrive. I will keep you posted as to my success or not!

Thanks again for keeping this magazine going, and I hope you will soon get yourselves straight after the 'quake'.

Tom Ridge
Surrey, England

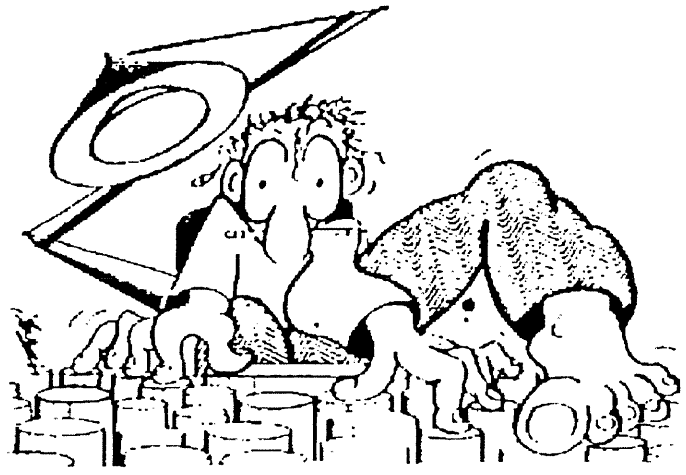
The "quake of '94" will be written about elsewhere in this issue, but let me just tell you that we at TRSTimes are approximately 5 miles from Northridge, and even closer to Reseda, which has now been pin-pointed as the epicenter. It was scary, everything falling down - TVs, video recorders, computers and everything else that literally was not nailed down came flying across the room. I don't believe I have ever felt that helpless before. We were lucky, though - not much damage, just a real mess.

One last word on the earthquake - now that it has officially been upgraded from 6.6 to 6.8 — do we get a new manual!!!

Good luck with the hard drive project. We did this a few months ago (we, meaning Roy Beck at a VTUG meeting), but had no success as the drive turned out to be faulty. He tells me that he will try it again shortly.

Ed.

PROGRAMMING TIDBITS



CAPTURING Z80 ROUTINES INTO BASIC

Some time ago Lance (our amazing publisher) suggested to Chris (Fara) that he (Chris) should supply his little assembly language inventions in form of strings for users to incorporate in BASIC programs. Presumably this would be easier in practical applications than Chris' favorite method of "self-loading memory modules" (for example as supplied on Microdex "Goodies" disk). Now, for some weird reason Chris seems to be in love with the memory module technique. Yet he could also see Lance's point. So he mulled the dilemma over and in a fine American spirit of "live and let live" came up with a sort of compromise: let the memory modules be, but make it easy to translate them into strings.

To begin with, let's recap the idea of "embedding" machine routines in BASIC strings. Machine routines are simply bytes with values 0 through 255 "stringed" together one after another somewhere in memory. If the routine is shorter than 256 bytes (machine routines for BASIC typically are) then the bytes can be represented by ASCII characters of a BASIC string. Strings may move around in memory during the execution of a BASIC program, so the other condition is that such machine routines must not contain any LD, JP or CALL instructions to fixed (absolute) memory addresses within themselves. Then, to execute such a routine, we use the VARPTR command to find the current address of the string and call the routine at this address via USR or CALL commands. The string data can be saved in files, and loaded into any BASIC program. There is no need to protect memory before entry into BASIC, because the routines "float" safely in the BASIC's string space.

So our job today is to "capture" a machine subroutine from some place in memory where it was pre-loaded as a "self-loading memory module", save it in a file, and then load it back into a BASIC string. Obviously the scheme can be used to capture from memory not only "memory modules" but any machine routine, or for that matter any chunk of memory shorter than 256 bytes.

The following "capture and save" procedure needs 3 pieces of information to be supplied in variables.....

ZI%	start address of memory to capture
ZJ%	end address of memory to capture
ZF$	name of file to save the string

The procedure is pretty self-explanatory. Just keep PEEKing and printing to the file the individual memory bytes until all done. One reason why we save the routine as numbers and not as a string is that the routine may contain "null" bytes or other strange codes that would make future input more complicated. Another advantage is that such numerical data may be also easily input into an integer array which overcomes the 255-byte string restriction (more about it later).

```
1000 open "O", 1, ZF$
1010 for X%=ZI% to ZJ%
1020 print #1, peek (X%);
1030 next: close
1040 print ZF$ " saved": RETURN
```

The result is a file with a list of numbers similar to a DATA statement in a BASIC program. A general-purpose program to capture any area of memory could then look like this.....

```
100 input "Enter start address: "; X!
110 input "Enter end address: "; Y!
120 if Y! <= X! then END
130 line input "Enter filespec: "; ZF$
140 if ZF$ < "A" then END
150 ZI% = X! + (X!>32767)*65536
160 ZJ% = X! + (X!>32767)*65536
170 gosub 1000: END
1000 ..... here add the "capture" procedure
```

CAPTURING MEMORY MODULES

The problem with capturing "self-loading memory modules" is that we normally have no idea where a particular module was loaded and how long it is. But we do know three facts about "well-behaved" memory modules (be it user routines or DOS filters, drivers, etc).....

1. All such modules have a standard "header"
2. All sit tightly "packed" in high memory
3. Memory is "protected" up to the last-loaded module

The standard header format might be written in assembly language like this.....

```
HEAD:    JR      EXEC
LAST:    DEFW 0              ; HEAD+2
NICK:    DEFB  HOOK-NAME ; HEAD+4
NAME:    DEFM 'UNIQUE'      ; HEAD+5
HOOK:    DEFW 0
DOSS:    DEFW 0
EXEC:    .....   actual routine begins here
```

The first 2-byte instruction is always JR (Jump Relative) over the header data to the actual routine. The first byte has the value 24 (the machine code for JR) and the second byte is the "length of the jump" which is the count of bytes generated by all the remaining header instructions.

The next instruction (at 2-byte "offset" from the beginning of the header) reserves a dummy 2-byte "word". When the module is loaded into memory, its "loader" puts here the address of the last byte of the actual routine. You can surely see the light now: this will be our variable ZJ% needed by the "capture" procedure.

The one-byte value at 4-byte offset is the length of the module's name which follows at 5-byte offset (names can be any length, but should not be longer than 8 bytes).

The remaining pieces of the header are used in some DOS modules and have no relevance for our capturing scheme.

Now, since modules are packed one after the other, next module (if any) will have its header located at the address ZJ%+1. And since the last-loaded module has its header located one byte above the end of protected memory, we can initially find that end of memory and then repeatedly calculate all starting and ending addresses of all modules.

The program to capture all or selected resident modules could be called MEMOGET/BAS and could look like this.....

```
100 L%=peek(17425): H%=peek(17426)     'Mod-III
100 L%=peek(1038): H%=peek(1039)        'Mod-4
110 ZJ% = L% + 256*H% - 65536

200 ZI% = ZJ% + 1
210 if peek (ZI%) <> 24 then END

220 L% = peek(ZI%+2): H% = peek(ZI%+3)
230 ZJ% = L% + 256*H% - 65536

300 ZF% = peek(ZI%+4)
310 ZF$ = string$(ZF%,"=")
320 L% = asc( mki$ (ZI%+5) )
330 H% = int( (ZI%+5)/256 ) AND &HFF
340 poke varptr(ZF$)+1, L%
350 poke varptr(ZF$)+2, H%

400 ZI% = ZI% + peek(ZI%+1) + 2

500 ZF$ = ZF$ + "/DAT"
510 print "Enter filespec <" ZF$ ">: ";
520 line input K$
530 if K$ > "" then ZF$ = K$
540 if ZF$ >= "A" then gosub 1000
550 goto 200

1000 ..... here add the "capture" procedure
```

Lines 100-110 calculate the end of protected memory (use appropriate version of line 100 for Mod-III or Mod-4).

Line 200 calculates ZI%, the beginning address of the header. Line 210 peeks at that address and if the value stored there is not 24 (the expected JR instruction) then the program ends: evidently there are no more modules (at any rate no standard, well-behaved modules). Otherwise assume it's a valid module and calculate its end address ZJ% from the data in the header.

The 300-group of program lines gets the name of the module from the header. First we peek to find the length of that name and create a dummy string of that length. Then we poke into its VARPTR the address of the module's name. This will be the "default" name for our file. Just to make things more amusing, we can try two different ways of getting the values for the pokes: the low byte is the ASCII code of the address converted to string form with MKI$, and the high byte is obtained by dividing the address by 256 and then "masking" the result with hex'FF to chop it down to single-byte size needed by the poke.

So far the variable ZI% contains the starting address of the header. But since we will be storing the routine in a string, we don't need the header anymore. So in line 400 we increase the starting address ZI% by the length of the header. This way only the actual routine will be copied, saving valuable string memory.

The 500-group of program lines constructs a "default filespec" from the module's name and an extension /DAT (or whatever extension you wish to use), presents that filespec on the screen and asks for input. Hit ENTER to accept the default, or type and enter another filespec. The variables ZI%, ZJ% and ZF$ are then passed to the "capture and save" procedure discussed earlier today.

Sometimes we don't want to capture all modules that happen to sit in memory, only some. To skip a module, instead of a valid filespec enter some character "lower" than "A" in the ASCII order (for example press SPACE bar or type a "period" and ENTER).

In any event the program then loops back to line 200 to look for a next module. At this point ZJ% is the address of the last byte in the previous module, so ZJ%+1 is the address of the next header, if any, and so on.

LOADING AND CALLING STRING ROUTINES

Simply input the saved numbers, convert them to ASCII characters and concatenate into a string. If the string should turn out longer than 255 bytes then a message says so and the program terminates (for handling longer routines use the integer array option discussed later). The filespec ZF$ is supplied by the program calling this procedure.....

```
2000 open "I", 1, ZF$: Z$ = ""
2010 input #1, K%: Z$ = Z$ + chr$(K%)
2020 if eof(1) then close: RETURN
2030 if len(K$)<255 then 2010
2040 print ZF$ " too long!": close: END
```

Upon return from this subroutine variable Z$ contains the machine routine for use by the BASIC program. A generic initialization of a program that needs to load one or more string routines could look like this.....

```
100 ZF$="ROUTINE1/DAT": gosub 2000: Z1$ = Z$
101 ZF$="ROUTINE2/DAT": gosub 2000: Z2$ = Z$
    ..... and so on
```

Right before each and every call to a machine routine in a string we must find its current address, because strings may move around in memory during program execution.....

```
L% = peek (varptr (Z1$) + 1)
H% = peek (varptr (Z1$) + 2)
Z% = L% + 256*H% + (H%>127)*65536
```

Immediately after that in Mod-III or Mod-4.....

```
def usr = Z%
X = usr(0)
```

or more conveniently in Mod-4 only.....

```
call Z%
```

INTEGER ARRAY OPTION

Like strings, integer arrays also float in memory as continuous blocks of bytes, so occasional machine routines longer than 255 bytes can be loaded into such arrays which can be any reasonable size. Since arrays must be dimensioned in advance, we must know the size of the routine before inputting our data from the disk. Computers are very good at "brute force" work, so the simplest solution is to scan the data file twice: first to count the bytes, then to load the routine into the array. Even the longest machine routines for BASIC are still very short and the small one-time delay caused by a double scan near the beginning of a program is hardly noticeable.....

```
3000 open "I", 1, ZF$: Z%=0
3010 input #1, K%: Z%=Z%+1
3020 if eof(1) then close else goto 3010
3030 Z%=fix((Z%-1)/2)
3040 dim Z% (Z%)
3050 open "I", 1, ZF$
3060 for X%=0 to Z%: input #1, L%
3070 if eof(1) then H%=0 else input #1, H%
3080 Z%(X%) = L% + 256*H% + (H%>127)*65536
3090 next: close: RETURN
```

The bytes must be packed into 2-byte integer values, so the number of elements of the array is half the number of bytes in the routine. We figure that in line 3030. If the routine has an odd number of bytes then to prevent a wrong value in the last element, we should stick a "zero" into its high byte (line 3070).

Upon return from this procedure the array Z%() contains the machine code. The address to call is simply the address of the first element of the array.....

```
    def usr = varptr (Z%(0))
    X = usr(0)

    or in Mod-4.....

    Z% = varptr (Z%(0))
    call Z%
```

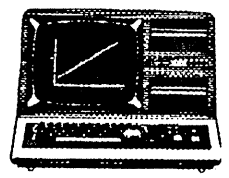
Another approach to the dimensioning of the array could be to modify our "capture" procedure so that the first number in the file would be the size of the routine. In that case the input procedures would have to be also modified to skip that first number.

A more radical alternative would be to store the data in binary form in a random-access file with record length 1, because then the number of items for the array could be directly found with the LOF command, without adding extra information to the file.

But the procedures outlined today are probably the simplest and most flexible. One side benefit of storing the data in plain numerical ASCII files is that they can be easily modified or "patched" with any text editor.

So now if you have some "memory modules" that you'd prefer to use in strings or arrays, load them as usual from DOS, enter BASIC and run the MEMOGET program. Again, remember that the string and array methods work only with routines that don't have any LD, JP or CALL to absolute addresses within themselves. If they do then strange things will happen, so watch out. Self-loading memory modules don't have this problem and are always safe, because the loader automatically adjusts all fixed addresses to the location in memory where the module will reside.

Actually there is a way to "eat the cake and keep it, too". The method to directly call memory modules from a BASIC program without knowing their addresses, is described along with an extensive discussion of the whole business of modules in Volume II of "Z80 Tutor" published by Microdex (see ad elsewhere in this issue). But since today's essay was instigated specifically for the purpose of exploring other options, we shall say no more.



FONTS GALORE

FANTASTIC DOT WRITER FONTS

CREATED BY KELLY BATES

$3.00 PER DISK

CONTACT

MICKEY MEPHAM
9602 JOHN TYLER MEM HWY
CHARLES CITY. VA 23030

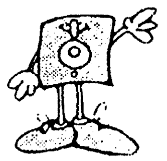
TRSTIMES ON DISK #12

IS NOW AVAILABLE, FEATURING THE PROGRAMS FROM THE JUL/AUG, SEP/OCT, NOV/DEC 1993 ISSUES

U.S. & CANADA:	$5.00 (U.S.)
OTHER COUNTRIES:	$7.00 (U.S.)

TRSTIMES ON DISK

5721 TOPANGA CANYON BLVD., SUITE 4
WOODLAND HILLS, CA 91367



TRSTIMES ON DISK
#1 THROUGH #11
ARE STILL AVAILABLE
AT THE ABOVE PRICES

SOFTWARE GLITCHES
(over the years)

by Henry A. Blumenthal

Here's a lesson in reality: Computer software has been written, and still is being written, by humans. New and old — especially old — software for the Model III and 4 has led users to wonder at times whether it is their brains that have glitches, or whether there is something wrong with their machines or the peripherals they have plugged in.

In some cases, the software developers just didn't look ahead to see how their products might be used. In other cases, the calendar caught up with them. In still other cases, simple errors in the machine language have had to be corrected, if indeed they ever were.

The most well known example of software that has gone astray is Model 4 TRSDOS through version 6.2 and Model I/III LDOS through version 5.1, and some of the older application programs written to operate under them. In the directories they create, they still carry the old dating structure, which won't accept years past 1987. Any year past 1987 will have 8 or 16 subtracted — whatever it takes to keep the date from rising past 1987. So be wary of old formatting utilities, especially what was included as part of the old initialization software for the Radio Shack hard drives. If you use them, you'll have to follow formatting with a DATECONV command before moving files to any floppies or hard drives formatted the old way, or the dates will be wrong and the times will be missing. The old DOSes didn't even post times — just dates. Patches and replacements for DOS have been available for years from MISOSYS and CN80.

Here are some other examples:

When TYPITALL, an easy-to-use and versatile word processor, is loaded, it stops the computer clock until it is exited. This means that the time stamping remains the same in the directory for each document filed during that session. The clock will start again immediately after exiting TYPITALL, but the time will be wrong. However, TYPITALL allows DOS commands to be issued from within, so the clock can be updated before each filing, if the exact time is important.

Another word processor, LeScript, which doesn't usually stop the clock, will do so when the Timclock system has been installed. (I recently acquired Timclock along with some old hard drives; it fits between the computer and the drive.) However, LeScript doesn't stop CN80's own I/O clock, which does not require a DOS patch. On the other hand, TYPITALL works just *fine* when Timclock is resident!

If printer spooling is active while Typitall is loaded, the data will not be passed to the printer until Typitall is exited. What *is* printer spooling? It's a great feature that shunts printer data to a memory partition or temporary disk file that feeds data to the printer as fast as the printer can accept it. This lets the user work while the printer is running.

LeScript has trouble with spooling, too. It has its own keyboard map and printer drivers, so when you want to utilize the spooling feature, LeScript has to be loaded with a parameter that accesses the DOS printer module rather than its own so that the data can be routed to a temporary holding area. If it is so loaded, the DOS "hot" keys will bleed into what is supposed to be LeScript's own keyboard mapping. For instance, clearing text from the position of the cursor to the end usually can be done with clear-colon or control-colon. But if DOS's own printer module has been accessed, control-colon becomes a command to print the screen contents as well.

If spooling is active while VisiCalc is loaded, the printer will run, and there will be no keyboard problems, but if too much memory is reseved for spooling, it will bleed into the loaded VisiCalc file and corrupt what's on the screen. That's because VisiCalc does not always respect other software, such as background utilities, that may also be resident in memory.

If spooling is active when the database programs pfs:File and pfs:Report for the Model 4 are loaded, they will be corrupted.

pfs:File and pfs:Report for the Model 4 exit to

LS-DOS with a cursor shape that once was used by TRSDOS. I don't like the shape, so I programmed my A key — using the KSM (keystroke multiply) feature of DOS — to restore it to my favorite block shape when pressed along with the clear key.

Here's something that's not a software error; it's just a quirk: If you have a Model 4 and have opted for a large cursor (using the DOS command SYSTEM (BLINK,LARGE), then have a program with reverse-video that doesn't use its own cursor, your cursor will become a reverse-video "e".

MISOSYS wrote its own KSM called KSM plus, with additional features. If the DEVICE command of DOS is envoked, it's supposed to show whether KSM is active. But it doesn't recognize KSMPLUS when it's installed, even though KSMPLUS may be working just fine.

My version of software for CN80's I/O clock misstated the time between 8 p.m. and midnight until I was sent a replacement disk.

Has anyone had trouble trying to move LDOS 5.3.1 system files to a double-sided diskette and then sysgen a configuration? I haven't been able to do it; it locks up. And this is a recent release. The most current version of the Model 4 operating system — LS-DOS 6.3.1 — has no such problems.

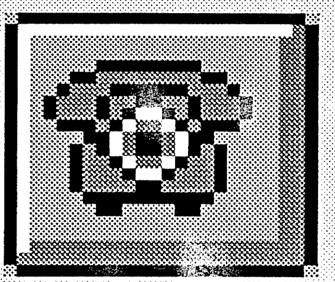
The TRS-80 is a good machine with a lot of life left in it, especially with the right software and peripherals. Sometimes it's hard, in troubleshooting, to decide when a glitch is due to user error, hardware, or software — or, if it turns out to be software, *which* software, since application programs and DOS interact. All in all, the computer itself should be considered the culprit only when all other possibilities have been exhausted, especially if you have been using up-to-date software.

Have you had any experience with a bug you don't think was of your making? I certainly haven't reported on them all! Send your suspicions on to TRSTimes; let's see what we can find out!

If it's any consolation, computers in the MS-DOS world suffer the same problems from time to time. Don't think the problems are exclusive with the TRS-80. I use an IBM AT at work: For text editing, blocks of type can be placed in up to 10 different temporary storage areas for calling out later. Nice touch, eh? But woe unto the user who uses temporary storage area 2: It will corrupt every KSM key combination established by the user. That beats anything I've seen with my trusty Model 4!

TRSuretrove BBS

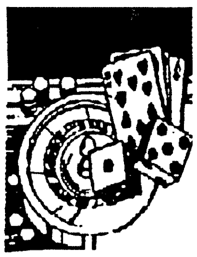
8 N 1 - 24 hours
Los Angeles
213 664-5056



where the TRS-80 crowd meets

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 and published by computer columnist and math professor Dr. Michael W. Ecker, REC features programs, challenges, puzzles, program teasers, art, editorial, humor, and much, 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, Attn: Dr. M. Ecker
909 Violet Terrace
Clarks Summit, PA 18411, USA
or send $10 ($13 non-US) for 3 sample issues, creditable.

BEAT THE GAME

by Daniel Myers

The Scott Adams Adventures

MYSTERY FUNHOUSE

Time for a little spy stuff! A Fun House may seem a strange place for espionage shenanigans, but secret agents have a habit of popping up in the darndest places! if you just had some money, you could get inside. The first thing to do is drop the watch, because you won't be needing it. Now go East to the parking lot. Ignore the five dollar bill (it's a bill for $5!), and look at the tree.

Get the branch, then look in the grate. Aha, that's where the real money is! Chew the gum (tastes HORRIBLE!) then stick it on the branch. Use the branch to get the coin, then drop both the branch and the gum. Return to the Fun House entrance.

Wear the shoes, then give the dollar for a ticket. Go Fun. You are standing in the Magical Mirror Room, and just up ahead is (gasp!) a maze. Fortunately, it's not a bad one, if you're careful (if you aren't careful, well....).

Go North three times, then West twice. This brings you to a small room. Go West from here to the room with the knobs. Pull the green knob, and you will find yourself in another room. Get the trampoline that's here, then go South to the shooting gallery and pick up the strange spectacles. You have a few minutes, so, if you want to, you can amuse yourself by shooting a few clay pigeons. When you're finished, go back North, then Up, to the knob room.

Now, as you proceed through the adventure, you will from time to time get messages about your shoe heel being loose. Ignore them, and don't touch the heel for now. There will be time for that later. At the moment, you have other things to do. So, head along West to the tank room, and from there, Up to the ledge over the pit. Drop the trampoline here, then go East to the barrel room.

Once in the barrel room, get the match and the comb, then crawl (that's how you get out of the barrel). Drop the match by the trampoline, then head South to the rickety stairs and down to the landing. Go down the slide into the tank. Get the rusty key, then give the comb to the mermaid. Go up the secret stairs she reveals, and you will be back on the landing.

Go East into the Windy Hall, and East again into (sigh) the maze. Now, carefully make your way South, East, South, East, and you will be in the Mirror Room again. Wear the spectacles, and look in the mirror. Sonuvagun! There's a hidden door! Open the door and go inside. Here you find a valve handle. Drop the spectacles and get the handle, then go East back to the mirror room.

Once again, go through the maze to the room with the knobs. Drop the key, then continue on West and then up until you come to the ledge. Get the trampoline, then go down the ladder to the pit. Drop the trampoline, then put handle and turn handle. You have now turned off the calliope in the merry-go-round room (which is where you're going next). Now go trampoline, and jump. Wheeee! You're back up on the ledge.

Get the match, and return to the knob room. Drop the match, get the key, and pull the blue knob. Now you're in the room with the Fortune-telling machine. However, it won't be telling your fortune today, because it's broken. However, in an odd sort of way, it's going to be very helpful to you. But right now, go on along East to the merry-go-round room.

Once in the merry-go-round room, push the blue button to stop the ride. Go merry, then go horse. Climb the pole in the horse's back, which brings you to the top of the ride. Look up, and you will see a rope. Jump! Now look, and you will see you are on a catwalk. Go East, and unlock the door. Drop the key, and look at the shelves. Grab the flashlight and the wrench, then head back West, and climb all the way back down again, then return to the knob room. As you pass the fortune-telling machine, pick up the "out of order" sign.

Pull the green knob, then go South to the shooting gallery. Drop the sign there, then return to the knob room. Get the match, then pull the yellow knob. This takes you to a small room with strange music. Go North into the maze (again!), then go East, South, East, South, East, and you're in the Mirror Room. From there, go South out of the Fun House.

Now it's back to the parking lot. Open the grate. You will only be able to open one bolt, but you can slide the grate to make room for yourself to go down. Drop the wrench and get the gum. Turn on the flashlight, then go down the manhole, and East to the room with the second grate.

Close the door, and drop the ticket. Now, the big moment has arrived! Remove the heel from your shoe. A couple of things will fall out. One is a note expaining what this is all about, the other is a fuse. Drop the heel and the note, and get the fuse.

Chew the (yuck!) gum again, then stick it to the fuse, then stick it to the grate. Light the fuse, and POP! the grate blows open. Get the ticket, then go through the hole. Now up through the shooting gallery (good thing you put that sign there!) and South to the hidden lab. There they are...the secret plans! Grab them and...congrats! You have successfully completed your assignment! After all this, you could use some peace and quiet. How about a trip to some faraway place, like a nice sandy island beach...

PIRATE ADVENTURE

Yo-ho-ho and a bottle of Coke (or whatever!). It's time for Pirate Adventure! So, don't just stand there, grab the crackers, sneakers and rum, then "Go Stairs." Hmmmm, wonder if there's anything interesting to read in that bookcase? Let's find out. "Get Book." Aha! A secret passage is revealed! "Go Passage," then East. Get the torch and duffel bag, then examine the bag. Some matches will fall out. Drop the bag (you won't need it), and get the matches. Now head back West twice. Read the book, which tells you that the magic word is "Yoho." Next, "Go Window," and say the word. Amazing! You're now on a sandy island beach.

Drop the book and sneakers, then go East. There's a shack here, so let's Maybe the pirate's thirsty, so give him the rum. Ah, he takes it and runs off! Now it's your turn to run off, since you don't need the parrot right now, and you can't open the chest yet either. So, go West, then East, which brings you to the cave-ridden hill.

"Go Path," and you're at the top of the hill. There isn't much here except a crack, but it looks like you just might make it through, so "Go Crack." This brings you into a cave, which is a bit dark, so you had better light the torch. That's better! Now you can see that there is a shed here, as well as some lumber and sails. Go into the shed, pick up the hammer and the water wings, then head North, and go back through the crack. "Unlight Torch" (because it won't last forever), then go back down the hill and continue West until you return to the beach.

Okay, drop the wings, torch, matches, and sack, and get the book and sneakers. Say the magic word, and you will find yourself back on the window ledge again. Go inside, and make your way to the secret passage. Head East, and find the pirate sleeping off the rum. Don't disturb him; just pick up the empty rum bottle and tiptoe out again. Now go downstairs to where the rug is. The rug is nailed down, so "Get Nails," then "Get Rug."
Underneath is a ring of keys. Drop the rug, get the keys, and head on back upstairs to the window ledge. Say the magic word once again.

On the beach, drop the book, hammer, sneakers, and nails, then get the water wings and "Go Lagoon." The tide should be coming in now. If it isn't, you'll have to wait for it. Go North, and you will be in the ocean. Get the fish, and also some water (that's what you need the bottle for; how else could you keep them alive?). Then it's back South twice to the beach.

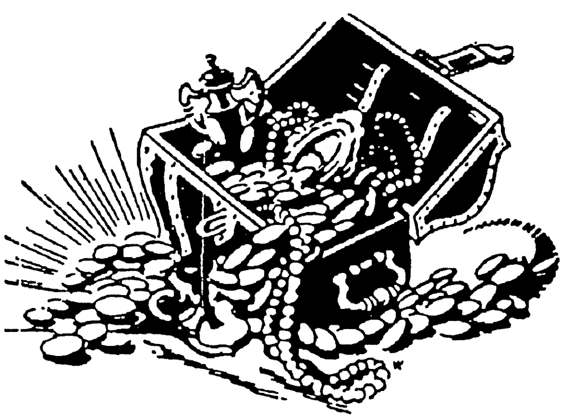
Drop the wings, get the torch and matches, and move along East twice to the bottom of the hill. Light the torch and go down. Hmmmm. Hungry-looking crocodiles! Good thing you have the fish with you! Feed the crocs, drop the bottle, and unlock the door. "Go Hall" and East. Surprise! There are lumber and sails here (you didn't really think you could drag this stuff out through the crack, did you?). But first, go into the shed and get the shovel. Now pick up the lumber and sails, and go West into the hall. From there, go to the pit, go up, then West, and you're out of the cave. Time to unlight your torch and make yet another trip back West to the beach.

Once there, drop the lumber, sails, torch, and matches, and return to the shack. Now you can open the chest with your keys. Look inside two times, because there are two items inside: a map and plans for building a pirate ship. Get both of those and the parrot and, once again (sigh!), go back to the beach. Wait for the tides to change, then go into the lagoon again. This time, the tide should be out, and you can dig up the anchor. Get that, and go back South to the beach. Drop the anchor. The magic moment is almost here. "Build Ship," and there, by golly, is a pirate ship! However, before you go sailing off on the bounding main, you do need someone to run the ship. (By the way, you can drop the plans now.)

Grab the sneakers and book, and (in case you hadn't guessed by now), say "Yoho." Now go wake up the pirate, and return to the beach. Drop the book and sneakers again, then "Go Ship," and "Set Sail." Finally! Treasure Island! Go to the beach and dig. The pirate will grab some of the rum and take off. Now go South through the graveyard (being careful not to awaken the pirate), then East into a field. "Pace 30," then dig, and you will uncover a wooden box. Get that, then drop the shovel and "Go Monastery." Oh boy! Deadly black mamba snakes! Good thing you still have the parrot with you. Drop the parrot. He will chase off the snakes, and you can pick up the "dubleons" (well, that's how they spell them in this game!).

Okay, head West twice, wake the pirate, then go North to the beach. "Go Ship" and "Set Sail." You're back on the pirate's island now. "Go Beach" and get the hammer. Now you can open the box and get the stamps (stamps? that's a pirate treasure??). Drop the hammer and box, and get the book and sneakers. Say "Yoho," then go inside and down the stairs. Drop the two treasures and say "Score." WHEW! You won the game!

It's time to sit back and relax with a mug of grog...or maybe even two!!



YES, OF COURSE !

WE VERY MUCH DO TRS-80 !

MICRODEX CORPORATION

SOFTWARE

CLAN-4 Mod-4 Genealogy archive & charting ***$69.95***
Quick and easy editing of family data. Print elegant graphic ancestor and descendant charts on dot-matrix and laser printers. ***True Mod-4 mode***, fast 100% machine language. Includes 36-page manual. ***NEW!***

XCLAN3 converts Mod-3 Clan files for Clan-4 ***$29.95***

DIRECT from CHRIS Mod-4 menu system ***$29.95***
Replaces DOS-Ready prompt. Design your own menus with an easy full-screen editor. Assign any command to any single keystroke. Up to 36 menus can instantly call each other. Auto-boot, screen blanking, more.

xT.CAD Mod-4 Computer Drafting ***$95.00***
The famous general purpose precision scaled drafting program! Surprisingly simple, yet it features CAD functions expected from expensive packages. Supports Radio Shack or MicroLabs hi-res board. Output to pen plotters. ***Includes a new driver for laser printers!***

xT.CAD BILL of Materials for xT.CAD ***$45.00***
Prints alphabetized listing of parts from xT.CAD drawings. Optional quantity, cost and total calculations.

CASH Bookkeeping system for Mod-4 ***$45.00***
Easy to use, ideal for small business, professional or personal use. Journal entries are automatically distributed to user's accounts in a self-balancing ledger.

FREE User Support Included With All Programs !

MICRODEX BOOKSHELF

MOD-4 by CHRIS for TRS/LS-DOS 6.3 ***$24.95***
MOD-III by CHRIS for LDOS 5.3 ***$24.95***
MOD-III by CHRIS for TRSDOS 1.3 ***$24.95***
Beautifully designed owner's manuals completely replace obsolete Tandy and LDOS documentation. Better organized, with more examples, written in plain English, these books are a ***must for every TRS-80 user.***

JCL by CHRIS Job Control Language ***$7.95***
Surprise, surprise! We've got rid of the jargon and JCL turns out to be simple, easy, useful and fun. Complete tutorial with examples and command reference section.

Z80 Tutor I Fresh look at assembly language ***$9.95***
Z80 Tutor II Programming tools, methods ***$9.95***
Z80 Tutor III File handling, BCD math, etc. ***$9.95***
Z80 Tutor X All Z80 instructions, flags ***$12.95***
Common-sense assembly tutorial & reference for novice and expert alike. Over 80 routines. No kidding!

Add S & H. Call or write MICRODEX for details
1212 N. Sawtelle Tucson AZ 85716 602/326-3502

DR. PATCH

UTILITY FOR TRS-80 MODEL 4 AND LS-DOS 6.3.1

A *'MUST HAVE'* FOR ALL LS-DOS 6.3.1 OWNERS.

DR. PATCH MODIFIES LS-DOS 6.3.1 TO DO THINGS THAT WERE NEVER BEFORE POSSIBLE.

COMPLETELY SELF-CONTAINED - MENU-DRIVEN FOR MAXIMUM USER CONVENIENCE.

FAST & SAFE - EACH MODIFICATION IS EASILY REVERSED TO NORMAL DOS OPERATION.

DISABLE PASSWORD CHECK IN FORMAT/CMD
FORMAT DOUBLE-SIDED AS DEFAULT
FORMAT 80 TRACKS AS DEFAULT
DISABLE VERIFY AFTER FORMAT
CHANGE 'DIR' TO 'D'
CHANGE 'CAT' TO 'C'
DIR/CAT WITH (I) PARAMETER AS DEFAULT
DIR/CAT WITH (S,I) PARAMETERS AS DEFAULT
CHANGE 'REMOVE' TO 'DEL'
CHANGE 'RENAME' TO 'REN'
CHANGE 'MEMORY' TO 'MEM'
CHANGE 'DEVICE' TO 'DEV'
DISABLE THE BOOT 'DATE' PROMPT
DISABLE THE BOOT 'TIME' PROMPT
DISABLE FILE PASSWORD PROTECTION
ENABLE EXTENDED ERROR MESSAGES

DISABLE PASSWORD CHECK IN BACKUP/CMD
BACKUP WITH (I) PARAMETER AS DEFAULT
BACKUP WITH VERIFY DISABLED
DISABLE BACKUP 'LIMIT' PROTECTION
DISABLE PASSWORD CHECK IN PURGE
PURGE WITH (I) PARAMETER AS DEFAULT
PURGE WITH (S,I) PARAMETERS AS DEFAULT
PURGE WITH (Q=N) PARAMETER AS DEFAULT
IMPLEMENT THE DOS 'KILL' COMMAND
CHANGE DOS PROMPT TO CUSTOM PROMPT
TURN 'AUTO BREAK DISABLE' OFF
TURN 'SYSGEN' MESSAGE OFF
BOOT WITH NON-BLINKING CURSOR
BOOT WITH CUSTOM CURSOR
BOOT WITH CLOCK ON
BOOT WITH FAST KEY-REPEAT

DR. PATCH IS THE ONLY PROGRAM OF ITS TYPE EVER WRITTEN FOR THE TRS-80 MODEL 4 AND LS-DOS 6.3.1.

DISTRIBUTED EXCLUSIVELY BY TRSTIMES MAGAZINE ON A STANDARD LS-DOS 6.3.1 DATA DISKETTE, ALONG WITH WRITTEN DOCUMENTATION.

DR. PATCH $14.95

NO SHIPPING & HANDLING TO U.S & CANADA. ELSEWHERE PLEASE ADD $4.00 (U.S CURRENCY ONLY, PLEASE)

TRSTimes magazine - dept. DP
5721 Topanga Canyon Blvd. #4
Woodland Hills, CA 91367

DON'T LET YOUR LS-DOS 6.3.1 BE WITHOUT IT!

Graphics Image Compression

by J.F.R. "Frank" Slinkman

Required:
TRS-80 Model 4/4P/4D
High Resolution Graphics Board
Pro-MC Compiler from Misosys, Inc.
Slinkman Hi-Res Graphics Library

While we all want to make the best possible use of disk space, the size of graphics image files isn't the most major of concerns for TRS-80 users. But it is for others. For example, the "raw" size of one SVGA full-color image is 1024 x 768 x 3 bytes, or a whopping 2,304K! Even a simple 8" x 10" fax image, in raw form, is 1,728 pixels wide by (at least) 1,013 pixels high (double that if the image is in "fine" resolution -- quadruple that if "superfine"). At eight horizontal pixels per byte, that's almost 214K of image data (over 427K for "fine" resolution, nearly a meg if "superfine").

Many compression techniques have been utilized or invented to get the size of these files down to manageable proportions. But all data compression must rely on one or both of two characteristics of the data: Redundancy and repetition. The Lempel-Ziv-Welch (LZW) method in wide use, notably in archive programs and GIF, relies on repetition. It records sequences of characters (or pixels) it has seen, and references these sequences by a code. If it sees a sequence repeated, it replaces the sequence with the much shorter code.

The Run Length Encoded (RLE) technique, which is used in the /CHR format known to TRS-80 users, is also based on repetition.

The /CHR method is very similar to the MacPaint method, which is well covered in the article, "A Tale of Two File Formats," TRSTimes 2.3 (May/June 1989). This technique works well with simple images, but results in reverse compression (i.e., a "compressed" file larger than the original) when used with complex images.

The "bit packing" technique is used in fax image compression. Fax data is very similar to TRS-80 graphics data in that it is stored 8 horizontal pixels per byte. However, unlike our format, 1 = black and 0 = white. Bit-packing achieves compression by only recording bytes which contain black (set) pixels. Each 216-byte line of 1,728 pixels is "mapped" by 27 preceding bytes, one bit per byte. For example, if the first byte on the 216-byte line is non-zero, then the first of the 216 bits in the 27-byte map is set to one. Otherwise, this bit will be reset to zero. Then the 27 bytes are mapped to the first 27 bits in 4 preceding bytes, and those 4 bytes are mapped by a nibble in a single preceding byte. Of all the major "non-lossy" compression techniques, bit packing is the only one which takes advantage of vertical repetition in the image. It does this by XORing lines with each other, which means that if there is no change from one line to the next (which happens often with images of text, which is what most faxes are), the lower line will be full of zeroes. Under this scheme, this compresses extremely well -- one line of 1,728 zero pixels representing either the white space between lines or no change from the line above compresses to a single byte!

The main players in the "lossy" area are BTC and JPEG. These methods rely on the inability of the eye to detect subtle changes in color and/or brightness from one pixel to the next; and "lose" those differences by smoothing the image before compression, which increases both redundancy and repetition.

With JPEG, the amount of smoothing is variable, but the more smoothing that is done to improve compressability, the greater the image degradation. I refer those interested in JPEG to the PICS forum libraries on CompuServe. A BROwse LIB:ALL KEY:JPEG,SOURCE command will bring up the relevant files.

BTC is designed for use with 256-level greyscale images, and standard BTC always provides exactly 4:1 compression. I have suggested some improvements to the original spec, which usually result in about 10% better compression (i.e., a compressed file about 22.5% of the size of the original, as opposed to 25%). I refer those interested in BTC to the DDJFORUM on CompuServe. A BRO LIB:ALL KEY:BTC command will bring up the relevant files.

Interestingly, the "newest" and "best" compression methods, namely TIFF 6.0 and JPEG, rely on the old standby, the Huffman compression algorithm, to accomplish the real work when it comes to compression. There are reasons for this, even though LZW and arithmetic compression often outperform Huffman. Unisys holds a patent on the LZW algorithm, which has scared off many potential users (including Aldus, apparently, and therefore TIFF 6.0). There are some questions whether patent law applies here (i.e., whether Welch's comparatively minor contribution makes LZW sufficiently different from the original, public domain, Lempel-Ziv algorithm to warrant a patent, and whether or not patent law even applies to pure software implementations), but nobody seems to want to spend the money to test these issues in court. Likewise, there are some interesting mathematical compression techniques out there, which are also largely unused because of similar patent concerns.

Interestingly, our TRS-80 "/HR" images could be considered mathematically compressed. For example, we could look at an /HR file not as 19,200 bytes of data, but as a 153,600-bit binary number, with an implied radix point to the left of the first bit. This way, an image with only the one upper left pixel set could be compressed to a single bit simply by calling it the number 0.5 decimal, or 0.1 binary. Likewise, an image with every other horizontal pixel set would be either the binary fraction 0.10101010... or 0.01010101... (each out to 153,600 places), which can be expressed in much shorter form as the fractions 2/3rds and 1/3rd, respectively. Realistically, however, arithmetic compression would be of little use for our simple, monochrome images, but it has great use with multi-megabyte "full color" images, where it takes 24 bits to describe a single pixel. I refer those interested in arithmetic compression to the article "Arithmetic Coding and Statistical Modeling," Mark R. Nelson, Dr. Dobb's Journal, Feb. 1991.)

The patent concerns I've mentioned above leave the public domain Huffman algorithm as the best legally safe method available for graphics compression. But, other than the use of ARC4 to "squeeze" /HR files, Huffman compression has never, to my knowledge, been used to compress TRS-80 /HR graphics files.

Not until now, anyway. Enter "dohuf.c" and "unhuf.c," programs which use Huffman compression to compress 640 x 240 /HR files to "/HUF" output files, and decode the /HUF files to images on the monitor screen.These programs are presented for those who wish to store their graphics images in minimum space, and as a way to demonstrate many C programming language features and concepts to TRSTimes readers.

Dohuf.c is not fast. It takes almost as long to compress an image as it would to archive it using ARC4. However, unhuf.c will display files created by dohuf.c very quickly. Also, because of overhead, dohuf.c does not always compress very simple images as compactly as Mel Patrick's /CHR format. With moderately complex images, /HUF files will be smaller than /CHR files, but larger than GIF files created by HR2GIF/CMD in my GIF4MOD4 package. However, with extremely complex images, especially converted GIF images with colored backgrounds, /HUF files will be somewhat smaller than HR2GIF-created files and substantially smaller than /CHR files.

Before we get to the actual programs, we need to understand how the Huffman algorithm achieves compression. This is best explained by example. Suppose we wish to compress a file which contains only the ASCII codes for the phrase, "HOW NOW BROWN COW." First, the file is passed to get a count of how many times each ASCII code is used (see Fig. 1). Once this data is obtained, the file is rewound, and a Huffman data structure "tree" is constructed.

The tree consists of an array of "elements," each of which is a "data structure" containing more than one variable (the structure's "members"). Pointers in each element can point to other elements, enabling each to be linked to up to three other elements. Each element contains members to hold the ASCII character, the count of how many times the character appears, a pointer to a "parent" structure of the same type, and pointers to both left and right "children," also structures of the same type. Each "leaf" of the tree will have a parent, but no children. Each "branch" will have both a parent and children, and the root of the tree will have children, but no parent. The tree can be transversed from a leaf to the root by looking at the parent in each step, until an element is reached which has no parent. It can be transversed from root to leaf by using recorded data to tell you whether to go right or left at each branch until you find an element which has no children.

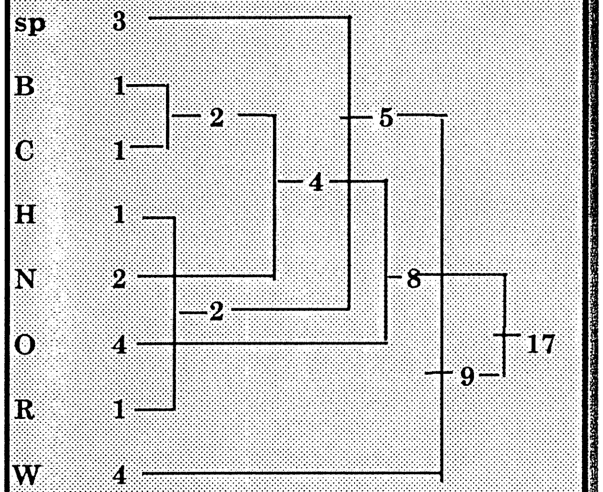
Figure 1
Character Count
HOW NOW BROWN COW

space	=	**3**
B	=	**1**
C	=	**1**
H	=	**1**
N	=	**2**
O	=	**4**
R	=	**1**
W	=	**4**

total = 17

Now look at Fig. 2 while the tree building process is described. The actual tree configuration will be different than the one depicted, but drawing it accurately would be too cluttered and confusing. The illustration, however, accurately illustrates the CONCEPT.

Figure 2
Building the Huffman Tree



At the outset, each of the eight leaves of the tree have only their characters and counts. The array of structures is scanned to find the two "orphans" (i.e., elements without a parent) with the lowest counts. In this example, the first pair selected will be the "B-C" combination, with a count of one each. This information is used to construct a new branch element with a count of 2 (the sum of the counts of the two children). Both the B and C elements will have a pointer to this new branch put in their "parent" members (so they're no longer orphans), and the new branch element will have a pointer to the B element in it's "right child" member, and a pointer to the C element in it's "left child" member.

Now the array is scanned again to find the next two orphans with the lowest counts, which are the "H-R" combination. Again, the H and R elements are given a parent with a count of 2, and the new element will have the H and R leaves as the right and left children.

The next scan selects the "B-C" branch and the "N" leaf, each with a count of 2, creating a new element with a count of 4.

The next scan selects the "space" leaf and "H-R" branch, with counts of 3 and 2, respectively, creating a new element with a count of 5.

Next, the "B-C-N" branch and "O" leaf are selected, each with a count of 4, creating a new element with a count of 8.

Next the "space-H-R" branch and "W" leaf are selected, with counts of 5 and 4, respectively, creating a new element with a count of 9.

Finally, the two remaining orphans, with counts of 8 and 9, are selected and used to create the last new element (the root) with a count of 17.

In this example, let's say a zero is the code to go right, and a one is the code to go left. The code, then, for "space" is 100 (binary). In other words, starting at the root, the "1" tells us to pick up the pointer to the left child. The first "0" tells us to look at that element's right child, and the second "0" tells us to look at the right child's right child. Because this child has no children, we know we're at a leaf; so we pick up the character from the character field of this element, and output it. Refer to Fig. 3 for the Huffman codes to navigate the tree. Note each code is unique. Note also that the higher the count for a character, the fewer bits are in it's code. This is how compression is obtained. In this example, the 17-character phrase "HOW NOW BROWN COW" would be encoded in 47 bits (6 bytes).

After the tree is constructed, the input file is read a second time, and the Huffman compression codes determined by stepping from the input value's leaf to the root, and then retracing those steps back to the leaf, writing bits to the output code stream as we go.

In addition to using the Huffman algorithm, one more trick is used in the dohuf.c and unhuf.c programs to attempt to increase the compressability of /HR images. Instead of looking at the data as each byte containing 8 horizontal pixels, the screen is instead divided into 4 pixel by 2 pixel "squares." Remember, TRS-80 pixels are twice as high as they are wide; so a 4 x 2 pixel group represents a physical square on the screen. So, instead of looking at a screen of 240 lines of 80 bytes each, dohuf.c looks at it as 120 lines of 160 squares. This is an attempt to take advantage of vertical repetition in the image. It usually does, but not always, since the success of this tactic depends on the nature of the image.

Figure 3
Huffman Codes

sp	100
B	0000
C	0001
H	1010
N	001
O	01
R	1011
W	11

It is beyond the scope of this article to be a full tutorial on C, but you need to look at the listings to fully understand the nature of the Huffman tree and its elements. Close to the beginning, you'll see a section of code which begins with the line:

```
TREE        { char   code;
```

This is where the structure of the Huffman tree elements are defined by listing the types and names of its members. "char code;" means a one-byte variable named "code" is included.

"int count;" means a 16-bit integer variable named "count" is included.

"TREE *parent," "TREE *rite," and "TREE *left" mean that three pointers-to-structure of type tree are also included (the "*" designates a variable as a pointer).

Since each pointer is a 16-bit address, the size of the structure is nine bytes. At the end of the structure definition, after the closing brace, the "huftree[511];" tells the compiler to reserve space for an array of 511 of these 9-byte structures. Also, the variable name "huftree" will henceforth refer to the address of the first byte of this array. In other words, "huftree" becomes a pointer, rather than a normal variable.

Now skip down the listing until you get to the start of the buildtree() function (starting with the line "void buildtree()").

The first line of code declares for variable (ptr1, ptr2, temp and limit) as pointers-to-structure of type tree. The next line, "limit = huftree + 256;" means "point 'limit' to the 256th element of the huftree array." This is a feature of C which frees the programmer from having to calculate physical addresses. It is the equivalent of the assembly code:

```
LD          HL,HUFTREE
LD          DE,256*9
ADD         HL,DE
LD          (LIMIT),HL
```

or the BASIC code:

```
LIMIT = VARPTR( HUFTREE(256) )
```

The next line:

"while (temp->parent | | !temp->count) ++temp;" means, in English:" if the 'parent' member of the structure pointed to by 'temp' is other than zero or the 'count' member of the structure pointed to by 'temp' is equal to zero, then increment 'temp' (i.e., point 'temp' to the next array element) and start over." To put it simply, it says, "step through the array until you find the first orphan with a count greater than zero." This line is a good example of the power of the C language.

In BASIC, just to duplicate the structures, you'd have to set up five separate arrays, or build a difficult-to-access "option base 1" array which could only be accessed via MID$, ASC, CHR$, MKI$ and CVI instructions. For example, to replace this short line of C code, you'd have to write something like:

```
50 IF (CVI(MID$(HUFTREE(TEMP),3,2)) <> 0)
 OR (CVI(MID$(HUFTREE(TEMP),1,2)) = 0)
 THEN TEMP = TEMP + 1 : GOTO 50
```

This power is one of the reasons I'll never go back to BASIC. Another reason is the much faster running time of compiled C programs. Anyway, once such an array element is found, it's address is copied to "ptr1," and the process repeated to find the next such element. If a second qualifying element is not found, then the tree is complete, and the endless loop created by the "while (TRUE)" statement is exited via the "break" statement. If a second element is found, it's address is copied to both "temp" and "ptr2."

The next few lines swap "ptr1" and "ptr2" if the higher of the two counts is in the structure pointed to by "ptr2," because the following array-scanning code expects that ptr1->count will always be >= ptr2->count.

Now, if you'll skip up the listing to the "get_counts()" function, you'll see, at the end, another way to address members of elements of an array of structures. The "->" ("points to") designator only works with pointers-to-structure.

When we need to address a member of a structure in an array, we use the array_name [element_number].member_name method. In this "for" loop, dohuf.c assigns the codes and counts to the first 256 elements of the array by addressing the element's members as "huftree[i].code" and huftree[i].count" respectively.

As I mentioned earlier, this article cannot be a C tutorial. The listings are provided for those who are, or who which to become, proficient in C. You'll note that in the "Required" box, you need both the Pro-MC compiler and my High Resolution graphics library for Pro-MC. Unfortunately, Misosys, Inc., is just about gone. Except for the LDOS and LS-DOS operating systems, they no longer sell or support their previous product line. Thus, if you want to "get into" C on your Model III or 4, you'll have to find a used copy of Pro-MC. I strongly recommend this package over the limited version of C promoted by another publication if you want to learn real C, and not a non-standard and very limited version of the language.

If you already have Pro-MC, but don't have the Slinkman Hi-Res graphics library, you can order it direct from me. It was originally available on TMQ DiskNotes 6.3, but this has also been discontinued by Misosys, Inc. Even if you have DiskNotes 6.3, you may still want the new version, which includes several improvements, including a MUCH faster paint() function, and new getimage() and putimage() functions, which load and save /HR files.

Also, if anyone reading this has a version of my GIF4MOD4 package earlier than Version 2.0, you may wish to upgrade to take advantage of a couple of bug fixes (including one which lets it work properly under LS-DOS 6.3.1), and a much faster HR2GIF/CMD program. The Hi-Res library and GIF4MOD4 update are $12.50 each, which includes S&H (please add $2.00 for addresses outside North America). If you wish to order both the library and upgrade, the total is $22.50 ($24.50 outside N.A.). If you are an original purchaser of GIF4MOD4, there is no need to return the original disk. However, if you bought your copy second-hand, you MUST return the ORIGINAL disk so I can use the program serial number to update my records.

My address is 1511 Old Compton Road, Richmond, VA 23233-4055. If you want faster delivery via CompuServe E-Mail, please specify when ordering. Also, if you would like to see a series of C language tutorial articles here in TRSTimes, please let Lance Wolstrup know. I'd be happy to write them if there's enough interest out there.

DOHUF.C

```
        /* dohuf.c
 *
 * Author:      J.F.R. "Frank" Slinkman
 * Date:        20-Nov-93
 * Compiler:    Pro-MC
 * Libraries:   Slinkman Pro-MC Hi-Res graphics library
 * Source:      "C Programming Column," Dr. Dobb's Journal, Feb. 1991.
 *                Note: listings accompanying article contain bugs!
 *
 * This program reorganizes the 640 x 240 TRS-80 hi-res screen into 19,200
 * 4 pixel by 2 pixel "squares." Then, using the Huffman algorithm, it
 * compresses that data, and writes the compressed data to the output.
 */


#include   <stdio.h>
#option INLIB
#option REDIRECT   OFF
#option FIXBUFS ON
#option MAXFILES   1
#define XREG   128
#define YREG   129
#define GFXDAT  130
#define TREE struct tree


void   loadHRfile(), get_counts(), buildtree(), dataout(), compress();
void   outbit();


FILE   *fp;
char   f_name1[15], f_name2[15], cant_open[15] = { "Can't open %s\n" };
TREE   {  char code;        /* code value          */
          int count;        /* code count          */
          TREE *parent;     /* pointer to parent   */
          TREE *rite;       /* pointer to right child  */
          TREE *left;       /* pointer to left child   */
       }  huftree[511];

/*=*=*=*=*=*=*=*/

main( argc, argv )
int argc; char **argv;
{
/* check command line parameters */

  if ( argc != 3 )
  {  puts( "usage: dohuf infile[/hr][:d] outfile[/huf][:d]" );
     exit(EOF);
  }

/* build filenames from command line args */

  addext( strcpy( f_name1, argv[1] ), "hr" );
  addext( strcpy( f_name2, argv[2] ), "huf" );

/* null out huffman tree */

  zero( huftree, sizeof huftree );

/* load /HR file into graphics board RAM */

  loadHRfile( f_name1 );

/* open /HUF file */

  if ( !( fp = fopen( f_name2, "w" ) ) )
  {  printf( cant_open, f_name2 );
     exit(EOF);
  }

  get_counts();
  buildtree();
  dataout();

  fclose( fp );
}

/*=*=*=*=*=*=*=*/

void loadHRfile( filename )
char   *filename;
{
  if ( !( fp = fopen( filename, "r" ) ) )
  {  printf( cant_open, filename );
     exit(EOF);
  }
  gfx_mode(1);
  gfx_cls();
   if ( getimage( fp ) == EOF )
  {  gfx_mode(0);
     printf( "Can't read %s\n", filename );
     exit(EOF);
  }
  gfx_mode(0);
  fclose( fp );
}


/*=*=*=*=*=*=*=*/

void get_counts()
{
  int    i, *counts, sqr_ctr = 19200;

/* allocate zeroed memory block for array of 256 ints */

  counts = calloc( 256, sizeof(int) );

/* get the 19,200 pixel squares, and count how many of each */

  while ( sqr_ctr-- )
     ++*(counts + get_square() );

/* write the 256 counts to the output file */
```

```
    fwrite( counts, sizeof(int), 256, fp );

/* assign code values and copy the counts
 * to the first 256 huffman tree members */

   for ( i = 0; i < 256; i++ )
   {  huftree[i].code = i;
      huftree[i].count = counts[i];
   }


/* release allocated memory */

   free( counts );
}


/*=*=*=*=*=*=*/

int get_square()
{
   static int  dat1, dat2, x = 0, y = 0;
   int         retcode;

/* if on left side, read vertical pair of bytes from graphics board
 * RAM, then put high nibble of upper byte in high nibble of retcode,
 * and high nibble of lower byte in low nibble of retcode */

   if ( !( x & 1 ) )
   {  outport( XREG, x >> 1 );
      outport( YREG, y );
      dat1 = inport( GFXDAT );
      outport( YREG, y + 1 );
      dat2 = inport( GFXDAT );
      retcode = ( dat1 & 0xf0 ) | ( dat2 >> 4 );
   }

/* If on right side, graphics board data is still in dat1 and dat2; so no
 * graphics board read needed. Put low nibble of upper byte into high
nibble
 * of retcode and low nibble of lower byte into low nibble of retcode. */

   else
      retcode = ( ( dat1 << 4 ) | ( dat2 & 0x0f ) ) & 0xff;

/* If at end of line, set up for next line. If
 * at end of image, set up for next screen read. */

   if ( ++x == 160 )
   {  x = 0;
      if ( ( y += 2 ) == 240 )
         y = 0;
   }

   return retcode;
}

/*=*=*=*=*=*=*/

void buildtree()
{
   static TREE    *ptr1, *ptr2, *temp, *limit;   /* statics for speed */

   limit = huftree + 256;

   while ( TRUE )
   {
      temp = huftree;

      while ( temp->parent || !temp->count )
         ++temp;
      ptr1 = temp++;

while ( ( temp->parent || !temp->count ) && temp < limit )
         ++temp;
      if ( ( ptr2 = temp ) == limit )
         break;


   /* always keep lower count in ptr2 */

      if ( ptr1->count < ptr2->count )
      {  ptr2 = ptr1;
         ptr1 = temp;
      }

      while ( ++temp < limit )
         if ( !temp->parent && temp->count )
         {  if ( temp->count < ptr2->count )

      /* if ptr1 >= ptr2 > temp, then if ptr1 == ptr2 keep ptr1 */

            {  if ( ptr1->count > ptr2->count )
                  ptr1 = ptr2;
               ptr2 = temp;
            }
            else if ( temp->count < ptr1->count )

      /* if ptr1 > temp >= ptr2, replace ptr1 w/temp */

               ptr1 = temp;
         }


      ptr1->parent = ptr2->parent = limit;
      limit->count = ptr1->count + ptr2->count;
      limit->rite = ptr1;
      limit->left = ptr2;
      ++limit;
   }
}


/*=*=*=*=*=*=*/

void dataout()
{
   int    sqr_ctr = 19200;

   while ( sqr_ctr-- )
      compress( huftree + get_square(), NULL );
   outbit( EOF );
}


/*=*=*=*=*=*=*/

void compress( this, prior )
TREE    *this, *prior;
{
/* recursive calls step from leaf to root of tree */

   if ( this->parent )
      compress( this->parent, this );

/* recursive returns send TRUE (1) if left or
 * FALSE (0) if right, in order of root to leaf */

   if ( prior )
      outbit( prior == this->left );
}

/*=*=*=*=*=*=*/

void outbit( bit )
int    bit;
{
   static int  byte = 0, ctr = 0;

   if ( bit == EOF )             /* code to flush 8-bit buffer */
   {  if ( !ctr )
         return;                /* do nothing if buffer empty */
      else
      {  byte <<= 8 - ctr;       /* else move bits to extreme left */
         ctr = 8;               /* and force write to file */
      }
   }
   if ( ctr == 8 )
   {  if ( putc( byte, fp ) != byte )
      {  perror( "putc()" );
         exit(EOF);
      }
      ctr = byte = 0;
    }
   byte = ( byte << 1 ) | bit;
   ++ctr;
}
```

UNHUF.C

```
/* unhuf.c
 *
 * Author:      J.F.R. "Frank" Slinkman
 * Date:        20-Nov-93
 * Compiler:    Pro-MC
 * Libraries:   Slinkman Pro-MC Hi-Res graphics library
 * Source:      "C Programming Column", Dr. Dobbs Journal, Feb 1991.
 *          Note:  listings accompanying article contains bugs!
 *
 * This program reads the Huffman-compressed files created by the
companion
 * program, "dohuf.c," decompresses the data, displays it on the hi-res
 * graphics screen, then writes the uncompressed image data from screen
to
 * an /HR file.
 */

#include    <stdio.h>
#option INLIB
#option REDIRECT   OFF
#option FIXBUFS ON
#option MAXFILES   1
#define XREG    128
#define YREG    129
#define GFXDAT  130
#define TREE struct tree

void   get_counts(), buildtree(), decompress(), put_square(), writeHRfile();

FILE   *fp;
char   f_name1[15], f_name2[15], cant_open[15] = { "Can't open %s\n" };
TREE   {  char code;       /* code value           */
          int  count;       /* code count           */
          TREE *parent;     /* pointer to parent    */
          TREE *rite;       /* pointer to right child  */
          TREE *left;       /* pointer to left child   */
       }  huftree[511], *root;

/*=*=*=*=*=*=*/

main( argc, argv )
int argc; char **argv;
{
/*  check command line parameters  */

   if ( argc != 3 )
   {  puts( "usage: unhuf infile[/huf][:d] outfile[/hr][:d]" );
      exit(EOF);
   }

/*  build filenames from command line args  */

   addext( strcpy( f_name1, argv[1] ), "huf" );
   addext( strcpy( f_name2, argv[2] ), "hr" );

/*  null out huffman tree  */

   zero( huftree, sizeof huftree );

/*  open /HUF file  */

   if ( !( fp = fopen( f_name1, "r" ) ) )
   {  printf( cant_open, f_name1 );
      exit(EOF);
   }

    get_counts();
   buildtree();
   decompress();

   fclose( fp );
   writeHRfile( f_name2 );
}

/*=*=*=*=*=*=*/

void get_counts()
{
   int  i, *counts;

   counts = calloc( 256, sizeof(int) );

/*  read counts array from input file  */

   if ( fread( counts, sizeof(int), 256, fp ) != 256 )
   {  perror( "fread()" );
      exit(EOF);
   }

/*  assign code values and copy the counts
 *  to the first 256 huffman tree members  */

   for ( i = 0; i < 256; i++ )
   {  huftree[i].code = i;
      huftree[i].count = counts[i];
   }

   free( counts );
}

/*=*=*=*=*=*=*/

void buildtree()
{
   static TREE    *ptr1, *ptr2, *temp, *limit;   /*  statics for speed  */

   limit = huftree + 256;

   while ( TRUE )
   {
      temp = huftree;

      while ( temp->parent || !temp->count )
         ++temp;
      ptr1 = temp++;

      while ( ( temp->parent || !temp->count ) && temp < limit )
         ++temp;
      if ( ( ptr2 = temp ) == limit )
      {  root = ptr1;
         break;
      }

   /*  always keep lower count in ptr2  */

      if ( ptr1->count < ptr2->count )
      {  ptr2 = ptr1;
         ptr1 = temp;
      }

      while ( ++temp < limit )
         if ( !temp->parent && temp->count )
         {  if ( temp->count < ptr2->count )

      /*  if ptr1 >= ptr2 > temp, then if ptr1 == ptr2 keep ptr1  */

            {  if ( ptr1->count > ptr2->count )
                  ptr1 = ptr2;
               ptr2 = temp;
            }
            else if ( temp->count < ptr1->count )

      /*  if ptr1 > temp >= ptr2, replace ptr1 w/temp  */

               ptr1 = temp;
         }

      ptr1->parent = ptr2->parent = limit;
      limit->count = ptr1->count + ptr2->count;
      limit->rite = ptr1;
      limit->left = ptr2;
      ++limit;
   }
}

/*=*=*=*=*=*=*/

void decompress()
{
   TREE *ptr;
   int  bytectr = 19200;

   gfx_cls();
   gfx_mode(1);
```

```
    while ( bytectr-- )
    {  ptr = root;

/* starting at "root," step through tree, using direction bits from /HUF
 * file, until end leaf (i.e., an element with no children) is reached,
 * and send it's code for screen display */

       while ( ptr->rite )
          if ( inbit() )
             ptr = ptr->left;
          else
             ptr = ptr->rite;

       put_square( ptr->code );
    }

    gfx_mode(0);
}
/*=*=*=*=*=*=*/

int inbit()
{
   static int  byte, mask = 0;
   int retval;

   if ( !mask )
   {  if ( ( byte = getc( fp ) ) == EOF )
      {  perror( "getc()" );
         exit(EOF);
      }
      mask = 0x80;
   }
   retval = byte & mask;
   mask >>= 1;
   return retval;
}

/*=*=*=*=*=*=*/

void put_square( code )
int code;
{
   static int  x = 0, y = 0, dat1, dat2;

   if ( !( x & 1 ) )

/* if on left side, mask out low nibble of "code" to init upper data byte,
 * and shift low nibble of "code" to high nibble to init lower data byte */

   {  dat1 = code & 0xf0;
      dat2 = code << 4;
   }

   else

/* if on right side, OR high nibble of "code" with lower nibble of upper
 * byte, and OR low nibble of "code" with lower nibble of lower byte */

   {  dat1  |= code >> 4;
      dat2  |= code & 0x0f;

/* write combined left and right squares to gfx board RAM */

      outport( XREG, x >> 1 );
      outport( YREG, y );
      outport( GFXDAT, dat1 );
      outport( YREG, y + 1 );
      outport( GFXDAT, dat2 );
   }

/* Bump "x" for next call.  If at end of line, set up for next line */

   if ( ++x == 160 )
   {  x = 0;
      y += 2;
   }
}

/*=*=*=*=*=*=*/

void writeHRfile( filename )
char   *filename;
{
   if ( !( fp = fopen( filename, "w" ) ) )
   {  printf( cant_open, filename );
      exit(EOF);
   }
   if ( putimage( fp ) == EOF )
   {  perror( "putimage()" );
      exit(EOF);
   }
   fclose( fp );
}
```

ITEMS OF INTEREST

VTUG

Valley TRS-80 Users Group meets the third Friday of eacgh month at 7:30 pm at the Learning Tree University. DeSoto & Knapp Streets, Chatsworth, CA Contact Lance Wolstrup (818) 716-7154

Mid-Cities Tandy Radio Shack Users Group

MCTRUG supports all of the Tandy computers plus IBM compatibles. We have software acailable for TRS-80 Models I, III, 4, Color Computers, Model 2000 and MS-DOS. Write us for more information. Please include your name, mailing address, computer model and which Disk Operating System you use.

Write to: MCTRUG
P.O. Box 171566
Arlington, TX 76003

VTHG

Valley TRS-80 Hackers Users Group Los Angeles, CA. meets the first Friday of each month at 7:00 pm. for more information contact Roy Beck (213) 664-5059

SAGATUG

San Gabriel Tandy Users Group meets the second Friday of each month at 7:-- pm at the Arcadis Park Senior Citizen's Center. 405 South Santa Anita Avenue. Arcadia, CA Contact Barbara Beck (213) 664-5059

LOTSA DRIVES FOR MODEL 4

by Kelly Bates

I have 4 slimline drives in my Model 4, accessed from the top down as 0, 1, 3, and 4. I skipped 2 on purpose. As configured, there is 360K x 4 on-line or 1.4 meg. If you have the 128K upgrade then Memdisk can format the upper banks for an additional 60K (as drive 2). And then, if you could disconnect 2 drives without without opening the case and hook up 2 external drives of 720K each, then your on-line storage would be about 2 meg, Read on if interested.

1. Modify the tower to accept 4 slimline drives. If you have the plastic tower, there are 8 supports for the installed 2 drives. Shave off about 2/3 of the top of each support. This will let your new drives drop down a bit to allow access for the drive door handles, and you also won't have to resize the case opening for drive access on the case upper half. Drill some new screw holes to mount the drives in the sides of the tower.

2. Mount a power supply for the additional drives in the bottom of the chassis just behind the keyboard, or a bit further back if you have a speaker mounted behind the keyboard. This new power supply should feed the 2 lower drives, so be sure your power feed is long enough to get to the drives. Hook'em up.

3. The drive controller card just behind the drives or just ahead of the CPU feeds all 4 drives. The top connection normally goes to the original two internals configured as :1 and :1 so, connect it to the two upper drives. The lower or external connection of the drive controller feeds two external drives and/or the additional two drives you have just installed. Looking at the bottom of the chassis you see the two openings for the printer and external drives. I enlarged each of these as I frequently change printers and drives. Do it, makes removal of the edge card connectors a lot easier.

4. Just forwards of the printer and drive holes on the chassis are some air flow vents. Make one of them big enough to feed an edge card connector through. Then connect the two lower drives through your larger air vent to the external drive card edge connection and configure the drives as :0 and :1, just like the first two. Tuck the excess ribbon cable into your new air flow vent. You will note that the leg and the foot is slightly in the way. Drill a new hole for it and remount it. Now your printer cable can be easily connected towards the front of the case if it is set up like mine.

Now, if you have an additional set of drives to hook up as externals, disconnect the lower drives and plug'em in. Your mod is complete. Put it all together.

I had originally thought of adding a muffin fan for cooling, but I don't think it is necessary. I had also thought of powering the two additional drives with the original power supply, but I am not so sure that it is advisable. Suit yourself!

I have purposely left out the tidbit instructions, such as the length of cables, how big to make the new openings, where to drill the holes and what size drill to use. Use your own judgement and common sense.

As an aside, I recently learned that Radio Shack. sells a tool to put connectors on flat ribbon (276-1596) for about 14 dollars. Good investment.

When using 720K (3 1/2) externals, your DOS will think that they are just LARGE 5 1/4 floppies. Don't tell it differently! If you do a BACKUP after formatting the 720K and reconstruct, you should actually be able to use the 720K as a boot disk, and four of those on-line will net about 3 meg to play with.My first attempt at this netted a good Mod 4 TRSDOS 6.0 and a good TRSDOS 1.3 (double-sided, 40 track). The NEWDOS, LDOS and CPM would not boot. What you may have to do is waste some space on the boot disk and just format it as a 40 track disk. The problem is probably related to relocating the /SYS files, and DOS can't find the stuff where it thought it ought to be. But once you have the boot disk, the other drives can be formatted as 720K — that part worked well on my machine. The mechanics are easy, set up the 720K's as externals first. Make your boot disk and then put the 720K floppy in as drive :0 and see if it will boot.

This project was for my own use, but I thought that the TRSTimes readers would like to know what other readers are doing, trying to stretch the use of a Model 4. So, building fonts primarily and tinkering with the machine keeps me pretty busy.

A question. Does anybody know where I can get a PAL chip to upgrade memory on my 2nd Model 4? I have some PAL chips, but the number is a bit different and I don't know if I can use any of them:

PAL16R6ACN, PAL16L8CN and PAL10L8CN.

Would any of those workin place of the PAL16L8ACN that is supposed to be used? I can get the D4164C-15 memory chips locally, so they are not a problem. Radio Shack wants about $30 for the PAL chips and that is a bit steep-. So — HELP! Will also swap parts and expertise. Contact me at:

1125 SE 23rd Street
Oklahoma City, OK 73129
(405) 670-3753

VARIABLE TIME DELAY ROUTINE

by Cass R. Lewart

To provide variable time delays during program execution, you can use the subroutine listed below. It consists of only eight bytes of code and provides a very large range of possible delays.

The routine can be used by Model I, III and 4, as well as any other Z80 machine. The length of the desired delay is passed to the subroutine in register pair HL. The value of HL can be in the range 0-65535.

```
DLAY    LD      B,L         ;get delay from HL
DLAY1   DJNZ    DLAY1       ;main delay loop
        DEC     HL
        LD      A,L         ;check if HL=0
        OR      H
        JR      NZ,DLAY     ;continue if not
        RET                 ;return to caller
```

FASTTERM II ZAPS

by Gary W. Shanafelt

If you use Mel Patrick's FastTerm II terminal program, you know the major settings can be changed and saved to disk in a configuration file which is loaded every time you load the program. A number of default settings can't be customized this way, though, so if you don't like them you have to change them manually every time you run FastTerm. What follows are patches to modify two of them in the program itself.

First, the default FastTerm setting is for pull-down windows. You hit <F1> and a window on the left of the screen appears, showing you the current status of Echo and Print mode. You press the arrow keys to move to different windows. I rarely do anything with the echo settings; it seems to me it would be a lot more convenient for the dialer to be the first window you get when you hit <F1>, not the echo screen.

The following patch makes the dialer the initial default:

```
PATCH FTII/CMD (D48,63=04;F48,63=01)
```

You can make any of the windows the default by inputting a different number than 04, for the dialer is simply the fourth window going across the screen from the left.

If you don't like the pull-down windows at all, FastTerm allows you to invoke a big custom window/menu when you hit the <F1> key. But to toggle to this mode from the arrow-activated pull-down window mode, you have to hit <SHIFT><F3> every time you start the program -- which can get tiresome after a while.

The following patch makes the custom window/menu the default when you run FastTerm:

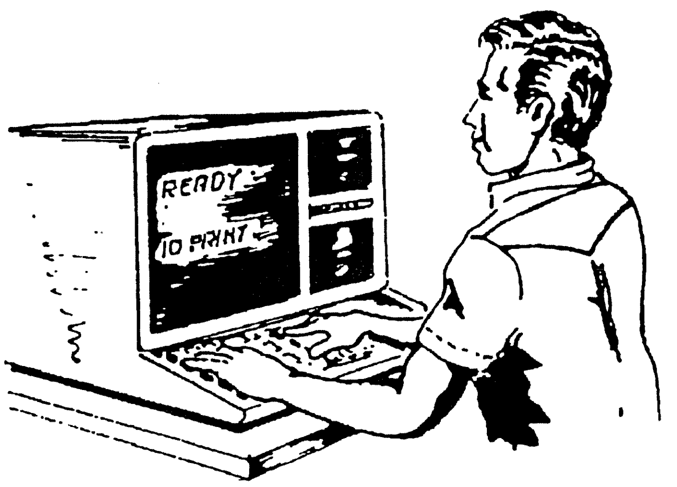
```
PATCH FTII/CMD (D5D,F0=01;F5D,F0=00)
```

These patches are for version 4.65 of the program, the latest one on Mel's BBS. You should not try them on earlier versions because the bytes to be changed are in different memory locations. If you don't have version 4.65, you can download it from him by calling his BBS at (604) 574-2072.

How did I figure this out? There is probably an easy way, but the way I used was to run FastTerm with the default window setting, exit the program, and dump the memory; then to run the program again, change the setting to what I wanted, and make a second dump. Finally, I ran the two dumps through a byte-by-byte comparison program and investigated the discrepancies: obviously one of the different bytes in the second dump was the toggle between the two modes that I wanted to change. I zapped in changes with LSFED-II on the original program until I got a version that showed the change I wanted when I ran it. Having figured out the first patch, I then repeated the procedure to determine the second.

TRANSFERRING FILES TO NEWDOS/80 v.2

by Lance Wolstrup



In our last issue, I wrote a short article about using NEWDOS/80 double-density disks with the Model I emulator. I presented patches to the MAKFILE/CMD and MAKFIL4/CMD programs that would allow them to read and write 18 sectors per track, instead of the normal 10. I then proceeded to say that you could now use NEWDOS/80 as the transfer vehicle, and to "simply copy all files from whatever DOS they were on over to NEWDOS".

Well, it seems that copying files to and from NEWDOS/80 v.2 is becoming an ancient and forgotten art. Phonecalls and letters have reminded me that many readers, both new and old to the TRS-80 world, need help in this mystical matter. So, let me clean off the earthquake dust from the laborious NEWDOS/80 manual and see if I can shed some light on this mess.

First we will discuss transferring files to NEWDOS/80 v.2 on a Model I. This article assumes that your Model I is capable of reading and writing double-density, single-sided, 40-track disks and that you have at least two drives — if you have more than two, bully for you, but two drives is the minimum configuration. It is further assumed that, like me, you have the Radio Shack double-density modification, and the mandatory diskzaps from Apparat have been correctly installed.

MODEL I

1. Be sure to make several copies of your original NEWDOS/80 v.2 system diskette before we begin. Then put the master away in a safe place.

2. Insert one of the backups in drive :0. Boot your Model I, answer the date and time prompts, and you should now have NEWDOS/80 READY on the screen.

3. Type:

```
PDRIVE,0 ENTER
```

This command will display the PDRIVE table. It should look something like this:

```
0* TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
1* TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
2* TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
3* TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
4* TI=CM,TD=E,TC=40,SPT=18,TSR=3,GPL=6,DDSL=17,DDGA=2
5* TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
6* TI=CK,TD=E,TC=39,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
7* TI=A,TD=C,TC=80,SPT=20,TSR=2,GPL=2,DDSL=17,DDGA=2
8* TI=C,TD=E,TC=40,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
9* TI=C,TD=G,TC=80,SPT=36,TSR=3,GPL=8,DDSL=17,DDGA=2
```

This table is the heart and soul of NEWDOS/80 v.2; it is the key to recognizing alien disk formats. If you find this somewhat confusing — don't feel bad, many good people have jumped off tall buildings because of this dreaded command.

4. We now must change all the TI specs to reflect the Radio Shack double-density mod. Type:

```
PDRIVE,0,0,TI=D ENTER
PDRIVE,0,1,TI=D ENTER
PDRIVE,0,2,TI=D ENTER
PDRIVE,0,3,TI=D ENTER
PDRIVE,0,4,TI=D ENTER
PDRIVE,0,5,TI=A ENTER
PDRIVE,0,6,TI=DK ENTER
PDRIVE,0,7,TI=A ENTER
PDRIVE,0,8,TI=D ENTER
PDRIVE,0,9,TI=D ENTER
```

Notice that you will get a message telling you that the following error has occurred:

```
*** TI= SPEC BETWEEN DRIVES INCOMPATIBLE
```

Don't worry about this. The message will disappear when you have changed TI specs on all drives as shown above.

5. Assuming that the above error message has disappeared, type:

PDRIVE,0,A ENTER

6. Set up drive :1 to allow the creation of a double-density, single-sided, 40 track bootable system diskette.
Insert a blank diskette in drive :1 and type:

COPY 0,1,,USD,FMT,DPDN=6 ENTER

Answer Y to the System/Source disk prompt.

Press ENTER to the destination mount prompt.

The diskette in drive :1 is formatted in double-density and the system and user files are copied from drive :0 to :1. Then track 0 of the disk in drive :1 is formatted to single-density to allow Model I booting.

7. Remove both disks from their drives and insert the newly-made double-density boot disk in drive :0 and press the RESET button. If everything went according to plan you should now boot up in double density.
At this point I suggest that you make a few backups of your new boot disk. Insert the disk in drive :1 and type:

COPY,0,1,,USD,FMT,DPDN=6 ENTER
and answer the prompts as explained above.

8. Now PURGE all user files from the boot diskette to make as much room as possible. Type:

PURGE,0,USR ENTER

and press Y to each of the KILL prompts.

9. Now, Model I NEWDOS/80 v.2 cannot read alien double-density formats — the problem is the same as in the Model I emulator — the directory of the alien disk is not where NEWDOS can find it. The exception to this is TRSDOS 1.3 for Model III. NEWDOS/80 has a special PDRIVE setting to recognize a TRSDOS 1.3. disk. More on that in a second.

If you are disappointed that we cannot transfer double-density Model I LDOS, DOSPLUS, or MULTIDOS disks to the emulator via NEWDOS/80, don't fret. Simply boot with the particular DOS, format some single-density data diskettes and copy the files from the double-density disk to the single-density ones. You can now transfer the single-density disks with MAKFILE/CMD as described in previous articles.

OK, back to NEWDOS/80 and TRSDOS 1.3. The PDRIVE setting to configure for TRSDOS 1.3 should be in slot 4. It should be:

TI=DM,TD=E,TC=40,SPT=18,TSR=3,GPL=6,DDSL=17,DDGA=2

10. If slot 4 of the PDRIVE table reads as just described, type:

PDRIVE,0,1=4,A ENTER

If slot 4 of the PDRIVE table reads differently, you must then bite the bullet and type the entire configuration:
PDRIVE,0,1,TI=DM,TD=E,TC=40,SPT=18,TSR=3,
GPL=6,DDSL=17,DDGA=2 ENTER

11. Insert the TRSDOS 1.3 disk in drive :1 and type:

COPY,1,0,,NFMT,CBF,CFWO,USR ENTER

Answer Y to the System/Destination prompt.

Press ENTER to the Source mount prompt.

Each user file will now be displayed and you will be asked to press Y if you wish to copy it. Step through the list of files, pressing Y if you want to copy it. When you reach the end of the file list, all marked files will be copied to the NEWDOS/80 disk in drive :0

12. Set drive :1 to handle a true double-density data disk (all 40 tracks are double-density) so we can copy the user files from the system diskette to it. Type:

PDRIVE,0,1=8,A ENTER
This assumes that slot 8 of the PDRIVE table reads:
TI=D,TD=E,TC=40,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2

If slot 8 of the PDRIVE table does not read the above, you will simply have to change it.

13. Format drive :1 and then copy the user files from the double-density system diskette in drive :0 to the double-density data disk in drive :1. Type:

COPY,0,1,,FMT,CBF,USR ENTER

Press Y to the System/Source prompt.

Press ENTER to the Destination mount prompt.
The disk in drive :1 will have all 40 tracks formatted double-density, and the user files from drive :0 will be copied over. When done, the data disk is ready to be used with the DD2FILE/CMD or DD42FILE/CMD programs, as explained in the article on page 31 of our last issue.

MODEL III

We will now move on to the Model III portion of this article. This machine is much more capable of handling our task, as it has the advantage of being a natural 'double-density' machine — that is, the Model III drives are all double-density drives. This means that we can go directly to our task without having to change the TI specs as we had to for the Model I.

1. Make a copy of your NEWDOS/80 v.2 boot disk. Put the master away, then boot with the newly made disk.

2. Type:

PDRIVE,0 ENTER

The PDRIVE table will now be displayed, and it should look something like this:

```
0* TI=A,TD=E,TC=40,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
1* TI=A,TD=E,TC=40,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
2* TI=A,TD=E,TC=40,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
3* TI=A,TD=E,TC=40,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
4* TI=AM,TD=E,TC=40,SPT=18,TSR=3,GPL=6,DDSL=17,DDGA=2
5* TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
6* TI=AK,TD=E,TC=39,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
7* TI=A,TD=C,TC=80,SPT=20,TSR=2,GPL=2,DDSL=17,DDGA=2
8* TI=D,TD=C,TC=80,SPT=20,TSR=3,GPL=2,DDSL=17,DDGA=2
9* TI=A,TD=G,TC=80,SPT=36,TSR=2,GPL=8,DDSL=17,DDGA=2
```

3. Make as much room as possible on the new boot diskette by getting rid of all user files. Type:

PURGE,0,USR ENTER

and press Y   T F4 each one of the KILL prompts.

4. Model III NEWDOS/80 v.2 cannot read alien double-density formats — except for TRSDOS 1.3, which has a special PDRIVE setting. It is usually in slot 4 of the table (if slot 4 in your PDRIVE table is different than the one listed above, change it to match). Then insert a TRSDOS 1.3 disk in drive :1 and type:

PDRIVE,0,1=4,A ENTER

followed by:

COPY,1,0,,NFMT,CBF,CFWO,USR ENTER

answer Y to the System/Destination prompt.

press ENTER to the Source mount prompt.

Each user file on the TRSDOS 1.3 diskette will now be displayed in turn, and you will be asked to press Y if you wish to copy it. Step through the files, pressing Y to copy, or N if not. When you reach the end of the file list, all marked files will be copied to the NEWDOS/80 v.2 diskette in drive :0.

Note that programs on Model III LDOS, DOSPLUS, or MULTIDOS can be transferred to the emulator. Simply boot with the particular DOS, format some single-density data diskettes and copy the files from the double-density disk to the single-density ones. You can now transfer the single-density disks with MAKFILE/CMD as described in previous articles.

5. Remove the TRSDOS 1.3 disk from drive :1, and in its place, insert a blank disk. Reset the PDRIVE table to reflect that slot 1 is a standard NEWDOS/80 v.2 double-density diskette. Type:

PDRIVE,0,1=0,A ENTER

Now, copy the files from the system disk in drive :0 to a formatted data diskette in drive :1. Type:

COPY,0,1,,FMT,CBF,USR ENTER

Answer Y to the System/Source prompt.

Press ENTER to the Destination mount prompt.

The diskette in drive :1 will be formatted, after which the user files from drive :0 will be copied to drive :1.

You can now transfer this disk to the emulator with the DD2FILE/CMD or DD24FILE/CMD programs, as explained in the article on page 31 of our last issue.

TRSTimes covered the PDRIVE mess in two previous articles: 'COPYAID' in the May/Jun 1988 issue (1.3), and 'PDRIVE WITHOUT TEARS' in the Mar/Apr 1991 issue (4.2). Both article contained programs that would tame the PDRIVE command and we will make them available on a NEWDOS/80 v.2 double-density data diskette for $5.00 in North America and $7.00 anywhere else.

LIGHT (bulb) FILLER:

Q: How many members of the Impossible Missions Force does it take to change a light bulb?

A: Five: While Cinnamon creates a diversion by wearing a skimpy dress, I use a tiny narcotic dart to knock out the fascist dictator and remove his body. Rollin, wearing a plastic mask, masquerades as the dictator long enough for Barney to sneak up to the next floor, drill a hole down into the light fixture, remove the burned-out bulb, and replace it with a new super-high wattage model of his own design. Meanwhile, Willie has driven up to the door in a laundry truck. Just before Rollin's real identity is revealed, we escape to the laundry truck, drive to the airfield, and return to the United States.

ATTENTION TRSDOS 1.3 USERS!

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

MORE SPEED!! MORE POWER!! NEW LOW PRICE!!

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

config=y/n	creates config boot up	filedate=y.n	date boot up prompt on/off
time=y/n	time boot up prompt on/off	cursor='xx'	define boot up cursor character
blink=y/n	set cursor boot up default	caps=y/n	set key caps boot up default
line='xx'	set *pr lines boot up	wp=d.y/n	write protect any or all drives
alive=y/n	graphic monitor on/off	trace=y/n	turn sp monitor on/off
tron=y/n	add an improved tron	memory=y/n	basic free memory display monitor
type=b/h/y/n	high/bank type ahead on/off	fast	4 mghz speed (model 4)
slow	2 mghz speed (model 3)	basic2	enter rom basic (non-disk)
cpy (parm,parm)	copy/list/cat ldos type disks	sysres=h/b/'xx'	move/sys overlay(s) to hi/bank mem
sysres=y/n	disable/enable sysres	macro	define any key to macro
spool=h/b.size	spool is high or bank memory	spool=d.size='xx'	link mem spooling to disk file
spool=n	temporarily disable spooler	spool=y	reactivate disabled spooler
spool=reset	reset (nil) spool buffer	spool=open	opens, reactivates disk spooling
spool=close	closes spool disk file	filter *pr.adlf=y/n	add line feed before printing0dh
filter *pr.iglf	ignores 'extra' line feeds	filter *pr.hard=y/n	send 0ch to printer (fastest tof)
filter *pr.filter	adds 256 byte printer filter	filter *pr.orig	translate printer byte to chng
filter *pr.find	translate printer byte to chng	filter *pr.reset	reset printer filter table
filter *pr.lines	define number of lines per page	filter *pr.width	define printer line width
filter *pr.tmarg	adds top margin to printouts	filter *pr.bmarg	adds bottom margin to printout
filter *pr.page	number pages, set page number	filter *pr.route	sets printer routing on/off
filter *pr.tof	moves paper to top of form	filter *pr.newpg	set dcb line count to 1
filter *ki.echo	echo keys to the printer	filter *pr.macro	turn macro keys on/off
attrib :d password	change master password	device	displays current config

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

SYSTEM 1.5. is now distributed exclusively by TRSTimes magazine.
ORDER YOUR COPY TODAY!

NEW LOW PRICE $15.95

TRSTimes - SYSTEM 1.5.
5721 Topanga Canyon Blvd., Suite 4
Woodland Hills, CA. 91367

Some Memory Meanderings

by Roy T. Beck

Recently I had occasion to peruse the memory configurations of the ubiquitous IBM clone, and in the process, I also began to reflect on the memory capabilities of our favorite TRS machines, the Models 4 and 4P. Sure, both of these are said to be capable of 128K, and many of them actually have the extra 64K installed.

But how does the Z-80 CPU make use of the extra memory? After all, the memory map of the Z-80 is limited to 64K, or so they tell us. How can it access more than 64K? In fact, some of the aftermarket packages offer up to 1 MEG of memory for a Mod 4 with a Z-80 CPU. What gives here? Perhaps it is germane to ask what is the property of a Z-80 which allows it to access a specific memory cell anywhere in its memory map.

The key to selecting a memory cell is setting an address on the address lines, each of which operate in a binary manner, representing either a one or a zero, depending on the voltage on the line. Since the address lines operate in a binary fashion, the number of possible combinations of logic ones and zeroes is two raised to the power n, n being the number of lines present. The Z-80 has 16 lines, and two to the 16th power is 65,536. Since we like to use the term K (1024) to more easily represent large numbers, 65,536/1024 = 64, hence we say the Z-80 has a 64K memory map. Note the 16 address lines required to achieve control of a 64K memory map. Since there are only 16 address lines, the Z-80 simply cannot address more than 64K memory cells with its address lines.

The TRS-80 solution to the problem of addressing more than 64K in a Z-80 machine is bank switching. In a none-too-good analogy, you could imagine a deck of cards, each of which represents a 64K memory map. The deck of cards thus can represent multiple 64K memory fields. With some electronic trickery, the Z-80 chip can determine which memory map (card) it wishes to connect to. By interchanging memory fields, it can address an unlimited number of such memory maps, each of which is 64K in size. But the inevitable gotchas are numerous.

First and foremost, the CPU can only address one of the memory maps at a time, no matter how many of them there are in the machine. Secondly, the instructions which direct the Z-80 are themselves stored somewhere in the presently active memory map. If, between instructions, you change the memory map in use, where will the next instruction come from, and will it be the correct one? How do you insure this will work correctly?

The method implemented in the Mods 4 and 4P is bank switching of HALVES of the total memory map, each half being 32K in size. Think of the 128K TRS as having four 32K memory blocks, or banks, any two of which can appear in the memory map of the Z-80 at a given instant. The actual selection of a 32K memory bank is done by means of logical latches, which appear as "ports" in the Z-80's other "memory map". I say OTHER memory map, because it would be perfectly possible to add another block of memory, accessed via the ports. But this is not viable, because there are only 256 ports in the other memory map, and besides, these ports provide access to the outside world for things such as printers, modems, alternate character sets, floppy and hard disk drives, etc, etc. When you get into the application of ports to real designs, 256 ports are not very many, and there really is a need for more of them as ports; use of them as additional memory would provide very little memory, and would further diminish the number of ports available for communication with hardware.

Going back for the moment to the problem of storing and providing instructions to the Z-80 while swapping (bank-switching) its fields of memory, the solution adopted for the Models 4 & 4P is to leave one 32K bank always present and always addressed as 0000h to 7FFFh, with the other three 32K banks addressed selectively at 8000h to FFFFh, but only one at any given time.

It is desirable that one 32K bank remain always present in the lower half of the memory map, because this is where the DOS kernel resides. By keeping this bank always present, the DOS is happy and can swap the other three banks in and out of the upper half of the memory bank freely as desired.

How is this extra memory capability used? Actually, there are not all that many programs which use the extra banks of memory, but some do exist and the use of the extra memory is quite elegant.

One you may know of is MEMDISK. This is built into your DOS, and utilizes the extra memory banks as a small disk drive. I say small, because we are accustomed to flopppies of 360K and larger, and the two 32K banks net out, after allowing for overhead, to 63 K, which is smallish. But the virtue of MEMDISK is its speed of access, and with some imagination you can load your essential SYSTEM files there, and really crank up the effective speed of your machine.

Another application is AllWrite. It will automatically make use of the extra 64K, if present and unused in your machine, to accommodate larger documents than a 64K machine can handle. I am told LeScript also has this capability, but not having used LeScript, I cannot confirm this.

Still another application program is LB (formerly named Little Brother) by MISOSYS, a data base program. I use LB to hold and operate the TRSCLUBS data base which I use for all my mailing functions, among other things.

I previously touched upon the principal limitation of the extra memory in our Mod 4's, and that is the small amount of extra memory available. Even 128K seems laughable in this day of 16 Meg clones.

There is an answer to this. Many of you have addon memory boards named "XLR8er", formerly sold independently and later through MISOSYS. These boards added 256K of memory, which combined with your original 128K yielded 384K in your Model 4 or 4P. This was a significant boost over the original machine capacity. Several people developed software drivers to control the necessary bank switching within the original software concepts by LSI and MISOSYS.

Yet another answer was created by Peter Ray of Anitek, who also publishes LeScript. He offers several versions of memory expansion kits, the largest of which makes the 4 or 4P a One Meg machine. Quite an accomplishment! This latter case may have been overkill, as I don't know for sure what was available in the way of driver software, nor what programs could make proper use of this much memory.

Since both of these methods switch memory in 32K blocks, the bank switching driver necessarily becomes a large effort.

Don't think this is an outlandish solution; One of the forms of increased memory in the IBM bank switches up to 16 MEGS in only 16K banks! It can be done, and within the kludgy framework of the clones, it works well. That system gives you what is called "Expanded" or LIM 3.2/4.0 memory in the clone world.

In conclusion, bank switching is an interesting concept, works well when designed well, both in hardware and software, and greatly increases the effective memory space available in a given machine with a given CPU.

Next time, I hope to explain some of the IBM/clone techniques used to increase the effective memory size in a machine which is presently totally inadequate with "only" 640K of working RAM available in its original design.

HOW TO REPAIR ELECTRONIC EQUIPMENT

Humor from the TRSTimes vaults

1 Approach the ailing instument in a confident manner. This will give the instrument the mistaken idea that you know something. It will also impress anyone who happens to be looking, and if the instrument should suddenly start working you will be credited with its repair. Should this fail, proceed to step 2.

2 Wave the service manual at the instrument. This will make the instrument assume that you are at least familiar with the source of knowledge. Should this fail, proceed to step 3.

3 In a forceful manner, recite Ohm's Law to the instrument. (Caution: BEFORE TAKING THIS STEP, REFER TO A HANDBOOK TO BE SURE OF YOUR KNOWLEDGE OF OHM'S LAW.) This will prove to the instrument that you do know something. This is a drastic step and should only be taken if the first two steps fail. If this step fails, proceed to step 4.

4 Jar the instrument slightly. This may take anything from a three to six foot drop, preferably on a concrete floor. However, you must be careful with this step because, while jarring in the approved method of repair, you must not mar the floor. Again, this is a very drastic step. If it should fail, proceed to step 5.

5 Brandish a large screwdriver in a menacing manner. This will frighten the instrument and demonstrates the deadly "SHORT CIRCUIT" technique. If this step fails, proceed to step 6.

6 Add a tube...even if the instrument is solid state. This will prove to the instrument that you are familiar with the design of the instrument. Also, this will increase your advantage and confuse the instrument. If this step fails, proceed to the most drastic and dangerous step of all, step 7. It is very seldom used and is the last resort if all else fails.

7 Think...!? Is the most dangerous step of all! It is very seldom used and is the last resort if all else fails.

continued from page 2

We found some of Lance's CDs ten feet from the cabinets they'd been stored in, as if they'd been shot out of a canon. Soccer and baseball trophies were smashed, picture glass broken, frames bent.

In the kitchen, the refrigerator and china cabinets had flung open and dumped their contents on the floor. Yet, strangely, a few cabinets spilled nothing. The refrigerator had moved a foot out from the wall. There was maple syrup all over the floor. The glass legs of the glass-topped dining-room table had smashed, tipping the table over. There were cracks in the walls (ugly, but not structurally damaged it turns out). Tiles in the entrance hall had popped up. We had no lights, no water, no gas.

Then came the first big aftershock. At the time we didn't know whether it was going to be an aftershock, or if what we'd already experienced was merely a foreshock. We grabbed onto whatever was near and rode it out.

After a second aftershock, there was a deadly silence. Even though there are 49 other townhouses in our complex, there wasn't a sound. No voices. No commotion. Eerie. Then a knock at the door. Lance opened the door. It was our neighbor's son. His parents were trapped in their bedroom and he needed help rescuing them. Lance and Alan went to help. Just seeing another person made me feel better. We weren't alone. The earthquake hadn't just happened at our house!

It was still completely dark, but people started coming out of their houses. I finally found a flashlight. When Lance came back from the neighbor's, I made him accompany me back upstairs to help me look for the cat. I couldn't rest until I knew where she was. We found her under the bed, clearly unwilling to budge.

We joined our neighbors at the pool. Nothing could fall on our heads out there. I had a small purse-size TV. Others had radios. People gathered around to get news of what had just happened to us all. I was relieved to hear that the epicenter was closer to us than to my daughter. I hoped that meant she hadn't been as shaken up. (Two days later I reached her and she was fine). My other daughter, hearing about the quake on CNN in Paris, had been frantic. She'd stayed up all night with the phone in her lap, hitting the redial button, unable to get through.

It was the longest dawn on record. From 4:31 until it finally got light around 7 seemed an eternity. On the other hand, the stars over L.A. were beautiful that night. Without any city lights to dull them, they sparkled elegantly.

When it got light, somebody asked me, "Sylvia, what happened to your leg?" I looked down and saw blood all over the front of my nightgown. I pulled it up saw a large gash on my lower left leg. It must have been that flying VCR. I hadn't even realized I'd been cut.

Later that day, after we managed to dig our cars out of the garage, Lance drove me to the hospital to have my leg treated and bandaged up. On the way to the hospital there was another aftershock. Steering the car was suddenly tricky, like driving on flat tires. I saw huge street lights sway gracefully. We passed one of the big malls and saw that parts of it were badly damaged. Store windows by the dozens were broken.

While at the hospital, another aftershock. The huge building swayed. The patients in the Emergency waiting room all looked at each other, many giggling nervously when the movement passed. It was strangely comforting to know that all the doctors and nurses were going through the same experience we "patients" were going through. The earthquake was a great equalizer. I overheard one doctor say, "As soon as there's a lull I've got to take a break and go home. My family's freaking out."

That night, along with our neighbors, we slept by the pool in deck chairs. We still had no electricity, gas or water. Lance and the boys played cards by candlelight. I was glued to my little portable TV. I had the cat next to me in her carrier. I didn't want to leave her alone in the house.

For days, you couldn't buy food, water, gas or batteries. We realized we'd been badly prepared. We ate junk food and leftovers. Steven had no school for a week. I'm a psychotherapist, and I had to cancel patients (or they cancelled me). A curfew was imposed for after dark.

After a few days, we began to get phonecalls from out-of-state relatives and friends who were finally able to get through. We got calls from TRSTimes subscribers from everywhere. They were relieved to hear we hadn't been swallowed up by the sunny California earth.

We were lucky. Our house was "green tagged" (meaning it's structurally okay). But everybody knows somebody who got "yellow tagged" (limited entry) or, worse, "red tagged" (condemned). The apartment complexes both to the north and the south of have been "yellow tagged." People are now moving out in droves. As you go around town these days, you find yourself automatically looking to see if a building has been "green tagged" before going inside. Lots of buildings are to be demolished. Lots of businesses have gone out of business. Luckily, the TRSTimes equipment, despite dents and spills, is still functioning, so we're still publishing.

But it's not just what you can see from the street. Inside each home and each apartment, inside each room, there's been an earthquake, messes that it's going to take months to clean up, anxieties that it's going to take even longer to recover from.

Since January 17th we've had more than 5000 aftershocks. Many have been large enough to kick-start my heart again. For over a week, neither the cat nor I would sleep in the bedroom — because that's where earthquakes happen! I slept on the couch, and the cat found a "safe house" for herself under the kitchen sick behind the dishwasher.

Now we're venturing upstairs again, but every once in a while I see the cat stalking around the bed, crouched down low as though waiting for that earthquake under the bed to start up again.

The epicenter of the Northridge quake, we hear, has been relocated to Reseda. That's even closer to our house. Plus it's been officially upgraded from 6.6 to 6.8. Of course those of us in the area of the epicenter are not at all surprised at the upgrade. We knew it was a big, bad earthquake. Maybe it wasn't the Big One, but it will do!

LITTLE ORPHAN EIGHTY

For obvious reasons, we do not have much space left for this column, so let's get right down to business with some good news and some bad news. Let's do the good news first —

Chris Fara tells us "you'd think that with the 'Pentium' on the horizon, the Z80 would be dead by now. Not so! I nearly fell off the chair when a college in Texas ordered 22 sets of all four volumes of my 'Z80 Tutor' for a programming class! The books are listed in the yearly ISBN 'Books In Print' bible. They apparently found it there, first asked for preview copies and then went full hog."

And now for the bad news — I am told that as of March 1, 1994, Roy Soltoff will have closed down Misosys permanently. Good luck in your new endeavors, Roy — we will miss you.

As you can imagine, getting this issue of TRSTimes to you was not an easy task, and I wish to acknowledge all who helped making it possible. To Chris Fara, Henry Blumenthal, Danny Myers, Frank Slinkman, Kelly Bates, Cass Lewart, Gary Shanafelt, Roy Beck, and my sweet wife, Sylvia — a big, big thank you from the bottom of my heart.

Lance W.

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 optional.
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 cost add to all prices

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