THE alternate source

Volume I

The magazine of advanced applications and software for the TRS-80.

TABLE OF CONTENTS

(continued)

PREFACE

Things were not progressing fast enough for this TRS-80 owner during the summer of 1979. Articles for the TRS-80 were few and far between in the 'REAL' publications. There were at least a half dozen magazines devoted to the TRS-80 -- some appeared only a few times, if at all. Some supported the games crowd. Others were 'beginners' magazines that, if anything, were a drag to read.

I felt that there should be a source of information for those who had mastered the delicate art of CLOADing; those who didn't feel entirely incompetent after booting up DOS -- yet realized the existence of many worlds yet unconquered.

If The Alternate Source were directed to one particular group, I would suggest that it be the 'systems analyst' at each TRS-80 installation (though humble it may be!). In order to fully utilize a system, there must be at least one person who understands the basics of the system: CLOADing versus SYSTEM loads, subtle differences between Level II and Disk BASIC, a general familiarity with the various software components in memory at any given time and how they interrelate, or any number of a thousand other minor 'rules and regulations'. Before we can truly test the limits of any system, we must understand the rules -- then we can break them! That's where TAS plugs in.

We have yet to implement many of our initial goals (some are being worked out even as I write this!). For this reason, many have labeled us as a 'Z-80 Assembler' journal. We're not ashamed of this, by any means, but our collective vision encompasses so much more. As you view the articles in this collection, I hope you find as much valuable information as I have. You will at least experience the growth of a magazine.

Charles W. Butler,
Publisher

One word of caution: This compilation was made from original copies of early issues. Some of the advertisements may not be in effect, for various reasons. We suggest that you contact any company with an attractive ad for current availability.

A heartfelt thanks to all who have supported TAS for the past months. It is only with your help that we can continue to grow and serve.

May 1981

The Alternate Source is published by Charles W. Butler and Joni M. Kosloski at 1806 Ada Street, Lansing, Michigan, 48910. Subscriptions are $18.00 for twelve issues; $12.00 for six issues USA; $24.00 and $15.00 for Canada and First Class subscriptions; and $36 for twelve issues overseas ($48.00 for airmail). All monies must be in U.S. currency. Subscriptions may be addressed to the above, as well as any advertising rate inquiries. We actively solicit meaningful articles and information relating to Tandy's I and III, and compensate for each article published. Call or write for more information.

TWO DOLLARS VOLUME 1, NUMBER 1

THE ALTERNATE SOURCE

A COMPENDIUM OF INFORMATION TO ENHANCE THE USERS ABILITY TO EFFECTIVELY UTILIZE THE RADIO SHACK TRS-80 MICROCOMPUTER, LEVEL II AND ABOVE.

IN THIS ISSUE:

BOOTSTRAP

Welcome to the first issue of The Alternate Source News. Undoubtedly, no issue of any publication can be as complicated as the first. Everything has to be done! We've lucked into some pretty special people while trying to put this together. All of them have helped better this product for you.

Not the least of our problems was what format in which to present the information. We want to keep it consistent...too many publications start one way and then change, just about the time you get used to the way they do things. We decided on 5½ x 8½ for a couple of reasons:

1. With the price of paper rising every day, we want to hold our own against it for as long as possible. Our subscription prices reflect the most recent increases to our printer, and, barring some unexpected shortage of materials (we don't get paper from Iran, do we?) we can pretty much guarantee price stability for our first year of publication. Planned expansion is in the thickness department, instead of length by width.

We've made allowances on our binding margin for people who would like to punch the mag for ring binder storage.

2. While we would like to appeal to everyone, our main target area is for persons who have already climbed the hurdles necessary to be ranked 'beginner' and are ready to move into the 'intermediate' and 'advanced' categories. Several of the major publications, both exclusively TRS-80 and otherwise, have already proclaimed their support for the beginner. We feel there is a group of TRS-80 owners at least slightly more intelligent than some of the articles appearing give them credit for--thus we will project our magazine at this more intelligent creature. Our desire is to be an Alternate resource to persons who have exhausted the conventional info media. Depending on our timing, resources, and our ability to entice enough people to participate, this may prove to be quite a job--and expensive. By allocating less funds to the printer, we can compensate authors better, ideally attracting a higher quality of information for all.

3. Five and one-half by eight and one-half is working pretty good for TV Guide, Readers Digest, and IASFM. We feel they make pretty good company.

A special thanks goes to THE LANSING STAR, Mid-Michigan's ALTERNATE newspaper, for working out an arrangement for us to use their IBM Composer for typesetting. We had tried using the Electric Pencil, but it inadvertently jumped to Level II Basic once too often.

For your general information, any material in this publication not otherwise credited is the work of Charley Butler and Joni Kosloski with idea stimulation and valuable feedback provided by Dennis Kitsz and CMTUG members Allan Moluf, Bill Brown, Gordon Williams and Dan Poorman. We welcome your comments, criticisms, evaluations, complaints and information.

We've received some very good packages in our marketing department and have been very busy getting all the loose ends tied up--not to mention readying a magazine for publication! Our special thanks to the people who have responded to our requests for articles and information. We were scared that development of every issue would completely be an in-house thing. So far, this hasn't been the case. A quest for endurance is second only to quality.

HELP FOR THE SINGLE DRIVE OWNER

When Radio Shack released their 2.2 disk operating system last summer, it contained a couple of utility programs to allow single drive owners to convert their machine language files to DOS 2.2. Only once did I try to use the 'GETDISK/BAS' program. I was going to convert one of my Fortran modules to 2.2. Things went fine until 'GETDISK/BAS' told me I would need 22 ten minute cassettes to transfer this one module! The Microsoft Fortran package has four huge modules, not to mention several auxiliary routines! I didn't even bother to calculate the time necessary to convert any of my other machine language programs. I had neither the time nor the cassettes.

It seemed pretty logical that most people wouldn't have many machine language files larger than their RAM, so why not use memory to transfer the files?

Needless to say, it works like a charm...which is why you're reading it here! Some additional benefits: it can be used not only to transfer 2.1 machine language files to 2.2, but can also be used to transfer 2.2 (or greater) files from diskette to diskette. Just before submitting this article, I added routines that will allow you to loop and save the same file on more than one diskette.

Some cautions: this program is designed to be run with 2.2 or 2.3 DOS or possibly any DOS with 256 byte record lengths. To my knowledge, neither VTOS, NEWDOS, or 2.1 has this. Thus, make sure your operating system was booted from a 2.2 or 2.3 diskette.

Also, wait for the 'insert destination diskette' prompts before you change diskettes. This will make sure all files are closed properly. In case you haven't experienced it yet, improperly closed files will result in improperly updated directory tracks, which will in turn result in lost files--invariably they will be the ones you least expect to go, and most want to keep (Murphy's Law).

You certainly don't need to hurry. You can probably back up every 'system' file you have twice (!) while one cycle of 'GETDISK/BAS' is executing!

****

TRANSFER Program Listing

```
25    CLEAR15000:DEFSTRF,R:DEFINTI:CLS
50    INPUT"FILENAME TO BE MOVED";F:OPEN"R",1,F:DIMF2(LOF(1)*4)
75    FIELD1,255ASREC,1ASR1:I2=1:I3=4:I4=1:I9=LOF(1)
100   FORI=1TOI9:GET1,I:PRINT@640,"GETTING RECORD";I
125   FORI1=I2TOI3
150   F2(I1)=MID$(REC,I4,64):I4=I4+64
175   NEXTI1:F2(I3)=F2(I3)+R1
200   I2=I3+1:I3=I2+3:I4=1:NEXTI
225   CLOSE:INPUT"INSERT DESTINATION DISKETTE";I
250   INPUT"DESTINATION FILE (ENTER=SAME AS SOURCE FILE)";FD:
      IFFD=""FD=F:OPEN"R",1,FD
275   FIELD1,64ASR1,64ASR2,64ASR3,64ASR4:I1=1
300   FORI=1TOI9:PRINT@640,"WRITING RECORD NUMBER";I
325   LSETR1=F2(I1):LSETR2=F2(I1+1):LSETR3=F2(I1+2):LSETR4=F2(I1+3)
```

(continued on page 25)

*We're really proud to announce this issue that Dick Clope of the *80 Software Critique has supplied us with reviews of some game software. The *80 Software Critique is dedicated to supplying TRS-80 owners with factual reviews--each program is summarized, then rated on the following scales: Fun, Originality, Bugs, Instructions, Technique, and Dollar Value. Each issue also contains the top ten programs of the quarter, several comments to authors and users, fixes for some packages that contain bugs, as well as various other features. The current issue features mostly games, and the list of reviews for the second issue contains programs from several of the top software producers, including Automated Simulations, Hayden Books, Personal Computing, Instant Software, Acorn Software, DOG, Creative Computing, and several others. I'm looking forward to it!--ed.*

PORK BARREL

Program by Rev. George Blank for Level II, 16K
Available from TRS-80 Software Exchange for $9.95

This game is a political simulation. One to four players can play, and the object of the game is to keep your seat in the House of Representatives. You must continue to be re-elected in order to continue in the game. Your Congressional District consists of the following precincts with a total of 340,000 registered voters:

1. Professionals	10,000	2. Technical	30,000
3. White Collar	60,000	4. Defense	25,000
5. Industry	50,000	6. Service	50,000
7. Farmers	15,000	8. Retired	40,000
9. Unemployed	20,000	10. Welfare	40,000

You are also provided with the current unemployment rate--around 8%. Each government department brings its budget to Congress for approval. You are provided with the departments' budgets for the last year, and you must then decide how much money to provide for next year.

Several House Bills are then presented. For each Bill, you are given information on public opinion polls and how each special interest group feels about a given Bill. You then vote on each Bill and you are provided with the results of the roll call vote. On some Bills, the President will inform you that he expects you to vote his way on the Bill.

When election time comes, you must decide how to spend your campaign funds. Ten different advertising media are available for your funds. The results of the election are then presented. You can see which groups of your constituents were pleased with your performance and which groups weren't. If you were re-elected, you continue playing. This game came without instructions. They should have been provided.

Summary: This is an excellent game. The strategy would be better understood if instructions were included. The game accepts improper input on the roll call voting. We accidentally pushed the ENTER key several times without entering a command, and never were sure how we voted on that particular Bill.

MAKE DATA

(A way to merge your Machine Language and Basic programs! ed.)

CMTUG is blessed with one 'resident genius', Mr. Allan Moluf. Allan originally contributed the following program to CMTUG News. As with most resident geniuses, Allan's time is rather limited, yet when asked to contribute something, he usually manages to find a few table scraps--to us, they materialize as a feast! Having a 'resident genius' to ponder your immediate problem can have a more exhilarating effect than all the periodicals on the market. Speaking especially for The Alternate Source, and most probably for CMTUG, whether Allan is directly responsible or not, a substantial number of good ideas recently evolving in this area have felt and appreciated his guiding hand.

Due to the last minute urgency that frequently pops up in journalism, the following program originally had a couple of bugs. They have now been worked out and the following is a listing of the actual working program.

But What Does It Do?

Glad you asked. A fellow member of CMTUG had a particular situation where he wanted to merge a machine language routine with a Basic program. At the same time, he wanted to keep the operation simple. This program creates data statements from the machine language routine you want to merge, as well as a routine for reading them and poking them into the proper memory locations. The line number of the routine starts at 60000, which should be high enough for most Basic programs; if not, change the value of 'S' in the initialization routine.

If you want to use the routine by itself, you may want to throw in a line similar to: 59000 GOSUB60000:VARIABLE=USR(0):END

A word of advice--when you use a machine language routine which overlays the Basic program at some point, you may want to avoid potential problems by finding the beginning of free memory (PEEK 16633 and 16634). Most Basic programs start at 27171; the following program will print out the starting location of the machine language file. These addresses are for DOS.

With a little thought and a bunch of hard work, this program could be converted to Level II. A possible approach would be to load the machine language routine into memory and PEEK the memory locations it occupies, to get the 'data' to create the new program. Good luck.

Allan Moluf is also the author of the machine language DVR and COPYDISK programs offered by The Alternate Source.

****

Program Listing

```
0 REM MAKEDATA V1.22 -- 1979 OCT 18, 22:00
90 CLEAR1000:DEFINT A-R,T-Z:DEFSNG S
100 LINEINPUT"FILE NAME?";F$:OPEN"R",1,F$+"/CMD":
    LINEINPUT"DATA FILE?";D$:IFD$=""THEND$=F$
105 OPEN"O",2,D$+"/DAT"
110 FIELD1,255 AS O$
120 N=VARPTR(O$):Q=PEEK(N+1)+256*PEEK(N+2)
130 X=0:S=60000:S1=10:GOSUB3000
```

```
140 P=256:R=0:A=1
150 GOSUB1000:T=C:GOSUB1000:L=C:
   PRINT"REC";R;"  POS";P;"  TYPE";T;"  LEN";L;
160 IFT=2GOTO250
    ELSE IFT<>1THENPRINT"--IGNORED--":P=P+L:GOTO150
170 GOSUB1000:B=C:GOSUB1000:IFC > 128THENC=C-256
180 B=B+256*C:IFB<>A
    THENA=B:C= -1:GOSUB2000:C=B:GOSUB2000
190 PRINT"  ADDR";B: L=L-2:IFL<=0THENL=L+256
200 GOSUB1000:GOSUB2000:L=L-1
    IFA<>32767THENA=A+1ELSEA= -32768
210 IFL > 0GOTO200ELSEGOTO150
250 GOSUB1000:B=C:GOSUB1000:IFC > 128THENC=C-256
260 B=B+256*C:PRINT"  END";B
270 C= -2:GOSUB2000:C=B:GOSUB2000
280 IFX<>0THENPRINT#2, " "
290 CLOSE 2
300 END

1000 IFP > 255THENP=P-256:R=R+1:GET1,R:GOTO1000
1010 C=PEEK(Q+P):P=P+1:RETURN

2000 IFX=0THENX=10:
     PRINT#2, MID$(STR$(S),2);"  DATA";:S=S+10
     ELSEPRINT#2, " , ";
2005 IFC < 0THENPRINT#2," ";
2010 PRINT  2,STR$(C);:X=X-1
2020 IFX=0THENPRINT#2," "
2030 RETURN

3000 PRINT#2,MID$(STR$(S),2);"  READ B:  ON ERROR GOTO";
     S+6*S1:S=S+S1
3010 PRINT#2,MID$(STR$(S),2);"  IF B = -2 THEN READ A:
     DEFUSR0=A:RETURN":S=S+S1
3020 PRINT#2,MID$(STR$(S),2);"  IF B<>-1 THEN STOP":S=S+S1
3030 PRINT#2,MID$(STR$(S),2);"  READ A:  A=A-1":S=S+S1
3040 PRINT#2,MID$(STR$(S),2);"  READ B:  IF B < 0 GOTO";S-3*S1:S=S+S1
3050 PRINT#2,MID$(STR$(S),2);"  IF A> 32767 THEN A=A-65536:  GOTO"
     MID$(STR$(S),2);" ELSE A=A+1: POKE A,B: GOTO";S-S1:S=S+S1
3060 PRINT#2,MID$(STR$(S),2);"  IF ERR/2+1=23 AND ERL=";S-5*S1;
   " GOTO ";S+2*S1:S=S+S1
3070 PRINT#2,MID$(STR$(S),2);"  PRINT"CHR$(34);" ** UNEXPECTED
     ERROR, LINE=";CHR$(34);"  ERR/2+1;ERL:STOP":S=S+S1
3080 PRINT#2,MID$(STR$(S),2);"  POKE 16526, A AND 255: POKE 16527,
     (A/256) AND 255: RESUME";S+S1:S=S+S1
3090 PRINT#2,MID$(STR$(S),2);"  RETURN":S=S+S1:RETURN
```

A special note when running this program: Notice that the program as listed wants only the first eight (or less) characters and that it assumes the extension "/CMD". You may wish to modify lines 100 and 105. If you respond to the line-input WITH the extension, the extension will also be used to open the data file--thus messing up the EOF markers for the original /CMD file!

BULLETIN BOARD

Gadzooks! There are a number of people that have a need for various types of specially designed programs. Whether your business is programming, or has a need for programming, why don't you let us know? Perhaps we can match you up with someone who has complimentary needs or abilities. Address your info to TAS, Special Services Dept., 1806 Ada Street, Lansing, MI 48910. We would like to get responses from all over!

****

The Marfam Company has developed its own accounting and inventory packages based on the Hawley-Walz method of accounting which is fully interactive with the ledger (inventory works through receivable unit). Interested persons should contact the Marfam Company at 6351 Almaden Road in San Jose, CA. Zip is 95120.

****

Murname and Associates now have available three quick reference guides. For just $1.95 you can purchase either the TRS-80 Level I Basic Quick Reference Guide or the TRS-80 Level II Basic Quick Reference Guide. The third guide is Microcomputer Basic Quick Reference Guide, and costs 95 cents. The guides are small, well-organized, and contain information necessary to answer most questions programmers have (i.e., edit commands, Basic commands, ASCII codes, etc.). Write to Murname & Associates at 1056 Metro Circle, Palo Alto, CA. Zip is 94303.

****

We currently have requests for information pertaining to the following:

A job-costing program for a furniture & floor covering business

A payroll program that DOESN'T require you to write checks

A bug-free, rule-following cribbage game

A program that will calculate the right-justification codes for an IBM Composer

Any information that will assist someone in making the Small System Software RS232 interface work with non-SSS CPM and a Datel printer--needed desperately!

****

The Alternate Source will list your specific wants and needs on the Bulletin Board for a $2.00 handling charge. New product listings are free, but, when feasible, we do require a sample of the product (will be returned if requested).

FROM THE SOURCE'S MOUTH

The whole concept behind The Alternate Source originated with us in October, 1979. Between that time and now, quite a few interesting things have transpired.

In mid-November we went to the printer and ordered 2500 copies of our freshly typeset promo letter. Only AFTER we got them back did we discover a glaring syntax error in the actual letter portion. We were trying to relate that we wanted to share IDEAS with you; instead it read as if we wanted to share hassles! No way. Our goal is to eliminate hassles. Everybody around this place is walking around with one arm about two inches shorter than the other. Remember when the teacher had you write 1000 times, 'I will not chew gum in class'? It was worse, trying to correct those 2500 copies! Oh, well. Let's talk about something more pleasant...

Those of you who didn't receive our promo letter probably saw our flyer in the TCS Newsletter. Between now and March, our ads will be appearing in at least five magazines, three of them exclusively TRS-80 publications. We've been trying to keep our advertising budget to a minimum in order to hold costs down, but we have to make ourselves known somehow. Our hats are off to the folks at TCS who have helped us both ways!

Moving on--I recently had the pleasure of visiting a gentleman here in town; he's the proud owner of an Exidy Sorcerer. I don't blame him for being proud--he's got it trained so that when he loads a particular program the beast comes back with "MEMORY SIZE?". After a carriage return, "RADIO SHACK LEVEL II BASIC". He's working on DOS, I presume.

The highlight of the evening was a call to California. We had a modem hooked up to the RS232 and the Exidy. We were on line with the TRS-80 Users Group in Orange County for thirteen minutes, and filled a good portion of the Exidy's 52K of memory with information from their net. We figured the cost of the call to be about $2.12, better than the price for some newsletters! Needless to say, I'm real excited about The Alternate Source going on line--just think! You'll be able to call and get your software packages immediately! Author submissions will be more fun, too!

Allan Moluf, resident genius of CMTUG, currently has a Dumb Terminal program. I wouldn't dream of mentioning his name, but one Radio Shack store manager has said they prefer Allan's program to the one offered by Radio Shack! Allan tells me it isn't yet exactly the way he wants it, but it should be by the time you read this!

With on line networks only about six months old, I can hardly wait until a year from now!

Would you be interested in a part-time job? Great hours (you set your own!) and only slightly below mediocre pay! We're interested in a wide variety of articles and programs, both short and long. To name a few: Z-80 techniques and programs, routines that can be interfaced with a wide variety of programs, reviews of good quality, low cost software packs (unfortunately, they don't all meet this criteria!), interesting applications for which you've applied your computer, and even games! We like to feature at least one article per month that even the inexperienced

TRS-80 owner can use. There is no way we'll ever be an all games magazine, though. We believe that ultimately most TRS-80 users are looking for more than that.

And about Model II? Some recent conversations about this new configuration have been most enlightening. We understand the index hole makes it incompatible with any other eight inch drive system. Also, Radio Shack is using the lower 3K of memory for system functions, device control blocks and the like. I'm still confused about whether or not the machine has any type of peek or poke function. A recent issue of CMTUG published a small article showing how to simulate PEEK and POKE. A more recent conversation with an outlying Radio Shack store manager led me to believe the function is there, although with a different key word. Anyone interested in making contact with a Model II Users Group might do well to write to National TRS-80 Model II Users Group of Madison, c/o Peter Daly, 430 Jean Street, Madison, WI, 53703. Send SASE for first newsletter. (This information appeared in On_Line, November 23, 1979, Volume 4, Issue 14.)

A special reminder for those who are contemplating submitting software, either for publication or distribution: For publication, we offer compensation on all articles published. Also, we will notify you of acceptance or rejection within three weeks--guaranteed if you enclose a postage paid return envelope! For distribution, it takes a bit longer to get the appropriate news releases out, ads prepared, ad spots purchased and duplication set up--but we definitely will respond quick. We also advise you to compare percentages given to authors!

****

LETTERS

Charley/Joni:

Thanks for the preview; here's some comments on your new magazine. It sure looks ambitious, and chock full of good things. (How's that for general?) The very first thing I see that I don't like, though, is the emphasis on disk, at least from the viewpoint of DOS. The point of TRSDOS, NEWDOS, and the rest, are the ability to change and to rebuild. Basing a large portion of your magazine on a product with so few users (only 20% of TRS-80 owners, last figures I read) is to me questionable. Okay, here's more of why: I want a disk, but what I want it to do rules out DOS. It takes too much space, and I'm not about to buy a second drive merely to accommodate all the space needed. What I want is mass storage, not another layer of high-level goop. But then, I don't believe that viewpoint is representative, either.

Let me recommend avoiding bulk mailing. When you say "hold things up a bit", let me tell you about Vermont (you've heard of it? Postal Service hasn't much.). We get good first class service on some things; On_Line comes a week after responses to my ads start to arrive, and that's mailed first class. My Byte never before the month it's dated, and some of the bulk stuff that doesn't look critical to the postal people arrives VERY late...announcements of performances sent bulk usually arrive after the performance, etc. Perhaps you could offer the option of first class mail, anyway.

Dennis Kitsz
Roxbury, VT

Believe me, Dennis, we're not ignoring Level II, nor do we intentionally place an emphasis on DOS--it's just that most of our support staff works with DOS. We are actively seeking Level II articles, and will give special consideration to articles that coincide with both systems. And you're probably right about the first class mailing option; however, this is where we stand--postage is our second biggest expense, next to advertising. So this is what we'll do: If you, or any other subscribers, would like to receive The Alternate Source via first class mail, all we ask is that you send us an additional $1.00 to cover postage and handling.--Kos

Dear Sirs:

My primary reason for subscribing is interest in the "annotated documentation of disassembled Level II and DOS"--mainly DOS. Please let me know how to order just as soon as any such information is available.

Andrew Law
Dallas, TX

The Alternate Source will not be selling any disassembled listings, Andrew, but we will be covering the subject in our magazine. In the next issue, there will be at least four pages depicting RAM memory pointers and entry points. Many of these are applicable to both Level II and DOS. There will also be several articles in following issues dealing with how to utilize these memory locations. Thanks for the feedback!!

Charley and Joni:

On reviewing the material you sent me, I notice that you mention "software" and "hardware" types. I would like to mention that there is a growing contingent of "user" types. To some extent, these will be software types, but not in the sense of doing their own programming. They will be interested in your software prices, distribution, reviews of products, etc. There may well be market, or one to be developed, that addresses the issue of what to do with the machine once the game playing gets old, or how to effectively integrate bits and pieces of software into a system that assists one in doing complex things with the computer, or just how to integrate the computer into one's life-style in such a way as to thoughtfully make it (the computer) part of it (the life-style), rather than an add-on gadget.

I encourage you to find ways of introducing people to FORTH, and to give it utility and peripheral software support. I think this or something like it is going to have a significant place in the software field, once people figure out how to make use of it. I recognize that the low level of its current use will probably not warrant your giving it that much attention for a while, but I can see its potential. Again, I urge you to keep it in mind.

Bill Brown
Haslett, MI

Editor's Note: There is no doubt that Bill's influence is reflected in many of the finer points of this magazine--his feedback has been very valuable!!

EMBED MACHINE LANGUAGE IN BASIC

Dennis Kitsz

And simple, too. It's an idea that has been around for a while, but certainly bears repeating because it is convenient and efficient. It isn't very transparent (in fact, once it's in a program it is most definitely obscure), but for completed programs with class, it does the job. Here's how and why:

In the TRS-80, all variables can be located with a wonderful Level II command known as VARPTR. Probably only a few of you have ever used it, so here is some of what VARPTR will find for you. Assume A$ is a string variable in a program. The statement

X = VARPTR(A$)

assigns the address of the string's length to X...so, PEEK(X) is the length of A$. But PEEK(X+1) and PEEK(X+2) are really the important things for us. PEEK(X+1) is the least significant byte of the string's starting address, PEEK(X+2) is the most significant byte. These values are in decimal, of course, because Level II doesn't deal in hex. So, convert them to a complete decimal address value with this formula:

AD = PEEK(X+1) + 256 * PEEK(X+2)

Do you see what is happening here? And what can be done with it? If we knew, for example, that A$="XXXXXXXXXX", then we can change A$ by POKEing values into the addresses we have calculated! Using the variable AD from the formula above, POKE AD,65 will change the string to "AXXXXXXXXXX". Going furthur, the statement POKE AD+9,191 will leave a graphics block in the final position of the string. Here's the point: you can create a dummy string containing the same number of bytes, putting it in your BASIC program. Find out where the string is located with VARPTR, then POKE a series of DATA statements in place of the string! The string will now list or print as a meaningless collection of characters, tabs, carriage returns, etc., but it will actually be a machine language program which BASIC thinks is an ordinary string variable. What are the benefits of this? And how ultimately do you use the thing? The benefits are simple: once the data is POKEd in place, the DATA statements can be dropped from the program, and the string remains intact (and remember, as the program is edited, this string is now moved around memory just like any other string), thus, it can save 75% of the space needed for all those decimal DATA lines--plus there are no worries about messing up with a wayward RESTORE in the program. Next, you don't need to set MEMORY SIZE, because it is now an integral "part" of the BASIC program. So that leaves us with the question, how do you get to it? Here VARPTR is again the answer. Addresses (decimal) 16526 and 16527 contain the USR(0) function in Level II Basic. POKE 16526 with PEEK(X+2). No matter how the program is changed or edited, X has been defined in terms of VARPTR(A$), and will always place the correct address into USR(0)! Again, it's slick. Only one thing to note: all the machine language programs must be relocatable, because the simple editing of one character surrounding BASIC program will alter the absolute memory position of the string variable holding the machine routine. And you know that if you release the program commercially, someone will instantly modify it--and oh, the complaints! Now here's a summary:

1. Write the Basic program.

2. Create a dummy string of any unused variable name (example A$)
3. Make the string the exact length of your machine language routine (example A$ = "LOON")
4. Write a program line that sets a variable to VARPTR of the string (example X = VARPTR(A$))
5. Find the starting address of the string by converting the decimal bytes to a single decimal value (example AD = PEEK(X+1) + 256 * PEEK(X+2))
6. Create a (temporary) set of READ and DATA lines in your program which will POKE the machine language from program into place (example FORN=AD TO AD+3: READ L: POKE N,L:NEXT: DATA 33,16,16,204 will put this short and relatively meaningless routine in the place of "LOON": LD HL,1010 RET)
7. Set the USR(0) starting addresses to PEEK(X+1) and PEEK(X+2)
8. RUN the program, which will do the POKEing
9. Delete the READ, DATA, and POKE loop lines used for the routine

One final note: take care when putting a zero into the POKEd string changes. The string will seem to have been truncated when listing or printing, and modifications to program will cause havoc. Avoid a zero if you possibly can, especially if the program is going out to the tampering public.

And that's all there is to it!

****

CURSOR CONTROL

I recently discovered that the capability for manipulating the on-screen cursor for the Model I is better than I thought.

The situation I encountered was this: I was designing a program that allowed a variable amount of data input from several different screen locations. After data was checked for validity, if the information was okay, the next piece of data could be entered. If not, a branch was made to a subroutine where an error message was flashed on another part of the screen. My problem arose while trying to get back to the part of the screen where the input originally began, to allow it to be re-input.

Some other considerations which complicated matters: at all times, a heading was to be displayed at the top of the screen so the user would know which portion of the program he was in, thus I couldn't use 'CLS'. Also, 'PRINT@' was out because of the varied input locations. The data was being input into an array using a FOR-NEXT loop. The PRINT@ would have caused too much additional coding.

The solution to this problem lay in the VIDEO(DO) control block whose starting location is at 401DH, 16413D. The current cursor location is stored in locations 4020 and 4021H. Although the problem below is greatly simplified, it shows one solution:

```
500   FOR I = 1 TO 10
510   ? "VARIOUS STRING LENGTHS TO MESS UP MY NEAT
      INPUT!!"
520   X = (PEEK(16416) + 256 * PEEK(16417)) - 15360
530   LINEINPUT I$(I)
```

(continued on page 25)

VISA & MASTER CHARGE WELCOME! 3C PAYS SHIPPING!

3C Contemporary Communications Corporation
P.O. Box 1707, East Lansing, MI 48823
24 hour phone orders, (517) 339-1028 3C

MODEL I SOFTWARE
(10% off on orders over $40.00!)

NEWDOS *	$49.95
NEWDOS+ *	$99.00
Infinite Basic	$49.95
Infinite Business Basic @	$29.95
Accounts Receivable *	$79.95
On-Line Invoice * $	$39.95
Remodel †	$24.95
Remodel + Proload †	$34.95
Copsys	$14.95
Comproc	$19.95
GSF †	$24.95
DOSORT *†	$34.95
Radex-10 Data Base *	$99.00
Payroll *	$59.95
Relax 48K *	$49.95
Mail-80 48K & GSF *	$69.90

MODEL II SOFTWARE!!

PACKAGE 1: Editor Assembler
Disassembler
Superzap (Mach. Lang!)
***
All three programs, just $100.00!

PACKAGE 2: GSF includes
PEEK & POKE!
***
Only $50.00!

FOR MODEL II SOFTWARE ONLY:
TO ORDER: Include 8" diskette with TRSDOS & proof of purchase of NEWDOS and Radio Shack EDTASM (paid receipt, page of manual, etc.--will be returned). Write for additional details.

*Requires Disk †Specify RAM size @Requires Infinite Basic $Requires Accts Rec.

UTILITIES NOW AVAILABLE FROM THE ALTERNATE SOURCE:

ISAR--*Information Storage And Retrieval is THE Information Management System for micros! Whether your needs are for hobby or business, mailing lists or formatted reports, nothing on the market beats ISAR's speed (uses random file structures created with easy user prompts) and price: just $16.95 on disk with documentation.*

DVR--*machine language Driver with special commands allows you to input lower case shifted upper case or echo screen output on printer. Provides keybounce fix and repeating key function and other extras! Sorry, not compatible with NEWDOS at this time. On cassette, just $9.95. Specify Level II or DOS and Memory size.*

DISKLIB--*uses a special machine language routine to read all your diskette names and directory entries and creates a master file of same. Screen or formatted hardcopy output. Works with TRSDOS, NEWDOS, and VTOS. On cassette, $9.95.*

COPYDISK--*a machine language utility which allows disk owners to copy ANY and several files using easy user prompts! Allows single drive owners to access and copy formatted only diskettes or multiple drive owners to copy files without system diskette. On diskette, $15.95 with documentation.*

TO ORDER: Address all software orders to TAS, 1806 Ada Street, Lansing, MI. Zip is 48910. Please include 50 cents per program to help defray postage costs.

THE ALTERNATE SURVEY

I, personally, have been asked to fill out no less than a handful of surveys this past year. Some I didn't mind, others I didn't fill out. The reasons given for the survey, either explicitly or implied, were often the decisive factors.

We don't particularly want to include a survey; they're a hassle. Yet, we do want to edit and publish a magazine that offers a service to its readers, either in the form of providing instruction, a market place for software and supplies, resource material, whatever. It's the 'whatever' that keeps us concerned.

The reasons I can give you for filling out this survey are thus:

1. Our circulation is relatively small, and not everyone is going to respond. The input you provide at this time could have a strong influence on the development of future issues.

2. We shall choose, in the usual random fashion, three survey participants and reward them with their choice of a year's subscription (or extension) of The Alternate Source News, or a $10.00 certificate good on any software pack we offer.

Alas, at this early stage we aren't offering post paid mailers, but they are a definite possibility in the future.

If you do choose to participate, you are by no means limited to the space allocated. If you choose not to tear your issue, answers may be supplied on a plain sheet of paper. Just copy the number and corresponding answer--no need to rewrite the question.

Mail all responses to The Alternate Survey, 1806 Ada Street, Lansing, MI, 48910.

****

1. Are you a subscriber?

2. Did any one article have a peculiar, repugnant odor?
   If yes, which one?

3. Which article did you find most favorable?

4. Was there something you wanted or expected to see, and didn't?
   If yes, what?

5. What suggestions or comments would you have that could help make The Alternate Source more valuable to you?

6. Was there an over-emphasis or under-emphasis on any one area?
   If yes, which one?

7. What is your TRS-80 hardware configuration (how many printers, disk drives, stringy floppies, how much memory, etc.)?

MORE SOFTWARE FROM THE ALTERNATE SOURCE!!!

ADVENTURE SERIES--*featuring ADVENTURE 8, PYRAMID OF DOOM! Adventureland, Pirates Adventure, Mission Impossible, Voodoo Castle, The Count, Space Odyssey and Mystery Fun House also available! On cassette, $14.95 each; on diskette as follows: 1 & 2 for $24.95; 3, 4 & 5 for $39.95; 6 & 7 for $24.95 and 8 for just $19.95.*

TSHORT--*the machine language utility that loads into low memory and allow you to key in programs in a minimum of time and with a minimum of mistakes. DOS version and Level II version on same tape. $9.95*

GSF--*the machine language utility package that features possibly the best sort routine available for the TRS-80, as well as many other machine language functions. Specify DOS or Level II and memory size. $24.95*

TEMPLE OF APSHAI--*For the experienced Dunjonmaster or the novice. Everyone should have at least one Dungeons & Dragons type game. Definitely not the kind you master in ten minutes. $24.95*

MICROSKETCH III--*The graphics package has been updated. This is the last time they will be available at this price. We also have support utilities for Microsketch. Write for information. Microsketch III (until January 30th) is just $3.95.*

SARGON II--*Never has a micro-chess game been closer to playing with the finesse of a master. Sargon II is the best chess game available for the TRS-80 at this time. Only $29.95.*

THIS COUPON MAY BE DUPLICATED, OR YOU MAY WRITE YOUR ORDER ON AN ADDITIONAL SHEET OF PAPER OR YOU MAY USE THIS COUPON. ORDER NOW--IT'S AS EASY AS FILLING OUT A TAX FORM!!

ADVENTURE *on cassette:*

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5
( ) 6 ( ) 7 ( ) 8

ADVENTURE *on diskette:*

( ) 1 & 2 ( ) 3, 4 & 5 ( ) 6 & 7
( ) 8

TSHORT ( )

GENERALIZED SUBROUTINE FACILITY ( )

TEMPLE OF APSHAI ( )

MICROSKETCH III ( )

SARGON II ( )

Total your order using prices from ads above, and enter the amount here:

____

Multiply the above amount by .10 (10%) and enter the amount in this blank:

____

Subtract the second amount from the first amount, and enter the amount in the blank here:

____
(Amount Due)

Write check or get money order for Amount Due, and enclose with coupon.

MAIL TO: The Alternate Source, 1806 Ada Street, Lansing, MI 48910.
VISA & MASTER CHARGE ORDERS BY PHONE: (517) 487-3358

IS THIS YOUR FIRST ISSUE
OF

THE ALTERNATE SOURCE?

Or, heaven forbid, someone else's? We hope there were at least a few things you enjoyed. Also, may we point out a few things? Be sure to check out 'In The Buffer'--a list of up and coming features. Look over our growing selection of software--and not just games, either! We're looking forward to making your TRS-80 the valuable tool you always knew it could be!

Our rates are not the cheapest--but they're far from the highest! $9.00 will cover you for a year (6 issues), or you may sample another issue for just $2.00.

We won't ask you to clip the coupon from this page; if you'd like, just write the information on a sheet of paper. Include your name and address, and whether you're subscribing or sampling. We'll guarantee you'll be pleased, or you may receive a refund on unused subscription monies at any time!

Please, send check or money order only. Yes, we have received a couple cash orders; we don't know how many we didn't receive. Mail to: The Alternate Source, 1806 Ada Street, Lansing, MI 48910. Phone (517) 487-3358 for Visa or Mastercharge.

SUBSCRIPTION ORDER

____
(Name)

____
(Address)

____
(City)

____
(State, Zip)

I have enclosed $9.00 check or money order.

( ) I would just like a sample issue. Enclosed is $2.00.

ADVANCING YOUR KNOWLEDGE

How many times have you seen a reference in a magazine article to 'Searching and Sorting Algorhythms' by Donald Knuth or one of his other classic books? We've seen several, and our curiosity got the best of us. We ordered a limited supply to have on hand for persons who are interested in considerably advancing their general computer knowledge. There are three books in the series, and each volume retails for $22.50 plus postage. For a limited time, we will make these volumes available to our readers for $20 each plus $1.50 per volume postage. If you order all three, we will pay shipping. Recommended only for the serious--not light reading! Mail orders to Book Dept.

DOCUMENT

Document is a unique way for you to relay messages to someone, either in DOS or Level II, using your computer.

Probably your most frequent use will be to pass along instructions or remarks in the form of a disk file or system tape file, which would precede another program. In DOS, you can 'AUTO FILENAME' and instruct the recipient to boot from your diskette. With Level II, you will have to clue them that the first program is a system program along with the system name.

Implementation is easy. You merely enter the following source code, replacing the DEFM literals with the message you want to pass along.

0FFH is the control byte for 'END OF PAGE'. Every time the program encounters this, it waits for any key to be depressed. If the next byte is 00H, the program returns control to DOS or Level II, otherwise it clears the screen & prints more DEFM statements until it encounters another 0FFH byte. Make sure you use a 'DEFB 0FFH' after every screen you wish to display, and a DEFB 00H after the last DEFB 0FFH.

****

DOCUMENT Program Listing

```
7A00           00100            ORG     7A00H
7A00 21247A    00200  BEGIN     LD      HL,TEXT
7A03 CDC901    00300  NEXT      CALL    CLS
7A06 3EFF      00400            LD      A,0FFH
7A08 11003C    00500            LD      DE,SCREEN
7A0B 010004    00600            LD      BC,PAGE
7A0E EDA0      00700  CONT      LDI
7A10 BE        00800            CP      (HL)
7A11 20FB      00900            JR      NZ,CONT
7A13 2803      01000            JR      Z,INKEY
7A15 EA0E7A    01100            JP      PE,CONT
7A18 CD4900    01200  INKEY     CALL    KEYIN
7A1B 23        01300            INC     HL
7A1C 3E00      01400            LD      A,00H
7A1E BE        01500            CP      (HL)
7A1F 2802      01600            JR      Z,RETURN
7A21 18E0      01700            JR      NEXT
7A23 C9        01800  RETURN    RET
3C00           01900  SCREEN    EQU     3C00H
0049           02000  KEYIN     EQU     0049H; ROM-LIKE INKEY$
01C9           02100  CLS       EQU     01C9H
0400           02200  PAGE      EQU     400H
7A24           02300  TEXT      EQU     $
7A24 41        02400            DEFM    'Any text placed in these'
7A3C 20        02500            DEFM    'DEFM statements . . . .'
7A51 FF        02600            DEFB    0FFH
7A52 2E        02700            DEFM    '. . . . .Will be displayed on '
7A70 53        02800            DEFM    'the screen when RUN!'
7A90 FF        02900            DEFB    0FFH
7A91 00        03000            DEFB    00H
7A00           03100            END     BEGIN
```

(Undoubtedly, there are some instructions in Basic or any other language that are used more than others. The general reason is lack of documentation and specific examples of some of the more complex instructions, thus many people don't understand the internal results of these complex instructions. Here we hope to fill the gap in documentation for the TRS-80 AND and OR instructions and show a potentially new application for them. This info will be good for developing assembly language programs, too! Kos)

BYTE PACKER

C. W. Simpson

One of the definite problems with the Model I TRS-80 is on-line capacity to store large amounts of data, an important consideration in business and other application programming. For persons using tape I/O, time considerations make data storage intolerable. Diskette I/O is more tolerable, but space allocations quickly become frustrating. Let's see if we can't overcome these inherent deficiencies somewhat. We'll begin with a quick review of what you should already know, and then show you how to use it.

There are techniques that assembly language programmers have always used--they were taught to from the beginning! Up to now, most of us Radio Shack folk have been so busy just learning to program that we haven't had much time for 'technique'. Let's see if we can't change that trend. Now pay attention!

First, we need to make sure you understand logical 'AND' and 'OR' functions. We'll start with the 'OR' function, and we're going to 'OR' two binary numbers (base two). When we 'OR' two numbers, we're looking for a '1' in any column--if we find AT LEAST one (we could find two), then the result is a '1':

```
     0110              1010
OR:  1100         OR:  0011
     ----              ----
     1110              1011
```

Using the 'AND' function, we're looking for a column with TWO 1's. If we find two, the result is a '1':

```
     0110              1010
AND: 1100         AND: 0011
     ----              ----
     0100              0010
```

As you are undoubtedly aware, the computer thinks in binary, or the 'on' and 'off' (1 and 0) status of thousands of tiny 'switches'. This is the true machine language we hear so much about. For our immediate project, we need to think in bytes. It just so happens that a byte contains 8 bits. A bit is best thought of as one of the 'on-off' switches just mentioned, or, when talking about binary, a one or a zero.

Now, some background on general trends in application programming, and we can begin.

Usually, in business data processing, there are several 'fields' that must be stored pertaining to each entry. For example, let's look at an employee record. There is probably a marital status field, a field to indicate whether he is salaried or

hourly, a field to indicate number of dependents, and any number of other items. As a general rule, these fields are stored in one byte each, and (if it's an integer) sometimes two.

Is there any way to compress this data to make storage more efficient? Of course there is!

Let's all pretend for a few minutes that we work for XYZ Realty. Let's also assume that we get ten or more calls a day from people inquiring about a particular type of house. In all probability, our filing system doesn't permit us to retain enough information to immediately match up a particular house with a particular caller. Because of this, most inquiries are probably lost...we just don't have an effective way of accessing this data. How about taking our TRS-80 to the office and setting up a data base program to improve our efficiency? (And make our TRS-80 deductible, too!)

Naturally, everyone wants different 'options' in their dream house. A basement here, a garage there, a fireplace is mandatory for one guy, while nearness to schools is essential for this family. We'll assume there are eight options for our example: basement, garage, large yard, nearness to schools, city or suburbs, fireplace, central air, and a built-in pool. Except for option 5, all desired options can be answered 'Y' or 'N'. We could field each option as one byte each, but why don't we try something different? Study the following code:

Beginning of Program...

```
010 DEFINT C
.
.
.
200 REM ** DETERMINE AND STORE SELLING POINTS **
210 CLS:CHOICE=0:BASEMENT=128:GARAGE=64:YARD=32:
    CITY=16:SCHOOLS=8:FIREPLACE=4:AIR=2:POOL=1
220 PRINT@576,"WOULD CUSTOMER LIKE...
230 PRINT@640,"A BASEMENT?"CHR$(31);:GOSUB1000
240 IFQ$="Y"CHOICE=(CHOICE OR BASEMENT)
250 PRINT@640,"A GARAGE?"CHR$(31);:GOSUB1000
260 IFQ$="Y"CHOICE=(CHOICE OR GARAGE)
270 PRINT@640,"A LARGE YARD?"CHR$(31);:GOSUB1000
280 IFQ$="Y"CHOICE=(CHOICE OR YARD)
290 PRINT@640,"HOUSE IN CITY?"CHR$(31);:GOSUB1000
300 IFQ$="Y"CHOICE=(CHOICE OR CITY)
310 PRINT@640,"CLOSE TO SCHOOLS?"CHR$(31);:GOSUB1000
320 IFQ$="Y"CHOICE=(CHOICE OR SCHOOLS)
330 PRINT@640,"FIREPLACE?"CHR$(31);:GOSUB1000
340 IFQ$="Y"CHOICE=(CHOICE OR FIREPLACE)
350 PRINT@640,"AIR CONDITIONING?"CHR$(31);:GOSUB1000
360 IFQ$="Y"CHOICE=(CHOICE OR AIR)
370 PRINT@640,"SWIMMING POOL?"CHR$(31);:GOSUB1000
380 IFQ$="Y"CHOICE=(CHOICE OR POOL)
.
.
.
999 END
```

```
1000 REM **UTILITY SUBROUTINES START HERE**
1010 Q$=INKEY$:IFQ$=""THEN1010ELSEPRINTQ$:RETURN
```

Please note: In order for this routine to work properly, the 'OR' and it's arguments must be enclosed in parenthesis!

To see what's happening, let's look at a sample customer, one that's easy to please. His only requirements are a large yard (32) and nearness to schools (8):

First, 'CHOICE' has been set to zero:	00000000
We 'OR' it with 32 (YARD):	00100000
Now 'CHOICE' equals:	00100000
We 'OR' our new 'CHOICE' with 8 (SCHOOLS):	00001000
And now 'CHOICE' equals:	00101000

Assuming you've typed in the preceding code, and RUN it (selecting the above options, as in our example), if you 'PRINT CHOICE', your answer should be 40, which is 00101000 in binary, or a '1' in both the yard and schools columns.

Now let's put the value in the customer's record. At this point, it is really a two-byte integer. You did a 'DEFINT C' in line 10, so let's convert it to one byte by 'fielding' C$ as '1', then LSET C$=STR$(CHOICE). Now 'PRINT C$'. Your answer this time should be @, the ASCII value for 40.

Assume that now we have all the vital information for the customer, and it's fielded properly, etc. and we've written the record to disk, and closed the file. (Important!!)

Our program has another subroutine called 'FIND HOUSE'. We still have our customer's information in memory to fit his criteria. Let's see if we currently have a house available that he might be interested in. (When we brought our system to the office a few weeks back, we added all of our available houses for sale and have kept it up to date through today.) We add our 'houses available' diskette in response to the prompt, and go house-hunting by computer!

When we added the houses, we had a subroutine very similar to the 'determine and store selling points' routine above. With some careful programming, it could be the same one, substituting 'DOES HOUSE HAVE' for 'WOULD CUSTOMER LIKE'...possibly with data statements.

When we set up the data base for the houses, we incorporated a field named OPTION$ to store the features of each house. We're now ready to scan the 'house' file and make a comparison using the 'AND' function we learned earlier. All routines not shown are commented to assist your understanding. Pay special attention to line 550!

```
500 'FILE HOUSEKEEPING IS DONE -- FIND A HOUSE
510 FOR REC=2 TO LSTREC
520 PR%=INT(REC-1)/30:SR%=REC-30*(IP-1)
    'FOR LEVEL II YOU WOULD SIMPLY INPUT THE VARIABLES
     DESCRIBING EACH HOUSE AND EXECUTE 550 ON. PR%=PHYSICAL
    RECORD NUMBER, SR%=SUB-RECORD NUMBER. SEE RADIO SHACK
     MANUAL ON RANDOM FILE STRUCTURES FOR MORE DETAILS.
    ALSO, WE'RE ASSUMING THAT ALL DATA FOR A HOUSE CAN BE
```

```
     STORED IN 30 BYTES -- WE'VE DONE A WHOLE BUNCH
     OF BYTE PACKING ELSEWHERE!
530 FIELD1, (SR% - 1)*30 AS DUMMY$, 28 AS OTHERDATA$,
     2 AS OPTION$: 'THIS AND THE LAST STATEMENT WOULD
     PROBABLY BE SUBROUTINES IN A REAL APPLICATION TO
     PERMIT ACCESS FROM VARIOUS POINTS IN THE PROGRAM
540 OPTION=CVI(OPTION$):'IN LEVEL II YOU WOULD SIMPLY INPUT
     AN INTEGER VARIABLE. CVI MEANS CONVERT THE STRING
     PARAMETER TO AN INTEGER
550 IF(CHOICE AND OPTION) < CHOICETHEN570:'THIS HOUSE
     DOES NOT FIT THEIR REQUIREMENTS!
560 GOSUB1300:PRINT"CONTINUE?":GOSUB1000:IFQ$="N"THEN580:
     'THIS LINE IS EXECUTED ONLY IF THE HOUSE HAS THE
     OPTIONS THE CUSTOMER DESIRES. SUBROUTINE 1300
     WOULD PERMIT A HARD-COPY DETAILED HOUSE DESCRIP-
     TION. AS EACH MATCH IS MADE, THERE IS AN OPTION
     TO EXIT FROM THE PROGRAM (TEST FOR "N").
570 NEXT REC:'GO LOOK FOR SOME MORE HOUSES
580 CLOSE:RETURN
```

You should be able to see what's happening. Taking our original new customer and his desired options (00101000) and the options available for each house, we can 'AND' the two fields and hopefully come up with a match. By 'AND'ing the house description field and the option field, any house not having the required options would be less than the value of the option field.

Assume a particular house has a basement (128), garage (64), a large yard (32) and is close to schools (8). The value of this house would be 232, or 11101000 binary. If we have accessed this house from our data base, and set all the variables pertaining to it, especially the variable 'OPTION', line 550 would be doing something like this:

```
CHOICE = 00101000
OPTION = 11101000
        ---------
RESULT = 00101000
```

Here, the result is not less than choice. Thus, we have found a house that fits the minimum criteria!

I will admit that this type of programming takes much more coding, but the efficiency of storage more than offsets that detriment.

Our purpose here is to provide you with a routine that can be used in several programs, rather than an actual program that's typed in and forgotten.

Anyone who has gotten bored while waiting for tape I/O, or frustrated with 'DISK FULL' messages should give this technique a try. Why store one piece of information in one or two bytes when you can store several? If you are using integers, you could store up to fifteen different pieces of information in one integer variable by assigning codes to the higher powers of two (256, 512, 1024, 2048, 4096, 8192, and 16384). Some really wild and crazy programmer could pack a life history into a single or double precision variable using these techniques! You aren't restricted to yes or no data, either. For example, you might AND or OR any four bits with a number representative of 'NUMBER OF KIDS'. That would handle up to 15; if

you feel any doubts about that, use five bits to handle up to 32. The point is, why do it with 16 bits, when you can get away with using just four? You have, at best, 445,000 bits on a diskette. Next time you get a 'DISK FULL' message, ask yourself where they went! By using this and similar techniques, you could save as much as 25%! If you don't think that's significant, just ask somebody who's using the TRS-80 for any major application. For cassette owners, you tell me if one quarter less I/O time would be helpful!

Can you think of any applications where this might be useful? The Alternate Source will award a $30.00 software package (your choice from over $400.00 worth of great programs!) plus regular remuneration to the programmer who can best demonstrate the above techniques. Deadline: April 31, 1980.

****

A DOUBLE DEAL FOR DISK DRIVE OWNERS!!!

Directories starting to overflow?
Let us help!!

Deal Number One:

A box of Verbatim diskettes, (ten, count 'em!) the top quality kind you've come to know and love! No dirty deals! Diskettes come in soft box with shucks and no surprises! A mere pittance at $26.50 plus $1.00 for postage!

Deal Number Two:

(A must for anyone who uses diskettes for unusual purposes; swatting flies, etc.). TWENTY of the above type diskettes for 2*26.50. That's (carry the one, put out the cat) $53.00 (only). No postage!

No additional discounts of any kind ap- to this offer! (Shucks!!)

Order today by mail or phone. Make checks payable to "A Double Deal for Disk Drive Owners" or "The Alternate Source" (your choice). Shipped promptly! Mail to: 1806 Ada Street, Lansing MI 48910. Phone (517) 487-3358.

CHEAPER BY THE DOZEN!

* A Baker's Dozen, no less! *

The Alternate Source has top quality cassettes at a price you can live with--not twelve, but thirteen (an even Baker's dozen) C-10 cassettes with plastic covers for $10.95 plus $1.50 postage. Stock up for life--order TWO dozen (26 cassettes) and we'll absorb the postage costs!

Order today by mail or phone. Mail to 1806 Ada Street, Lansing, MI 48910, or phone (517) 487-3358.

TRS-80™ OWNERS
SUBSCRIBE TODAY TO
80 SOFTWARE CRITIQUE

80 Software Critique is a new publication devoted to in depth reviews of TRS 80 cassette software. Issue #1 is now available for immediate delivery via First Class Mail. It is 50 pages long and contains detailed reviews of over 50 programs or program collections.

Avoid buying disappointing software from now on. Read **80 Software Critique** before you buy. We review programs from a user's viewpoint. If a program is useful or fun, we say so. If a program is boring or contains bugs, we will tell you that, too. Many of the programs do contain bugs and we will tell you not to buy. We also include a game called Telephone Directory which you can use your computer to play.

Money Back Guaranty - 80 Software Critique is a quarterly publication. A one year subscription is $24. Single copies sell for $7. Subscribe for one year or buy a single copy. If you are not satisfied with Issue #1, return it to us and we will promptly refund your money.

This offer is good only while supplies of Issue #1 last. Subscribe today. You won't be disappointed. Send check or money order to

80 SOFTWARE CRITIQUE
P.O. Box 134 Waukegan, IL 60085

HANGMAN

What can be said about this classic? We figured everybody had a version, but we've seen it selling for as much as $19.95 within the last few months. The program also presents a good introduction to the TRS-80 SET and RESET commands.

The program requires that you enter your own words as data lines, like the examples in line 36. You are limited only by the amount of memory you have. A suggestion for letting the kids play: enter their spelling words as data lines!

Every good hangman player knows that 'E' is the most frequently occuring letter of the alphabet. Do you know the remaining letters, with respect to frequency? Here they are, most frequent first:

E T A O N I S R H L D C V P F M W Y B G V K Q X J Z

Just a couple of other points...the letters are all displayed at the bottom of the screen as play begins. As you guess a letter, that letter will be removed. For this reason, if a word contains two or more of one letter, all will be printed out when that letter is guessed.

****

HANGMAN Program Listing

```
1 CLEAR250:DIMA(27):DEFINTA-Z
2 CLS:GOSUB18:G1=0:READW$:S$="":T$=""
3 K=731:FORJ=1TOLEN(W$):PRINT@K,"-";:K=K+2:NEXT
4 A$="":PRINT@0,CHR$(32):PRINT@0,"WHAT IS YOUR GUESS";
5 GOSUB34:RESET(10+(ASC(A$)-64)*4,42):GOSUB12
6 FORI=1TOLEN(W$):IFA$=MID$(W$,I,1)THENGOSUB17
7 IFLEN(S$)=LEN(W$)THEN
  PRINT"WE HAVE US A WINNER!  WANT TO PLAY AGAIN?":GOTO32
8 NEXT:IFA$=B$THEN4
9 G1=G1+1:T$=T$+A$
10 ONG1GOSUB21,22,23,24,25,26,27,28,29
11 GOTO4
12 FORT=1TOLEN(S$):IFA$=MID$(S$,T,1)THEN16
13 NEXT
14 FORT=1TOLEN(T$):IFA$=MID$(T$,T,1)THEN16
15 NEXT:RETURN
16 PRINT"YOU ALREADY GUESSED THAT!":FORU=1TO300:NEXT:GOTO4
17 B$=A$:PRINT@0,STRING$(60," "):PRINT@0,"GOOD GUESS!!":
   FORT=1TO1025:NEXT:GOSUB33:RETURN
18 FORA=9TO10:FORB=54TO67:SET(B,A):NEXTB,A:RETURN
19 FORA=54TO56:FORB=11TO24:SET(A,B):NEXTB,A
20 SET(63,11):X=903:FORY=65TO90:
   PRINT@X,CHR$(Y);:X=X+2:NEXT:RETURN
21 FORA=61TO65:FORB=12TO13:SET(A,B):NEXTB,A:RETURN
22 SET(63,14):RETURN
```

(continued on page 25)

VTOS STILL VEILED

VTOS 3.0 is now VTOS 3.01, so I'm led to believe. I couldn't get the original diskette to back up--fatal in the case of VTOS; the master diskette orders you to make a backup and run from that. Upon receiving 3.01 my enthusiasm was renewed--for a few minutes. Source disk read errors on every attempt. Upon the inspirational guidance of my local Radio Shack dealer, I took my VTOS diskette and disk drive to his store. To my surprise, the diskette DID back up on his system.

HI HO, HI HO, Off to the repair shop we go! Three days later the word is that there is nothing wrong with my drive. Apparently, I discover later, there's a discrepancy between the speed that the inner tracks are accessed on TRSDOS 2.2 and VTOS, thus causing the high track number read errors. I understand Percom now has a hardware fix for this problem, although I haven't ordered it yet--I'm not sure if I will.

The reasons are this:

1. VTOS is only sparsely documented. Everything comes on the diskette and you have to print it out. The listing of commands and their implications are exciting, yet to this date I have yet to verify if any work.
2. VTOS tears up the top of memory--I assume it's the top--by creating various buffers and variable shuffling. For anyone hoping to utilize machine language routines, the ill-documented access to this area can be disastrous.
3. VTOS is too well protected. Making a backup is a chore with its own operating system, impossible with any other. One of the early sectors has been encoded not as a data record, something which most backups and formats look for or create.

I believe the attempt to 'protect' the operating system is more harmful than good. In the first place, the majority of disk owners that I know are not swappers seeking to save $14.95 on a popular game. They are business or professional people seeking help with real applications or programmers who are by no means into software swapping. At best a programmer would provide a modification to improve the system to someone who already had it--if the other person were a friend. Every other person pays cash.

The sad part of this store is that VTOS potentially has the options to assist the people with serious computer applications: a chaining utility so that a businessman could set up a job stream, go to lunch, come back and find his work all caught up; a system history log so that he can easily track down where things went wrong while he was chowin' down; a purge command to quickly eliminate all data (or other) work files; variable record lengths to permit maximum utilization of disk space. Assembly language programmers could enjoy a wider variety of Debug commands; a patch command for incorporating upgrades to machine language files; the diskette contains several modules that are supposed to enhance telecommunications--an RS232 driver, a keyboard send/receive terminal emulator and an advanced (?!) communication package--I really wish I could try some of these out.

I will advise you that there are some people who are seen smugly enjoying (and improving) VTOS. While on-line to the Orange County TRS-80 Users Group people in California, I picked up a tip for creating VTOS patch files.

For now, does anyone want to buy a practically brand new operating system? First $39.95 gets it. For later, I hope Randy Cook (author of VTOS 3.0 and TRSDOS 2.1) quits playing games and makes his system compatible with serious applications.

****

TRANSFER *(continued)*

```
350   PUT1,I:I1=I1+4:NEXTI:CLOSE:
      INPUT"DO YOU WANT TO PUT THE FILE ON ANOTHER DISKETTE";
      Q$:IFLEFT$(Q$,1)="Y"THEN225
```

****

HANGMAN *(continued)*

```
23 FORA=61TO65:FORB=15TO19:SET(A,B):NEXTB,A:RETURN
24 FORA=58TO59:FORB=15TO18:SET(A,B):NEXTB,A:SET(60,15):RETURN
25 FORA=67TO68:FORB=15TO18:SET(A,B):NEXTB,A:
26 SET(66,15):RETURN
26 FORA=64TO65:FORB=20TO22:SET(A,B):NEXTB,A:RETURN
27 FORA=61TO62:FORB=20TO22:SET(A,B):NEXTB,A:RETURN
28 SET(60,23):SET(61,23):SET(62,23):RETURN
29 SET(64,23):SET(65,23):SET(66,23)
30 K=731:FORI=1TOLEN(W$):PRINT@K,MID$(W$,I,1)" ";:K=K+2:NEXT:
   PRINT@0,"SORRY' YOU LOSE!  WANT TO PLAY AGAIN?"
31 GOSUB34:IFLEFT$(A$,1)="Y"THEN2ELSE34
32 S$=S$+A$:K=731+(I-1)*2:PRINT@K,A$:K=K+2:RETURN
33 A$=INKEY$:IFA$=""THEN33ELSERETURN
34 END
36 DATA"NO","DIRTY","WORDS","ARE","ALLOWED"
```

****

CURSOR CONTROL *(continued)*

```
540   GOSUB1000:REM 1000 IS USED TO VERIFY THE VALIDITY
      OF DATA.  IF DATA IS INVALID, VARIABLE SW IS SET
      EQUAL TO 1.
550   IFSW=1THENPRINT@X,CHR$(30);:GOTO530ELSENEXT
```

This will allow you to maintain a neat screen. If we were to simply branch back to line 510, after a few inputs, scrolling would erase already input lines to scroll off the screen.

LAST MINUTE ODDS & ENDS

We've received some flack about our pricing policies. It appears that 'suggested retail prices' are actually 'mandatory retail prices'! (Price fixing, anyone?) Only the people who received our initial promo flyers will be really familiar with what we were doing as far as pricing--apparently some dealers did and were furious! Our current stance is this: if we feel the software holds merit, we will continue to offer it; some dealers have been so kind as to let us offer their software at list price and will allow you to discount a certain percentage from the TOTAL order. Our special deals, at this point, will have to come from vendors who pose no restrictions on dealer pricing. Wish us luck.

Ironically, the only vendor who explicitly stated that their dealer prices had to follow 'suggested retail prices' (in their initial dealer package) had a full page spread in one of the major mags (Jan., 1980) with some discounted prices! We chose not to handle their software at this time because of the restrictions, even though most of their software is very good. Now, I'm glad.

Received a letter from Jeff Lasman at Practical Applications^{TM}. Jeff reminded us that the word Bootstrap should be accompanied with the Trademark symbol (^{TM}) since it is a copyrighted trademark for a program his company offers. We feel that Bootstrap^{TM} would be inappropriate for the column located on the inside front cover of this mag, especially after having to attribute the name to an entity not specifically associated with our publication; therefore, Bootstrap^{TM} will have another name, if any. Until we can communicate with our attorney, or get a call from someone else's attorney about copyrights, we will direct your attention to copyright information via the prompt on this month's cover.

We just caught a glimpse of the availability of a new operating system for the TRS-80. Has anyone purchased it yet? We'll investigate this further, and try and have more information next issue. We definitely recommend a 'wait and see' attitude before investing another fifty bucks! Let somebody else help 'em debug the thing!

We definitely solicit your feedback on this first issue. Was there too much of anything? Not enough? Something missing? You tell us. We most probably won't be able to answer each letter personally, but we will take heed to what you say. Our number of subscribers will never be lower (we hope!) so your comments have a much better chance in directing our development.

****

TAS wishes to acknowledge the following TRADEMARKS and/or copyrights:

TRS-80 *is a trademark of The Tandy Corporation.*
Z-80 *is a trademark of Zilog.*
Electric Pencil *is a trademark of Michael Shrayer Software.*
Bootstrap *is a trademark of Practical Applications.*
Sorcerer *is a trademark of Exidy.*
VTOS *is a trademark of Randy V. Cook.*
NEWDOS *is a trademark of Apparat.*
CPM *is a trademark of Digital Research.*

****

IN THE BUFFER

UP AND COMING FEATURES IN THE ALTERNATE SOURCE NEWS:

*** How to disable your break key so no one can look at your Basic programs!

*** Information on self-modifying programs.

*** How to modify Basic ASCII files so they can be edited with The Electric Pencil!

*** A program to peek into Level II ROM and return the entry points for every Level II command! Plus, a ton of information about the Level II ROM and DOS RAM, including many routines you can use in your own Z-80 programs!

*** A very sneaky way to 'fingerprint' your programs which is dozens of times better than REM statements (no, this hasn't appeared in any other magazine--yet!)

*** More software reviews!

*** Information on how to recover lost DOS directories!

*** Which printer for your TRS-80? Productivity, cost, reliability and service comparisons of the top printers available today!

THE NEXT ISSUE OF THE ALTERNATE SOURCE NEWS WILL BE MAILED ON OR ABOUT FEBRUARY 20, 1980.

AFTERWARD FOR ISSUE 1

From our advertisements, Jeff Lasman determined that we were violating his trademark for a bootstrap command processor program by using "BOOTSTRAP" as the title for our editorializing. I considered trademarking "Jeff Lasman" as the name for our editorial column, but, my saner self won out. We decided to change the column to Editorial RAMbling even though several friends, including a couple of attorneys, suggested there was no conflict here.

Our first issue was typeset on an IBM Composer. Because of the large number of typos we decided to use our Selectric (interfaced to a TRS-80, naturally) to set subsequent issues.

Our thanks to the many people who pointed out the split infinitive on the cover page.

Our goal going into this issue was to keep the magazine a size that Joni and I could fill ourselves, should this be necessary. I would consider this impossible by our current standards.

We were doing much phone work at this time, lining up potential authors. To our dismay we had contacted an enthusiastic Dennis Kitsz in Roxbury, Vermont just about a week after he had been contacted by another new TRS-80 publication -- 80 Microcomputing.

"Would he still write for us?" we wondered.

Two Dollars

Volume 1, Number 2

THE ALTERNATE SOURCE

The magazine of advanced TRS-80 applications and software.

IN THIS ISSUE:

All trademark acknowledgments are on page 39.

WELCOME TO TAS ISSUE #2...
Editorial Rambling by Charley Butler

You folks are great. That's all there is to it! Sure, we've received a couple of flattering letters (thanks, mom!), but I'm referring directly to your responsiveness! Not only comments and suggestions, but also with articles! We have established contact with several persons who have good ideas for articles; look for good things in issues to come!

Some comments about TAS in general I'd like to pass along. Most of these reflect your inquiries:

** You're more than welcome to call about whatever your heart desires (not collect!). In order to make ma Bell more bearable, please be advised that you can call as late as 10:00 PM, possibly later, although after that hour many of us become pumpkins, werewolves and various and assorted creatures of the night.

** We were chastised for not elaborating on two very important 'groups' now operating in TRS-EIGHTYDOM (huh?). One is TCS, also known as the Tidewater Computer Society. Four to six times a year, they mail free (news)letters to individuals who will respond to their questionaire. Each letter is packed with many tidbits about people, products, program patches and the like. Joni has communicated several times with Les Logan, one of the key figures at TCS. She relates that right now, unfortunately, they are not seeking to expand their circulation. However, my current attitude is this: all Les has to do is ask, and he is free to use this publication to further the goals of his club and its worthy project.

The second group is CMTUG. Several readers wrote asking us to identify CMTUG. This is the 'Central Michigan TRS-80 Users Group.' For over a year and a half, Joni and I toyed around with our system, learning tidbits here and there. Only last year did we discover, Lo and Behold, that there were many other TRS-80 users around us! It's a great feeling to know that there are people you can call (even though you feel 'bothersome' at times) when you have a problem. We would recommend that, whatever your interest, you check out your local users group by attending the meetings periodically. Members of CMTUG have proven to be extremely helpful, both on a personal level, and as far as supporting TAS. I would especially like to acknowledge A. J. Rogers, who took time to provide us with valuable consultation, who assisted us in contacting local authors, and also provided us with reasonable printing rates.

One other topic: we've had a couple of complaints about discussing our problems of getting started. We will try to keep this to a minimum.

"NO FRILLS" TAPE DISPLAY PROGRAM
By Dan Yerke

After owning and using my cassette-bound TRS-80 for over a year and a half, I had accumulated a pile of unlabeled and mostly forgotten tapes, some of which I wanted to keep.

A means of efficiently screening my tape pile was needed, and Basic alone could not perform this task. To meet this need I wrote my first machine-language program with the idea of learning the Z-80 instruction set, using ROM subroutines, interfacing with Basic, and have a solution to my predicament in one shot. It took weeks!

What finally surfaced from this ambitious project was a very simple routine for performing a combination ASCII and hexidecimal "veritical dump" of any TRS-80 tape onto the display screen. I call it a "No Frills" tape display program because it's only function is to do a simple-minded display without error-checking routines or logic for any of the record-types the program would encounter. It doesn't even know when the end of a file has been reached.

The routine was assembled and then translated into Basic data statements so there is no need to respond to "MEMORY SIZE?" or even load the program with the SYSTEM command. The routine is copied into a string variable (PG$) and connected to Basic's USR(function with VARPTR and some PEEKs and POKEs.

When the Basic USR(function is entered the machine language routine starts the cassette and searches for a file leader. When a file is found the data is displayed on the screen. Pressing any of the following keys will return control to Basic: CLEAR, SPACE, ENTER, BREAK, or the arrow keys. When Basic has control, the number of bytes displayed is available from the USR(function. If the display should go beyond the end of the block or file, the computer will "hang-up" with the cassette still running until another block or file is found on the tape, or the user pushes the reset button.

Since the program is too simple to know better if the file has ended, it is necessary to stop the program when it begins displaying the leader of the next file, and then restart from where it stopped to give it an opportunity to resynchronize with the leader of that file. If this is not done, the display will not necessarily be sensible.

The display format is intended to show as much data as possible at once on the screen. It is organized as five groups of 3 display lines. The ASCII character (as upper case unless you have lowercase installed) is on the top line of each group with the left and right nibbles arranged on the second and third lines respectively. The "cursor" is on the third line of the current line group and is positioned to the next display location that will be used. The cursor helps you to see where the display of a long file has "wrapped-around" over previous data.

As it stands now, the program has a number of uses associated with it's ability to display any block or file without regard for it's type:

1) Identify Basic program and data tapes
2) Identify machine language files and programs
3) Experimenting with the proper volume settings for a difficult tape
4) Finding for reconstruction data that cannot be processed normally by Basic

** General Operating Procedure **

1) CLOAD the Basic program.

2) Rewind and remove the Basic program tape.

3) Load and position the tape to the leader of the file you want to display.

4) Type "RUN" (ENTER)

5) To stop the display hold down the space bar, the program halts before displaying the last byte read and returns to the Basic program. The "bytes displayed" count does not include the length of the file leader or the "sync byte" of the current file. If the files on the tape are not separated by a silence gap the byte count will include the leader bytes of the next file (or print-# block) that have been displayed.

6) To continue displaying data, rewind to the leader again, and hit the / (slash) key.

7) To discontinue the program, use "RESET" or press (BREAK).

8) Disk users can make the following changes to this program to run under Disk BASIC:

```
1180 AD=PEEK(B+1)+PEEK(B+2)*256:IFAD>32767 THEN
     AD=AD-65536
1190 DEFUSRØ=AD:CMD"T"
```

THE BASIC LISTING IS ON THE NEXT PAGE, FOLLOWED BY THE SOURCE LISTING.

BASIC LISTING FOR "NO FRILLS"

```
1100 CLEAR 500
1110 FOR I = 1 TO 90       ' READ 90 DATA ITEMS,
1120 READ B
1130 PG$ = PG$ + CHR$(B)  ' STRING THEM TOGETHER IN PG$
1140 NEXT I
1150 B = VARPTR(PG$)       ' POINT B TO PG$ CONTROL BLOCK
1160 IF PEEK(B) <> 90 PRINT B, "INCORRECT LENGTH IN PG$." :STOP
1180 I = PEEK(B+1) :POKE 16526,I   ' TRANSFER LOCATION OF STRING
1190 I = PEEK(B+2) :POKE 16527,I   ' PG$ AS USR( ENTRY ADDRESS.
1200 CLS
1210 K% = USR(0)           ' ENTER PROGRAM STORED IN PG$
1220 PRINT@0, K%; "BYTES DISPLAYED.";
1230 IF INKEY$ <> "/" THEN 1230
1240 GOTO 1150' RESTART TAPE AND DISPLAY.
1250 DATA 33,0,0,62,0,205,18,2,205,150,2,14,0,221,33,128
1260 DATA 60,6,64,221,54,64,191,205,53,2,87,58,64,56,183
1270 DATA 40,6,205,248,1,195,154,10,35,122,221,119,192,230
1280 DATA 15,198,144,39,206,64,39,95,122,15,15,15,15,230
1290 DATA 15,198,144,39,206,64,39,87,221,114,0,221,115,64
1300 DATA 221,35,16,198,17,128,0,221,25,12,62,5,185,32,185
1310 DATA 40,177
```

SOURCE LISTING FOR "NO FRILLS"

```
00000 ; =====================================================
00001 ;
00002 ;              "NO FRILLS TAPE DISPLAY PROGRAM
00003 ;               ASSEMBLY LANGUAGE SUBROUTINE
00004 ;
00005 ;              BY  DAN YERKE,  NOVEMBER 1979
00006 ;
00007 ; =====================================================
00008 ;
00100 ORIGIN   EQU    32600    ;ORIGIN FOR ASSEMBLY
00110 SCREEN   EQU    3C00H    ;SCREEN UPPERLEFT CORNER
00120 KBDRW7   EQU    3840H    ;KEYBOARD ROW SEVEN:
00130 ;               CLEAR, BREAK, SPACE, ENTER OR ARROWS.
00140 TAPOFF   EQU    01F8H    ;ROM SUBRTN: TURN OFF CASSETTE
00150 TAPEON   EQU    0212H    ;ROM SUBRTN: DEFINE CASSETTE UNIT
00160 TAPESY   EQU    0296H    ;ROM SUBRTN: FIND LEADER & SYNC.
00170 TAPEIN   EQU    0235H    ;ROM SUBRTN: READ ONE BYTE INTO A
00180 BASIC    EQU    0A9AH    ;ROM SUBRTN: RETURN W/HL TO BASIC
00190 LN       EQU    64       ;CHARACTER LENGTH OF SCREEN LINE
00200 NBRLNS   EQU    5        ;NUMBER OF LINE GROUPS TO USE
00210 ; * * * * * * * * * * * * * * * * * * * * * * * * * * *
00220 ;
00230          ORG    ORIGIN   ; LOAD ASSEMBLY ADDRESS
00240 START    EQU    $        ; ENTRY POINT (ALWAYS)
00250          LD     HL,0     ; RESET DISPLAYED BYTES COUNTER
00260          LD     A,0      ; USE CASSETTE UNIT 0
00270          CALL   TAPEON   ; TURN ON TAPE UNIT
00280          CALL   TAPESY   ; BYPASS LEADER & SYNC BYTE
```

<LISTING CONTINUED NEXT PAGE...>

```
00290 NEWSCR  EQU   $          ;LOOP: START NEW SCREEN
00300         LD    C,0        ; RESET LINE GROUP COUNTER
00310         LD    IX,LN+LN+SCREEN ; POINT TO LINE 3
00320 NEWLIN  EQU   $          ;LOOP: START NEW DISPLAY LINE
00330         LD    B,LN       ; RESET LINE-BYTE COUNTER
00340 OLDLIN  EQU   $          ;LOOP: CONTINUE CURRENT LINE
00350         LD    (IX+LN),191      ; DISPLAY "CURSOR"
00360         CALL  TAPEIN     ; READ ONE BYTE INTO REGISTER A
00370         LD    D,A        ; SAVE BYTE IN D TEMPORARILY
00380         LD    A,(KBDRW7)       ; CHECK KEYBOARD ROW 7
00390         OR    A                ; SET FLAGS
00400         JR    Z,NOKEYS         ; IF NONE GOTO NOKEYS
00410         CALL  TAPOFF           ; ELSE STOP TAPE UNIT,
00420         JP    BASIC            ; AND RETURN TO BASIC.
00430 NOKEYS  EQU   $          ; CONTINUE PROGRAM
00440         INC   HL         ; DISPLAYED BYTES COUNTER + 1
00450         LD    A,D              ; RESTORE BYTE TO A
00460         LD    (IX-LN),A        ; DISPLAY TOP LINE ASCII
00470         AND   0FH        ; MASK OFF LEFT NIBBLE TO ZEROS
00480         ADD   A,90H      ; CONVERT
00490         DAA              ;   RIGHT
00500         ADC   A,40H      ;     NIBBLE
00510         DAA              ;       TO ASCII.
00520         LD    E,A        ; SAVE ASCII RIGHT NIBBLE
00530         LD    A,D        ; RECALL BYTE SAVED IN D
00540         RRCA             ; SHIFT LEFT NIBBLE
00550         RRCA             ;   OVER TO THE
00560         RRCA             ;     RIGHT NIBBLE SIDE
00570         RRCA             ;       FOR CONVERSION.
00580         AND   0FH        ; ZERO LEFT NIBBLE
00590         ADD   A,90H      ; CONVERT
00600         DAA              ;   LEFT
00610         ADC   A,40H      ;     NIBBLE
00620         DAA              ;       TO ASCII
00630         LD    D,A        ; SAVE ASCII LEFT NIBBLE
00640         LD    (IX),D     ; DISPLAY IN MIDDLE LINE
00650         LD    (IX+LN),E ; DISPLAY RIGHT IN LAST LINE
00660         INC   IX         ; BUMP SCREEN POINTER UP ONE
00670         DJNZ  OLDLIN     ; TEST END-OF-LINE: NO - LOOP.
00680         LD    DE,LN+LN ; GET LINE-GROUP DISPLACEMENT
00690         ADD   IX,DE      ; POINT TO NEXT 3-LINE GROUP
00700         INC   C          ; BUMP LINE COUNTER UP ONE
00710         LD    A,NBRLNS ; COMPARE NUMBER OF LINES LIMIT
00720         CP    C          ; TO CURRENT LINE COUNT,
00730         JR    NZ,NEWLIN ; >= THEN GOTO NEWLIN.
00740         JR    Z,NEWSCR ; ELSE, GOTO NEWSCR.
00750 LENGTH  EQU   $-START ; HAVE ASSEMBLER SHOW PGM-LENGTH.
00760         END   START     ; END ASSEMBLY, SET ENTRY POINT.
```

<END OF LISTING>

ERRATA ALREADY! Some versions of EDTASM will not assemble the line 460 (LD (IX-LN),A). In order to get around this, make the instruction like this:

```
00460      LD      (IX+256-LN),A
```

<END OF ARTICLE>

CAUTION! ADVERTISEMENT AHEAD...

We're convinced that the problem with learning Z-80 Assembly language is the direct result of a lack of good source programs to serve as study guides. Remember when you were learning Basic? Every time you encountered a problem, there were dozens of examples to help you work it out. Not so with assembly language. Combined with the fact that there are more and more intricate concepts to understand, things have been rough going for the persons desiring to master this excellent language. We believe that the general availability of some programs - especially for the TRS-80 - will aid to better all who choose to investigate this fascinating field, ideally providing faster and more competent programs down the road! For this reason, we have been encouraging authors to make the EDTASM source code available, at least as an option. And, we're proud to say, we've been successful!

****

Z-80 Source Code from
The Alternate Source!

Each source listing listed below is a valuable tutorial in itself! All programs are designed for the TRS-80 and include a Z-80 source file!

*** DVR--a Driver package that adds several features to the TRS-80 including: screen echo to printer, repeating key, lower case driver and more! Level II or Dos. Cassette. $15.95; diskette, $18.95. Also, DVR now works with NEWDOS!

*** KEEPIT--Level II only! Preserves your program from any execution point. Excellent for debugging and saving current program status. Cassette only, $9.95.

*** COPYDISK--a program which allows copying of several DOS files at once with no system diskette! Complete Disk I/O routines! Great for diskette library maintenance and single drive owners! On cassette, $17.95; diskette, $20.95.

*** MICROSKETCH III package--Microsketch, Microscreen and Screen Save Utility. These three programs (on 3 tapes) make graphics as easy as 1-2-3! All 3 programs sell for $23.85. Included FREE are the source codes for FIFTEEN machine language subroutines that are utilized! If you're already a Microsketch owner, purchase the two support packages and the source codes for the fifteen machine language utilities for only $19.85 total!

Owners of DVR and COPYDISK may return original media and purchase the source codes for the difference in price!
Mail your order to: TAS SOURCE, 1806 Ada street, Lansing, MI 48910. Visa and MC accepted by phone 517/487-3358. Include 50 cents per program for First Class Mail. More to come!

>>>>>>>>>> SOUNDEX <<<<<<<<<<
A TRS-80 SOUND UTILITY

By Roxton Baker, 56 South Rd., Ellington, CT 06029
(Software in the Public Domain)

The SOUNDEX utility was written to make it easy for BASIC programmers to add sound to their programs. This machine language driver can be poked anywhere in memory and can even be embedded in one or two lines of **BASIC** at the beginning of a program. Once it is available, cassette-port sounds can be generated with the USR call under Disk or Level II BASIC.

The SOUNDEX driver offers tone-producing capability to the programmer. Complete ready-made sound effects are not available in it. It is up to the programmer to generate these by appropriate calls to SOUNDEX from BASIC. A compromise has been made between code length, simplicity, applicability, and ease of use on one hand, and overall sound capability on the other. SOUNDEX does offer the following features to the software author considering the use of sound:

It is easily called from BASIC via the USR command.
It works identically under Level II or Disk BASIC.
It is completely relocatable, and thus can be poked anywhere in memory without reassembly.
It offers a wide range of frequencies (0 to 8 KHz).
It offers a number of different "sounds", both loud and soft.
It is suitable for embedding into string space or REM statements (no need for memory protect).
It is short (72 bytes).

And

It is NOT copyrighted (though author credit in a REM line is requested).

HARDWARE

The computer's cassette output line (the large gray plug that goes to the recorder's AUX input) is what carries the sound. It can provide enough power to drive a small earphone or set of headphones directly. For more volume, a separate amplifier and speaker should be used. One low-cost posibility is to connect a speaker to the recorder's "EAR" output, and to set up for a CSAVE as usual, with the recorder in the RECORD mode and the AUX plug in place. This works well with the CTR-80; I'm not sure about the CTR-41.

Radio Shack also offers a "Microsonic" box, cat. no. 277-1008, which can be plugged into directly. It runs from a battery, has it's own amp and speaker, and costs about $11.00.

The best sound of all results from connecting the cassette output to a nice, powerful hi-fi system. Phaser blasts through this can really shake up the troops.

USE OF SOUNDEX

Once the sound driver has been put in memory (as described later), calling it from BASIC is fairly simple. The user must first specify TONE DURATION, which is done by writing:

```
X = USR ( dd )
```

In this, X is a dummy variable. Any variable name can be used instead. The value "dd" must be a number between -1 and -32767. The larger in magnitude this number is, the longer the tone will be. The range is 0 to 2 seconds. Once a duration has been specified, it will control all subsequent tones until a new duration is specified.

To actually generate a tone, the user would then write:

```
X = USR ( pp )
```

As before, X is a dummy variable. The value "pp" can be any number between 1 and 32767; it defines the PITCH of the tone that is generated. The larger the value of pp, the lower will be the pitch of the tone. Actually, only values for pp between 1 and 100 or so are useful. A blank, or silent, tone is easily generated by using a value of 0 for pp.

As an example, the following statements in a BASIC program would put out two tones. The first tone would be long and low-pitched, the second tone short and high-pitched:

```
10 X=USR(-20000)
20 X=USR(50)
30 X=USR(-2000)
40 X=USR(2)
```

If line 30 were deleted, both tones would be long.

The USR calls are simple enough, and often they are all that will be used.

There are a few preliminaries and options, however. First, the user must select a "voice" or type of sound by poking a value into address 16672. This value can range from 0 to 255, but only ten different values give distinct sounds. These are:

1 5 6 17 18 22 25 86 90 102

Some of these do not give different types of sound; they give a tone of twice the normal pitch. Just as with TONE

DURATION, the voice need not be respecified until a different one is desired. As an example, near the beginning of a program, the user might write:

```
POKE 16672,18
```

All tones generated after this would be fairly smooth "sine wave" approximations. For a noticeably softer tone, he might write POKE 16672,5. For a thin, reedy sound he would use POKE 16672,25.

Under Disk BASIC, the user has the option of disabling interrupts with CMD"T". If he doesn't, the tones will all have a "popping" quality about them, not unlike radio static.

By working with a combination of tone duration, pitch, voice, and speed of call from BASIC, one can eventually arrive at a suitable sound effect for almost any application.

Although usually of no concern, there is one further option available to the user. Under BASIC it is possible to use OUT 255,4 or OUT 255,12 to turn the cassette motor on. SOUNDEX will normally turn off the cassette motor whenever it is called. Should the user wish to run the cassette during his program (possibly even during the generation of sound), he should inform SOUNDEX of this fact with a POKE 16697,4. To go back to having the cassette motor disabled, POKE 16697,0.

Finally, one must of course define the USR entry point at the beginning of the program. This is discussed in the Level II and Disk BASIC manuals. The address to be used depends on where SOUNDEX has been placed in memory. The start of SOUNDEX is its entry point.

Note that SOUNDEX achieves its relocatability by using as temporary data storage six bytes in low memory - the "reserved" section of RAM. These bytes are hex 411C-4120 and 4139. It is believed that these are unused by Level II or any DOS.

CREATING THE SOUNDEX CODE

Probably the easiest way to get the 72 bytes of code into memory is to POKE it in with a BASIC program like the one listed at the end of this article. If high memory is to be used (the usual approach), then MEMORY SIZE must be answered to protect the code from BASIC. To place SOUNDEX at the top of memory, appropriate numbers would be 32695 for 16K, 49079 for 32K, or 65463 for 48K. If TRSDOS 2.2 is used, subtract 51 from these.

Since SOUNDEX does not contain any zero bytes, it is feasible to poke the code into a dummy REM statement instead. Then memory protection is not required, since SOUNDEX is an integral part of the BASIC program itself. This technique is presented later.

The program listed may look complicated because it has been set up to handle different combinations of BASIC level and memory size. Any single application will really require only the READ, DATA and POKE statements; these can be extracted and used as needed in the user's program.

As an example, one might wish to have sound in a 16K Level II game. MEMORY SIZE would then be set at 32695, assuming there are no other high-memory machine language routines to be used. If there were, SOUNDEX would have to be located below them, with a lower memory protect. Once into BASIC, the program below would be loaded and RUN. The SOUNDEX code could then be saved as a SYSTEM tape using TBUG. This SYSTEM tape would, later, be loaded in conjunction with the user's BASIC program.

A better alternative would be for the user to incorporate the DATA statements into his own program, and POKE them into (protected) high memory himself, starting at 32696.

Either way, once the code was in memory, the user would define the USR entry point (32696) to BASIC with:

```
POKE 16526,184 : POKE 16527,127
```

as explained in the Level II manual.

Finally, in his program, the user would specify a type of sound - perhaps POKE 1672,17. At this point, sound could be generated via the USR calls described earlier.

STORING SOUNDEX IN A REM STATEMENT

The advantage to poking a machine language routine into the space normally reserved for a REM statement is that the code becomes part of the BASIC program. Memory protection is not required. Furthermore, it is possible to save the BASIC program with the code in it, meaning that it never again need be poked in. The READ, DATA, and POKE statements can be deleted. The machine language has become a permanent part of the "target" program.

Although in theory any machine language code can be put into REM statements, the process gets difficult for code that is lengthy or that contains bytes with the value 00. SOUNDEX does not present these problems, and since it is the subject of this article, it will be used as an example.

Briefly, the technique used is to put a dummy REM statement in the target BASIC program as the first line. Line number 1 is used for this line to keep anyone from ever putting another statement in ahead of it, which would dislocate it. Then the SOUNDEX code is poked, by the target program itself, into the space that was allocated to the REM line. Note that the REM must have at least as many dummy characters in it as there are bytes in machine code. Finally, the USR entry point is defined to point at the string of code now in the REM line.

As mentioned above, this poking can be done by the target program every time it is run, or it can be done just once after which the DATA, READ and POKE statements can be deleted, and the target program saved.

The advantage of the former is that the program remains entirely listable and readable, whereas if the program is saved after the code has been poked into line 1, that line will not list properly. Nor can the program then be saved as a disk ASCII file, meaning that some renumbering utilities will not work with it.

Either way, the first line of the user's target BASIC program should be:

```
1 REM***********(continue for 72 *'s total)******* ....
```

These dummy *'s (any character could be used) reserve enough space for the SOUNDEX code. Next, it is necessary to locate the address of the first asterisk so that the poking of SOUNDEX can begin there. This address is fixed in the sense that we are always using the first line; but Level II, TRSDOS and NEWDOS all put the first line at a different place. However, they all point to it the same way - its address is kept at locations 16548, 16549. The first asterisk is always the fifth byte in the first line, so it's address (and thus the location for the first byte of SOUNDEX) is simply:

```
PEEK(16548) + 256*PEEK(16549) + 5
```

This is also the address that should be defined as the USR entry point using DEFUSR in Disk BASIC or its POKE equivalent in Level II.

Putting this all together, the following lines of BASIC, placed at the beginning of the user's target Disk BASIC program, would serve to embed the SOUNDEX routine into the program. After running the program once, the user could delete lines 4 through 12, and save the target program with the machine language now permanently integrated into it. In fact, lines 1, 2 and 3 could be saved alone as a "sound preamble" that could be put at the front of any Disk BASIC program to give it sound.

```
1 REM*******(72 *'s total)********* ...
2 S=PEEK(16548) + 256*PEEK(16549) + 5
3 DEFUSR = S
4 FOR J = 1 TO 72
5 READ X
6 POKE S+J-1,X
7 NEXTJ
8 DATA ..............
9 DATA ..............
10 DATA .............
11 DATA .............
12 DATA .............
13 and so on--user's program
   from here on out
```

Note: These are the same data statements as in the program on the next page...

The above is for Disk BASIC; line 3 would have to be changed for Level II. For an example of how to convert a decimal memory address into the correct POKE statements for setting up a Level II USR entry address, see lines 200 through 280 in the program on the next page.

PROGRAM TO POKE SOUNDEX INTO MEMORY

```
10 REM    >>>>>>  SOUNDEX  <<<<<<
20 REM    BY ROXTON BAKER - NOT COPYRIGHTED
30 CLS
40 DEFINT J,K,L
50 INPUT"WHAT MEM SIZE DID YOU SET ";DS
60 REM SOUNDEX WILL START AT MEM SIZE PLUS ONE
70 MS = DS
80 IF MS>32767 THEN MS=MS-65536
90 REM POKE THE 72 BYTES IN
100 FOR J = 1 TO 72
110 READ K
120 POKE MS+J,K
130 NEXT J
140 REM NOW SEE IF WE'RE IN LEVEL II OR DISK BASIC
150 IF PEEK(16433) = 0 THEN 200
160 REM MUST BE DISK
170 CMD"T"
180 DEFUSR=MS+1
190 GOTO 290
200 REM IT'S LEVEL II
210 N=DS+1
220 REM CHANGE N FROM DEC TO HEX FOR USR ENTRY POINT POKE
230 FOR I = 1 TO 4
240 D(I)=INT(N/16[(4-I))
250 N=N-D(I)*16[(4-I)      -------->    NOTE:  "]" = up arrow
260 NEXT I                                     in these 2 lines
270 POKE 16527, 16*D(1)+D(2)
280 POKE 16526, 16*D(3)+D(4)
290 REM CHOOSE SQUARE WAVE FOR DEMO
300 POKE 16672,102
310 REM THIS IS THE BIG DEMO . . .
320 X=USR(-100)
330 FORL=65TO1 STEP-1 : X=USR(L) : NEXT
340 FORL=1TO70 STEP 2 : X=USR(L) : NEXT
350 REM
360 REM THESE FIVE DATA STATEMENTS CONTAIN THE
370     RELOCATABLE CODE OF SOUNDEX:
380 REM
390 DATA 205,127,10,203,124,40,4,34,28,65,201,34,30,65
400 DATA 219,255,31,31,31,47,230,248,95,58,57,65,254,4,
         32,2
410 DATA 171,95,58,32,65,87,237,75,28,65,43,124,181,40,6
420 DATA 221,227,221,227,24,12,42,30,65,122,7,7,87,230,
         3,179
430 DATA 211,255,3,120,177,32,228,123,211,255,201
```

<END OF LISTING>

LETTERS!

Dear Sirs:

I see that you are selling Donald Knuth's series of books. I've written a MIXAL simulator/assembler which would go nicely with these. It requires 32K (48K for full 4000-word MIX memory), preferably disk. I would be willing to send a copy (for duplication costs) to anyone who writes.

Sincerely,
Phelps Gates
6 Crestwood Tr. Pk.
Rt. 4
Chapel Hill, NC 27514

(Thanks Phelps. I've received my copy and am looking forward to giving it a workout!--Kos.)

****

Gentlemen:

Your Hangman game in your first issue is excellent, but it has bugs!! I had a tough time getting your program to work properly. I inserted a number of remarks as an aid to studying the program, and I also renumbered your program to add a zero to each of your line numbers.

The program's biggest problem is that it lacks randomness. I have also corrected this flaw by employing a technique that I learned from Steve Macgregor. It forms the data words into an endless chain and the computer makes a selection from that chain, depending on the instant a key is pressed. Incidentally, there is no need for the quotes around the data words as shown in line 36 of your program. Elimination of these simplifies typing data statements.

I will be glad to reproduce the tape and listing for any of your readers for two dollars to include postage.

Sincerely,
C. W. Evans
9806 Amber Trail
Sun City, AZ 85351

(In our thus far short life span, CW has offered many suggestions and comments. I got a feeling we'll be reading some of his fine work in a future issue!--Kos.)

****

Dear Joni:

I enjoyed my first copy of your publication, and am looking forward to future issues. Dennis Kitsz and I have shared ideas in the past and I was pleased to see his contribution.

A suggestion. There does not seem to be much available (as far as I can find) on the use of DCBs for the TRS-80. I am belatedly learning that much machine language programming involves these blocks and that area of memory between 4000H and 4300H. I would sure appreciate a detailed article explaining the Device Control Blocks and their use.

Please remind your editors that many of us inspired "Computerists" are relatively speaking, still novices. As an example, the article on "Cursor Control" must be written for Disk Basic, though nowhere in the article or program is this fact mentioned.

Sincerely,
John Painter
Silver Springs, MD

(John, the main technique in Cursor Control is applicable for both Level II and DOS. Both store the current cursor location in the same area of the VCB. About the Control Blocks, I refer you to Allan Moluf's Ramstuff article this issue as well as our ad for source programs. Allan's work is ideally setting us up for exploration of many of these areas in future issues--we have one program demonstrating them that's a real KILLER! Hold tight!-Kos.)

****

TAS BULLETIN BOARD

Richcraft Engineering, Box 1065, Chautaugua, NY 14722 advises of the availability of:

--W4UCH TRS-80 Morse Code Transmit & Receive Program: no ancillary devices required, 5-25 words per minute, 20 prepared "Q" signals, auto-logbook, code practice, and much-much more. $15.00 cassette or disk postpaid.
--TRS-80 Disassembled Handbook: $10.00 postpaid.
--TV Satellite az-el-range (send your lat/longitude): $5.00 postpaid.
--Multi Base Both-Way Conversion Program: decimal-binary-hex-split decimal-split hex; $10.00 cassette or disk.

Phone (716) 753-2654 for COD orders.

Blechman Enterprises, 7217 Bernadine Ave, Canoga Park, CA Zip 91307, advises of the availability of:

--Telephone Dialer Program: store up to 500 names and numbers in 16K memory; request an alphabetized list; or one desired name; will even dial for you with the addition of a simple external interface (no modification required!). Keeps track of time and computes charges. And more. Detailed instructions. $10.00 postpaid. CA residents add tax.
--TRS-80 Programs for Amway Product Distributors (no disk or printer needed!) A money making opportunity! Four

programs put you in business, providing data services for Amway Distributors. There's one near you, and they need your help! $24.95 covers all four programs and postage. Write for details!

Club-80 Newsletter Exchange, Box 38, Douglass, KS 67039 tells us they assist groups in exchanging newsletters and disk files. Users groups should investigate, as this is an excellent method of sharing news, ideas and programs. Phone Laris Pickett at (316) 746-2650 for more info.

R. E. Henley, 4530 N. 16th St., Omaha, NB 68110, advises that he is available for contract programming. Persons needing customized programs are encouraged to contact him.

David K. Forbes, 3241 Briarcliff Drive, Anchorage, AK, zip 99504, would be interested in any comments (pro or con) on the Microtek MT-80 printer. He would also like a 100% reliable method of determining that the <BREAK> key has been pressed. (To be used in machine language under TRSDOS 2.2 -- method must intercept the <BREAK> prior to any action by DOS, such as a reset.)

Mr. Forbes will also consider free-lance software development on a custom basis. He relates the following qualifications: 12 yrs programming experience, 8 professionally, FORTRAN, BASIC, Assembly Language, APL, COBOL. He has an in-house TRS-80, and has previously developed commercial software for businesses (references on request). He works cheap 'mostly for fun after work', but will require a written specification of what the software is to do, and a contract.

Management Systems Software, Inc. has released two new business applications--PROFORMA CASH BUDGET and a LEASE-BUY PROGRAM. Both have significant documentation, are written by a professional educator, and have been tested for over 6 months to insure an error-free product. Briefly, PROFORMA CASH BUDGET will forecast cash needs for up to 12 periods, by specifying up to 25 different cash-inflows and 25 different cash-outflows. The LEASE-BUY Program evaluates the typical lease versus purchase decision, relates current tax laws and the impact of investment tax credit on this type of decision. Persons interested may obtain more information by writing: 5200 Brittany Drive, #1006, St. Petersburg, FL 33715

Allen Blanchard, of R.G.S. Supermarkets, Inc. would be interested in a program which would enable his cashiers to readily determine whether or not a check-cashing card produced by a customer should be honored or not. Several thousand cards are involved, and lists are updated daily. If anyone can help Mr. Blanchard, he can be contacted at 1001 Barrow Street, Houma, Louisiana 70360.

A REPORT ON THE STRINGY FLOPPY
By Roger Fuller

The long sought after missing link between cassette and diskette storage has been discovered by Exatron--The Stringy Floppy. When my first unit arrived, I quickly unleashed it from its box and promptly broke it--the edge card connector on the string's ribbon cable is not the exact size as the edge card on the TRS-80 (which is thicker than normal). Results: a torn contact caused by a slight burr on a trace of the TRS-80's card edge. When finally I got a new connector, I found the pins on the 80 were upside down. After carefully crossing pairs of ribbon cable wires and pushing them onto the contact prongs one at a time, I had the Stringy working.

The Stringy Floppy loads and saves data at 7.2 kilobaud, about 15 times faster than cassette. The data is stored sequentially on an endless loop of narrow chromium dioxide tape, mounted in a tiny plastic "wafer" cmaller than a business card. The actual recording is done using a self-clocking bi-phase method, the same technique used on floppy disks. However, since the Stringy uses only one track at 10 ips versus 35+ at about 30 ips for disks, the Stringy is more reliable.

The Stringy operating system (SOS) is stored in 2K EPROM mapped to the unused, reserved I/O addresses, and uses ports 240-247, **depending** on the number of drives. RAM usage is just 4 bytes between string space and reserved high memory. The SOS is linked to Level II or Disk Basic by a jump using BASIC's SYSTEM command. After loading SOS, respond with an easy-to-remember "/12345" and you get the sign-on message. The SOS does not use any reserved words, since all its commands are preceded by an "@".

Exatron sells extra wafers in standard sizes of 5, 10, 20 and 50 foot lengths for $2.00 each in boxes of ten. The wafers can be checked for dropouts by the 'NEW command; this will erase all files on the wafer. A 5 foot wafer will load 4K of data in 6 seconds. The SOS will automatically verify an @SAVE; thus the time is doubled when saving data. This is independent of the length of your program. Thus, a one line program could take almost two minutes to save on a 50 foot wafer, the worst case occurance. The SOS allows expressions to specify both drive number (0-8) and file number (1-99). Included with the Exatron Stringy Floppy (ESF) is a data I/O program which gives the Stringy owner a file handling capability. The new commands are: @CLEAR, @OPEN, @CLOSE, @INPUT, and @PRINT. This 1000 byte machine language program patches itself into RAM just as the 4 byte work area does.

One of the unique features of the ESF is the ability to @SAVE memory. You need only specify the start address and length of data in decimal form. Also an auto start address may be added, if desired. I can see a teacher (such as myself) storing records for an entire class on a wafer, updating them, and re-storing the whole memory out on tape. This would eliminate all the bother and extra coding required to keep up with the changing data.

The ESF can chain programs in BASIC, leaving the old values computed by the original program intact and accessible. This is possible by use of overlays of new programs @LOADed by the old one. If you @LOAD inside a program, the new program is loaded and executed just like a statement. This one feature could make living in a non-DOS 16K machine acceptable for many people.

The TRS-80 is the VOLKS-COMPUTER, but its usefulness in applications requiring data handling beyond a single dedicated program is limited. A business man who sees Radio Shack's $499 ad is misled to believe this will "computerize" him. He is usually surprised to find the total cost is nearer $2000. The Stringy brings that cost down to under $1300 for 16K and ESF. If I wrote programs for businesses, I would order an ESF just because of the market it can open for my software.

The ESF has drawbacks like anything else. One problem is the way files must be handled sequentially. This is a possible hazard for the sloppy user. You MUST keep an accurate record of what is where, on which wafer, or you might just lose something important. This can be reduced by using a wafer per program, but $2.00 each is quite high. Even so, the cooperation of Exatron with their customers can outweigh this in terms of support. The Exatron company provides an SOS version of the Electric Pencil for only $10.00 more than the normal price. If you are wondering whether or not you need a disk or are tired of cassette problems, think about an ESF. At $249.50, it is a bargain.

<END>

****

SAVE 15% !!

TRSDOS users! Now you don't have to buy a new operating system to enter 'ZAP's into your system! Here's a package especially designed to permit file recovery and system manipulation for 2.2 and 2.3 users! Check it out:

1. H. C. Pennington's "TRS-80 Disk & Other Mysteries". The disk owners bible. What more can we say? Lists for $22.50. 2. Z80ZAP--the MACHINE LANGUAGE monitor that contains valuable file recovery functions, hash code computation, and disk examine & modify functions. Lists for $29.95. 3. For fast delivery , First Class Postage. $2.25.

TOTAL LIST VALUE: $54.70
COMBINATION DISCOUNT: $9.70
YOU PAY ONLY: $45.00 !!

Over 15% discount for ordering both! If 4th class postage is OK, deduct $1.75. Mail your order to: TAS Combo #1, 1806 Ada Street, Lansing, MI 48910. For fastest service, call now! 517/487-3358

B17 REVIEW
By Dan Poorman

We all know the reputation the TRS-80 has for cassette loading problems. Especially us Level II folk. So, when I first heard about B17, it sounded like a dream come true!

Yes, it works. Quite well, as a matter of fact, and it does save a heck of a lot of time. B17 is a bit of software magic that costs $25.00 and makes reading and writing to tapes quite a bit faster than the 500 baud rate that we're used to. As a matter of fact, the 'b' in B17 stands for BAUD, and the '17' indicates 1700 baud; over three times faster than our usual 500 baud rate. ABS Suppliers (the originators of B17) actually claim the program operates at 2125 baud, the highest rate that can be accommodated by the TRS-80 electronics. I didn't find any reason not to believe it.

I'll admit that I can't really measure the baud rate accurately, but I can tell you this much: Using B17, programs--both SYSTEM and BASIC--load in one third of the time with no problems at all. My modified Radio Shack Editor/Assembler loads in 44 seconds when under B17 control. Invasion will load in approximately 73 seconds, compared to it's usual 201 seconds. Before, my RSM-2 took 87 seconds, now it zooms into memory in about 27 seconds. Sargon zips into memory in just 68 seconds, compared to it's standard two minute and 38 second time.

Like everything else in this world that's good, there is a small price to pay (and I'm not referring to the $25.00, either); B17 is no exception. If your TRS-80 has the XRX-2 cassette loading modification by Radio Shack, you'll have to bypass it in order for B17 to work. It's not difficult at all to do, it just means breaking the seal on your machine--a move that does not impress Radio Shack service folks, should you ever need service on your computer.

If your computer does NOT have the modification, B17 is strictly a software routine.

I do have the XRX-2 modification, and did have to bypass it. Installing the switch took me about 10 minutes, and is quite simple to do, following the instructions that come with the program. After that, duck soup. The program works beautifully loading and writing tapes at high speed, and if I don't use B17, it's just a matter of throwing the switch (I installed) to the opposite position. This move activates the XRX-2 modification once again, and I can load 500 baud tapes nicely.

The B17 program requires a SYSTEM load, the file name being 'B17'. The cassette comes complete with various modules for loading both BASIC and machine language programs, and for machines with up to 48K of memory. I doubt that anyone would have 48K of RAM and not have disc, but the lads at ABS have a program for everyone.

At first, you'll see the familiar asterisks blinking up in the corner. When it shifts into high gear, you'll see, instead of the asterisks, an 'I' and an 'O' with a row of arrows in between. When B17 is inputting, the arrows will be pointing and moving towards the 'I'. When outputting, exactly the opposite. Neat touch, I think.

After duping the modules you need--in my case, 16K BASIC and high and low memory system--you are ready to convert your programs.

In BASIC, you set memory size to 31750 and load the BASIC B17 module. Then load the BASIC program. After loading the BASIC program, list it to make sure it loaded correctly. ANOTHER IMPORTANT NOTE: If the program is long, DO NOT run it! B17 takes up a little over 1K of memory, and you might not have enough room for RUNning, if the program has any arrays or needs a lot of string space. Once the program is loaded, slip a new tape into the recorder, push both the PLAY and RECORD buttons, type SAVE "filename", and you're on your way. After a short 500 baud pre-loader (this is written onto every B17 produced tape) you'll see the B17 verifier in the upper right hand corner, with the arrows pointing and moving towards the 'O' for output.

Once the program has been converted, you can verify it by typing 'LOAD?' in BASIC or using the 'V' command in machine language.

One of the nicest things about the program, I feel, is the LOAD AND GO option for system tapes. When you use the SYSTEM module, you can tell it the execute address, and when it is finished loading the program executes automatically!

Are there any negative points? Yes. My Adventure program--one that takes over four minutes to load and needs B17 badly--is too long to fit in my 16K machine along with the module to convert it. If I had 32K, there wouldn't be any problem, but I don't.

Other considerations, both good and bad: The program does not support a filename search, although it requires one to be input, even for the first program on the tape. When using B17 even without the XRX-2 modification, the level of the recorder is non-critical. You can always tell if there is a loading problem by watching the arrows up in the corner.

I guess the big question about any software is whether or not it's worth the price. In this case, you have a program that you will undoubtedly use every single time you load or write a program to or from tape. True, it will set you back $25.00, but that's beans compared to the purchase of a disc drive and interface expansion. B17 is neither as fast nor as versatile as disc, since it won't support data files, but it is certainly an improvement over the normal TRS-80 computer's cassette input and output.

Before closing, I will also pass along that ABS Suppliers tells me they'll soon be releasing a 'B29' that will allow the Editor/Assembler to read and write at the 1700 baud rate. That will bear watching for, as it just might make Assembly Language programming for the TRS-80 Level II more enjoyable, instead of an exercise in patience.

(B17 is available from TAS, or can be ordered directly from ABS Suppliers, P.O. Box 8297, Ann Arbor, MI 48107--Kos.)

<END>

****

THE ALTERNATE SOURCE (NEWS)LETTER

is in it's first year of publication. We are bi-monthly this year for one especially important reason: We intend to meet all deadlines, publication and otherwise. Also, there are many projects we are currently undertaking:

1. Coordinating information about a variety of software packages.
2. Seeking to provide support for existing software packages.
3. Providing a viable communication media between users, programmers and vendors--without costing an arm and a leg.
4. Giving you a valuable resource to find the answers for questions not answerable elsewhere.

Won't you join us? Subscriptions to TAS are minimal--nine dollars for six issues. Throughout your tenure as a subscriber we have several ideas that should make your subscription pay for itself--from various bargains on software and other info from time to time, to dynamite programs you can key in, to bulletins on the latest rumors!

We appreciate your support. And, we'll do our best to help you prove your TRS-80 is the excellent investment you always knew it was!

To subscribe, mail $9.00 with your name and address to: THE ALTERNATE SOURCE, 1806 Ada Street, Lansing, MI 48910. If you're not thoroughly convinced, $2.00 will get you a copy of TAS #3.

DISKMAP
By Larry Sylvia

(Larry's article was submitted by a friend, Fred Smith, who thought it should be published. After running it, we agreed. The article/program itself is sparsely documented, but this is because the program itself describes what is being displayed. We intend this to be one of several pieces of info relating to the all-important DIRECTORY track, #17 on most operating systems for the TRS-80.

Several tidbits have appeared explaining how to reclaim lost files--but only when working through the directory. Larry's program prints out which file is in each granule--a very important consideration, especially with 2.2 and 2.3, which completely eliminate the directory entry.

Incidentally, this program was written for a 40 track Percom drive, but works fine with 35 track drives--just don't become confused as the printer continues to print beyond the 35 tracks expected!--C.W.)

DISKMAP LISTING

```
10 CLS:PRINT"                   NEWDOS+ DIRECTORY SUMMARY PROGRAM"
20 FORI=1TO 5:PRINT:NEXT
30 INPUT"THE OUTPUT OF THIS PROGRAM IS DIRECTED TO A PRINTER.
   MAKE SURE  THE PRINTER IS ON. PRESS <ENTER> WHEN READY TO
   PROCEED.   ";X
40 LPRINT"                   NEWDOS+ DIRECTORY SUMMARY    PAGE 1":
   LPRINT
50 LPRINT"THE 2-GRANULE, 10-SECTOR(0-9) DIRECTORY ON TRACK 11
   (HEX), CONSISTS OF:"
60 LPRINT"SECTOR 0 - GRANULE(GRAN) ALLOCATION TABLE (GAT-SECTOR)
70 LPRINT"SECTOR 1 - HASH-CODE INDEX TABLE (HIT-SECTOR)"
80 LPRINT"SECTORS 2 THRU 9 - CONTAINING 'DIRECTORY ENTRYS' AS
   FOLLOWS:":LPRINT
90 LPRINT"     2 LINES (32 BYTES) PER ENTRY (IF FIRST 2
   BYTES=0000, FILE IS DEAD),"
100 LPRINT"     8 ENTRYS (FIRST TWO FOR SYSTEM) PER SECTOR
    (8*8 SECTORS=64 TOTAL).":LPRINT
110 LPRINT"IN THE DATA BELOW, THE 1ST COLUMN DENOTES THE
    DIRECTORY SECTOR, THE 2ND COLUMN DENOTES THE ENTRY (0-7)
    WITHIN THAT SECTOR. THIS IS FOLLOWED BY"
120 LPRINT"THE NAME AND MODIFIER OF THE FILE. THE NEXT COLUMNS
    CONSIST OF PAIRS OF (8-BIT)BYTES. THE 1ST BYTE DENOTES THE
    STARTING TRACK FOR THE FILE. THE"
130 LPRINT"FIRST 3 BITS OF THE SECOND BYTE DENOTES THE TRACK
    GRANULE THAT THE FILE STARTS ON (0=1ST GRANULE, 1=2ND
    GRANULE). THE RIGHT-MOST 5 BITS PROVIDE"
140 LPRINT"THE NUMBER-1 (ADD 1 TO IT) OF CONSECUTIVE GRANULES.
    FOR EXAMPLE, SYS0  HAS 22 (=00100010) FOR ITS 2ND BYTE.
    THEREFORE, SYS0 STARTS ON TRACK 0,"
```

<LISTING CONTINUED NEXT PAGE>

```
150 LPRINT"2ND GRAN, FOR 3 CONSECUTIVE GRANS. SYS2 IS ON TRACK
    10,2ND GRAN,ONE GRANONLY. DIR IS ON TRACK 11, 1ST GRAN,
    2 GRANS LONG.":LPRINT
160 DEFINTA-Z:CLEAR1000
170 DIM F$(16,13)
180 DIM FF$(80)
190 S$="              "
200 FOR D=0 TO 80:FF$(D)=S$:NEXT
210 H$="0123456789ABCDEF"
220 OPEN "R",1,"DIR/SYS"
230 ON ERROR GOTO 750
240 FOR L=1 TO 8
250 FIELD1,(L-1)*32 AS D$,1 AS F$(L,1),2 AS D$,1 AS F$(L,2),
    1 AS F$(L,3),8 AS F$(L,4),3 AS F$(L,5),2 AS F$(L,6),2 AS
    F$(L,7),2 AS F$(L,8),2 AS F$(L,9),2 AS F$(L,10),2 AS
    F$(L,11),2 AS F$(L,12),2 AS F$(L,13)
260 NEXT
270 ON ERROR GOTO 0
280 FIELD 1,255 AS D$
290 L=VARPTR(D$):L=PEEK(L+1)+256*PEEK(L+2):L=L-33
300 POKE L+15,&H11 : POKE L+16,&H01
310 POKE L+14,0 : POKE L+13,10 : POKE L+9 ,0
320 ON ERROR GOTO 670
330 FOR Z=3 TO 10
340 GET 1,Z
350 FOR F=1 TO 8:V=ASC(F$(F,1))
360 IF V=0 THEN 540
370 LPRINTZ-1;F-1;
380 IF V=144 THEN LPRINT,"      ";:GOTO400
390 LPRINTF$(F,4);"/";F$(F,5);"    ";
400 FOR VV=9 TO 13
410 IF F=8 AND VV=13 THEN 510
420 CC$=F$(F,VV)
430 C=ASC(LEFT$(CC$,1))
440 IF C=255 THEN 530
450 IF C<254 THEN 480
460 C=ASC(RIGHT$(CC$,1))
470 LPRINT "    ";(C AND 7)+2;(CAND224)/32;:GOTO530
480 GOSUB 690:LPRINT" ";
490 NEXT
500 GOTO 530
510 CC$=CHR$(PEEK(L+33+254))+CHR$(PEEK(L+33+255))
520 GOSUB 690
530 LPRINT
540 NEXT F
550 LPRINTSTRING$(64,"-");
560 NEXT Z
570 LPRINT
580 ON ERROR GOTO 0
590 CLOSE
600 FORI=1TO12:LPRINT:NEXT
610 LPRINT"                          NEWDOS+ DIRECTORY SUMMARY    PAGE 2":
    LPRINT
620 LPRINT"IN THE DATA BELOW, THE 1ST COLUMN DENOTES THE TRACK
    NUMBER ON THE DISK. THE 2ND AND 3RD COLUMNS DISPLAY THE
    FILES ON EACH OF THE TWO GRANULES ONEACH TRACK.":
    LPRINT:LPRINT
```

<LISTING CONTINUED...>

```
630 FOR D=0 TO 78 STEP 2
640 C$=CHR$(D/2):GOSUB680:LPRINT" ",FF$(D),FF$(D+1)
650 NEXT
660 STOP
670 RESUME 350
680 LPRINTMID$(H$,((ASC(C$)AND240)/16)+1,1);
    MID$(H$,(ASC(C$)AND15)+1,1);:RETURN
690 C$=LEFT$(CC$,1):AT=ASC(C$):GOSUB 680:C$=RIGHT$(CC$,1):
    AR=ASC(C$):GOSUB680
700 AT=AT*2:AD=(AR AND 224)/32:AC=AR AND 31:AT=AT+AD
710 FOR LL=AT TO AT+AC
720 FF$(LL)=F$(F,4)+"/"+F$(F,5)
730 NEXT
740 RETURN
750 RESUME 270
```

<END OF LISTING>

Diskmap produces a map similar to that found on page 35 of H. C. Pennington's "TRS-80 Disk & Other Mysteries".

****

NEWDOS-80 !!

Now in the final stages of preparation! Here's the scoop: NEWDOS-80 contains many, many improvements over NEWDOS+! It also includes new features, such as: New parameters for BASIC! 256 byte fielding! Set BREAK, LIST or PASSWORDS on or off! DU--duplicate line numbers! Accommodates 35, 40 and even 77 track formats! Even the Users Manual is improved! All this and more, and you can reserve yours today!

The Alternate Source was the first dealer to place an actual order for NEWDOS-80; you can receive the fastest delivery possibde by acting now!

NEWDOS-80 is $149.00. If you already own NEWDOS+, and can prove it, you can upgrade for only $65.00.

We suggest you order by phone, as the copies we have ordered are going fast! Your credit card slip will not be deposited until your copy is in the mail. Copies shipped Priority Mail or UPS Blue Label (your choice) on the day we receive them!

For persons wishing to upgrade, the fastest method is to ship us your original NEWDOS+ diskette.

Yes, even the best can be improved. NEWDOS-80, the super improved version of NEWDOS+, is available from THE ALTERNATE SOURCE, 1806 Ada Street, Lansing, MI 48910, or by phoning 517/487-3358. Hurry--be the first person in your users group to own one!

Patchwork (orange?)

by Chuck

Old timers to the Alternate Source (i.e. from last issue) will remember the problems we had with the Electric Pencil. Kept jumping back to Basic and all. We had a similiar problem with Allan Moluf's DVR program. Whenever the <BREAK> key is depressed and SYS1 overlays are in RAM, there is an invalid op code (during the test to see if the <BREAK> key IS depressed) which gives control of the system to the left-fielder, who scratches his head and finally throws the ball into the cheap seats (Level II, would you believe?) OK, I'm semi-sorry. The fault wasn't entirely with E/P. (Some folks just never say die!-Kos.)

On most R/S diskettes, your SYS1 should be starting on track 10h (break out the Zapper!). The problem is within that first sector of SYS1. This is next to impossible to implement in memory; besides, if you Zap your diskette, you won't have to worry about it again.

The culpret is located at track 10H, sector 0, byte DFH. The instruction as it comes is: LD HL,(430FH)

The op code for this is 2A0F43. The instruction we actually want is: LD HL,430FH

The op code for this instruction is 210F43. Simply replace the byte at DFH, 2A, with 21. The effect of this instruction is to determine if interrupts are on or off as a machine language program is executed. Depending on whether the program has a DI (disable interrupts) or not, your program can find very interresting places to wonder--usually to ENTER MEMORY SIZE.

My sincere thanks to both Allan Moluf and Jim Mullin for pointing out this particuliar fix--I understand it is incorporated into NEWDOS and is documentated along with this should you be interested in more information.

<END>

(Note from Joni: Since 'TRS-80 Disk and Other Mysteries' I believe Superzaps and Z80zaps have gotten more workout than all combined in the last two years! We would give special priority to other information which help to make the TRS-80 the reliable machine it can be. If you have any suggestions, please pass them along. Reminder: all of our authors are compensated, at least minimally!

The Alternate Survey

1. The vast number of TRS-80's in the market place are resulting in a large number of software packages being made available--too many to review in one issue, and still maintain a balanced content. Would you object to possibly one issue per year being devoted to 'catching up' on various products available?

2. Would you like to see 'comparison studies' of similar products, for example, utilities, monitors, various business packages? Would you have any specific suggestions for reviews?

3. Is price a major concern when you purchase software?

4. Were you disappointed with the print quality of this issue?

5. Have you written a program that you would be interested in marketing?

As always, we appreciate your opinions in directing the future goals of TAS. One way to make this easier (for both of us) is via the 'psuedo survey' format. Like before, we will award a $10.00 software certificate to the three lucky persons who are blessed by the fickle finger of fate (actually, it's Peggy, Joni's sister). Winners from the last issue will be announced next issue.

Regarding question 4, a special apology to Mr. Rohr and Jim Mullin. They both commented favorably on the bold type font.

Mail your viewpoints to TAS Survey #2, 1806 Ada Street, Lansing, MI 48910. Winners will be announced Issue #4.

****

The Disassembled Handbook for the TRS-80

This is the book that does what the others don't! Learn how to set up and use TRS-80 ROM routines! Complete examples (in source code!) included! This book will teach you more about the 'inner workings' of the TRS-80 than any other on the market! The most informative and accurate source of information currently available!!

Just to give you an idea of what the book is like, type in and RUN this small program:

```
20 CLS:?"FUNCT=ADDRESS      LSB-MSB         FUNCTION=
   ADDRESS      LSB-MSB"
25 A=6176:FORX=5712TO6175:Y=PEEK(X):IFY>127THENY=Y-128
30 Z=X+1:IFPEEK(Z)>127THEN?CHR$(Y);" = ";ELSEGOTO45
35 A=A+2:IFA=6352THENA=5640
40 ?A,PEEK(A);"-";PEEK(A+1),:GOTO50
45 ?CHR$(Y);
50 NEXT
55 END
```

Disabling the BREAK Key

by Allan Moluf

This subject came to light when Ken Edwards told me of his research with the BASIC statement: poke 16396,n where he had tried the different values of N from 0 to 255 and recorded the effects of pressing the BREAK key for each different number. He said a note in the Chicago TRS-80 Users' Group newsletter (CHICATRUG) had said that the break key could be disabled by POKEing a certain value there. Although he did not know why it worked, some numbers did disable the BREAK key, some had no effect and others caused strange effects such as jumping back to the MEMORY SIZE? question. This article will explain how it works.

First of all, the TRS-80 ROMs have a subroutine which scans the keyboard to see if any keys are depresssed. (This subroutine is located at 03E3H-0457H; it does the work when 002BH is called to get a keyboard character.) It returns zero if no character has been pressed or else returns the ASCII value for the character. It also makes a special check for the BREAK key and calls 0028H if the BREAK key is detected; at 0028H is a jump to 400CH (which is 16396 in decimal).

```
0028   C30C40    JMP   400CH      ;This is RESTART 28H
002B   111540    LD    DE,4015H   ;4015H = keyboard DCB
002E   18E3      JR    0013H      ;To device I/O routine

0453   FE01      CP    01H        ;1 is the BREAK key
0455   C0        RET   NZ         ;Return if not BREAK
0456   EF        RST   28H        ;CALL 0028H for BREAK
0457   C9        RET              ;And then return
```

In Level II BASIC (but not Disk BASIC), we have the following instructions starting at 400CH:

```
400C   C9        RET              ;This is address 16396
400D   00        NOP
400E   00        NOP
400F   C9        RET              ;For RESTART 30H
4010   00        NOP
4011   00        NOP
```

Now if you change the instruction at 400CH to one which makes A zero, such as XOR A (which has the opcode 0AFH), the routine which looked for a keyboard character will get zero (meaning no key) instead of 1 (meaning the BREAK key). For example, you could POKE 16396,175 which puts 175 (or 0AFH) into 16396 (or 400CH), giving us:

```
400C   AF        XOR   A          ;Sets A to zero
400D   00        NOP
400E   00        NOP
400F   C9        RET
```

Now when the BREAK key is pressed and the keyboard scan routine calls 0028H, it jumps to 400CH and puts zero in A, does nothing twice and then returns. The net effect is to return saying no key was pressed.

If the BASIC function INKEY$ is used to get a single character, it would return with an empty string (zero characters) instead of a single character with ASCII value of 1.

Another instruction such as INC A would cause the BREAK key code of 1 to be changed to a code of 2. The BREAK key would still be disabled but INKEY$ would now return with a single character whose ASCII value is 2.

All of these instructions work because the RET at 400FH gets executed after the instruction in 400CH and the NOPs in the the next two bytes. If the instruction POKEd into 400CH is some kind of jump, the TRS-80 will jump to the address in 400DH-400EH. Since this is zero, we go back to the MEMORY SIZE? question.

All well and good until we get to Disk BASIC. Now we have the following instructions starting at 400CH:

```
400C  C3A24B    JP   4BA2H      ;For RST 28H in DOS
400F  C3B444    JP   44B4H      ;For RST 30H in DOS
```

Here 400CH is used for both the BREAK key detected by the keyboard subroutine and for DOS to load its overlays for things like OPENing and CLOSEing files. Now POKEing around in 400CH is going to cause trouble. The routine at 4BA2H determines which of the two cases it was (overlay requests always have A > 127 whereas the BREAK key calls always have A = 1).

```
                     ;--- DOS I/O vector area ---
4312  C34D5D    JP   5D4DH      ;For BREAK in KBD scan
4315  00        NOP             ;For BREAK to DEBUG
4316  00        NOP
4317  00        NOP

                     ;--- DOS processing area ---
4BA2  E3        EX   (SP),HL    ;BREAK, DOS overlays
4BA3  E1        POP  HL
4BA4  B7        OR   A          ;See if not overlay
4BA5  F21243    JP   P,4312H    ;If bit 7 is zero

                     ;--- Disk BASIC ---
5D4D  FE01      CP   1          ;See if BREAK key
5D4F  C8        RET  Z          ;Return if so
5D50  C34BA0    JP   4BA0H      ;Else go return a zero
```

Thus when you hold down the BREAK key and the keyboard is scanned, the RST 28H is performed which jumps to 400CH and from thence to 4BA2H which diddles around and determines that bit 7 is not set (so it was a BREAK key restart) and then proceeds to jump to 4312H which normally jumps to BASIC at location 5D4DH. We can prevent Disk BASIC from detecting the BREAK key by the following:

```
POKE &H4312,&HAF: POKE &H4313,&HC9
```

Those commands will zero A and return, simply disabling the BREAK key for Disk BASIC with no side effects.

An alternate way is to change the data byte at 5D4EH. We could change it to anything other than one so the BREAK key would be turned into no key. The Chicago area newsletter gives an address of 23461 for disabling the BREAK key for NEWDOS. This is hex 5BA5H, corresponding to 5D4EH in the above listing. The advantage of using 4312-4313H is that it will work with either TRS-DOS or NEWDOS.

The next locations 4315H-4317H are also useful. When you do a CMD"D" to get into DEBUG from Disk BASIC and then go back to BASIC with a G command, pressing the BREAK key always gets you to DEBUG. This is because DEBUG places a jump to DEBUG in 4315H-4317H and when the real-time clock interrupts (40 times per second), it checks to see if the BREAK key is depressed and 4315H is non-zero. You can get back to normal use of the BREAK key after by either

```
CMD"T"
```

which turns off interrupts or by:

```
POKE &H4315,0
```

which makes DOS think DEBUG is not wanted anymore.
****

ERRATA -- Issue #1

Make the following changes to the "HANGMAN" program...
Line 5, change GOSUB34 to GOSUB33. Line 7, change GOTO32 to GOTO31. Line 17, change GOSUB33 to GOSUB32. Line 18, delete RETURN. Delete the first line 26. Add SET(66,15):RETURN to the end of line 25. Line 31, change GOSUB34 to GOSUB33. Line 4, delete PRINT'0,CHR$(32):. Line 5, after GOSUB33, insert PRINT'64,CHR$(30):.

Re: EMBED MACHINE LANGUAGE IN BASIC. To set up the USR(0) functions, you must POKE 16526 with PEEK(X+1) and POKE 16527 with PEEK(X+2). It should also have been noted that when using a string array, you cannot POKE in either a CHR$(0) or a CHR$(34).

FROM THE SOURCE'S MOUTH

By Joni M. Kosloski

I think there are some assumptions and/or pretensions being propagated, especially by programmers and software houses, which only serve to weaken the software industry. If you even have a fantasy of writing a 'best seller' (program) just as soon as you get your machine mastered, give this some thought now.

I'm talking about software protection. I recently did some systems work for a local company. As I was explaining various aspects of computing to this very specialized firm, the subject of software protection came up. I got into an interesting debate with an attorney who believes that software is protected. He has yet to show me proof. Nor have I seen it in any of our favorite publications.

Creative Computing seems to be a leader in uncovering supposed copyright violations, yet they haven't given adequate documentation to the rules, rights and regulations to back up their stance.

At a recent users group meeting, I stirred up this hive of bees (at least one lawyer was present). Several comments were batted back and forth, from "You're buying the right to use it on one system", to "You can make and give away all the copies you want, as long as you don't sell them". Unfortunately (or is that fortunately?), the meeting was about to close, so debate was curtailed. I have a feeling the latter comment is closer to the truth. Prove me wrong.

TAS is concerned with getting a lot of good software into the hands of many people...at reasonable prices. By the same token, we're concerned for the programmer who puts many hours into developing a good program. How's he gonna feed his kids?

I have seen both ends of the spectrum; persons who distribute fast and freely with no discrimination, to persons who wouldn't give away a bit! (Pun intended?!?) I currently walk a fine line. I can't afford all the software I want (not that I'd ever use it all), yet I don't believe in unlimited exchange. My feelings toward co-op purchases, especially for major packages, are more positive than negative. How about trades for equal value? If you didn't swap comic books when you were a kid, you missed an important indoctrination into the capitalistic way!

The only difference between men and boys is the price of the toys.

TRS-80 RAM Locations

by Allan Moluf

This is a summary of the BASIC and DOS uses some of the RAM memory locations used by Level II BASIC, Disk BASIC and DOS itself. Locations whose use is not well understood are marked by "???". Locations by either BASIC are noted as "BASIC - " and the locations used by Radio Shack's TRS-DOS are noted by "DOS - ". (NEWDOS generally uses the same locations.)

This is only a brief index to some very interesting ideas. A proper listing of the purpose and usage of each location would require much more time than I have available, and undoubtedly cause this material to not see print for many more months. May it be of some use as it stands.

Some other sources of information do exist. I can especially recommend Bob Richardson's "Disassembled Handbook for the TRS-80" for its explanation of the numerical subroutines and Roger Fuller's "TRS-80 Supermap" for its detailed commentary on the entire Level II ROM. Also, the article by Wes Thielke in the February 1980 issue of "80 Microcomputing" gives a very readable discussion of some of the useful ROM subroutines.

This is the first revision of this material, which was originally published in the CMTUG newsletter, for the TRS-80 users in the Lansing Michigan area. I welcome corrections or any further information which may be included in later revisions and elaborations of this wonderful world of the TRS-80.

```
00010 ;        RSTnn is jumped to by ROM after 'RST nn'
00020 RST08    EQU     4000H   ;BASIC - CHECK CHAR, SKIP SPACES
00030 RST10    EQU     4003H   ;BASIC - SKIP SPACES
00040 RST18    EQU     4006H   ;COMPARE HL - DE
00050 RST20    EQU     4009H   ;BASIC - GET TYPE OF TOP OPERAND
00060 RST28    EQU     400CH   ;DOS - BREAK/OVERLAY REQUESTS
00070 RST30    EQU     400FH   ;DOS - BREAKPOINT FOR DEBUG
00080 RST38    EQU     4012H   ;DOS - INTERRUPT SERVICE ROUTINE
00090 KBDCB    EQU     4015H   ;KEYBOARD DCB
00100 VDDCB    EQU     401DH   ;VIDEO DISPLAY DCB
00110 LPDCB    EQU     4025H   ;LINE PRINTER DCB
00120 DOSJMP   EQU     402DH   ;RETURN TO DOS
00130 DOSEN2   EQU     4030H   ;RETURN TO DOS COMMAND LEVEL
00140 DEVECT   EQU     4033H   ;DOS - DEVICE VECTORING ROUTINE
00150 KEYMEM   EQU     4036H   ;KEYBOARD KEY MEMORY (7 BYTES)
00160 PORTFF   EQU     403DH   ;CURRENT PORT FF OUTPUT BITS
00170 ;                         Bit 3 selects 32 BIG chars per line
00180 ;                         Bit 2 turns on the cassette tape
00190 ;                         Bits 0 or 1 are set for positive and
00200 ;                         negative pulses to the cassette.
00210 ;                         These values take effect when OUTput
00220 ;                         to I/O port 0FFH.
```

```
00230 ;                   403E-403F        ???
00240 TIME      EQU       4040H    ;TIME: 25MS SEC MIN HR YR MON DAY
00250 DOSEND    EQU       4047H    ;DOS - END OF DOS RAM
00260 MEMTOP    EQU       4049H    ;DEFAULT TOP OF MEMORY
00270 INTBIT    EQU       404BH    ;DOS - COPY OF INTERRUPT REGISTER
00280 INTMSK    EQU       404CH    ;INTERRUPT MASK
00290 BRKEP     EQU       404DH    ;DOS - ADDR OF BREAK ENTRY POINT
00300 ;                   404F-4058        ???
00310 DSKWT     EQU       4059H    ;DOS - ADDR OF WAIT FOR DISK RTN
00320 DISP      EQU       405BH    ;DOS - ADDR OF TASK DISPATCHER
00330 ;                   405D-408D        ???
00340 USRADR    EQU       408EH    ;BASIC - USR ROUTINE ENTRY ADDR
00350 ;                   4090-4092        ???
00360 INPRTN    EQU       4093H    ;BASIC - START OF INP SUBROUTINE
00370 OUTRTN    EQU       4096H    ;BASIC - START OF OUT SUBROUTINE
00380 KEYCHR    EQU       4099H    ;BASIC - INKEY$ CHAR STORAGE
00390 ERRCOD    EQU       409AH    ;BASIC - ERROR CODE STORAGE
00400 PRTPOS    EQU       409BH    ;BASIC - PRINTER HORIZ POSITION
00410 DEVTYP    EQU       409CH    ;BASIC - I/O DEVICE TYPE
00420 ;                   409D-409F        ???
00430 STRBEG    EQU       40A0H    ;BASIC - START OF STRING SPACE
00440 ;                   40A2-40A3        ???
00450 PGMBEG    EQU       40A4H    ;BASIC - START OF BASIC PROGRAM
00460 CURPOS    EQU       40A6H    ;BASIC - CURSOR HORIZ POSITION
00470 IBUFAD    EQU       40A7H    ;BASIC - INPUT BUFFER ADDRESS
00480 ;                   40A9H            ???
00490 RNDVAL    EQU       40AAH    ;BASIC - RANDOM VALUE (3 BYTES)
00500 ;                            used as seed for next random number
00510 ;                            middle byte changed by RANDOM
00520 ;                   40AD-40AE                  ???
00530 ACCTYP    EQU       40AFH    ;BASIC - TYPE OF NUMBER IN ACCUMULATOR
00540 ;                   40B0             ???
00550 STREND    EQU       40B1H    ;BASIC - END OF STRING SPACE
00560 NEXTOP    EQU       40B3H    ;BASIC - STRING OPERAND POINTER
00570 OPSTK     EQU       40B5H    ;BASIC - STRING OPERAND STACK
00580 NSTLEN    EQU       40D3H    ;BASIC - NEW STRING LENGTH
00590 NSTADD    EQU       40D4H    ;BASIC - NEW STRING ADDRESS
00600 STRPTR    EQU       40D6H    ;BASIC - NEXT STRING POINTER
00610 TOKADD    EQU       40D8H    ;BASIC - NEXT TOKEN ADDRESS
00620 ;                   40DA-40DE        ???
00630 PROGEP    EQU       40DFH    ;BASIC - 'SYSTEM' - PROGRAM ENTRY
00640 AUTOF     EQU       40E1H    ;BASIC - AUTO NUMBERING FLAG
00650 CURLIN    EQU       40E2H    ;BASIC - CURRENT LINE NUMBER
00660 AUTINC    EQU       40E4H    ;BASIC - AUTO NUMBERING INCREMENT
00670 TOKPTR    EQU       40E6H    ;BASIC - TOKEN POINTER
00680 STKPTR    EQU       40E8H    ;BASIC - POINTER TO STACK POINTER
00690 ;                   40EA-40EB        ???
00700 EDITNO    EQU       40ECH    ;BASIC - EDIT LINE NUMBER
00710 ;                   40EE-40F4        ???
00720 LASTLN    EQU       40F5H    ;BASIC - LAST LINE NUMBER EXECUTED
00730 ;                   40F7-40F8        ???
00740 VARBEG    EQU       40F9H    ;BASIC - START OF VARIABLES
00750 ARRBEG    EQU       40FBH    ;BASIC - START OF ARRAYS
00760 FREBEG    EQU       40FDH    ;BASIC - START OF FREE MEMORY
00770 DATPTR    EQU       40FFH    ;DOS - NEXT 'DATA' POINTER
```

```
00780 DEFTYP   EQU  4101H    ;BASIC - DEFAULT TYPES FOR A - Z
00790 TRONF    EQU  411BH    ;BASIC - TRACE ON FLAG
00800 ACCUM    EQU  411DH    ;BASIC - 8 BYTES FOR PSEUDO-ACCUMULATOR
00810 ;                      first 4 bytes are only used for double
00820 ;                      precision values; next two bytes are
00830 ;                      the interger value or the least two
00840 ;                      bytes of a single precision number.
00850 ;            4125-4126        ???
00860 ACCUM2   EQU  4127H    ;BASIC - 8 BYTES FOR EXPRESSION VALUE
00870 ;                      First two bytes are for an integer, or
00880 ;                      the first 4 bytes are for a single
00890 ;                      number or all 8 for a double precision.
00900 ;            412F-4151        ???
00910 ;            4152H    ;BASIC - "CVI"
00920 ;            4155H    ;BASIC - "FN"
00930 ;            4158H    ;BASIC - "CVS"
00940 ;            415BH    ;BASIC - "DEF"
00950 ;            415EH    ;BASIC - "CVD"
00960 ;            4161H    ;BASIC - "EOF"
00970 ;            4164H    ;BASIC - "LOC"
00980 ;            4167H    ;BASIC - "LOF"
00990 ;            416AH    ;BASIC - "MKI$"
01000 ;            416DH    ;BASIC - "MKS$"
01010 ;            4170H    ;BASIC - "MKD$"
01020 ;            4173H    ;BASIC - "CMD$"
01030 ;            4176H    ;BASIC - "TIME$"
01040 ;            4179H    ;BASIC - "OPEN"
01050 ;            417CH    ;BASIC - "FIELD"
01060 ;            417FH    ;BASIC - "GET"
01070 ;            4182H    ;BASIC - "PUT"
01080 ;            4185H    ;BASIC - "CLOSE"
01090 ;            4188H    ;BASIC - "LOAD"
01100 ;            418BH    ;BASIC - "MERGE"
01110 ;            418EH    ;BASIC - "NAME"
01120 ;            4191H    ;BASIC - "KILL"
01130 ;            4194H    ;BASIC - "&"
01140 ;            4197H    ;BASIC - "LSET"
01150 ;            419AH    ;BASIC - "RSET"
01160 ;            419DH    ;BASIC - "INSTR"
01170 ;            41A0H    ;BASIC - "SAVE"
01180 ;            41A3H    ;BASIC - "LINE"
01190 ;            41A6H    ;BASIC - Disk BASIC error messages
01200 ;            41A9H    ;BASIC - "USRn"
01210 ;            41ACH    ;BASIC - Disk BASIC re-entry
01220 ;            41AFH    ;BASIC -
01230 ;            41B2H    ;BASIC -
01240 ;            41B5H    ;BASIC -
01250 ;            41B8H    ;BASIC -
01260 ;            41BBH    ;BASIC -
01270 ;            41BEH    ;BASIC -
01280 ;            41C1H    ;BASIC -
01290 ;            41C4H    ;BASIC - Scan keyboard extension
01300 ;            41C7H    ;BASIC - "RUN" extension
01310 ;            41CAH    ;BASIC - "PRINT" extension
01320 ;            41CDH    ;BASIC - "PRINT" extension #2
```

```
;                41D0H   ;BASIC - "PRINT" extension #3
;                41D3H   ;BASIC - "PRINT" extension #4
;                41D6H   ;BASIC - "INPUT" extension
;                41D9H   ;BASIC -
;                41DCH   ;BASIC - "READ" extension
;                41DFH   ;BASIC -
;                41E2H   ;BASIC - "SYSTEM" extension
DOSSTK   EQU     41E5H   ;DOS - STACK AREA
;                41E8 is the start of the BASIC input buffer
;                for Level II BASIC.
DIRREC   EQU     4200H   ;DOS - DIRECTORY SECTOR BUFFER
;                4288 is the stack pointer for SYSTEM.
CURTRK   EQU     4300H   ;DOS - CURRENT TRACK ON 4 DRIVES
DIRTRK   EQU     4304H   ;DOS - DIRECTORY TRACK ON DRIVES
CURDRV   EQU     4308H   ;DOS - CURRENT DRIVE #
DRVBIT   EQU     4309H   ;DOS - CURRENT DRIVE SELECT BIT
DCBADD   EQU     430AH   ;DOS - DCB ADDRESS FOR ERRORS
RETADD   EQU     430CH   ;DOS - RETURN ADDR FOR ERRORS
DOSFCN   EQU     430EH   ;DOS - LAST DOS OVERLAY FUNCTION
DBGFLG   EQU     430FH   ;DOS - FLAG FOR DEBUG ON/OFF
;                4310-4311       ???
BRKVEC   EQU     4312H   ;DOS - BREAK KEY VECTOR
DBGVEC   EQU     4315H   ;DOS - DEBUG VECTOR
CMDBUF   EQU     4318H   ;DOS - 64-BYTE COMMAND BUFFER
;                4358-43FF       ???
RSTDOS   EQU     4400H   ;DOS - RESTART DOS COMMAND LEVEL
DOSCMD   EQU     4405H   ;DOS - EXECUTE DOS COMMAND
DSPERR   EQU     4409H   ;DOS - DISPLAY ERROR MESSAGE
BEGDBG   EQU     440CH   ;DOS - BEGIN DEBUG (LIKE CMD"D")
ADDTSK   EQU     4410H   ;DOS - ADD REAL-TIME TASK N
DELTSK   EQU     4413H   ;DOS - DELETE TASK N FROM LIST
CHGTSK   EQU     4416H   ;DOS - CHANGE CURRENT TASK ADDRESS
ENDTSK   EQU     4419H   ;DOS - DELETE CURRENT TASK
PARCMD   EQU     441CH   ;DOS - PARSE DOS COMMAND
FINIT    EQU     4420H   ;FILESYS - OPEN OLD OR NEW FILE
FOPEN    EQU     4424H   ;FILESYS - OPEN OLD FILE
FCLOSE   EQU     4428H   ;FILESYS - CLOSE AND SAVE FILE
FKILL    EQU     442CH   ;FILESYS - CLOSE AND KILL FILE
FLOAD    EQU     4430H   ;FILESYS - LOAD PROGRAM FILE
FEXEC    EQU     4433H   ;FILESYS - EXECUTE PROGRAM FILE
FREAD    EQU     4436H   ;FILESYS - READ RECORD FROM FILE
FWRITE   EQU     4439H   ;FILESYS - WRITE RECORD TO FILE
FVERF    EQU     443CH   ;FILESYS - WRITE AND VERIFY
FREW     EQU     443FH   ;FILESYS - MOVE TO RECORD 0
FPOSN    EQU     4442H   ;FILESYS - CHANGE FILE POSITION
FBACK    EQU     4445H   ;FILESYS - MOVE BACK 1 RECORD
FSKIPF   EQU     4448H   ;FILESYS - MOVE TO END OF FILE
;                444B-4466       ???
DSPMSG   EQU     4467H   ;DOS - WRITE MESSAGE TO DISPLAY
PRTMSG   EQU     446AH   ;DOS - WRITE MESSAGE TO PRINTER
GETIME   EQU     446DH   ;DOS - GET FORMATTED TIME
GEDATE   EQU     4470H   ;DOS - GET FORMATTED DATE
;                4473H   ;DOS - OVERLAY SYS1 - 5
;                4476H   ;DOS - OVERLAY SYS1 - 6 - PARSE
;                4479-44AF       ???
;                44B0-44B3  FROM 4409
;                44B4-44B7  FROM 440D
;                44B8-44CE  FROM 4033
```

```
01910 ;                 44CF-44DE   FROM 4467
01920 ;                 44DF-44EE   FROM 446A
01930 ;                 44EF-44FF       ???
01940 TASK0    EQU      4500H    ;DOS - TASK (8/SEC) 0
01950 TASK1    EQU      4502H    ;DOS - TASK (8/SEC) 1
01960 TASK2    EQU      4504H    ;DOS - TASK (8/SEC) 2
01970 TASK3    EQU      4506H    ;DOS - TASK (8/SEC) 3
01980 TASK4    EQU      4508H    ;DOS - TASK (8/SEC) 4
01990 TASK5    EQU      450AH    ;DOS - TASK (8/SEC) 5
02000 TASK6    EQU      450CH    ;DOS - TASK (8/SEC) 6 - CLOCK
02010 TASK7    EQU      450EH    ;DOS - TASK (8/SEC) 7 - TIMER
02020 TASK8    EQU      4510H    ;DOS - TASK (40/SEC) 8
02030 TASK9    EQU      4512H    ;DOS - TASK (40/SEC) 9
02040 TASK10   EQU      4514H    ;DOS - TASK (40/SEC) 10
02050 TASK11   EQU      4516H    ;DOS - TASK (40/SEC) 11 - TRACE
02060 ;
02070 ;                  4518-4CFF  DOS RESIDENT ROUTINES
02080 ;                          IN NEW RELEASES OF DOS SOME OF
02090 ;                          THESE ADDRESSES ARE LIKELY TO
02100 ;                          CHANGE, BUT A FEW INTERESTING
02110 ;                          ROUTINES ARE DESCRIBED HERE:
02120 ;
02130 DSKRD    EQU      46DDH    ;DOS - READ PHYSICAL SECTOR
02140 DSKWR    EQU      46E6H    ;DOS - WRITE PHYSICAL DATA SECTOR
02150 DSKWRD   EQU      46EFH    ;DOS - WRITE DIRECTORY SECTOR
02160 GETDIR   EQU      4AC1H    ;DOS - READ FILE DIRECTORY ENTRY
02170 MPY24    EQU      4B6AH    ;DOS - MULTIPLY C * HL -> AHL
02180 DIV16    EQU      4B84H    ;DOS - DIVIDE HL / A -> HL, A
02190 ;
02200 ;                 4D00-4DFF  G.A.T. SECTOR BUFFER
02210 ;                 4E00-51FF  DOS OVERLAY AREA
02220 ;                          SYS1 - DOS command input
02230 ;                          SYS2 - file opens
02240 ;                          SYS3 - file closes
02250 ;                          SYS4 - displays DOS error messages
02260 ;                          SYS5 - the DEBUG routine
02270 ;                 5200-XXXX  IS AVAILABLE FOR PROGRAMS
02280 ;                          SYS6 - extended DOS commands
02290 ;                          such as 'DIR' and 'KILL'
02300          END
```

****

FYI...

The Alternate Source is compiled (or is that interpreted?) bimonthly by Charley Butler and Joni Kosloski at 1806 Ada Street, Lansing, MI 48910, along with the help of many fine folks. We have not decided on a formal Masthead because too many changes are taking place. Should you desire to communicate with any of the authors, we will gladly forward mail to them. Should you care to comment, criticize, evaluate, suggest, contribute or otherwise extract your two cents worth, we're all ears. All correspondence should be addressed to the editor at the above address.

AVAILABLE SOFTWARE...

QUILL DRIVER--Format your written reports, letters, documentation, articles, and more! Quill Driver is written in BASIC, but has embedded machine language routines to provide the speed you need! Text can be entered as BASIC REM statements (easy to edit!) or in EDTASM format; numerous commands will accommodate ANY type of formatting. Comes with extensive manual and sample document files, for just $29.95 complete! Manual alone, $5.00, refundable on later purchase.

SPOOLER--Don't be I/O bound! Put SPOOLER to work, and printing becomes a background activity! Spooler reserves a 4K buffer in memory, routes printed material to buffer area, and returns control of the machine to you. WORKS FOR LEVEL II OR DOS!! All versions on one tape, instructions for making disk file are included. The price? Just $16.95!

SUPERMAP--Now that you've spent hours disassembling Level II ROM and RAM, not to mention a fortune on computer paper, whaddaya do with it? You need SUPERMAP! SUPERMAP is an expertly annotated listing of all relevant memory locations in the Level II ROM and RAM. Hundreds and hundreds of comments and explanations--an essential tool! A SPECIAL PRICE, TOO--Just $8.95!

FROM AUTOMATED SIMULATIONS--Two new programs! Morloc's Tower and The Datestones of Ryn. Find and slay the vicious Morloc the Warlock! Recover the valuable datestones stolen by the notorious bandit! Hours and hours of action--just $14.95 each!

ISAR--Information Storage And Retrieval! ISAR is THE low-cost data base management system for micros! Hobby, Business, Mailing lists, Formatted Reports--ISAR handles them all! Uses random file structures for increased speed! Easy user prompts, clear documentation. Nothing beats ISAR! Seven modules--Menu, Create a File, Add Records, Change or Delete Records, Sort A File, Scan or Search File, LPRINT and Format Reports. Requires DOS. On diskette for $16.95, cassette for $13.95. Also available separately, ISAR8, which merges two ISAR files with the same specifications, and ISAR9 ,which allows you to add a field or lengthen a present field. On cassette only, $6.00 for both.

FRESH OFF THE PRESS AND AVAILABLE NOW--THE ALTERNATE SOURCE'S FIRST SOFTWARE CATALOG!

ASK FOR YOURS TODAY--NO CHARGE!! (Features special discounts!)

DISASSEMBLER 1.2--This program is written in machine language and will disassemble machine code into ZILOG Z-80 mneumonics , with symbolic labels for address and 16 bit references within the start-to-end disassembly request. Output can be directed to CRT, PRINTER, or TAPE CASSETTE. A source tape suitable for loading into the Radio Shack Editor/Assembler is produced. Specify memory size. FOR LEVEL II ONLY! TAPE I/O only. $15.00.

DISASSEMBLER 2.0--Provides disk I/O! This version is compatible with the Apparat disk extended EDTASM, too! Several useful commands in addition to the disk extension. Will run in 32K or 48K; TRSDOS, NEWDOS or VTOS required. $20.00.

CASSETTES--A Baker's Dozen! Thirteen (13) high quality Agfa tapes--they hang on to your bits like they were gold! Soft plastic cases included. Just $10.95 plus $1.00 postage. Order double quantity, we pay postage!

DISKETTES--Ten (10) Verbatim top quality diskettes, with sleeves and soft box, are just $26.50! Sorry, but this price is so low that we've got to ask you to include a dollar postage. Order double quantities, we pay postage!

ALSO AVAILABLE--Just $14.95 each: Android Nim, Snake Eggs, Life Two, and Beewary (all with sound!); Adventures 1 - 8 on cassette; Back-40, Dr. Chips, Space Battle, Super Invader, and Galactic Empire!

ALSO AVAILABLE--Just $9.95 each: Owl Tree, Great Race, Scramble, Lying Chimps, Challenge, Concentration (all with sound!)!

ALSO AVAILABLE--Just $3.95 each: Number Base Converter; Electric Pencil File Conversion; Freakout (a sound extravaganza!!); Machine Language to Basic Data Statement; Mail List File to Upper/Lower Conversion!

TO ORDER: Send check or money order to The Alternate Source, 1806 Ada Street, Lansing, MI 48910. Visa & Master charge by phone, 517/487-3358. Include 50 cents per program for first class postage, otherwise shipped fourth. Don't forget to ask for your FREE catalog, featuring our entire software selection!

PLAYING GOD WITH LIFE--The familiar game of Life, with an intriguing touch! In this version, you get to play GOD--enter BIRTH mode to give life! Enter DEATH mode to take life! In machine language, too! Order from Dennis Kitsz, Roxbury, VT zip 05669. (That's a full address!)

ODDS & ENDS

Considering Telecommunications?

We at The Alternate Source feel that telecommunications will soon be playing a major role in the field of microcomputers. So we've been investigating.

Our latest discovery is of a "UNIVERSAL MODEM" package that will be a combination modem and RS-232. It will plug right into the back of your Level II 16K keyboard, or your expansion interface, if you're a disk user. The modem directly connects to insure greater transmission reliability. the best part is the price--it will be less than the combined price for the similiar R/S configuration. The RS-232 is also user accessible. Right now there are only prototype models available and we have one on the way. Also, information is on the way--it should be here before the actual model. If you are interested in such a unit, why not let us know? Are there particular features you desire on such a unit? Please tell us about them. We'll look forward to hearing from you, and you can look forward to more information in Issue #3!

Monitoring the Media

We recently acquired a copy of "The Best of Personal Computing". The magazine sells for $7.50 and is jam-packed with applications for business or home. They all aren't directly written for the TRS-80, but could be modified with a minimum of hassle. Copies can be obtained by writing Personal Computing at 1050 Commonwealth Avenue, Boston, MA 02215.

The March, 1980 edition of Interface Age has an excellent tutorial on Fortran, using the Microsoft compiler for the TRS-80.

The first two issues of 80-Microcomputing were fabulous and terrific! Let's hope they can keep up the good work!

Would still encourage everyone to purchase a copy of the current 80 Software Critique. With more and more packages for the TRS-80 being sold, it's hard to know what to buy and what to shun!

Fuller Electronics

has a nifty little device that plugs into the back of your keyboard--fools the computer into thinking there's a printer attached. LPRINTs won't hang up the system anymore! Write Fuller Electronics for more info: 7465 Hollister Avenue, Suite 232, Goleta, CA 93017.

No Time For Entering Programs?

The Alternate Source now offers all programs contained within a given issue available of cassette or diskette for those persons desiring. This will benefit those of you who have little time for keying in programs, and will also eliminate potential 'BUG' problems. Cassettes can be ordered for $5.00, diskettes for $7.50. Mail check to TAS, 1806 Ada St., Lansing, MI 48910. Be sure to specify issue number.

****

Information for ISAR owners...

ISAR was originally designed to provide personal users with a low cost data management system. When introduced, however, its reception has been most frequent for small business applications. In order to fully support the myriad of applications to which it has been applied, several changes and improvements have been made. If you would like to take advantage of these improvements, The Alternate Source will provide, on diskette, the latest updates to the ISAR modules for $5.00, if you include your original diskette with your request. MAil to: ISAR Update, 1806 Ada Street, Lansing, MI 48910.

Current version numbers for the individual modules are:

Module 1=1.0	Module 2=1.0	Module 3=1.11
Module 4=1.2	Module 5=1.31	Module 6=1.2
Module 7=0.95	Module 8=1.12	Module 9=1.1
Module 5b=1.33		

****

In the Buffer...

We are going to discontinue this feature for this reason: No less than three of the articles in this issue came in with less than a week before press time. If we become more formatted, we will not be able to accomodate these late articles, be they exciting or not. In addition, no less than two people are sending in more data for our printer review (promo from last issue). If we don't maintain a certain degree of flexibility (by not obligating ourselves to fill any given issue with promised articles), we feel our service to you will not be as unique. We will provide clues sometimes. All promised articles will appear. Thanks.

****

MAY THE SOURCE BE WITH YOU! (Sorry, couldn't resist!)

TRS-80 is a registered trademark of the Tandy Corporation.

AFTERWARD FOR ISSUE 2

Probably the most valuable article in this issue is Allan Moluf's RAMSTUFF. We have received several comments on it and plan a more complete listing -- someday.

Note the first "Public Domain" software by Roxton Baker, who later penned "TRAKCESS".

B-17 from ABS Suppliers is still around and only recently has been receiving acclaim well deserved.

TWO DOLLARS

VOLUME 1, NUMBER 3

THE ALTERNATE SOURCE

THE MAGAZINE OF ADVANCED APPLICATIONS
AND SOFTWARE FOR THE TRS-80.

IN THIS ISSUE:

Editorial RAMbling...

Well, with taxes out of the way and Spring finally showing it's more pleasant side, I hope this issue of TAS finds you in good spirits!

Before you get issue #4 (we've got some dynamite articles for it already!) you should receive a small communique from Joni and/or myself called 'Between the Issues'. We're attempted these before with reasonable success and there are some projects we would like to develop that are more apropos in that type of communique. Some of our older subscribers will not be too surprised; the only distinction is that BTI will become, at least per our intentions, regular. This will allow us to handle certain items on a more timely basis, it will help interface with those who desire a monthly publication until that reality can be within our grasp and it will allow us to interface with you on a more personal level, something which may or may not be proper within the confines of this magazine as it is evolving.

We're really proud of some of the articles in this issue and hope they meet with your satisfaction. Please allow me to direct your attention to the survey (on the middle pages). The concept is not entirely new, but I believe it has a definite place in the development of any magazine providing compensation and appreciation for both readers and authors.

As of this date, we have very few commitments from authors for regular articles. I guess the primary implications of this for you is that you are guaranteed a wide variety of articles. We are looking for good articles for future issues and welcome your contributions. We pay for all articles published.

Above all, we welcome your comments, suggestions, feedback, hisses, boos, rahs and the like. We are truly amazed at the span of knowledge that exists among users for the TRS-80. We still seem to have the middle ground covered by our in-house folk, but branching from that are the experts and beginners. Our job now is to try and interface those two groups! Wish us luck...

April 19, 1980
Charley Butler

The Alternate Source is published bi-monthly by Charley Butler and Joni Kosloski at 1806 Ada, Lansing MI 48910. Any letters to any author will be forwarded expediently. Subscription rates are $9.00 per year. Write for advertising info.

HANDS OFF!

By Dennis Bathory Kitsz

(Dennis is destined to become a classic name amongst the TRS-80 inner circle. We've had several favorable comments on his last article, and he has two articles in this issue. This first article is for the Level II audience who are frustrated with reserving memory sizes for their SYSTEM programs.)

Are you tired of typing starting addresses for SYSTEM tapes? Or do you tend to forget a start address, and have to shuffle through your documentation? Take heart! Your own SYSTEM programs can be made to execute themselves in Level II. Once you have the idea, the process is easy.

There are several ways of interrupting the SYSTEM loading process to jam in your starting routine. Understanding this process will allow you to effect this auto-start a number of ways, two of which will be presented here.

How, then, does the SYSTEM process execute in the usual fashion? You type SYSTEM, and the machine responds with the special *? prompt. You then enter the program name, and the tape is searched for a corresponding name; the program is loaded into place, a closing address (if designated by the programmer) is left in the HL register, and a *? is returned, waiting for you to enter the starting address.

Question: what is happening when the *? prompt is waiting for you to enter the starting address? Yes, that's it. The keyboard is being scanned for a value. What can be done is simple: PATCH INTO THE KEYBOARD SCAN while the program is being loaded. In Level II, a program can be loaded in as many discrete blocks of memory as desired...including, of course, the two bytes employed by the keyboard scanning routine.

Here's how I did it: the loading program loads itself as well as two bytes into that keyboard scan routine at addresses 4016 and 4017 (hex). When the machine returns the *? prompt, it attempts to scan the keyboard, but instead is diverted to the entry point of the program you wish to execute. But there remains one end to tie up...the first command of your program must REPLACE the original bytes into 4016 and 4017. If you are using Level II without utilities (such as a debounce routine), these bytes will be E3 and 03, representing the jump address of 03E3. Here is my loader routine, with a few bells and whistles:

```
        ORG     3C00H           ;This is the first video
                                 location
```

```
     DEFM     'LOADING'         ;This writes "LOADING" on
                                 the screen
     ORG      4016H             ;This is the keyboard scan
                                 routine
     DEFB     nn                ;Define your program's
                                 entry LSB
     DEFB     nn                ;Define your program's
                                 entry MSB
     ORG      nnnn              ;This is where your program
                                 starts
     LD       HL,4016H          ;Get the keyboard scan loca-
                                 tion again
     LD       (HL),0E3H         ;Put original scan MSB back
                                 in place
     INC      HL                ;Increment address to 4017H
     LD       (HL),03H          ;Put original scan MSB back
                                 in place
                                ;
                                ;Your own program goes here
                                ;
     ORG      3C40H             ;This is second line on
                                 video screen
     DEFM     'GOOD LOAD'       ;This message helps indicate
                                 correct load
     END
```

This system has three advantages: first, it lets you know for sure that the program has STARTED to load. Often that's pretty vague, even if the stars do flash. Next, it lets you know as best it can that the load was IN SYNC. Incorrect bytes may get by (the "C", or checksum, error), but a lost bit will create true havoc. The message is reassuring. And finally, the program takes right over, executing without a pause.

You could also patch into the video driver, which is likewise used in virtually all applications of the computer, including the return from SYSTEM load.

Another interesting way of autostart is to patch into the SYSTEM load at its point of re-entry, which is a CALL to location 41E2. If you recall, I earlier mentioned that the starting address, if you have provided one in your END statement, is left in the HL register. We could execute at JP (HL) from location 41E2, as the usual byte found at 41E2 is merely C9, a RETURN command. Using the same method, then, the program below may be used (in this example, the LOADING and GOOD LOAD display instructions have been omitted):

```
     ORG      41E2H             ;This is the system patch
                                 before return
     DEFB     E9                ;This is the JP(HL) command
     ORG      nnnn              ;Your program starts here
```

```
    LD      HL,41E2H         ;Get the system patch back
    LD      (HL),0C9H        ;This restores the original
                              return
                             ;
                             ;Your program goes here
                             ;
```

I tend to use the former method (keyboard scan patch) because it is more "transparent". As one of the most used functions, it is likely to be examined often by machine-language enthusiasts. The 41E2 SYSTEM patch may be used by overlapping programs, and should be treated with more care.

In any case, this convenience of self-loading programs is not available only to those with disc (or even stringy-floppy); by taking this approach a few steps further, it is possible even to chain-load a group of machine language programs.

Thanks are due to Jeffrey Eisen of Huntington Valley, PA, for reminding me of patch found at 41E2. Correspondents may write to me (Roxbury, Vermont 05669), or call me at (802) 485-6112.

VARPTR TIP

Tip on how to find the variable name and variable type using VARPTR: Since you must already know the variable name in order to use VARPTR to find the name, there is probably not too much value in knowing how to do it. It is nonetheless one of those interesting pieces of information that may come in handy in a pinch, so here it is. If the variable name is AZ, then

```
PEEK(VARPTR(AZ)-1) = The first character of the variable
                     name = 'A'
PEEK(VARPTR(AZ)-2) = The second character of the variable
                     name = 'Z'
```

If the variable did not have a second character in it's name, that byte would be set to an ASCII zero, CHR$(0).

```
PEEK(VARPTR(AZ)-3) = A one-byte number indicating the
                     variable's type:  2) Integer,
                     4) Single Precision, 8) Double
                     Precision, 3) String.
```

I'll leave it to you to figure out where the number codes come from.

(This information is from an actual Meta Technologies Bulletin released September 10, 1979. For more information on how you can receive their Bulletin Service, see page 19.)

REAL B.S.

SUBJECT: TRS-80, Live Keyboard/Display Control

PROBLEM(S)/ISSUE(S): A very small, high speed, live keyboard/display routine featuring real-time character validation, restricted input field length, flashing "block" cursor and supporting full special character control functions is required.

IMPLICATIONS: Real-time character validation will limit data entry errors and simplify coding of option-selection procedures. If technology employed is of a level that is not critical from a competitive standpoint, it may be released in low-cost software, education, etc.

RECOMMENDED ACTION(S): Routine meeting these criteria (with demo program) follows:

Contact Bob Fiorelli
(216) 289-6600

```
5 GOTO100
6 REM
7 REM **** DATA ENTRY/DISPLAY  (C) SEPT. 1979
   METATECHNOLOGIES CORPORATION, INC. ****
8 REM Q=CURSOR POSITION, FL=FIELD LENGTH,
   ML=MIN. ENTRY LENGTH (OR = ZERO FOR N/A)
9 REM FV$=FIELD VALUE, VC$=VALID CHARACTERS,
   CC$=CONTROL CHARACTERS (RETURNS IC <> ZERO)
10 S$=FV$:LS=LEN(S$)
20 PRINT@Q,STRING$(FL,136);:PRINT@Q,S$;
30 PRINTC4$;:K$=INKEY$:PRINTC5$;:IFK$=""GOTO30
   ELSEIC=ASC(K$)
40 IFINSTR(VC$,K$)=0GOTO50   ELSEIFLS<FLTHENS$=S$+K$:
   PRINTK$;:LS=LS+1:GOTO30   ELSE30
50 IFIC=8ANDLS>0THENLS=LS-1:S$=LEFT$(S$,LS):GOTO20
60 IFIC=13AND(LS>=MLORML=0)THENFV$=S$:IC=0:RETURN
70 IFIC=24THENLS=0:S$="":GOTO20
80 IFIC=9GOTO10
90 IC=INSTR(CC$,K$):IFICRETURNELSE30
100 CLS:Q=540
110 VC$="0123456789.+-":FL=10:ML=3
120 CC$=CHR$(91)+CHR$(10)+CHR$(27)+CHR$(26)
130 C4$=CHR$(14):C5$=CHR$(15)
140 GOSUB10
150 PRINT@715,"FV$=";FV$,"IC=";IC
160 INPUT"HIT ENTER";A$:GOTO140
```

NOTE:

↑	=	CHR$(91)
↓	=	CHR$(10)
shift ↑	=	CHR$(27)
shift ↓	=	CHR$(26)

STORING VIDEO DISPLAY FRAMES IN MEMORY FOR LATER RECALL

By Robert M. Richardson

SYNOPSIS

Here is an interesting exercise that will allow the user to store 5 complete video display frames in MEM or later recall. Storage is called by pressing '456'. The 115 byte assembly language program and MEM storage only require 5235 bytes total, so it will operate easily with any 16K MEM system. It operates equally well with non-disk Level II, DOS 2.1, DOS 2.2, DOS 2.3, and NEWDOS+. The program may be entered in about 5 minutes using an Editor/Assembler.

INTRODUCTION

There are many occasions when TRS-80 programmers would like to store data from the video display to RAM memory for later recall and review, thus allowing selective 'JKL' LPRINT of that data desired for permanent hard copy record. Other applications include duplex telephone line MODEM operations where incoming data may be selectively stored in MEM for leisurely recall, as well as Morse code and/or radio teletype systems, et al.

This program is self-executing when loaded with DOS; i.e., it will operate in DOS or disk Basic with no further SYSTEM and /27000 commands. With non-disk Level II it is loaded using standard cassette procedures.

PROGRAM LOGIC AND FLOW

Since the 'JKL' keys, when pressed simultaneously, are the NEWDOS+ clue to LPRINT the video display, we will use the '123' keyboard keys to tell our assembly language program when to store a complete video display frame, and the keyboard '456' keys to tell our program when to recall a frame. Following Dave Lien's advice to "KISS" (Keep It Simple, Stupid), we will limit this demonstration program to storing 5 video display frames; i.e., 1024 X 5 = 5120 bytes. Further simplification includes:

1. Automatic sequencing of the stored video display frames.
2. If a 6th video frame is stored, it will "wipe-out" the previous #1 frame. If a 7th frame is stored, #2. Etc.
3. Recall also includes automatic sequencing.

4. The number of the stored frame is displayed in the lower right hand corner of the video display.

An assembled version of the source code for the program follows this article. This demonstration program is located at 27000 decimal so that it will work with any Level II 16K MEM system, whether non-disk or otherwise. It may be relocated in MEM anywhere you wish by changing lines 120, 130, 150, 160, and 170 appropriately.

Let's run through the program briefly, even though the comments are largely self-explanatory. The label COUNT is the 1 byte MEM location where the number of frames stored, 1 to 5, is stashed. The label PLACE is the 2 byte MEM location containing the address in MEM where each 1024 byte video display frame is stored. Lines 170-190 are a straightforward way of poking the START address into the video display control block's driver address at 401EH and 401FH, thus making the program self-executing when loaded via DOS. Lines 220 & 230 switch alternate Z80 register pairs so as not to foul-up any machinations that may be going on in either your BASIC or assembly language program that may be running. Remember, the alternate register pairs AF', BC', DE', and HL' are never used by LEVEL II ROM. Additionally , the EX and EXX opcodes take 4+ bytes less MEM than PUSH and POP opcodes. Lines 230-250 are optional and serve only to remind the user "how many frames" have been stored or "which frame" is being recalled. Lines 260-300 are the real 'work horses' of the program, in that they simply test the keyboard's number row to see if either '123' or '456' keys are pressed. Lines 310-330 restore the original register pairs and then jumps back to the normal video display routine at 0458H. The entire subroutine, so far, requires only about 24 microseconds, so be assured it will not disrupt most any program you are running.

STORE moves the 1024 byte video display to PLACE whenever it is called, resets COUNT to 1 if 5 frames have been stored, updates COUNT by +1, and then returns to lines 310-330 for a normal exit back to your program.

RECALL is virtually identical to STORE except that now we will reverse the procedure and move the stored 1024 bytes from PLACE to the video display before returning to exit lines 310-330. RESET simply reinitializes COUNT to +1 and PLACE to 27200 decimal, which is where we started. If you have enough memory to store 10 to 20 video display frames, it is very easy to modify the program to do so. All that is necessary is to change lines 390 and 540 to: "CP 20" (for storing 20 video display frames). If you chose to leave the frame number display, lines 230-250, in the program it would display:
frame 10 = :, 11 = ;, 12 = <, 13 = '=', 14 = >, 15 = ?, 16

= ', 17 = A, 18 = B, 19 = C, 20 = D, unless modified.

PROGRAM SUMMARY

This is really a teaching program from Volume II of the TRS-80 Disassembled Handbook. Its logic is straightforward and its primary purpose is to accustom the user to moving video display data around as desired. It is the predecessor to the following Chapter, "Split Screen Video Display", which allows the user two totally independent video displays with individual scrolling, storage, and recall. For those of you who wish to dig deeper, we suggest you order a copy of Volume II of the "TRS-80 Disassembled Handbook" @ $15.00 from either The Alternate Source or from Richcraft Engineering Ltd., Drawer 1065, Chautauqua Lake, NY, 14722.

PROGRAM LISTING

```
00100 ;STORING VIDEO IN RAM DEMONSTRATION PROGRAM - SV8
00110
00120         ORG     6978H           ;=27000 DECIMAL
00130 COUNT   EQU     6978H           ;PROGRAM SAVES SPACE HERE
00140         DEFB    1               ;INITIALIZE BYTE AT +1
00150 PLACE   EQU     6979H           ;PROGRAM SAVES SPACE HERE
00160         DEFW    27200           ;BEGIN STORAGE 27200 MEM
00170 START   EQU     27003           ;PROGRAM WILL CAREFULLY-
00180         ORG     401EH           ;POKE 27003 AT 401EH AND-
00190         DEFW    START           ;401FH VIDEO DISPLAY -
00200         ORG     START           ;CONTROL BLOCK.
00210         EX      AF,AF'          ;EXCHANGE ALT. REGISTERS
00220         EXX                     ;EXCHANGE BC,DE,HL REGS
00230         LD      A,(COUNT)       ;NO. OF FRAMES STORED
00240         ADD     A,48            ;CHANGE TO ASCII NUMBER
00250         LD      (16382),A       ;DISPLAY LOWER RT. CORNER
00260         LD      A,(14352)       ;KYBD 01234567 NOS. ROW
00270         CP      14              ;SUB 14 = '123' PRESSED
00280         JR      Z,STORE         ;GOTO 'STORE' IF ZERO
00290         CP      112             ;SUB 112 = '456' PRESSED
00300         JR      Z,RECALL        ;GOTO 'RECALL' IF ZERO
00310 ELFIN   EX      AF,AF'          ;RESTORE AF REGISTER
00320         EXX                     ;RESTORE BC, DE, HL REGS
00330         JP      0458H           ;GOTO NORMAL VIDEO
00340 STORE   LD      HL,15360        ;BEGINNING VIDEO MEMORY
00350         LD      DE,(PLACE)      ;LAST HI MEM STORE
00360         LD      BC,1024         ;BYTES VIDEO TO MOVE
00370         LDIR                    ;GOTO IT!
00380         LD      A,(COUNT)       ;NO. OF FRAMES STORED
00390         CP      5               ;SUBTRACT 5
```

```
00400           JR      Z,RESET         ;GOTO RESET IF ZERO
00410           LD      HL,(PLACE)      ;LAST HI MEM STORE
00420           LD      DE,1024         ;NO. OF BYTES MOVED
00430           ADD     HL,DE           ;ADD THEM UP
00440           LD      (PLACE),HL      ;UPDATE STORE LOCATION
00450           LD      A,(COUNT)       ;NO. OF FRAMES STORED
00460           INC     A               ;ADD +1
00470           LD      (COUNT),A       ;UPDATE FRAMES STORED
00480           JR      ELFIN           ;GOTO THE END
00490 RECALL    LD      HL,(PLACE)      ;MEM STORE LOCATION
00500           LD      DE,15360        ;1ST LINE VIDEO LOCATION
00510           LD      BC,1024         ;NUMBER OF BYTES TO MOVE
00520           LDIR                    ;DO IT.
00530           LD      A,(COUNT)       ;NUMBER OF STORED FRAME
00540           CP      5               ;SUBTRACT 5
00550           JR      Z,RESET         ;GOTO RESET IF ZERO
00560           LD      A,(COUNT)       ;NUMBER OF STORED FRAME
00570           INC     A               ;ADD +1 TO 'A' REGISTER
00580           LD      (COUNT),A       ;UPDATE FRAME COUNTER
00590           LD      HL,(PLACE)      ;CURRENT FRAME LOCATION
00600           LD      DE,1024         ;BYTES NOW DISPLAYED
00610           ADD     HL,DE           ;ADD +1024 TO HL REG
00620           LD      (PLACE),HL      ;UPDATE PLACE LOCATION
00630           JR      ELFIN           ;GOTO THE END
00640 RESET     LD      A,1             ;REINITIALIZE AT +1
00650           LD      (COUNT),A       ;LOAD +1 IN COUNT MEM
00660           LD      HL,27200        ;REINITIALIZED AT 27200
00670           LD      (PLACE),HL      ;STASH 27200 IN PLACE MEM
00680           JR      ELFIN           ;GOTO THE END
00690           END     COUNT           ;EL FIN = EL BEGIN
```

IT'S FINALLY HERE!!

We now have in stock Volume II of the Disassembled Handbook for the TRS-80 by Richcraft Engineering. We're very pleased with it; it's 180 pages of very useful information, with no duplication from Volume I! There's a complete disassembled listing (about 60 pages) of the Level II ROM in Z80 mnemonics, something that has yet to appear anywhere. There's a detailed commentary on all Level II functions not included in Vol. I, including all string commands. There's an informative discussion of disassembly in general, with specific application to the Level II ROM. And there's more!

We think you'll find Vol. II of the Handbook extremely valuable. The price is only $15.00 plus $1.00 shipping (it's heavy!)--and it comes with a 100% money back guarantee! If you're not pleased with it for some reason, simply return it for a full refund. Order yours today! Just send $16.00 to The Alternate Source, 1806 Ada Street, Lansing, MI 48910. ASAP delivery, too!

For those desiring, Vol. I is still available for $10!!

80 AIDS

Dear Jesse Bob,
I'm writing a game called "EARTHQUAKE". I want to simulate a quake by rapidly switching the screen between 32 and 64 character mode. How can I do this without clearing the screen? Also, are you really a software cowboy?
Arnie Richter

Dear Arnie,
The following line will make your screen shake like a long-tailed cat in a room full of rockers:

```
100 FORX=1TO1000:OUT255,8:OUT255,0:NEXTX
```

With regard to your second question, Son--'ol Jesse Bob has been bustin' bits since you was knee-high to a Horned Toad! Right now me an' the boys ride herd on over 700,000 bits, and pardner, that's a bunch!
JB

----- (J) -----

Dear Jesse Bob,
My TRS-80 is driving me up the wall! Periodically, it reports a SYNTAX ERROR on a perfectly good line. Sometimes it won't boot. Every time I take it to the Radio Shack Service Center, the problems don't show up, and it works good for two or three weeks. What can I do?
Baffled in Buffalo

Dear Baffled,
This problem is a burr under the saddle of almost every '80 owner sooner or later. It is caused by a corrosion in the connection between the computer and the Exapansion Interface or between the E.I. and the disk. The edge connectors from most cables are gold-plated, while the TRS-80 printed circuits are not. When electrical current flows through this junction, it causes an electrolytic reaction that causes corrosion. When this corrosion gets bad enough it becomes hard for the computer to discriminate between logic 1 and logic 0 signals. Then all kinds of strange things can happen.
When you take the connection apart some of the corrosion comes off and it works again for a while. A better cure is to take an eraser (Pink Pearl by Eberhard-Faber is a good one) and rub it over the exposed edges of each P.C. card where the cables connect. Notice the black "crud" that the eraser picks up. Repeat whenever

erratic operation is noticed.

JB

----- (J) -----

Dear Jesse Bob,

Yesterday I spent four and a half hours typing in a program. Before I could save it, my computer locked up in the dreaded "Silent Death". It took four hours for the police and my wife to talk me in off the ledge.

What causes this malady, and how can I recover? If I push the RESET button the system either reboots, or goes to "MEMORY SIZE?", neither one of which lets me get my program back.

Despondent in Detroit

Dear Despondent,

The "Silent Death" is a snake that bites all of us sooner or later. The term "Silent Death" is used to describe a situation in which the computer apparently locks up and refuses to BREAK or accept any keyboard input, with no visible activity taking place.

What causes it? Anything which causes the computer to get "lost" while executing machine language instructions. This may, in turn, be due to many things. A bad bit in RAM, a "noise" glitch on the power line, or poor cable connections (see the previous letter). Not all such deaths are silent. Some will result in a re-boot while others will result in the system stack overflowing into the video RAM, leaving a trail of trash on the screen. The important thing to remember is that the Z80 microprocessor in a TRS-80 will execute anything that it finds on its data bus as instructions and is quite incapable of sorting out the logical stuff from random nonsense.

How do you recover? Well, it is essential that you regain control of the computer. This can usually be accomplished only by pushing the RESET button. When you do this a special signal is fed into the Z80 forcing it to execute its next command from memory address 0. This is the initialization routine in the Level II ROM. The first thing that this routine does is to check for the presence of an Expansion Interface in the system (by looking for the disk controller chip). If an E.I. is found, a program is initiated to read track 0, sector 0 (the boot program) into memory. This causes the DOS to be reloaded. If you are using a disk and running TRSDOS 2.3 or NEWDOS, it is possible to type the command 'BASIC *', which will reload Disk Basic without disturbing all of the pointers in memory, so that any program in memory may be reclaimed to be saved on the disk. The procedure is dependent on nothing in memory being destroyed, which is not always the case.

If the BREAK key is held down during a RESET, the disk test is bypassed and control passes to Level II at the "MEMORY SIZE?" question. This procedure leaves no option for recorery of lost data, and will cause complete initialization of the Level II RAM.

If no E.I. is present when RESET is pressed, control passes to the "READY" state of Basic, which allows quick recovery. Perhaps this is why the "Silent Death" so seldom strikes when one has no disk -- recovery is automatic.

How does one recover from "Silent Death"? Well, except for the 'BASIC *' trick mentioned above, there is no good method. In some cases it may not be good to recover, since memory may have been farkled (a software cowboy term for messed up) by the Z80 in its wanderings. Probably the best recovery is good defense. Test the RAM you bought from the door-to-door salesman before using it on critical data. Don't run your computer from the same outlet that your brother-in-law is using a power saw in. Keep your connectors clean (Pink Pearl). Use a power line filter if line noise is a problem. Keep good backup. When entering long programs, stop from time to time and "SAVE" what you have entered. This will prevent you from having to key in everything from the start. Remember the words of the prophet Mur-fee: "Blessed are the pessimists for they shall never be disappointed!"

JB

----- (J) -----

If you have a technical question that you have been unable to get an answer for, send it to TAS for review by Jesse Bob. Only a few letters will be answered in each issue. Technical consultation is also available by purchase. Simply inchude $6.00 with your question. If the answer is published, or cannot be provided within 60 days your check will be returned. Otherwise Jesse Bob or one of his wranglers will send you an answer by mail.

ABOUT JESSE BOB:

Jesse Bob Overholt is the proprieter of the famed Circle J Software Ranch. Located in the fertile plains of Carrollton, Texas, the Circle J is 180,000 microacres of prime ranchland. Jesse Bob's herd of bits produces fine software for TRS-80's all over the world. As Jesse Bob says, "Every little bit counts!"

(Jesse Bob qualifies for the 'author of the month' contest -- see details under SURVEY on page 22!)

(Part two of this series is scheduled for Issue #4. Dennis has offered to continue the series, but we will leave that up to you--if you liked the article, and would like to see it continued, please let us know!)

WHEN YOU TURN IT ON

POWER-UP ROUTINES OF THE TRS-80

By Dennis Bathory Kitsz

(Part One)

The initialization routine of the TRS-80 is a complicated and very interesting aspect of the computer. it must, of course, set up all the parameters that will be used by Basic programs, but it also conducts a series of tests and makes hardware adjustments to the device.

It has double-checks to assure the proper operation of memory, and to be certain that the parameters needed for proper operation of programs will be present. This article will take a look at the process of initialization, and how understanding this procedure is fairly essential to writing elegant machine-language programs for the TRS-80.

Let's look at the first few instructions.

```
ORG     0000H
DI
XOR     A
JP      0674H
```

At power-up, the Z-80 chip "homes in" on address 0000, and begins its execution there. The first action is significant: DI, Disable Interrupt, keeps the clock heartbeat generated by the expansion interface from disturbing any actions of the computer - particularly since the necessary software for handling that interrupt request is not in ROM, but rather a part of the Disc Basic routines, or what is now offered as "Level III" Basic.

So the interrupt is masked out. XOR A is the process of "exclusive-ORing" the accumulator. Exclusive OR is a logical operation which states: of two elements, either may be zero or one, but not both. Thus, whatever is present in the accumulator is "XORed" with itself. since each bit is identical with is Exclusive-Or partner, each bit will be set to zero. This effectively clears the accumulator.

The final instruction of the group, JP 0674H, "gets out of the way" of the Z-80's low memory, for it is in this area that the chip's "restart" codes - very frequently used subroutines - are found. Going on:

```
ORG     0674H
OUT     (0FFH),A
LD      HL,06D2H
LD      DE,4000H
LD      BC,0036H
LDIR
DEC     A
DEC     A
JR      NZ,$-0DH
```

After the jump to 0674H, the routine resets the output flip-flop at port ff (255 decimal). This flipflop controls both cassette functions and 32-character video, and by outputting the value in A (0, since it was exclusive-ORed earlier), the cassette will be off, no data will be present at its input, and video will come up normally.

Following this is an interesting (and encouraging) piece of code. Using the Z-80's powerful "LDIR" command, 54 bytes stored at address 06d2H are transferred to the RAM address area starting at 4000H. These are the most important pieces of information the TRS-80 must have, so the writers of this program took great care to insure that this transfer is certain. This LDIR instruction itself, for those unfamiliar with it, takes data stored at an address specified by register HL (in this case, 06d2H), and moves a block whose length is specified by register BC (36H, or 54 decimal), to the location indicated by register DE (4000H).

The interesting part is found just below. The value in A (0) is decremented twice (to FE), and the identical transfer instruction is repeated. This goes on until a reaches zero again - a total of 128 times! We may draw the conclusion that the Z-80 chip probably reaches full power and begins operating before memory gets to the point where it can accept data...therefore, the instruction is repeated over a period of approximately 14 milliseconds.

Now a portion of RAM is cleared to zero with the following few commands:

```
LD      B,27H
LD      (DE),A
INC     DE
DJNZ    $-2
```

Recall that after the previous set-up process, the accumulator again contains zero. Here, a block of RAM specified by the DE register (essentially where we left off) is loaded with zero. Using the fast "DJNZ" (decrement B, jump if not zero) instruction, 39 bytes are fixed at zero.

A few instructions follow that are very significant at power-up. Address 3840H contains the keyboard row where the "break" key sits. It is connected to data line 4; thus the instruction and 04 checks to see if it is held down. If it is being held down, the result of the and instruction will not be zero...and a jump to address 0075H will be made. This is why expansion interface owners without Disc must press that key at power-up:

```
LD      A,(3840H)
AND     04
JP      NZ,0075H
```

Since we have mentioned Disc, then, how does the TRS-80 find out that a Disc drive is in fact connected to the interface? The answer to that - and to the reason the computer "hangs up" when an expansion interface is connected without a Disc - is found in the next few bytes of code:

```
LD      SP,407DH
LD      A,(37eCH)
INC     A
CP      02
JP      C,0075H
```

The stack pointer is set at 407DH for use by potential future programs; it is out of the way of all the Basic pointers set up in the first data transfer, for experienced machine-language programmers, an obvious but important action.

The accumulator is then loaded with the contents of memory location 37ECH. There is no "memory" per se at address 37ECH; it is instead an instance of "memory mapping". That is, when reading this memory cell, a signal is sent to the expansion interface. That signal strobes information from the "Floppy Disc Controller" to the TRS-80. What will it find?

If no expansion interface is connected, there is no signal to strobe. Hence, the value will be floating, not pulled to ground (zero) on any bit. The computer sees all bits apparently "high" at this location, and interprets it as Binary 11111111, that is, Hex FF.

The next instruction increments the accumulator, in this case resulting in FF+1, or 00. In the next instruction this value is compared to 2. A compare (in effect a subtraction, but with no "result") will cause the carry flag to be set, since 0 minus 2 is negative.* (*Footnote: Why compare with 02? Why not just 01, as a carry would still be generated? My suspicion is that it is possible for those data lines to "float" in the low state. IN that case the CPU would "see" 00000000, with the INC a instruction resulting in a value of 01 - ;which is still incorrect. So a compare with 02 guarantees the presence or absence of the Disc Controller.)

Once the carry flag has been set, the instruction JP C,0075 would be executed, sending the program to address 0075H. For the moment, however, let's assume that an expansion interface is connected to the TRS-80.

The Floppy Disc Controller, when queried by the command LD A,(37ECH), will respond with 80H. Incremented ;by one it becomes 81H, and comparing it with 02 generates no carry. The JP C,0075H is thus ignored, and the program simply goes on to find:

```
        LD      A,01
        LD      (37E1H),A
        LD      HL,37ECH
        LD      DE,37EFH
        LD      (HL),03
```

The accumulator is set to 1, and address 37E1H is made to accept the contents of the accumulator. Again, 37E1H is a location "memory mapped" in the expansion interface. It simply selects Disc drive number. one. You may recall that the Disc operating program is always in drive number one; this assures that the TRS-80 turns on only the correct drive.

That done, it loads HL with the Disc Controller address (37ECH), and sends out an "acknowledge" signal - 03, or 00000011 Binary. This tells the Controller to start the Disc rolling. Register DE is prepared by loading it with the Disc's data address, 37EFH. Now:

```
        LD      BC,0000
        CALL    0060H
        ORG     0060H
        DEC     BC
        LD      A,B
        OR      C
        JR      NZ,$-3
        RET
```

This is a short, but very useful, subroutine. You may in fact want to call this yourself from time to time. Found at address 0060H is a simple delay loop - load the BC register pair (as is done just before the call instruction), and it is decremented and tested until it reaches zero. When it finally reaches zero, a return instruction sends the Z-80 back to the main program flow.

Why a delay? Merely to give the Disc drive time to come up to speed - again obvious, but very important. Moving ahead with this branch of the program:

```
ORG     06B2H
BIT     0,(HL)
JR      NZ,$-2
```

This is a loop which waits until the Disc control chip says "Okay, Disc is up to speed and everything looks pretty good", and sends along a zero. The program loop tests this bit until it receives a zero. And it is this loop which is maddening to you expansion interface owners who have no Disc drive. Like all the previous memory-mapped addresses, 37ECH will never have that zero forever - but we'll address that later (pun intended?). In the meantime, the loop has found the acknowledging zero sent by the floppy Disc Controller:

```
XOR     A
LD      (37EEH),A
LD      BC,4200H
LD      A,8CH
LD      (HL),A
BIT     1,(HL)
JR      Z,$-2
LD      A,(DE)
LD      (BC),A
INC     C
JR      NZ,$-7
```

The accumulator is cleared again, and the BC register is set to 4200H. This will be an area of RAM set aside for Disc use. 37EEH is loaded with 0, and 37ECH is loaded with byte 8CH. This says "Let's go, Disc!", and the Floppy Disc Controller starts sending in the bytes - it sends out a "here comes the byte" message to the computer via address 37ECH, and the machine loops until that signal shows up. When it does, the accumulator looks for what it finds in the memory location specified by the DE register. DE, you recall, is set to address 37EFH. This is the memory-mapped location through which the actual data will flow.

The accumulator picks up the data from DE, stores it in the RAM memory location indicated by BC (4200H); the next instruction increments register C so that location 4201 is ready. The program loops back, waits for another message from the Disc Controller, picks up a byte, and stores it. When register C finally reaches 0, ;pointing to address 4300H), the loop terminates. Then:

```
JP      4200H
```

Here the Disc system takes over completely. As you recall, starting at 4200H data from the Disc has been stored. By jumping to that location, the program direction is wrested from ROM and given to the first 256 bytes of the Disc system.

In the next issue of "The Alternate Source", we will look at the rest of the initialization process - how the TRS-80 knows how much memory is on line, what happens when some memory is bad, and how the words "memory size" make their appearance on the screen.

NEW FROM TAS!!

We have increased our product line immensly in the last two months. Subscribers will soon be receiving an updated flyer listing some of our new products. Others may write or call for further info. Among them:

We are now handling subscriptions for the MTC Bulletin Service, which is $36.00/year. Bulletins are similar to that on page 6, and are mailed first class when the news breaks, not just monthly. Free year-end index with all subs. Order through TAS and receive free notebook for storage! Write either: Meta-Tech, 26111 Brush Avenue, Euclid, OH 44132, or TAS. Be sure to indicate whether you would like your sub starting from the beginning or with the NEXT bulletin.

Inside Level II is now available from TAS for $15.95 postpaid (see description in Bulletin Board).

Orders should be mailed to: The Alternate Source, 1806 Ada Street, Lansing, MI 48910. Master Charge or Visa by phone: 517/485-0344 or 517/487-3358. We will ship COD.

MYSTERY PROGRAM

By Charley Butler

(What does it do? One may never know, unless one enters it into their TRS-80! This program was written by Charley in one of his more sadistic moods. We hope you find it interesting!)

PROGRAM LISTING

```
0 'THE ALTERNATE SOURCE MYSTERY PROGRAM
5 'IT'S EASIER TO DEMONSTRATE THAN EXPLAIN!!
10 FOR X = 28707 TO 28814 : READ A : POKE X,A : NEXT
20 POKE 16633,143 : POKE 16635,143 : POKE 16637,143
30 POKE 16634,112 : POKE 16634,112 : POKE 16634,112
40 POKE 16548,36  : POKE 16549 ,112
50 CLS:LIST
60 DATA0,68,112,10,0,76,69,65,82,78,32,65,76,76,32
70 DATA65,66,160,32,83,69,76,70,206,77,79,68,143,89,73
75 DATA78,71,0,102,112,20,0,89,79,85
80 DATA82,32,79,87,78,32,80,82,79,71,82,65,77,83,206
90 DATA206,67,79,77,73,78,71,32,83,79,161 ,0,140,112,30
100 DATA0,73,78,32,58,147,251,84,72,69,32,65,76,84,69
110 DATA82,78,65,84,69,32,83,79 ,85,82,67,69,32,78,69
120 DATA87,83,39,33,0,0,0,28
130 'DOUBLE CHECK DATA BEFORE RUNNING!
140 'IF ALL ELSE FAILS, TRY LISTING THE PROGRAM AGAIN!
```

Enjoy...

THE CODE BOOK

All About Unbreakable Codes and How to Use Them

BY MICHAEL E. MAROTTA

What information do *you* want to keep safe from the prying eyes of Big Brother?

- ***Financial records***
- ***Secret formulas***
- ***Directions to buried or hidden valuables***
- ***Or other sensitive data you want to keep*** **PRIVATE**

Yes, now *you* can write codes that are *unbreakable*! This amazing new book shows you how.

Complete detaiuls for writing unbreakable codes are given for:

- ***doing it by hand***
- ***using a pocket calculator***
- ***using a microcomputer***

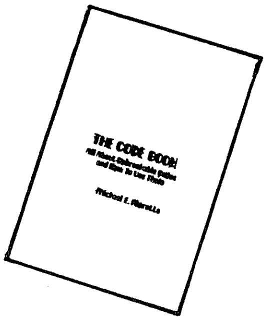
Contents include: What A Code Is; A Brief History Of Codes And Ciphers; How Codes And Ciphers Are Broken; How To Write An Unbreakable Code; Unbreakable Codes And The Home Computer; Obfuscation; Thoughts On Secrets In General; and more! Also iincluded are computer programs, tables of random numbers, instructions for generating your own random numbers, how to test a string of numbers for randomness, an extensive bibliography, and more!

Obscure secrets known only to international espionage agents and professional cryptographers -- now revealed for ***you*** *to use!!!* EVERYTHING you need to know to write your own unbreakable codes is covered in this astonishing new book!

Don't miss this one!!!

5½ x 8½, 76 pp, illustrated, perfect bound soft cover.
THE CODE BOOK: $6.95 + $1.00 shipping.

ORDER FROM:
Loompanics Unlimited
PO Box 264-S
Mason, MI 48854
USA



A "one-time" pad of the type used by international espionage agents to write unbreakable codes. Complete step-by-step instructions for writing unbreakable codes using a "one-time" pad are given in **THE CODE BOOK,** ***along with other ways of writing unbreakable codes.***

TAS SURVEY #3

First, the winners of our Survey #1 are:

Harry Maurer, of Succasunna, NJ
James Lisowski, of S. Milwaukee, WI
Patrick Morgan, of Los Angeles, CA

A ten dollar certificate has been awarded to these three gentlemen, which they can use in conjunction with ANYTHING offered by TAS. The certificates are just like cash, and may be used even with our special deals! We would like to extend our thanks to all who participated.

For this month's survey, we would like to rehash a theme used elsewhere. There shouldn't be much question that the thing that makes most magazines tick is the authors. We want to couple this month's survey with a method of rewarding both you and your favorite author.

Simply choose your favorite article (and author) from this issue of TAS, and write it on a separate slip of paper (along with your name and address), and send it to: TAS Survey #3, 1806 Ada Street, Lansing, MI 48910. We will award $10.00 certificates to three participants, and a $25.00 award to the winning author.

Winners of Survey #2 will be announced in Issue #4; winners of Survey #3 will be announced Issue #5.

THE ALTERNATE SOURCE
The Magazine Of Advanced Applications And Software

"I think your approach is terrific; a magazine devoted to someone other than the beginner has been sorely needed on the market for a long time."
Pendleton, IN

"If you keep this up, you'll not only have a winner, but you will also provide a valuable service to many, many TRS-80 computerists."
Houston, TX

"Along with the OCTUG Newsletter, yours is the best I've seen for important info on the TRS-80."
Atlanta, GA

"I'd like to say that I'm glad I took the gamble and subscribed to The Alternate Source News; it looks like I made the right choice. Keep up the good work!"
Makawao, HI

"Of the several TRS-80 publications I take, I like The Alternate Source News the best so far!"
Studio City, CA

"I was so pleased at being approached as if I knew something about computers and programming, that I not only read every word, but am also enclosing a check for nine dollars."
Canoga Park, CA

"I just received my copy of Issue #2 of TAS. I'm really impressed. You've done a great job!"
Waukegan, IL

"Okay, I'm hooked. Enclosed is my $9.00 check."
Franconia, VA

If you're serious about programming, you should seriously consider The Alternate Source News.

( ) Enclosed is $9.00 for six issues. Please start my subscription with Issue #_____ (1, 2, 3 or 4?).
( ) Enclosed is $2.00 for a single copy of Issue #___.

Name: ____________________
Address: ____________________
City: ____________________
State, Zip: ____________________

MAIL TO:
THE ALTERNATE SOURCE
1806 Ada Street
Lansing, MI 48910

OKIDATA PRINTER

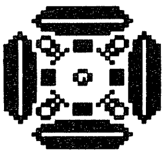
```
MICROLINE 80                                            AVAILABLE FROM:
9X7 DOT MATRIX                                          DREAM MACHINES

FEATURES :
          3 CHARACTER SIZES ( 16.5  10  &  5  CHAR./INCH)
          2 LINE SPACINGS  ( 6  &  8  LINES/INCH )
          2 LINE LENGTHS  ( LONG  &  SHORT  LINE MODE
          FULL TRS-80 +  GRAPHICS CHARACTER SET
          ALL PRINTING MODES SOFTWARE SELECTABLE
```

ELONGATED LETTER 5 CHAR./INCH

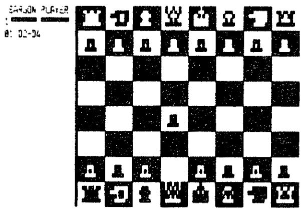
NORMAL LETTER 10 CHAR./INCH

CONDENSED LETTER 16.5 CHAR./INCH

```
 !"#$%&'()*+,-./0123456789:;(=)?@ABCDEFGHIJKLMNOPQRSTUVWXYZ↑↓←→_ abcdefgh ij mno
pqrstuvwxyz{|}~

         !"#$%&'()*+,-./0123456789:;(=)?@ABCDEFGHIJKLMNOPQRSTUVWXYZ↑↓←→_
         abcdefghijklmnopqrstuvwxyz{|}~

THIS IS 8 LINE/INCH                          THIS IS 6 LINE/INCH
THIS IS 8 LINE/INCH                          THIS IS 6 LINE/INCH
THIS IS 8 LINE/INCH                          THIS IS 6 LINE/INCH
THIS IS 8 LINE/INCH                          THIS IS 6 LINE/INCH
THIS IS 8 LINE/INCH                          THIS IS 6 LINE/INCH
THIS IS 8 LINE/INCH                          THIS IS 6 LINE/INCH
THIS IS 8 LINE/INCH
THIS IS 8 LINE/INCH
```



```
7 8 9  ROW=24
4 ■ 6
1 2 3  COL=24

C = CLEAR
D = DRAW
E = ERASE
F = FORTH
G = GO
L = LOAD
M = MOVE
R = REVERSE
S = SAVE
W = WHITE-OUT
! = INTERRUPT
GEN = 16
```



```
PRICE       $888.00     MICROLINE-80
            $120.00     TRACTOR FEED
            $ 35.00     CABLE
                +       TAX & SHIPPING IF APPLICABLE

FOR MORE INFORMATION CALL OR WRITE: BOB ZWEMER, DREAM MACHINES,
6408 S. WASH., LANSING, MI., 48910  (517) 393-9287 (AFTER 6 PM)
FOR A FREE SAMPLE SEND SASE
```

MICRODOS REVIEW

By William Perry

After finally making the decision to upgrade my 16K TRS-80 system to 32K and one disk drive, I was soon faced with another problem. TRSDOS (Radio Shack's disk operating system, required in drive 0) gobbles up about 40% of the usable diskette space. Many people have solved the problem by adding a second disk drive. This allows them to use all the diskette in drive 1, however another disk drive is a rather expensive solution. Some have simply used more diskettes than they had originally planned. I decided to try MICRODOS.

MICRODOS is published by Percom Data Co., Inc. of Garland Texas. It is easily used, and also very efficient. Although it was written for the 40 track (400 sectors, 255 bytes each) Percom drives, the documentation includes patches necessary to adapt it to the 35 track (350 sectors) Radio Shack drives. MICRODOS sells for $29.95 (much cheaper than a second drive!), and delivery time is about two weeks.

Initially, the printed documentation seemed somewhat lacking, but I found it sufficient and understandable after digging in. What really gets you on the right track is the close examination of the four BASIC language programs that are included on the system diskette. Briefly, these programs are:

PERCOM 5 1/4 INCH NOTEBOOK - This program describes all of the MICRODOS commands on the video display. This program also allows the user to add his own "pages" to the "notebook" for later reference. In addition to having the online documentation, listing the program will provide some insight on data file handling techniques.

DISK UTILITY PACKAGE - This program is menu driven and provides routines to format, backup, copy, erase, etc. all or part of a diskette.

FILE MANAGEMENT PROGRAM - This program provides a means of manually maintaining a directory on a data file. I was not too enthused about doing this manually. I found it simpler to write my own small menu program which would load and run the desired program by typing the number shown on the menu (see program following article for an example).

DISK DIAGNOSTIC TEST - This program puts the disk drive through its paces and reports any errors.

The advantages of MICRODOS are many; for example, LOADING SPEED. Basic programs that take 3 minutes to load from tape, or 16 seconds TRSDOS loading time, require just 4.6 seconds using MICRODOS. MICRODOS is MEMORY RESIDENT - at power-up, the entire disk operating system is booted into less than 7K of RAM, and no longer requires a system diskette in drive 0. But, if you decide to have MICRODOS on each diskette, it will only use sectors 0 through 19, leaving 380 sectors on a 40 track disk, or 330 sectors on a 35 track disk. MICRODOS is also efficient in its SECTOR ALLOCATION. Since disk space is allocated by sector, rather than 5 sector granules, there are no unavoidable gaps between programs or files. This obviously provides better diskette utilization.

Programming with MICRODOS is no headache--consider, for example, the following commands:

PUT X$,20	-	This will write the contents of X$ onto sector 20.
GET X$,20	-	This will read the contents of sector 20 into variable X$.
LOAD 20,R	-	This will load and run the program starting on sector 20. The R may be omitted if you do not want to immediately execute the program.
MERGE 40	-	This will add the program beginning on sector 40 to the one currently in memory.
CMD"I",0	-	This will format a blank diskette and write MICRODOS on sectors 0-19.

With MICRODOS, there is no need to worry about opening or closing a file (there is no directory). It also supports FIELD definition statements for packing several data items into one sector.

MICRODOS does have some shortcomings. For one thing, MICRODOS is strictly BASIC oriented; there are no provisions for storing a "system" loaded program on disk (other than MICRODOS itself). MICRODOS does not support such functions as DEBUG or CLOCK. And, as mentioned previously, MICRODOS doesn't feature a DIRECTORY. Since MICRODOS contains no automatic space allocation system, you must tell it which sector to read, write, save or load. This requires you to do some manual "bookkeeping" to make the most efficient use of a diskette or keep from overwriting another file. Although this sounds horrifying, it is no more trouble than recording your tape counter readings on your cassette labels. The SAVE command will tell you the last sector it used. The next SAVE, simply specify the next sector. Considering this in conjunction with the menu program following this article, I do not feel this problem is too significant.

Although MICRODOS has been heralded as THE disk operating system for a 16K, one drive system, few TRS-80 owners will find the remaining 8K sufficient space for most of their Basic programs.

In summary, I feel MICRODOS will compliment any disk system and will pay for itself in disk purchase alone. It is handy to have 25 or so programs plus the operating system on one side of one diskette. I intend to use it exclusively for Basic programs and data, but at present, TRSDOS is still a necessity for the machine language programs.

FIRST OFFERINGS FROM TAS !!!

BTRACE: LII or DOS. The TRS-80 TRON command can be a great aid to debugging programs. Unfortunately, it holds no respect for screen prompts. BTRACE displays the current line being executed in one convenient position, and preserves the current cursor location.

CPU: LII or DOS. Compress Program Utility removes all unnecessary spaces and remarks from a Basic program with complete regard for unclosed strings, unusual commands (such as ERL and when RUN is embedded in your program).

SEARCH: LII or DOS. Search lists any Basic program lines containing an occurance of a STRING select. You can break at any time to make changes, or continue. Option allows LPRINTED listing.

Each of these utilities are, we feel, the best (or only) available of their type. The price of each is **$19.95** on cassette. Add **$2.00** for diskette. Some factors you may wish to take into consideration when examining competitive software products:

1. Each is written in Z80 assembly language for fast execution.
2. Each is complete with easy to understand instruction.
3. Each comes complete with a special relocating module to allow you to create a disk or tape file at any address you specify! This will allow you to use each program in conjunction any other machine language routines or utilities you may wish to use.

AND THERE'S MORE !!

RELOC: allows you to incorporate the relocating module mentioned above with your own machine language programs. You can now have one version that will be compatible with drivers, GSF, RSM or whatever other machine language programs you use. RELOC is **$29.95 on cassette** or **$31.95 on diskette.** Sample file and instructions included.

REPLACE: allows you to search and replace occurances in your Basic programs saved (saved as ASCII files). **$14.95 on cassette, $16.95 on diskette.**

CHANGES: generates a screen or printed listing of differences between versions of programs you are developing. **$14.95 on cassette, $16.95 on diskette.**

PPD: is a parallel print driver that allows you to pass parameters to do such things as 'bypass' LPRINTED output, insure that the line printer is ready, and more. **$9.95 on cassette, $11.95 on diskette.** Add **$3.00** for source code.

SPECIAL INTRODUCTORY OFFER: (for as long as author permits) Two special package deals! 1) BTRACE, CPU, SEARCH, REPLACE and CHANGES are available at the package price of just **$49.95!** You save an incredible $39.80!! 2) ALL of the above programs are available for only **$69.95!** You save an unbelievable **$59.70!!** Specify Level II or DOS, and include **$2.00** shipping costs when ordering either package.

TO ORDER: Send a check or money order (no cash, please) and include **50 cents** per program shipping costs (**$2.00** for package). Mail to: **The Alternate Source, 1806 Ada Street, Lansing, MI 48910.** Orders shipped within 48 hours.

BULLETIN BOARD

An unsolicited, well deserved plug: three new books we've added to The Alternate Source Library. All have a very special purpose; the first two are written by Dr. Dave Lein, the last by Dr. John Blattner.

Number 1: Undoubtedly computer magazines are the cheapest source of legitimate software. Unfortunately, except for a few sparkling exceptions, not all software is written for the TRS-80. THE BASIC HANDBOOK compares idiosyncrasies in over 50 different computers. For most unusual commands, there are examples of how to program around this problem.

Number 2: will make you ready for some of the dynamite programs up and coming in TAS. LEARNING LEVEL II is the unofficial extension of the lucid Level I manual.

Number 3: is yet another well written, comprehensive guide to the internal operations of the Level II ROM and RAM. The book is 65 pages in length, and is broken down into two parts. Part one provides the assembly language programmer with everything he needs to access the sophisticated Microsoft routines already resident in ROM. Part two of the book covers the linking of assembly language with Basic.

All three books are in the $15.00 range, and worth their weight in gold! You can purchase the first two from Compusoft, 8643 Navajo Road, San Diego, CA 92119. (714/465-3322). Inside Level II can be purchased from TAS (see ad elsewhere).

We've had such a lack of response from Users Groups, we figured everyone had forsaken this valuable resource. Not entirely so:

MILWAUKEE AREA TRS-80 USERS GROUP meets at 7:00 on the last Thursday of each month at Nino's Steak Roundup, 3400 S. 108th Street, in Milwaukee, WI. Write MATUG, Box 184, 53172.

WESTPORT USERS GROUP 80 is just getting under way in the Norwalk, CT area. Write WUG-80, Box 726, Belden Station, Norwalk 06852, or phone Mr. Abrahamson @ 203/853-3861.

A couple of items on Radio Shack printers this issue. We have two in-house printers, but neither is a R/S. I hope the info herein is useful. Please advise. Regarding our own article on printers, our faces are slightly red. The May issue of Personal Computing has the most complete dossier imaginable on printers of all types--starting with the Quick Printer II and going through models costing several grand. If you haven't picked up a copy of Personal for awhile, you might want to consider it. They are undergoing some personnel changes which usually results in a change of format. The trend seeems to be toward business/serious software, with perhaps more support for the TRS-80 than any non-exclusive TRS-80 mag on the market. Twelve issues are still only $14.00!

Now that we got that out of the way, here's a couple of guys with printers that need advice. In their own words:

> "I have a need for a black box to connect between my TRS-80 interface and my Diablo Terminal. The black box should pick up LPRINT commands and convert them to RS-232 for the Diablo. I know there are many adapters available, but they all require a software driver. My problem is that many utility programs change the lineprinter DCB or clobber the memory I use to store my RS-232 driver software. So, what I need is a hardware device that needs no software driver." Contact Ken Knecht, 1340 W. 3rd St., Apt. #130, Yuma, AZ 85364.

and

> "Could you please help me respecting my Anderson Jacobson serial printer and the Radio Shack SCRIPSIT? I have just obtained SCRIPSIT and would like to use it, but have not been able to print any text via the Radio Shack RS-232 C (getting a RS-232 Interface NOT READY). In any event, there appears to be no provision for inserting the needed nulls in the serial output, at least to the point I have reached. There does not seem to be any provision either for using the Small Systems Hardware TRS232. So, what should I do in order to use SCRIPSIT and the AJ 841? Any ideas and suggestions would be very much appreciated. Thank you." Contact Michael Maw, 125 East 84th Street, New York, NY 10028.

Can anyone help these gentlemen?

Contract programming in New York? You bet--the best. Jack Bilinski has just announced the opening of 80-Microcomputer Services, Inc. at 118 Mastan Ave., Cohoes, NY (518/235-9007). Jack relates that he's spent over 7000 hours studying the TRS-80 (hope he has a green screen!) and we believe him. We haven't heard from his hoard of happy customers (yet), but we've seen examples of what he can do. He recently pointed out that the Spooler we sell (early models) were looking for a Control-M instead of a Control-N! (Blush!) Jack has reviewed several hundred pieces of software personally just to make sure that his customers get the advantage of having the best on the market. Not only this, but he offers 30 day consultation AFTER the sale! Better call now, before he gets busier than a couple of would-be editors we know!

Richard C. Vanderburgh has made some interesting enhancements to Sargon II (and some other programs). Using a Quick Printer (any printer with a bell, which is CHR$(7)) Sargon II beeps after his move. You can also LPRINT board configurations for the current move. He welcomes suggestions for other enhancements and makes all available at modest prices. Write to Richard at 9459 Taylorsville Road, Dayton, OH 45424.

SPECIAL NOTE TO USERS GROUPS: Recently Central MI TUG has experienced money problems. Dues have been maintained for over a year at just $3.00. The recent loss of an ally in the printing business as well as an unexpected jump in the amount of 'rent' for a monthly meeting place has jeopardized the clubs newsletter and possibly the club organization itself. Several fund raising considerations were proposed, one of which may have interest to other groups. The suggestion is to have members contribute original programs they have authored to the club library. These can be distributed to members for a token amount, such as one or two dollars, or even a donation. Members would also have to provide their own media, naturally. The idea was further expanded to offer this package to other clubs at a nominal price for similar distribution. The ideal goal is to help maintain the club treasury, while sustaining a members reason for remaining with the club. We at TAS think this is an excellent idea; the software will be inexpensive, the library will increase, the dues will remain the same, etc. Therefore, we'd like to know what your group thinks. If you're interested, TAS will list addresses of clubs with packages to sell (no charge for now). Whaddayathink?

And that's all (!) for this issue!!

```
02110 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * *
02112 ; PROGRAM  'TWODISK'      V1.1
02114 ;
02116 ; BY ROXTON BAKER         4/80
02118 ; 56 SOUTH RD., ELLINGTON, CT. 06026  (203) 875-2483
02120 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * *
02122 ;
02124 ;  TWODISK IS A UTILITY THAT WILL ALLOW YOU TO PUT ONTO
02126 ;  DISK THOSE 16K LEVEL II PROGRAMS THAT ARE, FOR VARIOUS
02128 ;  REASONS, INCOMPATIBLE WITH DISK BASIC. SUCH A PROGRAM,
02130 ;  WHICH TWODISK STORES AS PROG2/CMD ON THE DISK, CAN
02132 ;  LATER BE LOADED AND RUN (IN LEVEL II) RIGHT FROM DISK.
02134 ;
02136 ;  THIS ROUTINE HAS BEEN TESTED ON NEWDOS 2.1 AND
02138 ;  TRSDOS 2.3.  IT REQUIRES 48K.  IF YOU ENCOUNTER PROB-
02140 ;  LEMS WITH THIS SOFTWARE, PLEASE CONTACT THE AUTHOR!
02142 ;
02144 ;  ASSEMBLE THIS FILE AND SAVE IT WITH FILESPEC
02146 ;                 TWODISK/CMD.
02148 ;
02150 ;
02152 ;             ----INSTRUCTIONS FOR USE----
02154 ; (FOLLOW THESE EXACTLY, IN THE SEQUENCE INDICATED)
02156 ;
02158 ;  TO USE, LOAD TWODISK/CMD FROM DOS.  JUST USE THE
02160 ;  'LOAD' COMMAND - DON'T EXECUTE IT YET.
02162 ;  THEN PUSH BREAK AND RESET TOGETHER TO GET INTO LEVEL
02164 ;  II BASIC.  SET MEMORY SIZE TO 65345 (OR LOWER IF
02166 ;  REQUIRED BY THE LEVEL II PROGRAM TO BE SAVED).  THEN
02168 ;  LOAD IN THE LEVEL II PROGRAM - FROM TAPE OR HOWEVER.
02170 ;  IF IT HAS A SYSTEM TAPE THAT GOES WITH IT, LOAD THAT
02172 ;  TOO  (AFTER THE BASIC PART!).  YOU CAN LIST THE LEVEL
02174 ;  II PROGRAM, BUT SINCE IT MAY BE SELF-MODIFYING, DON'T
02176 ;  RUN IT.
02178 ;  WHEN THE PROGRAM (INCLUDING ANY SYSTEM PART) IS COM-
02180 ;  PLETELY LOADED, TYPE:  SYSTEM <ENTER> /65345 <ENTER>.
02182 ;  THE DISK DRIVE WILL RUN, AND DOS WILL BE BOOTED.  NOW
02184 ;  EXECUTE TWODISK/CMD BY TYPING: TWODISK <ENTER>.
02186 ;  THE LEVEL II PROGRAM WILL BE PUT ONTO DISK UNDER THE
02188 ;  FILESPEC PROG2/CMD.  YOU SHOULD IMMEDIATELY RENAME
02190 ;  THIS FILE TO SOMETHING MORE APPROPRIATE.  REMEMBER
02192 ;  THAT THE NEXT TIME YOU EXECUTE TWODISK IT WILL OVER-
02194 ;  WRITE ANY FILE CALLED PROG2/CMD ON THE DISK!
02196 ;
02198 ;  TO EXECUTE THE LEVEL II PROGRAM FROM DOS, SIMPLY CALL
02200 ;  IT AS YOU WOULD ANY /CMD FILE.  THAT WILL LOAD THE
02202 ;  PROGRAM INTO ITS PROPER PLACE AND JUMP YOU INTO LEVEL
02204 ;  II, FROM WHICH YOU CAN RUN IT.
02206 ;                 ----------------
02208 ;
02210 ;  ONE ASPECT OF TWODISK SHOULD BE NOTED.  ALTHOUGH
02212 ;  TWODISK DOES AUTOMATICALLY HANDLE THE CASE WHERE THE
02214 ;  LEVEL II PROGRAM HAS A SYSTEM TAPE PART, IT DOES THIS
```

```
;  BY SAVING OFF --ALL-- OF THE LOWER 16K OF MEMORY,
;  FROM 4000H TO 7FFFH.  IT HAS TO DO THIS BECAUSE IT
;  HAS NO WAY OF KNOWING EXACTLY WHERE THE SYSTEM PART
;  WENT.  THIS PROCEDURE IS OFTEN JUSTIFIED, BECAUSE
;  MANY LEVEL II SYSTEM ROUTINES DO GO AT THE TOP OF MEM-
;  ORY.  BUT THE RESULT IS A 14 GRAN PROGRAM ON THE DISK.
;
;  THERE IS AN ALTERNATIVE.  DEPENDING ON THE LOCATION
;  OF THE SYSTEM TAPE CODE IN MEMORY, YOU MAY USE THE
;  DOS UTILITIES TAPEDISK, DCV, TDISK, OR LMOFFSET TO
;  PUT THE SYSTEM CODE ONLY ONTO DISK.  THEN USE TWODISK
;  ON THE BASIC PART OF THE LEVEL II PROGRAM (NOT FOR-
;  GETTING TO SET MEMORY SIZE, IF REQUIRED).
;
;  THIS WILL RESULT IN TWO DISK FILES; TO RUN THEM FROM
;  DOS YOU WOULD 'LOAD' THE SYSTEM CODE FILE AND THEN
;  EXECUTE THE PROG2/CMD FILE CREATED BY TWODISK.
;
; - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;
PWRUP   EQU     0000H           ;FULL INITIALIZATION
LEVEL2  EQU     06CCH           ;ENTER LEVEL II NICELY
DOSRET  EQU     402DH           ;RETURN TO DOS
CMDBUF  EQU     4318H           ;DOS COMMAND BUFFER
SYSADD  EQU     40DFH           ;HOLDS SYSTEM TAPE ENTRY
CMHLDE  EQU     1C90H           ;ROM COMPARES HL,DE
EPROGP  EQU     40F9H           ;POINTS TO END BASIC PROG
BOTTOM  EQU     4000H           ;START OF MEM TO SAVE
BMOVE   EQU     09D7H           ;MOVES B BYTES
CMDINT  EQU     4405H           ;DOS COMMAND INTERPRETER
P2ENTR  EQU     8000H           ;ENTRY POINT PROG2/CMD
SAFETY  EQU     P2ENTR+18       ;WHERE LEVEL II PROG HELD
;
;  THE FOLLOWING START ADDRESS IS ALL THAT MUST BE
;  CHANGED TO RELOCATE TWODISK.  IT MAY BE SET AT
;  ANY LOCATION BETWEEN C012H AND FF45H.  THE OBJECT CODE
;  PRODUCED IS 184 BYTES LONG, AND TWO BYTES MUST BE
;  AVAILABLE BEYOND THAT FOR THE TEMPORARY STORAGE LOCA-
;  TION 'TSTORE'.  NOTE THAT CHANGING THE STARTING LOCA-
;  TION WILL REQUIRE THAT YOU ALSO USE A DIFFERENT MEMORY
;  SIZE UNDER LEVEL II, AND THAT THE  '/' ENTRY POINT YOU
;  USE TO GET OUT OF LEVEL II BE CHANGED.  FOR BOTH OF
;  THESE VALUES, USE THE DECIMAL EQUIVALENT OF THE START
;  ADDRESS YOU SPECIFY HERE :
;
START   EQU     0FF41H          ;DEC. EQUIV. IS 65345.
        ORG     START           ;  (EASY TO TYPE)
;
; - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;                ** ENTRY FROM LEVEL II **
L2ENTR  LD      HL,42E8H        ;ENTRY FROM LEVEL II.
        LD      DE,(SYSADD)     ;SEE IF SYSTEM TAPE WAS
        CALL    CMHLDE          ;  LOADED UNDER LEVEL II.
        JR      Z,BONLY         ;IF HL=DE, NO SYSTEM TAPE
```

```
02324          LD      BC,3FFFH        ;IF ONE WAS, WE MUST SAVE
02326          JR      MOVEUP          ;  MEM FROM 4000 TO 7FFF.
02328 ;
02330 BONLY    LD      BC,(EPROGP)     ;ELSE FIND END OF LEVEL
02332          LD      A,B             ;  II PROGRAM,
02334          SUB     40H             ;AND GET DIFF FROM 4000
02336          LD      B,A             ;  IN BC.
02338 ;
02340 MOVEUP   LD      (TSTORE),BC     ;SAVE #BYTES TO MOVE
02342          LD      HL,BOTTOM       ;  FROM 4000+.
02344          LD      DE,SAFETY       ;WHERE PROG WILL BE HELD.
02346          LDIR                    ;SLIDE IT UP THERE.
02348          JP      PWRUP           ;HAVING MOVED LEVEL II
02350                                  ;  STUFF TO SAFETY, BOOT
02352                                  ;  THE DOS.
02354 ;
02356 ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
02358 ;                ** ENTRY FROM DOS **
02360 DCODE    LD      HL,(TSTORE)     ;RETRIEVE #BYTES MOVED
02362          LD      (BYTNUM),HL     ;PUT IN MOVEDOWN CODE
02364          LD      B,12H           ;NOW SLIDE MOVEDOWN CODE
02366          LD      DE,MVDNCD       ;  DOWN INTO PLACE AT
02368          LD      HL,P2ENTR       ;  P2ENTR+.
02370          CALL    BMOVE
02372 ;
02374          LD      HL,(TSTORE)     ;CALC END ADDRESS, AND
02376          DEC     HL              ;  PUT ASCII IN DUMP CMD.
02378          LD      DE,SAFETY
02380          ADD     HL,DE           ;GETS END ADDRESS IN HL.
02382          LD      C,H
02384          CALL    ASCII           ;CONVERT H TO 2 ASCII'S
02386          LD      (DMP2),DE       ;  AND PUT IN COMMAND.
02388          LD      C,L             ;DO THE SAME WITH L.
02390          CALL    ASCII
02392          LD      (DMP2+2),DE
02394          LD      HL,DMPCMD       ;NOW POINT TO COMMAND
02396          JP      CMDINT          ;  AND EXECUTE IT.
02398 ;
02400 ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
02402 ;                 ** SUBROUTINES **
02404 ASCII    LD      A,C             ;RETURNS ASCII OF C IN ED
02406          CALL    SBR1
02408          LD      D,A
02410          LD      A,C
02412          RRCA                    ;GET NEXT DIGIT
02414          RRCA
02416          RRCA
02418          RRCA
02420          CALL    SBR1
02422          LD      E,A             ;DONE WITH BOTH DIGITS.
02424          RET
02426 ;
02428 SBR1     AND     0FH             ;CALLED BY ASCII.
02430          OR      30H
```

```
               LD      BC,3FFFH          ;IF ONE WAS, WE MUST SAVE
               JR      MOVEUP            ;  MEM FROM 4000 TO 7FFF.
      ;
      BONLY    LD      BC,(EPROGP)       ;ELSE FIND END OF LEVEL
               LD      A,B               ;  II PROGRAM,
               SUB     40H               ;AND GET DIFF FROM 4000
               LD      B,A               ;  IN BC.
      ;
      MOVEUP   LD      (TSTORE),BC       ;SAVE #BYTES TO MOVE
               LD      HL,BOTTOM         ;  FROM 4000+.
               LD      DE,SAFETY         ;WHERE PROG WILL BE HELD.
               LDIR                      ;SLIDE IT UP THERE.
               JP      PWRUP             ;HAVING MOVED LEVEL II
                                         ;  STUFF TO SAFETY, BOOT
                                         ;  THE DOS.
      ;
      ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      ;                  ** ENTRY FROM DOS **
      DCODE    LD      HL,(TSTORE)       ;RETRIEVE #BYTES MOVED
               LD      (BYTNUM),HL       ;PUT IN MOVEDOWN CODE
               LD      B,12H             ;NOW SLIDE MOVEDOWN CODE
               LD      DE,MVDNCD         ;  DOWN INTO PLACE AT
               LD      HL,P2ENTR         ;  P2ENTR+.
               CALL    BMOVE
      ;
               LD      HL,(TSTORE)       ;CALC END ADDRESS, AND
               DEC     HL                ;  PUT ASCII IN DUMP CMD.
               LD      DE,SAFETY
               ADD     HL,DE             ;GETS END ADDRESS IN HL.
               LD      C,H
               CALL    ASCII             ;CONVERT H TO 2 ASCII'S
               LD      (DMP2),DE         ;  AND PUT IN COMMAND.
               LD      C,L               ;DO THE SAME WITH L.
               CALL    ASCII
               LD      (DMP2+2),DE
               LD      HL,DMPCMD         ;NOW POINT TO COMMAND
               JP      CMDINT            ;  AND EXECUTE IT.
      ;
      ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      ;                   ** SUBROUTINES **
      ASCII    LD      A,C               ;RETURNS ASCII OF C IN ED
               CALL    SBR1
               LD      D,A
               LD      A,C
               RRCA                      ;GET NEXT DIGIT
               RRCA
               RRCA
               RRCA
               CALL    SBR1
               LD      E,A               ;DONE WITH BOTH DIGITS.
               RET
      ;
      SBR1     AND     0FH               ;CALLED BY ASCII.
               OR      30H
```

WHAT'S WHERE THE VARPTR POINTS

By Bill Brown

The VARPTR function in TRS-80 BASIC is a handy tool that gives you access to the storage addresses of the values you assign to variables in your BASIC programs. Equipped with these addresses and the PEEK and POKE functions, there are several interesting things you can do to enhance the operation of the programs you write. This article is the first of at least two articles dealing with how you can make use of VARPTR.

Most of the uses you will find for VARPTR in day-to-day programming have to do with string variables. Programs that use a lot of string space and do a lot of string manipulation, frequently have 'uncomfortable' pauses in execution while BASIC rearranges string space. This allows it to reuse the space that is occupied by strings that your program has abandoned when the values of string variables are changed. What goes on in this process, how BASIC manages string space and how you can use VARPTR to make it more efficient all warrant an additional article (next issue). By the end of this article, however, you should have the basics of what you need to know to make use of the general technique.

The core of the issue involving the use of VARPTR deals with how the computer stores the numbers and other information that you assign as values to variables. If you have a string variable (A$) and you set this variable equal to some value in the program, say

```
10 A$="TAS"
```

then the operating system of the computer (BASIC) must store the value someplace in the computer's memory. But more importantly, it must be able to get it back any time you use A$ in your program, say

```
20 PRINT"THIS NEWSLETTER'S INITIALS ARE ";A$
```

To do this BASIC stores a table (list) of variable names, and adds to this list the first time it encounters each new variable name in the program. The variables that are defined first in the program will be first in the list of variable names, and ones defined last will be at the end. Any time the program encounters a variable name it will search the list to see if it already exists, and if not, will add it. Actually, BASIC keeps two such lists, one for simple variables and one for arrays, and the appropriate

WHAT'S WHERE THE VARPTR POINTS

By Bill Brown

The VARPTR function in TRS-80 BASIC is a handy tool that gives you access to the storage addresses of the values you assign to variables in your BASIC programs. Equipped with these addresses and the PEEK and POKE functions, there are several interesting things you can do to enhance the operation of the programs you write. This article is the first of at least two articles dealing with how you can make use of VARPTR.

Most of the uses you will find for VARPTR in day-to-day programming have to do with string variables. Programs that use a lot of string space and do a lot of string manipulation, frequently have 'uncomfortable' pauses in execution while BASIC rearranges string space. This allows it to reuse the space that is occupied by strings that your program has abandoned when the values of string variables are changed. What goes on in this process, how BASIC manages string space and how you can use VARPTR to make it more efficient all warrant an additional article (next issue). By the end of this article, however, you should have the basics of what you need to know to make use of the general technique.

The core of the issue involving the use of VARPTR deals with how the computer stores the numbers and other information that you assign as values to variables. If you have a string variable (A$) and you set this variable equal to some value in the program, say

```
10 A$="TAS"
```

then the operating system of the computer (BASIC) must store the value someplace in the computer's memory. But more importantly, it must be able to get it back any time you use A$ in your program, say

```
20 PRINT"THIS NEWSLETTER'S INITIALS ARE ";A$
```

To do this BASIC stores a table (list) of variable names, and adds to this list the first time it encounters each new variable name in the program. The variables that are defined first in the program will be first in the list of variable names, and ones defined last will be at the end. Any time the program encounters a variable name it will search the list to see if it already exists, and if not, will add it. Actually, BASIC keeps two such lists, one for simple variables and one for arrays, and the appropriate

THE ALTERNATE SOURCE

June 8, 1981

Dear Bound Volume Purchaser:

After completion of printing, we were saddened to discover that the bound volume of TAS contained errata. Namely, pages 34 & 35 of issue #3 are not correct.

The correct pages 34 & 35, as they appeared in the original issue #3, are on the reverse of this page.

We are sorry for any inconvenience this has caused you.

TAS Staff

FROM THE SOURCE'S MOUTH

By Joni M. Kosloski

HOT RUMOR DEPT: (Maybe.) Radio Shack will be offering an on-line subscription service (Source, Micro-Net?). Price should be less than what's available now. If this is true, you'd better get a modem/RS-232 now.

BUGS DEPT: Our hats off to Jack Bilinski at 80-Microcomputing Services for these tips. Early versions of the Mumford spooler had an 'M' instead of 'N' when using the '<left arrow>N' command. Either use the 'M' or patch mem location ECOOH with 40H. This has been fixed in later versions. Thanks, Jack.

REGARDING NEWDOS-80 ORDERS: Unfortunately (and predictably) we're still awaiting our first shipment. We have been informed that it should be within a couple of weeks, most likely by the time most of you read this. Some initial questions have been asked (and answered):

1. It will be compatible with Scripsit. For current NEWDOS+ owners, a patch is available on the Source.
2. NEWDOS-80 will be provided in a special wrapper. According to our source, this wrapper will be all the receipt you need, should an update become available in the future. Please save it!
3. Unless you have requested otherwise, we will be shipping NEWDOS-80 by UPS on the day we receive it.
4. Your original diskette will be returned (this for those who are upgrading).

Thanks for your order. We hope you'll be hearing from us soon!

TIDBIT DEPT: If you haven't already, power up DOS. Any version, including NEWDOS. Type:

```
BOOT/SYS.WHO
```

Hit return and press the "2" and "6" numeric keys while DOS is looking for the program. You should see a very interesting message on the screen! (Sorry, no equivalent tidbit for Level II folks.)

May the Source be with you!!

```
                CP      3AH
                RET     C
                ADD     A,07
                RET
;
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;                       ** MISCELLANEOUS **
;   THIS IS THE DUMP COMMAND STRING.  IT IS MODIFIED AS
;   NECESSARY BY TWODISK TO MAKE IT PUT THE CORRECT
;   BLOCK OF MEMORY ONTO DISK.  THE ONLY THING THAT EVER
;   CHANGES IS THE END ADDRESS.
;
DMPCMD  DEFM    'DUMP PROG2/CMD (START=X'
        DEFB    '''
        DEFM    '8000'
        DEFB    '''
        DEFM    ',END=X'
        DEFB    '''
DMP2    DEFM    'XXXX'
        DEFB    '''
        DEFM    ',TRA=X'
        DEFB    '''
        DEFM    '8000'
        DEFB    '''
        DEFB    ')'
        DEFB    0DH
;
;
;   THIS IS THE BLOCK-MOVE CODE THAT WILL GET SLID DOWN
;   AND PREFIXED ONTO THE LEVEL II BLOCK PRIOR TO DUMPING
;   THE WHOLE THING TO DISK AS PROG2/CMD.
;
MVDNCD  DI                              ;INTERRUPTS FOUL UP LVL2.
        LD      BC,0000                 ;#BYTES TO MOVE (LATER).
        LD      DE,BOTTOM               ;START PUTTING IT HERE.
        LD      HL,SAFETY               ;FROM WHERE IT WAS HELD.
        LDIR
        LD      SP,4288H                ;WHERE SYSTEM PUTS STACK.
        JP      LEVEL2                  ;JUMP INTO LEVEL II.
;
;
BYTNUM  EQU     MVDNCD+2
TSTORE  EQU     MVDNCD+18
;
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        END     DCODE
```

****

WE'RE PROUD...

...to announce that there was NO ERRATA from Issue #2 to report!!

list is searched for the variable that is found in the program. (Note: The time that the computer spends searching these lists for variable names is a significant factor in the speed with which your program runs. It is possible to speed up a program noticeably by making sure that the most frequently used variables are the ones that get defined first; i.e., are encountered first in the course of the program's execution. It doesn't count if the variables are mentioned at the beginning of the program, but are jumped over in execution. The more variables that you use in the program, the more important it will be to define frequently used ones first, because of the increased time to search the whole list. A simple variable can be said to be 'defined' the first time it is given a value. Array variables are 'defined' when mentioned in the DIM statements.)

Associated with each variable name on one of these lists is some other information that the computer uses to handle its actual operations with that variable. One piece of this information is the variable's type: INTEGER, SINGLE PRECISION, DOUBLE PRECISION, STRING. Depending on which type it is, each variable will have associated with it some other information concerning the values of the variables and where the values are stored in memory. BASIC uses this information to quickly find and retrieve the values when it is executing your program, and it is this information that is made available to you by using the VARPTR function in BASIC.

The description of VARPTR is given on pages 8/8 to 8/10 in the Level II BASIC Reference manual. While the discussion that follows will not refer directly to those pages, the intent of the article is to clarify what you find there. You will probably find it helpful to have the manual available as you read the remainder of the article. One thing that may make reading those pages more easily comprehensible is to mentally insert 'PEEK' in front of the parenthesized expressions that you find there. For example, when the manual states that '(K+7) = exponent of value', what it means is that PEEK(K+7) will give you the exponent.

To quote the Level II manual, VARPTR

"Returns an address-value which will help you locate where the variable name and it's value are stored in memory."

All of the discussion that follows that in the manual is concerned with where the value is stored, and I can find no reference to information that would let me determine where the name of the variable is stored. Whatever the case, it is the values with which we will be mainly concerned here. (There is a tip on Page 5 which indicates how you use VARPTR to locate the storage of the name and variable type.)

The VARPTR function is used by specifying a variable name enclosed in parentheses following the function name, e.g.,

```
40 X=VARPTR(A$)
```

will cause X to be set to an address-value 'which will help you locate where' the value of A$ is stored in memory. Essentially, what you get from VARPTR (VARiable PoinTeR) is a pointer to the memory location, from which you can PEEK the information that is stored about the variable. The variable that is the argument for the function does not have to be a string variable, as used in the example. It can be any variable type. It can also be a subscripted variable; using VARPTR(A$(1)) will yield a different value than VARPTR(A$(2)). How you make use of the address-value that is returned by the function is different for each variable type, but is the same (or very similar) for subscripted and non-subscripted variables of the same type. In all cases, the address-value is a 'pointer' to a memory location. This memory location contains part of the variable's value, or other information about how memory addresses and numeric values are stored in the TRS-80 memory.

The Z-80 microprocessor that serves as the CPU for the TRS-80 can access (for reading or writing) 65535 bytes of memory (64K). This means that in order to address any given byte in memory, we must be able to store and use numbers as large as 65535. (Actually, this is not quite accurate; the discrepancy will be cleared up later.) To store a number this large requires two (2) bytes of memory, which means that all memory address-values are stored as 2-byte integers. The value returned by the VARPTR function is such a number; it is not the value of the variable, nor any other direct information about the variable--it is simply a memory address-value that points to where the information we want is stored. Actually, it points to the first byte in memory that contains the information we want. For all variable types, there are other bytes of information that immediately follow the one pointed to: for integer variables there are four (4), for double precision there are eight (8) and for string variables there are three (3). The 2, 4 and 8 bytes of information for the numeric variables contain the value of the variable. The string variables are a completely different story, and we will get to them a little later. For now, let's take an integer variable as an example and see how we can use VARPTR to access its value. This requires another little side trip to get some more background.

Recall that the values of integer variables are stored in two bytes of memory. The first of these is the one pointed to by the value that is returned by VARPTR. If our integer variable is N, then we can get the pointer and store it in the variable X with the following statement:

```
50 X=VARPTR(N)
```

Now we can use PEEK(X) to get one byte of the integer N, and we can use PEEK(X+1) to get the other byte. The question then is how to put these two bytes together to get the value of N.

A quick review of some basic arithmetic: If we had the number 2317 stored in memory so that the '23' is in one byte and the '17' is in the other (this is not how it is actually done, just an example leading to something else), and we PEEKed each byte individually so that R=23 and S=17, then we could put them together to get the whole value by

N = R*100 + S = 2300 + 17 = 2317

One way to look at this example is that the two bytes we PEEK are each a 'digit' and the '17' another (in this base 'digits' range between 00 and 99). To reconstruct the whole number above we have multiplied by 100 (actually, 100 to the first power) because we are working with base 100 digits, and because we need to shift the digit '23' over 'one place' before we do the add. In this simple example, we can refer to the digit '23' as the 'most significant digit' and the digit '17' as the 'least significant digit'. The most significant digit is the one that gets multiplied.

We must do something similar to this when we use the two bytes we get by PEEKing the memory locations pointed to by VARPTR to construct our integer N, except that we multiply by 256. This is because the bytes we get when we PEEK are numbers that could be thought of as base 256 'digits' that are each stored in one byte of memory (the range for a 'digit' is 000 - 255). We shift the 'most significant byte' (MSB) over 'one place' by multiplying it by 256, then we add the 'least significant byte' (LSB), i.e.,

N = MSB * 256 + LSB

The kink in the process and the factor that causes confusion is that the Z-80 stores numbers with the LSB preceding the MSB. This would be analogous to storing our 2317 number with the 17 first, followed by the 23. Most of the time when we are programming in BASIC, the computer takes care of this 'back byting' for us and we don't need to know about it. But when we want to PEEK information out of memory and make use of it, it is necessary that we keep this in mind.

What this boils down to is that in order to reconstruct the whole number that we PEEK from memory, we must reverse the order of the digits. For variable N, with X=VARPTR(N), PEEK(X) is the LSB and PEEK(X+1) is the MSB:

```
N = PEEK(X+1) * 256 + PEEK(X)
```

To reconstruct single precision and double precision values is more complex than this because there are more bytes to be PEEKed and multiplied by powers of 256, and there is an exponent to be dealt with. We will not go into this any further here. One parting note on this example, if we want to set variabel Q equal to variable N, we can do this in one statement:

```
60 Q=PEEK(VARPTR(N)+1)*256 + PEEK(VARPTR(N))
```

or equivalently by

```
65 Q=N
```

As you might imagine, there is not a lot of traffic in numeric variables reconstructed from PEEKs of separate bytes in memory.

The reason for going into this at such great length is that in order to get at the values of string variables using VARPTR, we must construct an integer address of the storage location for the string value.

As noted above, VARPTR of a string variable points to the first three (3) bytes that give us information about the string variable. The first of these is one byte that contains the length of the string (remember that the length of a string can be a maximum of 255, the largest number that can be stored in one byte). The two bytes that follow contain the LSB and the MSB, respectively, of the address-value of where the first character of the string is actually stored in memory. If A$="TAS", then PEEK(VARPTR(A$))=3, the length of the string. To get the address of the string storage for A$ and store it in variable X, we can specify

```
70 X = PEEK(VARPTR(A$)+2) * 256 + PEEK(VARPTR(A$)+1)
```

Almost!

Here is where we encounter another one of those wrinkles that needs to be explained. While it is possible to store an integer as large as 65535 in two bytes of memory, BASIC only allows integers in the range of -32768 to +32767. That is, one bit in the MSB is reserved for the sign of the number. This also applies to address-values, since they are stored as integers, and are assumed to be integers when used with PEEKs and POKEs. How then do we address 65535 bytes of memory, or access any memory address above 32767? We use the negative numbers. To PEEK the byte that is one memory location above 32767 we use PEEK(-32768), for one above that we use PEEK(-32767), and for the top byte of 65535 bytes we use PEEK(-1). Simple but complex.

When we are calculating addresses, as in statement 70 immediately above, the most straightforward way to approach it is as follows:

1. Make sure that the variable that will hold the result (in this case, X) is a single precision variable (if you use an integer variable you will get an overflow error when the result is larger than 32767).

2. However, if you try to PEEK or POKE an address with a single precision variable with a value greater than 32767, you will also get an overflow. To make sure that X in statement 70 is an appropriate value, modify the statement as follows:

```
70 X=PEEK(VARPTR(A$)+2) * 256 + PEEK(VARPTR(A$)+1):
   IF X > 32767 THEN X=X-65535
```

With that in hand,

```
CHR$(PEEK(X))="T"
CHR$(PEEK(X+1))="A"
CHR$(PEEK(X+2))="S"
```

Note: The PEEK of the character value returns the ASCII code for that character. To get its character representation we have to use the CHR$ function.

As the dust settles around the oasis at sunset, let's pull one more rug out from under the camel. The application mentioned at the beginning of the article that dealt with using VARPTR to optimize the use of string space doesn't really need to deal with the particular values of strings at all. As long as you are programming in BASIC, most of the manipulations of the string values can be taken

Almost!

Here is where we encounter another one of those wrinkles that needs to be explained. While it is possible to store an integer as large as 65535 in two bytes of memory, BASIC only allows integers in the range of -32768 to +32767. That is, one bit in the MSB is reserved for the sign of the number. This also applies to address-values, since they are stored as integers, and are assumed to be integers when used with PEEKs and POKEs. How then do we address 65535 bytes of memory, or access any memory address above 32767? We use the negative numbers. To PEEK the byte that is one memory location above 32767 we use PEEK(-32768), for one above that we use PEEK(-32767), and for the top byte of 65535 bytes we use PEEK(-1). Simple but complex.

When we are calculating addresses, as in statement 70 immediately above, the most straightforward way to approach it is as follows:

1. Make sure that the variable that will hold the result (in this case, X) is a single precision variable (if you use an integer variable you will get an overflow error when the result is larger than 32767).

2. However, if you try to PEEK or POKE an address with a single precision variable with a value greater than 32767, you will also get an overflow. To make sure that X in statement 70 is an appropriate value, modify the statement as follows:

```
70 X=PEEK(VARPTR(A$)+2) * 256 + PEEK(VARPTR(A$)+1):
   IF X > 32767 THEN X=X-65535
```

With that in hand,

```
CHR$(PEEK(X))="T"
CHR$(PEEK(X+1))="A"
CHR$(PEEK(X+2))="S"
```

Note: The PEEK of the character value returns the ASCII code for that character. To get its character representation we have to use the CHR$ function.

As the dust settles around the oasis at sunset, let's pull one more rug out from under the camel. The application mentioned at the beginning of the article that dealt with using VARPTR to optimize the use of string space doesn't really need to deal with the particular values of strings at all. As long as you are programming in BASIC, most of the manipulations of the string values can be taken care of more efficiently with the string functions that are available in BASIC.

Without going into a lot of detail here that will be covered in the next article, the optimization comes from reducing the number of string values that are abandoned when the values of two string variables, or the values of two elements of a string array ,are interchanged. When the values of any two variables are interchanged it is necessary to temporarily store one of the values in a third variable, copy the second to the first, then move the temporary value to the second variable. When this is done repeatedly with string variables, there accumulates a trail of 'dead strings' that were at one time temporary values; the inefficiencies come from BASIC having to go around cleaning up the mess. BASIC finds and uses string values by using the location of the value and the length of the string that you can get via VARPTR. This information is all that BASIC has to go on when it needs to find out what the value of a string is, or if a string variable has any value at all. If you change these three bytes of information, and do it properly to point to another string, BASIC will never know that it did not put them there.

If we have two string variables, A$ and B$, for which we want to interchange their values, then we can PEEK the three bytes for one and save them temporarily as integer values, e.g.,

```
100 T1=PEEK(VARPTR(A$)): T2=PEEK(VARPTR(A$)+1):
    T3=PEEK(VARPTR(A$)+2)
```

Then copy the appropriate values for B$ into A$, e.g.,

```
110 POKE VARPTR(A$), PEEK(VARPTR(B$)):
    POKE VARPTR(A$)+1, PEEK(VARPTR(B$)+1):
    POKE VARPTR(A$)+2, PEEK(VARPTR(B$)+2)
```

Then copy the temporaries into B$, e.g.,

```
120 POKE VARPTR(B$),T1: POKE VARPTR(B$)+1,T2:
    POKE VARPTR(B$)+2,T3
```

Clearly this takes a lot more code than copying the strings directly. The article in the next issue will discuss this in more detail, including situations where you might find it a valuable technique and some examples that reduce a job of several minutes to less than a minute by applying methods no more complicated than those just outlined.

LAST MINUTE ODDS & ENDS

TAS now has daily pick-up from UPS. Those who do not have UPS delivery in their area should state that fact when placing orders.

TAS now has another phone line for more effective communication: area code 517, phone 485-0344. 487-3358 is the original number, feel free to continue to use it, also.

Caution to Copydisk users: CD does not update the EOF marker properly when booted from NEWDOS. A patch is forthcoming.

A disk ($7.50) or cassette ($5.00) of this issue is available from TAS, as well as previous issues and forthcoming issues.

Microdos (see review, page 25) is also available from TAS. $29.95 on diskette with manual.

Stan Ockers has suggested we continue to upgrade ISAR, sort of on a group basis. The tools are there for both ease of use and programming is accessible for ease of modification. We've had many suggestions that have not been implemented yet. A new bug was uncovered when sorting on a double precision numeric field. Apologies to a gentleman in West Virginia who first pointed this out. He received an updated version, but it did not include this correction. We will publish it in the next BTI (Between The Issues, see page 2). Any modifications you care to suggest will also be included in future BTIs. I think Stan's ideal is to have an all-encompassing program for minimal cost. With that in mind, how can we refuse? A module 10 is about half finished. Upon completion, you may request a printed listing with a SASE (not yet!). It's going to be a 'Catch-All' to allow some requests to be implemented.

Mr. Andrews, please keep the faith. We haven't forgotten.

And last but not least: Radio Shack is coming out with a TRS-80 Applications Software Sourcebook which will list existing software for the TRS-80, both Model I and Model II. The Sourcebook will provide you with titles, descriptions, authors and prices. It won't, however, include any Z80 assembly language programs. Hmmm....

And with that thought, we'll leave you until next issue, scheduled to be mailed on or before June 20, 1980.

AFTERWARD FOR ISSUE 3

Sometimes my enthusiasm gets in the way of my common sense. Volume II of The Disassembled Handbook arrived prior to this issue. On page 3/10 is the statement "No duplication from Volume I", which caused me some regrets.

Two very significant appearances this issue: Dennis Kitsz reappears and continues with us consecutively through issue number 8. A new (our only) column called "80-Aids" makes its debut. Next issue, the title changes to "Bit Wits", and finally in issue 5 to "Bit Kickin' With Jesse Bob".

Would you believe there are people who don't think there really IS a Jesse Bob Overholt?

TWO DOLLARS VOLUME 1, NUMBER 4

THE ALTERNATE SOURCE

THE MAGAZINE OF ADVANCED APPLICATIONS AND SOFTWARE FOR THE TRS-80.

IN THIS ISSUE:

Editorial RAMbling...

Charley Butler

We welcome all the newcomers to TAS. Lots of goodies packed here for helping make your TRS-80 a more reliable system. We also have several new authors, some starting this issue and some next. Don't forget to vote for your favorite. There is still a limited amount of time to get votes in for last issue, but hurry! As of next issue, we will have at least three authors who will appear full time (every issue) and promises from a couple more. We hold to one of our original premises that authors are the most important asset to any publication and have excellent intentions along this line. Our rates are still modest, but we welcome your inquiries. If you have an article you would like published, right now you would do very well to submit it to The Alternate Source. Joni or myself would be delighted to discuss your plans for writing. Rates have just moved from the maximum amount we could afford to 'flexible'. Each article will be evaluated on an individual basis. We have received some inquiries about regular columns. Our current attitude is that we welcome the prospect. Formats are not to be fixed; that is, we would appreciate commentaries and elaborations on a variety of subjects. That can keep both the authors and readers exposed to a wide variety of subjects.

The most-appreciated Allan Moluf pointed out some time ago that much of the inferior software that has been hitting the market for some time now is due to the lack of exposure to professional techniques for the some half-million 'new' programmers that entered the marketplace in the last few years. Naturally we are only counting those who deal with the TRS-80. We are currently concentrating on 'unmystifying' some of the more elaborage techniques in business, science and education. More in the next BTI.

There seems to be a lot of exciting software hitting the market in the next couple months with the Microsoft Compiler (9% for use-rights or $195. per year) and Newdos-80 (I'm writing this on Monday, June 2) and several other fine programs. We'll try and keep you informed on the y hs and bahs of all. Ken Edwards (see 'Direct Statement' article in this issue) has started calling Newdos-80 Newdos-2001. He ordered it last March.

We appreciate and welcome your comments and support. The Alternate Source is published bi-monthly by Charley Butler and Joni Kosloski at 1806 Ada Street, Lansing, MI 48910.

WHEN YOU TURN IT ON

THE POWER-UP ROUTINES OF THE TRS-80

By Dennis Bathory Kitsz

(Part Two)

Last issue we found out about the initialization routines of the TRS-80 assuming that a disc drive and expansion interface are connected to the system. Here's a quick review of that activity:

Interrupts are disabled, cassette is turned off and data are cleared from that output, video is restored to normal, and significant pointers for BASIC program operation are set up. A disc drive is searched for, and if one is found, a group of procedures are initiated in order to transfer control of the TRS-80 to these disc instructions.

A series of control signals and acknowledgments are exchanged between the floppy disc controller and the CPU, a page (256 bytes) of data are poured into a RAM buffer area, and program control is given over to this new series of commands.

If a disc drive is not found, or if the break key is held down during power-up, control is transferred to address 0075H. At this point it should be noted that the "reset" button on the TRS-80 is a non-maskable interrupt, that is, the only interrupt which the DI (Disable Interrupt) command first executed by the TRS-80 cannot mask out. When pressed, the reset button goes directly to address 0066H, following a much shorter series of instructions reminiscent of the power-up routine.

Because it is likely most important RAM pointers are still intact, this sequence does not reset them:

```
;
0066                 00010    ORG    0066H
0066  310006         00020    LD     SP,0600H
0069  3AEC37         00030    LD     A,(37ECH)
006C  3C             00040    INC    A
006D  FE02           00050    CP     02
006F  D20000         00060    JP     NC,0000
0072  C3CC06         00070    JP     06CCH
;
```

This group of instructions sets up the stack pointer, checks for the presence of a disc drive, and jumps to the complete initialization routine ("reboot") if it finds one. If none is present, it goes to the "READY" sequence beginning at address 06CCH.

Now let us return to the initialization program flow we have been following, which is found at 0075H:

```
;
0075            00080     ORG     0075H
0075  118040    00090     LD      DE,4080H
0078  21F718    00100     LD      HL,18F7H
007B  012700    00110     LD      BC,0027H
007E  EDB0      00120     LDIR
;
```

Using the LDIR instruction described in the first part of this article, a block of information located at 18F7H is transferred to RAM beginning at 4080H. These bytes describe ports in use, error storage, INKEY$ information, and so forth, as needed in the general operation of Level II BASIC.

A few specific addresses are delineated:

```
;
0080  21E541    00130     LD      HL,41E5H
0083  363A      00140     LD      (HL),3AH
0085  23        00150     INC     HL
0086  70        00160     LD      (HL),B
0087  23        00170     INC     HL
0088  362C      00180     LD      (HL),2CH
008A  23        00190     INC     HL
008B  22A740    00200     LD      (40A7),HL
;
```

Next, more RAM bytes are prepared; these jump to the familiar "?L3 ERROR" message because they are disc commands not available to Level II BASIC. The result of the following program statements is to fill addresses 4152H to 41A5H with the direction "JUMP TO 012DH":

```
;
008E  112D01    00210     LD      DE,012DH
0091  061C      00220     LD      B,1CH
0093  215241    00230     LD      HL,4152H
0096  36C3      00240     LD      (HL),0C3H
0098  23        00250     INC     HL
0099  73        00260     LD      (HL),E
009A  23        00270     INC     HL
009B  72        00280     LD      (HL),D
009C  23        00290     INC     HL
009D  10F7      00300     DJNZ    $-7
;
```

Another group of ROM "breakout" points follows; these all become returns to the main program flow. But notice something interesting about them - three bytes are set aside, but only one is filled with the return instruction (C9). This means, of course, that a jump command could be placed there. Let's first look at the series of instructions, then examine the possible benefits of their alterability:

```
;
009F  0615      00310       LD      B,15H
00A1  36C9      00320       LD      (HL),0C9H
00A3  23        00330       INC     HL
00A4  23        00340       INC     HL
00A5  23        00350       INC     HL
00A6  10F9      00360       DJNZ    $-5
;
```

If we wanted to break into the BASIC operating system, this area of RAM is one place in which we could effect this action. Most of these are error codes of one kind or another. We could "rescue" a program from displaying an error message and halting by patching in one of our own routines. If our routine were located at 5000H, for example, the C9 instruction (followed by two unused bytes) could be replaced with a "JUMP TO 5000H" command, which needs all three bytes: C3 00 50. Essentially, the authors of Level II BASIC provided many areas of expansion.

Now let's move on. BASIC programs begin at address 42E9H. A pointer to that beginning is found as a zero at address 42E8H. The next instruction sets that in place:

```
;
00A8  21E842    00370       LD      HL,42E8H
00AB  70        00380       LD      (HL),B
;
```

The stack pointer is delineated, and a call is made to 1B8FH, a subroutine to turn off or reset various devices, including the printer and cassette player. It is in part redundant, but a double-check is often worthwhile.

```
;
00AC  31F841    00390       LD      SP,41F8H
00AF  CD8F1B    00400       CALL    1B8FH
00B2  CDC901    00410       CALL    01C9H
;
```

The call to 01C9H results in the screen being cleared and the cursor being placed at position 0,0. Not yet have we seen "MEMORY SIZE?". Well, here it is:

```
;
00B5  210501      00420        LD       HL,0105H
00B8  CDA728      00430        CALL     28A7H
00BB  CDB31B      00440        CALL     1BB3H
;
```

At address 0105H is a block of ASCII bytes which spell out "MEMORY SIZE". The subroutine starting at 28A7H displays a string of data at the present location of the cursor, a byte at a time, until finding a byte whose value is 00. That terminates the display and advances the cursor. The call to 1BB3 is identical to the BASIC "INPUT" command, in that it displays the question mark and cursor, and halts for keyboard input.

If that keyboard input is a <BREAK>, a carry is generated, and the program skips b ack to "MEMORY SIZE" and displays it again, waiting for keyboard input. The instruction RST 10 (RESTART AT 0010H) follows, which is the quick way of calling a routine to locate the first character of an input. If one is found, the result of an "OR" instruction will not be zero. Here are the instructions that perform those functions:

```
;
00BE  38F5        00450        JR       C,$-9
00C0  D7          00460        RST      10H
00C1  B7          00470        OR       A
00C2  2012        00480        JR       NZ,$+20
;
```

What if, on the other hand, there was no entry other than <ENTER>? You have no doubt noticed a slight pause in the action when you do not specifically set the memory size. Let's have a look at that code:

```
;
00C4  214C43      00490        LD       HL,434CH
00C7  23          00500        INC      HL
00C8  7C          00510        LD       A,H
00C9  B5          00520        OR       L
00CA  281B        00530        JR       Z,$+1DH
00CC  7E          00540        LD       A,(HL)
00CD  47          00550        LD       B,A
00CE  2F          00560        CPL
00CF  77          00570        LD       (HL),A
00D0  BE          00580        CP       (HL)
00D1  70          00590        LD       (HL),B
00D2  28F3        00600        JR       Z,$-0BH
00D4  1811        00610        JR       $+13H
;
```

For the moment we will start at the instruction LD A,(HL). HL contains the address of a byte of RAM memory, the contents of which are placed in the accumulator. From the accumulator, they are also saved in the B register. The accumulator is complemented, which inverts all the 1's to 0's and all the 0's to 1's. This complemented value is then placed in the memory location still specified by HL. The accumulator is compared with what has been placed in HL.

What, you ask? But this value was just placed in memory - why compare it? Because - and this is a very elegant piece of writing - if it does not compare:

1. The memory location is bad and only the block of memory below it should be used to be safe;
   - or -
2. This is the end of memory.

If this is good memory, then, the test for zero passes, the contents saved in register B are returned to memory, and the program loops back, incrementing HL to the next potential memory location.

We did skip a few instructions back there. They become important only after the first loop is complete. These commands "OR" the contents of H and L; when the result is zero, we are at address 0000 - full memory has been found, and the test is complete.

Here's what we would find, alternatively, if we entered some value (or other characters) in response to "MEMORY SIZE?":

```
;
00D6   CD5A1E     00620        CALL     1E5AH
00D9   B7         00630        OR       A
00DA   C29719     00640        JP       NZ,1997H
00DD   EB         00650        EX       DE,HL
00DE   2B         00660        DEC      HL
00DF   3E8F       00670        LD       A,8FH
00E1   46         00680        LD       B,(HL)
00E2   77         00690        LD       (HL),A
00E3   BE         00700        CP       (HL)
00E4   70         00710        LD       (HL),B
00E5   20CE       00720        JR       NZ,$-30H
;
```

The call to 1E5AH checks for numeric input, and jumps to 1997H (?SN ERROR) if it is not. If the input is properly numeric, then registers DE and HL are exchanged; this action puts DE (left off at the lowest usable memory location above pre-set RAM needed by BASIC) in HL, where it can be manipulated conveniently.

Memory size minus one is usable; memory size and above is protected. So HL is decremented before being tested, then it is tested (in a manner similar, but not identical, to that done earlier). If the memory test fails, it's back to displaying "MEMORY SIZE?" again.

We're not quite there yet, however, as the figure entered for memory size may be too small. BASIC needs a bit of room to work with, so DE is set to 4414H, and the subtraction subroutine at RST 18H is called. If a carry is generated, we're shipped off to the "?OM ERROR" message found at address 197AH. Here's what it all looks like:

```
;
00E7  2B          00730          DEC       HL
00E8  111444      00740          LD        DE,4414H
00EB  DF          00750          RST       18H
00EC  DA7A19      00760          JP        C,197AH
;
```

A little more work is left to do. Recall that a value for available string space is set aside, and it is 50 bytes. Here is how it is done:

```
;
00EF  11CEFF      00770          LD        DE,0FFCEH
00F2  22B140      00780          LD        (40B1H),HL
00F5  19          00790          ADD       HL,DE
00F6  22A040      00800          LD        (40A0H),HL
;
```

Register pair DE is set up with 0FFCEH, which, if you are not yet weary of manipulation of hex numbers, is the two's complement of 50 decimal. That is, when 0FFCE is added to 0000, the result is FFCE hex ,or 50 decimal less that the original figure. Try it to see that it works. This bit of code, then, saves the value for top of available memory in 40B1H, adds register DE to it, and saves that result (memory size minus 50 bytes for string space) in address 40A0H.

There follows:

```
;
00F9  CD4D1B      00810          CALL      1B4DH
;
```

Here let me quote Roger Fuller, whose TRS-80 Supermap identifies this subroutine this way:

Revelation 21:5
"And behold...he shall make all things new".

This subroutine identifies and sets up all pointers necessary for the start of a BASIC program: variables reset, previous programs deleted, etc.

And now, the moment you've all been waiting for! Here it is:

```
;
00FC  211101        00820          LD          HL,0111H
00FF  CDA728        00830          CALL        28A7H
0102  C3191A        00840          JP          1A19H
;
```

The call to 28A7H, you may recall, displays a string of ASCII characters. The string displayed in this case is...

RADIO SHACK LEVEL II BASIC

The final instruction is a jump to 1A19H, the address of the "READY" display.

To summarize this last portion of the initialization routine:

> All the BASIC pointers, disc error codes, and ROM return codes are set up, the screen is cleared, and the "MEMORY SIZE" prompt is displayed. A valid response to that question is accepted, and, if necessary, the entire bank of memory is tested. Error messages are generated as needed. Finally, the memory size and available room for strings is recorded, the "READY" prompt is displayed, and control of the TRS-80 is given to the user.

In a future issue of "The Alternate Source", some of the myriad uses of this initialization process will be examined, as well as the options for making devilish changes after the process is complete.

```
;
06CC                00850          END         06CCH
;
```

IN-MEMORY SORTING USING GSF

By Ron Johnston

One of the most powerful attributes of a computer is the ability to store information, sort the information, and generate specialized outputs. Unfortunately, sorting in BASIC is slow, particularly for large data files, and also, a new sort subroutine must be written for each sort variation desired. This article describes the basics of sorting and illustrates a simple solution to the problems of speed and flexibiltity desired for sorting with the S-80 using one of the RACET computes programs, GSF.

SORTING - IN GENERAL

What is a computer sort and how is it implemented? Sorting in a computer is accomplished on data in array(s). The data is first entered into the array(s). This can be accomplished a number of ways -- through READ and DATA statements, through INPUT statements from the keyboard, or through INPUT# statements from tape or disk files. Data is entered in arrays in 'elements' - one element for each item of data. The elements typically start at 0 or 1 and data is entered in consecutive elements until all data is included. The location where the data is stored is defined by element (or index) number. The sort subroutine rearranges the data within the arrays. Then the sorted information may be displayed on the screen, printed on a printer, or saved in sorted order on tape or disk.

HOW IS DATA SORTED?

In most sort subroutines, data is sorted in one of two ways depending on the data type. If the data is in a numeric array, data is sorted in numerical order - if data is in a character string array, the elements are sorted in their 'ASCII' order (including spaces and non-printable characters) left to right. Numbers won't sort as expected if in character string arrays unless they are very carefully defined in format (-13.9 is greater than 132909 in ASCII!!). Both numeric and character string arrays may be sorted in ascending or descending format. Ascending is from most negative to most positive for numeric arrays; ASCII 'alphabetized' for string arrays. Descending sorts are for most positive to negative for numeric arrays, reverse alphabetized for string arrays.

WHAT ARE SORT KEYS - PRIMARY, SECONDARY, ET AL?

Sort 'keys' are a way of defining a more complex sorting arrangement. The 'primary' sort key is the most important or main way you want it sorted. An example on a mailing list might be primary sort on ZIP code, descending (I live on the west coast). The secondary sort key is how you want the data arranged within the sort following the primary sort. In a mail list example, you might want a secondary sort key of last name (ascending) alphabetical. The example combination will give you a ZIP code list with the list of names alphabetized within each ZIP code (the names for ZIP 92665 would all be alphabetical - A to Z - and for ZIP 92664 would start over again alphabetical A to Z). A tertiary sort key might be included for first name - giving you a 'three' key sort. What happens if you have more arrays than 'ZIP', last name, and first name? You must 'hold them together' with the original data or the information will become scrambled. In other words, when you define your sort keys, you must also define which other arrays have data which must be kept with the sort keys to hold all the information together.

AN EXAMPLE:

An actual example of a general sort application follows. The example chosen is for a record library, however, with a few changes, this same program can be used for any number of applictions where it is desirable to look at data in different ways - inventory lists, menus, classroom grades, sports league results. With a little thought, the applications are nearly endless. The example program is not intended to be an example of consummate programming skills, but is rather intended to provide an easy-to-follow method of building such a program and to illustrate the simplicity of using the GSF sorts. The example is designed to run on a 16K Level II (or larger) TRS. If you are a sophisticated 'disk system user', you would, of course, use disk files and other more sophisticated programming techniques. You would also include a 'DEFUSR' statement in your first line of code to access the GSF routines. The actual sample is courtesy of my son, Chris (the 'C' in RACET).

```
100 REMARK *********************************************
110 REMARK **                                         **
120 REMARK **   GSF SORT EXAMPLE. BASED ON SIMPLE     **
130 REMARK **   LP RECORD FILE. USES GSF SORT #17.    **
140 REMARK **   2 FEB 80.  RACET COMPUTES.            **
150 REMARK **                                         **
160 REMARK *********************************************
170 REMARK
180 CLEAR (MEM-2000)
190 MAX=20:MAX=MAX+2
```

<Listing continued on next page . . .>

```
200 DIM R(MAX),C$(MAX),A$(MAX),S$(MAX)
210 FOR I=1 TO MAX-2
220 READ R(I),C$(I),A$(I),S$(I)
230 IF R(I)=-999 THEN I=I-1 : GOTO 250
240 NEXT I
250 CLS
260 PRINT "MENU:"
270 PRINT "1)       SORT BY RECORD NUMBER"
280 PRINT "2)       SORT BY COMPOSER"
290 PRINT "3)       SORT BY ALBUM TITLE"
300 PRINT "4)       SORT BY SONG TITLE"
310 PRINT "5)       DISPLAY LIST"
320 PRINT
330 INPUT "ENTER OPTION NUMBER ";O%
340 IF O%<1 OR O%>5 THEN 330
350 ON O% GOTO 500 ,1000,1500,2000,2500
500 REMARK               SORT BY RECORD NUMBER
510 SV$="+R,S$,A$,C$"
520 GOTO 4000
1000 REMARK              SORT BY COMPOSER.
1010 SV$="+C$,+S$,R,A$
1020 GOTO 4000
1500 REMARK              SORT BY ALBUM TITLE (GROUP)
1510 SV$="+A$,+S$,R,C$"
1520 GOTO 4000
2000 REMARK              SORT BY SONG TITLE
2010 SV$="+S$,+R,C$,A$"
2020 GOTO 4000
2500 REMARK              DISPLAY SECTION.
2510 FOR C=1 TO I
2520 PRINT "ENTRY #";C
2530 PRINT "RECORD #",R(C)
2540 PRINT "COMPOSER ",C$(C)
2550 PRINT "ALBUM TITLE",A$(C)
2560 PRINT "SONG TITLE  ",S$(C)
2570 PRINT
2580 PRINT:INPUT"PRESS <ENTER> TO SEE NEXT ENTRY";A:PRINT
2590 NEXT C
2600 INPUT "END OF LIST. <ENTER> TO CONTINUE.";A
2610 GOTO 250
4000 REMARK              SORTING ROUTINE.
4010 K=USR(17) OR USR(VARPTR(SV$)) OR USR(1) OR USR(I)
4020 IF K=0 THEN PRINT "SORT COMPLETE." : GOTO 4090
4030 ON K GOTO 4040 , 4050 , 4060 , 4070 , 4080
4040 PRINT "NULL STRING IN 'SV$'" : GOTO 4090
4050 PRINT "MISSING VARIABLE IN 'SV$'" : GOTO 4090
4060 PRINT "ARRAY SPECIFIED NOT FOUND." : GOTO 4090
4070 PRINT "ARRAY FOUND NOT SINGLY DIMENSIONED." : GOTO 4090
4080 PRINT "ARRAY FOUND TOO SMALL." : GOTO 4090
4090 PRINT
4100 INPUT "PRESS <ENTER> WHEN READY.";A
4110 GOTO 250
```

Continued--->">Continued--->

```
5000 STOP
9000 REMARK             DATA GOES AFTER HERE...
9010 DATA 1,TOTO,HYDRA,NINTEY NINE
9020 DATA 3,BEETHOVEN,THE NINE SYMPHONIES,DA DA DA DUM
9030 DATA 6,EAGLES,LONG RUN,THOSE SHOES
9040 DATA 2,SYNERGY,SEQUENCER,CHATEAU
9050 DATA 7,ELO,BEST OF,TELEPHONE LINE
9060 DATA 4,BACH,VIOLIN CONCERTO,HEIFITZ
9070 DATA 5,FLEETWOOD MAC,TUSK,TELL ME YOU LOVE ME
9999 DATA -999,THIS IS,THE,END
10000 END
```

PROGRAM EXPLANATION

Line 180 clears sufficient string space for variables.

Lines 190 and 200 set the dimensions (maximum number of elements allowed per array). When using GSF, the arrays must be dimensioned to two greater than the maximum number of elements.)

Lines 210 - 240 read the data in the data statements. Data statements are handy because they are easy to edit.

Lines 250 - 340 display the menu.

Line 350 uses a computed GOTO to provide the setup for the desired sort.

Line groups 500, 1000, 1500, and 2000 are identical except for the contents of SV$. SV$ defines the way that the data is to be sorted when using GSF (any string variable name may be used in place of SV$).

Line 510 establishes a sort by record number, keeping the 'composer', 'album title', and 'song title' data together with the record number. The '+A$' says sort ascending primary on array 'A$'.

Line 1010 establishes a primary sort (ascending) by composer, with a secondary sort key by song title - keeping the other two arrays with the key data.

Line 1510 establishes a primary sort (ascending) by album or group title, with a secondary sort by song title.

Line 2010 establishes a primary sort by song title with a secondary key by record number.

Lines 2500 - 2610 provide for the printout on the screen of the sorted data.

Line 4010 is IT! This is all it takes to do a sort using

GSF! USR(17) says use the GSF multi-variable sort. USR(VARPTR(SV$)) says 'sort it the way I told you' in SV$. USR(1) says start the sort in array element '1', and USR(I) says sort 'I' elements.

Lines 4020 - 4080 are included to indicate the use of the 'Return Code' feature of GSF. GSF has its own error code routines which are very useful when debugging your programs.

Lines 9000 on are the data statements where you can store variable data.

IN SUMMARY

Sorting with GSF gives sorting speed, flexibility, and simplicity of programming compared to a BASIC sort. The GSF routines (or similar routines in RACET's Infinite BASIC) can sort and hold together up to 15 keys and arrays. With a little experimentation on variations built on the above example, a large number of types of lists can be stored and sorted. The list of 'lists' is practically endless. The effort to modify a program to handle these other types of lists is very simple, and the sorted outputs allow you to realize more of the true value of your computers.

****

PACKAGE DEAL!

For a limited time only, TAS is featuring a special discount on a package of five versatile utilities! BTRACE displays the current line being executed in one convenient location instead of all over the screen. CPU is a Compress Program Utility which allows you to remove all unnecessary spaces and remarks from your Basic programs. SEARCH will list any Basic program lines containing an occurance of your choice. REPLACE will search and replace occurances in your Basic programs. And CHANGES will generate a screen or printed listing of differences between versions of programs you are developing.

The TOTAL LIST PRICE, if purchased separately ,is $89.75. But right now, you can purchase all five at a package deal price of just $49.95! You save over $39.00! (Unfortunately, this package is for DOS users only.) Act now, before it's too late: Send $49.95 to The Alternate Source, 1806 Ada Street, Lansing, MI 48910. Visa, Mastercharge or COD phone 517/485-0344 or 517/487-3358.

KILLER

By Allan Moluf

This article shows a program using some of the DOS routines described in "RAMSTUFF" in issue #2 of The Alternate Source. The program, KILLER, kills one or more files with a single command. It is intended to demonstrate the elementary use of the file system calls, as well as showing how a machine language program can use parameters typed on the command line. The first twenty lines of the program include some equates for symbols mentioned in "RAMSTUFF", which we will discuss in this article.

When you type a DOS command, DOS stores the characters you type in a 64-byte buffer (at 4318-4357H), as well as echoing them onto the screen. When you press the ENTER key, DOS stores a CR code in the buffer (the CR is decimal 13 or 0DH). (Thus, the number of characters in the command is limited to 63.) DOS uses the first part of your type-in as the command or program file. The rest of the characters you typed can be used by that program. Commonly, the parameters are separated by spaces. The parameters that KILLER expects are the names of the files to kill.

To describe the actions of the program, I am going to use something called "structured psuedo-code". Don't be alarmed. You can forget that name right now. The important thing is using English words and a few keywords (in CAPITALS) to describe the decisions and jumps in the program.

A statement is a simple English sentence. Statements in sequence are separated by semicolons. A subroutine is called by just using its name, or by its name followed by arguments in parentheses. Comments for the reader are enclosed in << and >>.

Decisions are indicated by using the keyword IF:

```
1.  IF condition THEN statements ENDIF
2.  IF condition THEN statements ELSE
    statements ENDIF
```

Three types of looping statements are used:

```
1.  FOR variable <- initial value TO
    final value BEGIN statements ENDFOR
2.  WHILE condition DO statements ENDWHILE
3.  REPEAT statements UNTIL condition
```

The FOR loop is like the Basic FOR statement, except that the loop is not executed if the initial value is greater than the final value. (An optional STEP increment may be used.) The WHILE loop first checks the condition if true, then executes the statements once and tests the condition again. The REPEAT loop executes the statements once and then checks the condition; if not true, it executes the statements again.

The IF statement or any of the loop statements can be used wherever statements are allowed; in other words, nesting is allowed. Indentation is usually used to make the amount of nesting clear.

The reason for using these constructs instead of whatever you are familiar with is that these have proven helpful in writing programs that do what we want. Note that no labels or GOTO statements are used. It is easier (with some initial practice) to write a program that works the first time when we think in terms of these statements. Please bear with me, even if you are not yet convinced. The KILLER main program may be described in words:

> Start looking at the first character of the command buffer. Skip any non-blanks except CR. (This skips the program name the first time, or skips the name of the file we just killed.) Then skip any blanks. This leaves us at a non-blank which is either a CR or the start of a parameter. If not a CR, go back to the top of the loop and skip this file name. Eventually, we will get to the CR at the end of the command. Then we return to DOS.

Or, we can describe the main program like this:

```
Point to start of command buffer;
REPEAT
  WHILE char <> blank and char <> CR
    DO skip to next char ENDWHILE;  << skip
    non-blanks >>
  WHILE char = blank
    DO skip to next char ENDWHILE;  << skip
    blanks >>
  IF char <> CR
    THEN Killfile ENDIF;  << try to kill the
    file >>
UNTIL char = CR;
Go back to DOS
```

The description in psuedo-code seems easier to follow because it concisely reflects the program structure. The desired effect of this routine is to find each parameter and call the routine Killfile to process it. In this case, the processing consists of trying to kill the file identified by the parameter.

This is an example of a paradigm (pronounced pair-ah-dim), a model of how to perform a general function.

We minimize the details in a clear description of the essentials of the process. Experience in programming is partly familiarity with a given language, but is largely learning the paradigms for a given class of problems.

Knowing the appropriate paradigms for a problem can save a great deal of time. Instead of trying to figure out how to attack the problem, the experienced programmer can see that the program should be done in a certain way, because it is similar to another program he has worked on. As he proceeds, he may notice minor variations from the previous versions and generalize his model.

Note that the paradigm is not how the entire program works; it is how a given routine is structured. As the program is broken up into smaller and smaller pieces, either an appropriate paradigm is used for a routine (quick and easy) or the routine needs a new structure (more errors are likely here). Sometimes this is a clue that the routine is too big and should be broken into simpler ones. Other times, this is just a structure he has not encountered before.

In KILLER, the work of killing a single file is made a subroutine called Killfile (KFILE in the assembler listing). Killfile has to use the TRS-DOS file system calls to get rid of the file. Roughly, it uses the file name parameter and the FOPEN call to identify which file it wants and then uses the FKILL call to get rid of it. We will discuss all these things after describing the Killfile routine in psuedo-code:

```
Save registers;
Copy the file name parameter into the DCB;
FOPEN the file;
IF no error
  THEN FKILL the file ENDIF;
IF errors in the FOPEN or FKILL
  THEN
    Display the file name parameter;
    Display the error message for the error code
  ENDIF
Restore registers
```

Subroutines in machine language programs may or may not save some or all of the registers they will use. It is usually better for a routine to save any registers it does change (unless it is passing results back to the caller in that register). If this is not done, then any time the routine is called, the programmer must check which registers can be changed by the routine and either make sure (make absolutely positive) that the caller is not using the registers or protect himself by saving the registers before calling the routine and restoring them after the call. Furthermore, the programmer must keep track of which registers are changed by the routine; any modifications to the routine which change more registers make him check all the places the routine is called from.

In summary, it is simpler and safer to always have routines restore all registers they change, except for registers returning results to the caller. (There may be some system routines such as DOCHR which do not save all registers this way. It is a good idea to write a routine which saves the registers and then calls the system routine. WRCHR is an example of this type.)

The TRS-DOS file system uses a 32-byte area to hold information for an opened file. This is called the file DCB (Device Control Block). File DCB's use space reserved within your program or its data areas. Before you open a file, you put its filename in the DCB you want to use. After a successful file open, the DOS file system changes all the information in the DCB. DOS uses the new values for you whenever you call any of the file system routines. Each file you have open at the same time needs its own DCB, because that is how you tell DOS which file you are reading or writing or whatever.

Before the file is opened, you put the name of the file in the DCB. Then you can call either of the two file open routines, FOPEN or FINIT. The FOPEN routine will attempt to open an existing disk file; if it can not open it, it will return with an error code. The FINIT routine will attempt to open an existing disk file; if it can, fine. Otherwise, FINIT will try to make a new, empty file for you. It will return with an error code only if it can not find an old file or make a new file. KILLER uses FOPEN instead of FINIT because it just wants to kill files that already exist; if it used FINIT, it would end up creating a new file and then killing it, instead of simply saying the file did not exist. FINIT is usually used only when making an output file.

You use the DE-reg, the HL-reg and the B-reg for both file open routines. The DE-reg contains the address of the file DCB, which should contain the name of the file in the format FILENAME/EXT.PASSWORD:D althought the extension, password and/or drive number may be omitted. The HL-reg contains the address of a 256-byte buffer which the file system will read disk sectors into or write them from. The B-reg contains the logical record length (LRL) you will use for reading and writing records. Use LRL=0 if you will get and put data directly from and to the disk sector buffer given in the HL-reg.

If you wish, you may have the file system give you logical records of any fixed length from 1 to 255 by putting that LRL in the B-reg for the FOPEN or FINIT. Then whenever you use a FREAD or FWRITE call, you will give the address of a user record of that length; DOS will transfer your user record into the sector buffer and read and write sectors as it packs and unpacks the records from the sectors. (It is really easy, just hard to explain in words.)

In KILLER, we will not be reading or writing, but the FOPEN routine still expects the LRL and sector buffer address, so we do set them. When the FOPEN routine returns, it may have opened the file, or some error may have occurred. The zero flag is set and the A-reg contains zero if no error, i.e. the file was opened okay. If zero was returned, the DCB now contains the data that the DOS file system needs to keep track of the state of this file instead of the file name which was there before the open call. The DCB now tells what drive the file is on, where the disk directory for the file is, where the data sectors are written on disk, how many records are out there and what the current position is, among other things.

Any of the other file system routines expect the address of the DCB in the DE-register so they can read or write or do whatever to the proper file. They all return zero if okay or a non-zero error code. Some of the routines need other parameters in other registers; some routines need just the DCB address. All of them will check that the file has been opened (the first byte of the DCB, which used to contain the first character of the file name, has bit seven set while the file is open).

If there was an error on the FOPEN or FINIT call, a non-zero code is returned in the A-reg and the DCB is not changed. The error code is from 1 to 63, as described in the TRSDOS manual on pages 6-12. The error description can be displayed by calling the DSPERR routine.

It expects the error code in the A-reg and looks at the top two bits to decide how to display the error. If bit six is set, it will just give the error description; otherwise it will tell the error code number, the return address of the routine that detected the error and try to display the file name. (That does not work very well in DOS 2.2 unless the file was successfully opened, so we will be able to set bit six.) If bit seven is set, DSPERR will return to you; if not, it will immediately jump to the "DOS READY" entry point and your program is effectively terminated. (We will also set bit seven to prevent this from happening.)

Assuming that we opened the file with no error, killing the file is easy. We just call the FKILL file system routine with the address of the DCB in the DE-reg. It will also return with a zero for okay or a non-zero for an error, so we do check the zero/non-zero flag and display the error message if an error occurred.

Filekill might reasonably be split into two routines, with the second just handling the error messages. We display "***", the file name parameter from the command buffer, " - " and then call DSPERR with 80H+40H added to the error code to just get the error message. We use WRCHR to display single characters to demonstrate saving registers from a system call. (DOCHR does not save and restore the DE register, so WRCHR does it for us.)

MISCELLANEOUS COMMENTS:

1. I use "label EQU $" as a way of ensuring that I do not have to retype the label line when program revisions require adding an instruction right after the label. It also makes a convenient point for comments relating to the program structure instead of to the current instruction on the line.

2. When copying the file name parameter from the command buffer to the DCB at the start of KFILE, I assume the entire file name will fit in the 32-byte DCB. A full file name is 24 bytes long counting 8+1+3+1+8+1+1 plus a terminating blank or CR. The DOS file system uses as much of the name as is there and ignores anything else in the rest of the DCB.

3. The LDIR instruction that does the move is a commonly used Z80 instruction, but may seem complex. What it does is use the HL register to point to data to copy and the DE register to point to the destination. The BC register counts the number of bytes to move.

   First, it moves one data character to the destination; that is (DL) <- (HL). Then it increments HL and DE and decrements BC. Finally, it checks if BC is zero; if not it repeats the entire instruction, moving another byte, incrementing HL and DE and decrementing BC and testing BC again until eventually HL <- HL+BC, DE <- DE+BC and BC <- 0 and all the bytes were moved.

4. The POP HL / PUSH HL after KFIL10 gets the address of the start of the parameter into HL and also puts it back on the stack so the restore registers after KFIL90 will have the right number of registers on the stack.

Most of this information is also presented in the TRSDOS manual in Chapter 6, Technical Information. Hopefully, this article will help you to use that material. It does cover the format of the file DCB and the arguments for the system calls in more detail than this article. KILLER program listing and instructions are on the following pages.

```
;          KILLER  VER 1.1 - 1979 NOV 18 10:30
;          BY  ALLAN MOLUF
;
ORIGIN     EQU     5600H
;
;-------------------------------
;
CR         EQU     0DH        ;THE CODE FOR THE ENTER KEY
;
DOCHR      EQU     0033H      ;ROM I/O - SEND CHAR TO DISPLAY
DOSJMP     EQU     402DH      ;RETURN TO DOS
CMDBUF     EQU     4318H      ;DOS - 64-BYTE COMMAND BUFFER
DSPERR     EQU     4409H      ;DOS - DISPLAY ERROR MESSAGE
FOPEN      EQU     4424H      ;FILESYS - OPEN OLD FILE
FKILL      EQU     442CH      ;FILESYS - CLOSE AND KILL FILE
;
;-------------------------------
;
           ORG     ORIGIN
;
KILLER     EQU     $          ;--- PROGRAM ENTRY POINT
           LD      HL,CMDBUF          ;START OF COMMAND BUFFER
KILL10     EQU     $          ;--- SKIP NON-BLANKS (EXCEPT CR)
           LD      A,(HL)             ;GET NEXT CHAR IN COMMAND
           CP      ' '
           JR      Z,KILL20           ;IF BLANK
           CP      CR
           JR      Z,KILL20           ;IF END OF COMMAND
           INC     HL                 ;ELSE SKIP THIS NON-BLANK
           JR      KILL10             ;AND CHECK THE NEXT CHAR
;
KILL20     EQU     $          ;--- SKIP BLANKS
           LD      A,(HL)             ;GET NEXT CHAR IN COMMAND
           CP      ' '
           JR      NZ,KILL30          ;IF NOT A BLANK
           INC     HL
           JR      KILL20             ;SKIP THIS BLANK
;
KILL30     EQU     $          ;--- AT NEXT PARAMETER OR CR
           CP      CR                 ;SEE IF END OF COMMAND
           CALL    NZ,KFILE           ;NO, TRY TO KILL THE FILE
           JR      NZ,KILL10          ;KEEP KILLING UNTIL CR
           JP      DOSJMP             ;THEN GO BACK TO DOS
;
;-------------------------------
;
;          KFILE TRYS TO KILL THE FILE WHOSE NAME IS POINTED
;          TO BY THE HL REGISTER.  IF AN ERROR OCCURS, IT
;          DISPLAYS AN ERROR MESSAGE.
;
;          ENTRY:  HL      ADDRESS OF FILE NAME PARAMETER
;
```

```
00530 ;         EXIT:    NO REGISTERS ARE CHANGED
00540 ;                  EITHER THE FILE HAS BEEN KILLED OR
00550 ;                  AN ERROR MESSAGE HAS BEEN DISPLAYED
00560 ;
00570 KFILE     EQU      $          ;--- TRY TO KILL A FILE
00580           PUSH     AF             ;SAVE REGISTERS
00590           PUSH     BC
00600           PUSH     DE
00610           PUSH     HL             ;PARAM ADDR ON STACK TOP
00620           LD       DE,DCB         ;ADDRESS OF FILE DCB
00630           LD       BC,32          ;LENGTH OF THE DCB
00640           LDIR                    ;COPY THE FILE NAME
00650           LD       DE,DCB
00660           LD       HL,BUF         ;B, DE AND HL FOR OPEN
00670           LD       B,0
00680           CALL     FOPEN
00690           JR       NZ,KFIL10      ;IF AN ERROR
00700           CALL     FKILL          ;NOW KILL THE FILE
00710           JR       Z,KFIL90       ;IF OK
00720 KFIL10    EQU      $          ;--- ERROR ON FILE OPEN OR KILL
00730           POP      HL             ;GET ADDR OF PARAMETER
00740           PUSH     HL             ;AND SAVE IT AGAIN
00750           PUSH     AF             ;SAVE ERROR CODE
00760           LD       B,3
00770 KFIL20    EQU      $          ;--- DISPLAY ASTERISKS FIRST
00780           LD       A,'*'
00790           CALL     WRCHR
00800           DJNZ     KFIL20
00810           LD       A,' '
00820           CALL     WRCHR          ;AND THEN A SPACE
00830 KFIL30    EQU      $          ;--- LOOP TO DISPLAY FILE NAME
00840           LD       A,(HL)
00850           CP       ' '
00860           JR       Z,KFIL40       ;IF END OF PARAMETER
00870           CP       CR
00880           JR       Z,KFIL40       ;IF END OF COMMAND
00890           CALL     WRCHR
00900           INC      HL
00910           JR       KFIL30         ;CHECK NEXT CHARACTER
00920 ;
00930 KFIL40    EQU      $          ;--- REACHED END OF FILE NAME
00940           LD       A,' '
00950           CALL     WRCHR
00960           LD       A,'-'
00970           CALL     WRCHR
00980           LD       A,' '
00990           CALL     WRCHR
01000           POP      AF             ;GET SAVED ERROR CODE
01010           OR       80H+40H        ;0C0H = 80H + 40H
01020 ;                       80H ASKS FOR RETURN INSTEAD OF A
01030 ;                           JUMP TO DOS AFTER THE ERROR MSG.
01040 ;                       40H ASKS FOR NO EXTENDED ERROR
```

```
01050 ;                                 DISPLAY (WHICH IS UNREADABLE IF
01060 ;                                 IT WAS A FILE OPEN ERROR).
01070           CALL      DSPERR              ;HAVE DOS SHOW ERROR MSG
01080 KFIL90    EQU       $               ;--- NOW RESTORE THE REGISTERS
01090           POP       HL
01100           POP       DE
01110           POP       BC
01120           POP       AF
01130           RET
01140 ;
01150 ;------------------------------
01160 ;
01170 ;         WRCHR WRITES A SINGLE CHARACTER FROM THE A-REG
01180 ;         TO THE SCREEN DISPLAY.  IT SAVES DE BEFORE IT
01190 ;         CALLS THE ROM ROUTINE TO DO THE WORK.
01200 ;
01210 ;         ENTRY:  A       CHAR CODE TO DISPLAY ON SCREEN
01220 ;
01230 ;         EXIT:   A AND FLAGS HAVE BEEN CHANGED
01240 ;
01250 WRCHR     EQU       $               ;SEND A CHAR TO THE DISPLAY
01260           PUSH      DE
01270           CALL      DOCHR                   ;ROM ROUTINE
01280           POP       DE
01290           RET
01300 ;
01310 ;------------------------------
01320 ;
01330 DCB       DEFS      32              ;DEVICE CONTROL BLOCK FOR FILE
01340 BUF       DEFS      256             ;SECTOR BUFFER AREA FOR FILE
01350 ;
01360           END       KILLER
```

INSTRUCTIONS FOR USING KILLER

Killer is executed directly from DOS mode, in the following manner:

```
KILLER AAAA BBBB CCCC DDDD EEEE FFFF GGGG
```

where AAAA, BBBB, CCCC, DDDD, EEEE, FFFF, and GGGG are files that you want to kill. You may enter as many files as you can fit on the command buffer. Extensions and passwords must be included where necessary; drive number is optional. Killer has been tested using both TRSDOS and NEWDOS.

Killer just may become one of your most valuable tools!

TEACH YOUR TRS-80 TO TALK!

A Product Review by Jim Stutsman

Communicating with a computer has always been a one sided proposition. You spend hours typing in a program using the language that the computer understands, only to get a response of "SN ERROR IN 100". Who among us has not at one time or another wished that the blasted thing could speak using words we understand instead of jargon? True man-machine communication by spoken word is still a long way off , but thanks to a new product from Percom Data Company, it is now possible to program your TRS-80 to talk.

The new product is called Speak-2-Me-2. No, it's not a 'droid, but a small printed circuit board which installs in a Texas Instruments (TI) Speak and Spell learning aid. This enhancement allows the Speak and Spell to be attached to the printer port (or printer adapter cable) so that the computer has command of it. The full vocabulary of the Speak and Spell may then be selected by the TRS-80. Although the repetcire is limited to only those words that Speak and Spell "knows", the list can be expanded through the addition of word modules available as Speak and Spell accessories. Words can also be spelled out, since Speak and Spell can say the entire alphabet and the numbers from zero to ten.

While the relative scarcity of verbs in Speak and Spell's vocabulary makes sentence construction a bit difficult, the overall quality of speech is far superior to most other speech synthesizers. The technique used to generate speech, while held proprietary by TI, is not the commonly used phoneme method which causes speech to sound mechanical and strange. Because of this, new w ards cannot be added to the vocabulary by joining sounds, as with some other speech synthesizers.

As previously stated, Speak-2-Me-2 is a small circuit board which is designed to fit in the battery compartment of the Speak and Spell. Installation of Speak-2-Me-2 requires that the Speak and Spell be modified. The mod consists of three printed circuit etch cuts and the attachment of six wires. Since the battery compartment is occupied by the new board, the Speak and Spell must also be modified to accept power from an external source. A nine volt battery eliminator, of the type sold in most Radio Shack stores, serves this purpose. Once the modifications are complete the Speak and Spell will no longer function in its old capacity, as the keyboard and display are completely dead.

Although Speak-2-Me-2 is furnished as an assembled and tested board, it is really more of a kit, since the purchaser must install it and modify the Speak and Spell (not included with Speak-2-Me-2) himself. The modifications, while not terribly difficult, are definitely not for amateurs. Only persons with prior experience in soldering on small printed circuit boards should attempt them.

Software control of Speak-2-Me-2 is achieved through a simple driver which is provided as a short BASIC program that the user keys in. The driver operates as a USR function, with the integer parameter signifying which word or phrase is to be spoken. After being keyed in and RUN, it indicates successful installation by announcing "READY". Two simple example programs are provided. The first is a simple pre-school alphabet game in which the player must push the key corresponding to a spoken letter. Perhaps hinting at future products, the second program causes Speak-2-Me-2 to recite "Yesterday I was a young machine. Today I talk, tomorrow I should learn to walk."

While the number of possible applications for Speak-2-Me-2 seems limitless, the areas of education and aid to the handicapped seem full of promise. Connected to a TRS-80, a Speak and Spell could literally become the voice of someone unable to speak. It might also replace the screen for a blind person. With regard to education, Speak and Spell by itself has already proven to be a powerful learning tool. With Speak-2-Me-2, it can be used in even more ways.

Speak-2-Me-2 is available for $69.95 from Percom Data Company, Inc., 211 North Kirby, Garland, TX 75042. Orders may be placed by calling toll-free (800) 527-1592. For answers to technical questions, call (214) 272-3421. The purchaser of Speak-2-Me-2 is expected to already have a TI Speak and Spell which he must modify himself. It will also be necessary to provide nine-volt power for operation of the modified unit.

****

(Timothy S. Smith from Monmouth, IL submitted this short assembly language routine which will eliminate the slash in the zero on line printer output. Very useful when printouts are to be read by the general public, for mailing labels, billings, etc. ed.)

```
BFF3 F5         PUSH    AF        ;SAVE FLAGS/ACCUMULATOR
BFF4 79         LD      A,C       ;LOAD CHARACTER INTO ACCUM.
BFF5 D630       SUB     30H       ;SUBTRACT VALUE OF ZERO
BFF7 2002       JR      NZ,BFFB   ;IF NOT ZERO GO ON
BFF9 OE4F       LD      C,4FH     ;IF ZERO CHANGE TO LETTER O
BFFB F1         POP     AF        ;RECOVER FLAGS/ACCUMULATOR
BFFC C38D05     JP      058DH     ;ON TO LINE PRINTER ADDRESS
```

TO USE: Load beginning of above routine into the line printer driver address 4026H=MSB, 4027H=LSB (From above, 4026H=F3, 4027H=BF)

BIT WITS

Dear Jesse Bob,

I use PEEK and POKE a lot, as well as VARPTR for manipulating machine language programs and experimenting. In some programs I can do whatever I want, but in others all I seem to get is "OVERFLOW" errors on every PEEK or POKE. What am I doing wrong? Is there something wrong with my Level II ROM?

Nervous Ned

Dear Ned,

Calm yourself, your ROM is fine. Your problems stem from the way Level II BASIC handles addresses in PEEK and POKE statements.

Whenever you PEEK or POKE, the address is converted internally by BASIC to a 16 bit integer. If the address is above 32767 decimal then the most significant bit will be a 1, which will cause BASIC to regard the 16 bit result as a negative number. If the number it started out with was positive, BASIC panics and reports an overflow error. To overcome this problem, you must start out with a negative number whenever using PEEK or POKE with addresses above 32767 decimal. For example, address 49152 (C000 hex) must be specified as -16384 in a PEEK or POKE.

The boys and I don't like to hassle with these phony negative addresses, so we use the following defined function to take care of it for us:

```
100 DEF FNAD(A!) = A!+(A!>32767)*65536
```

The key to the success of this function is the logical expression "(A!>32767)". In evaluating this, Level II will return a value of 0 if false and -1 if true. Thus, if A! is less than 32768, the result will be A!+0*65536, or simply A!. If the address is greater than 32767, the result will be A!+(-1*65536) or A!-65536, which converts it to the proper negative number. The nice thing about this function is that if you inadvertently use it with an address that is already negative, the address will be left unchanged.

----------(J)----------

Jesse Bob,

My DOS just got me again! I wrote a program which reads several long files and creates a very short file summarizing the input. I just ran it for four and a half hours summarizing data, and lost it all trying to write to a write protected diskette. No error was reported until I tried to CLOSE the file.

I also have difficulty writing programs which other people will use. If an output file is OPENed on a write protected diskette, a "DISK FULL" or "TOO MANY FILES" error is reported, which is very confusing. As a result, I have a severe phobia of diskettes and break into hysterical laughter at the sight of the write protect tabs. Can you help me?

Tense in Toledo

Dear Tense,

Don't get hung up on write protect tabs! Next time let your program check the disk BEFORE you open the file. This can be done with the following line:

```
100 POKE 14305,N : POKE 14316,208: WP=PEEK(14316) AND 64
```

The value of "N" should be "1" for drive 0, "2" for drive 1, "4" for drive 2, and "8" for drive 3. "WP" will contain zero if a drive is NOT protected, i.e., may be written on, or a 64 if it is write protected.

----------(J)----------

Dear Jesse Bob,

My printer recognizes the ASCII form feed character, CHR$(12), as a request to feed paper to the top of the form that I have defined at the printer. Whenever I do a PRINT CHR$(12), though, it prints an extra line past the top of the form. In fact, every page gets an extra line so that my text "walks" down as I print. I've had the printer checked and it seems to be in good working order. Any ideas?

Puzzled in Los Angeles

Dear Puzzled,

What you really have is two problems. Your first problem is that the Level II printer driver intercepts the form feed character and does not print it directly. Instead, it subtracts the current line count from the number of lines per page in the printer DCB to determine how many lines remain on the current page. It then issues that many line feed characters to simulate a form feed. This technique was probably used so that even printers which do not recognize the form feed character can be forced to the top of a new page by printing a CHR$(12). The disadvantage of doing the form feed in software is that the program has to "know" how many lines are on the form. To control the printer directly with the form feed character, try the following one-liner:

```
     100 IF (PEEK(14312) AND 240)=48 THEN POKE 14312,12
         ELSE GOTO 100
```

The IF statement first checks the printer status lines for a ready state. Only the four most significant bits are used for status, hence the "AND 240" to mask off the four lower bits. Note that the expression "(PEEK(14312) AND 240)" will return the printer status, and may be used in an IF statement to prevent a program from going into suspended animation due to an inoperative printer. Once the printer has been found to be ready, the form feed (decimal 12) is poked directly into the printer port, bypassing the driver.

Your second problem is due to an anomoly in the Level II ROM. The most common line spacing used in computer printers is six lines per inch. Standard paper runs eleven inches per page. Most third graders can compute a page to be sixty six lines long, but the ROM plugs the printer DCB with 67. This causes every page to be one line too long, which causes the "walking" text. The solution is to simply POKE 16424,64 at the beginning of any program which is to use a PRINT CHR$(12) to go to the top of form.

----------(J)----------

Jesse Bob and TAS wish to thank all the readers who sent in questions. According to Jesse Bob, "Me an' the boys were busier than mosquitoes in a nudist colony answering all the mail. Y'all had some dang good questions!"

If you have a technical question that you have been unable to get answered, send it to TAS for review by Jesse Bob or one of his wranglers on The Circle J Software Ranch. While only a limited number of letters can be published in each issue, a response can be guaranteed by including $6.00 with your question. If the question is published or cannot be answered within 60 days, your check will be returned. Otherwise you will get a letter from Jesse Bob or one of the boys with your answer.

****

Could anyone inform us of any activities, applications, modifications, etc. that are being used to allow the visually impaired make use of the TRS-80? Please contact Charley at TAS, (517) 485-0344. Thanks!

BULLETIN BOARD

CONTEST!

Richcraft Engineering is sponsoring a "Write the Fastest Disassembler in Basic Possible" contest. The best time wins; total of three prizes; guaranteed $100 minimum for first prize. Interested persons may contact Richcraft Engineering at Drawer 1065, #1 Wahmeda Industrial Park, Chautaugua, NY 14722 (or TAS) for contest rules and entry blank. $10 entry fee. Hurry, deadline for submissions is September 30, 1980. Winners announced not later than October 15th. Winners names will appear in 80-Microcomputing!

PATCHES

Centronics owners having problems with Mumford's Spooler program, take note! Form-feed problem can be corrected by changing form feed character from 13 to 138. This character occurs at the start address of the program plus 467. To be safe, PEEK this location to be sure it is a 13 before poking in 138. Earlier versions of Spooler may be offset from this location by a few bytes. If you are unwilling or unable to make this change, Mumford will take care of the problem for just $2.00. Return tape to Mumford Micro Systems, Box 435, Summerland, CA 93067.

USER GROUP INFO

Milwaukee Area Users Group now has a (CBBS) Bulletin Board! Phone area code (414) 282-8118.

NEW PRODUCTS

THE AMAZING BLACKJACK MACHINE! An extraordinary program designed for the development, study , and analysis of blackjack playing strategies and betting systems. It features high speed test runs with the computer playing both sides under an almost endless variety of conditions that can be set by the user. At any time you may interrupt the test run and get a printout of additional data (player busts, dealer busts, cards per hand, bet per game, etc.). This program could return its purchase price a hundredfold! Available from Richard A. Ramm, 64 Division Ave, Levittown, NY 11756, for just $25.00.

FASTLOAD! Load at 8000 baud, 16 times faster than the usual baud rate! JOY-80! A joy-stick for games as well as an input device for such things as light level sensing, voice input, noise sensing, etc. ROM EXTENDER! Plugs into the back of the TRS-80 to provide full decoding to access the unused 2K space in the TRS-80 memory map. Can provide up to 2014 bytes of extra memory! PRO-80! A prototyping printed circuit board with a 40 pin connector for conveniently assembling special peripheral circuits for the TRS-80. For info on any of the four above products, contact Microsette Co., 475 Ellis Street, Mt. View, CA 94043.

SUPER STEP! A machine language program designed to be used in conjunction with TBUG. Super Step displays, in various formats, a scrolling field of disassembled RAM locations corresponding to the PC of an animated Z80 Processor. Super Step can single-step or TRACE any Z80 machine code, and is displayed in a before/after format that includes CPU registers, stack elements and an expanded flag register configuration. Also displayed is an intelligent RAM window that selectively posts local RAM environments or a user-designated area. Super Step is $19.95 and available from Allen Gelder Software, Box 11721 ,Main Post Office, San Francisco, CA 94101. Please include 75 cents postage; CA residents add 6%.

TIME & SCHEDULE PROGRAM! Make your time more productive. Organize your life at home or at the office. Plan ahead the easy way with Softronics Time & Schedule Program. Features like selective screen message erasing, automatic day of week calculating, debounced key and search modes for fast alteration, and more makes this a pleasing and easy to use system. Available on tape for $19.95, disk for $29.00. Write Softronics Computer Services, P.O. Box 1465, Mesa, AZ 85201.

Now, a LINE PRINTER III! Features a 9 X 9 dot matrix, 150 characters per second print speed, 132 columns, 6 or 8 lines per inch, 2K buffer, various print sizes and more! Just $995. Information available from 80-Microcomputer Services, 118 Masten Ave, Cohoes, NY 12047.

INFORMATION WANTED

Sigrep Data, Box 14-522, Panmure, Auckland, New Zealand, is interested in programmes suitable for operation in their own bureau or for use as part of the package in our "Turn Key" systems. Details and prices of programmes suitable for TRS-80 Model II 64K and written for use in the commercial, medical, educational, scientific, or technical fields would be appreciated. Contact I. H. Taylor, Operations Manager.

COMPUTING ELAPSED TIME

(This information is from an actual Meta Technologies Bulletin released February 22, 1980. Subscription information can be found at the end of this article.)

SUBJECT: TRS-80 Elapsed Time Calculations

PROBLEM(S)/ISSUES: While the TRS-80 does provide a function that returns the time (TIME$), a function for obtaining elapsed time is not provided.

IMPLICATIONS: Elapsed time is valuable when measuring the performance of a subroutine or other code. It may also be employed to determine response rate, or the lack of response (is the computer being used?).

RECOMMENDED ACTION(S): Key in, run and study the code listed below.

```
10 CLEAR500
20 ' FUNCTION BELOW CONVERTS STRING RETURNED FROM THE
    "TIME$" FUNCTION INTO SECONDS
30 DEF FNTM(TM$)=3600*VAL(MID$(TM$,10,2)) +
   60*VAL(MID$(TM$,13,2)) + VAL(MID$(TM$,16,2))
50 '
60 PRINT:PRINT "HIT ANY <KEY> TO START TIMER ..."
70 IF INKEY$="" THEN 70 ELSE T1$=TIME$
80 PRINT "HIT ANY <KEY> TO STOP TIMER ..."
90 IF INKEY$="" THEN 90 ELSE T2$=TIME$
100 PRINT "ELAPSED TIME ="; FNTM(T2$) - FNTM(T1$);
    "SECONDS"
110 GOTO 60
```

SOURCE/CONTACT: Bob Fiorelli

TO SUBSCRIBE: Subscriptions to Meta Technologies Corporation Bulletin Service are $36.00 per year. Bulletins are similar to the one above, and are mailed first class when the news breaks, not just monthly. Free year-end index with all subs. Subscriptions also available through TAS; those choosing this route also receive FREE a no-punch storage binder. Write to Meta Technologies at 26111 Brush Ave in Euclid, OH 44132, or to The Alternate Source at 1806 Ada Street in Lansing, MI 48910. Please indicate whether you would like your subscription to begin from the beginning (default if not specified) or with the next available bulletin.

TAS SURVEY #4

First, the winners of survey #2 are:

Darrell M. Younts, of Matthews, NC
Wes Kar, of Pheonix, AZ
Ronald J. Ungashick, of Canton, OH

A ten dollar certificate has been awarded to these three gentlemen, which they can use in conjunction with ANYTHING offered by TAS. The certificates are just like cash, and may be used even with our special discounted deals! We would like to extend our thanks to all who participated in Survey #2.

For this month's survey, we are simply going to repeat the theme from last issue, thus providing us with a method of rewarding both you and your favorite author.

Simply go through this issue and select your favorite article (and author). On a separate slip of paper, write down the article you chose, and your name and address. We would also be interested in any other general or specific comments you have regarding TAS. Mail this information to: SURVEY #4, The Alternate Source, 1806 Ada Street, Lansing, MI 48910.

Certificates worth $10.00 each will be awarded to three lucky participants. We will also tally the votes and reward the winning author with a $25.00 certificate.

Winners of TAS SURVEY #3 will be announced next issue, along with the winning author from issue #3. Entries for SURVEY #4 will be accepted until August 1, 1980, and winners will be announced in TAS V1, N6.

****

BRAND NEW!

You'll be challenging all members of your family and friends as well! PINBALL is a fast Z-80 simulation of the real thing, and just as challenging! Beware of the "Bermuda Square"! Good graphics, complete with sound, and just $14.95. In stock and available now, from The Alternate Source.

FLIGHT SIMULATOR--A whole different kind of challenge. It's as realistic as anything we've ever seen before! From the control board to the 3D graphics, this is one you'll never tire of.

Microsoft's BASIC COMPILER has been released! Before you buy, check our price: just $175.00! (Save $20.00 off the list price! That's a week's worth of lunch and gas money!). We'll guarantee you the lowest price; if you find one lower, call us first.

SUPERZAP PATCHES

By Bruce Hansen

This article will give some enhancements to SUPERZAP 3.0 as well as provide one major fix.

One of the major complaints I had about SUPERZAP was that it wouldn't display lower case letters on the ASCII display of a sector dump. But, by changing the machine language display routine, I could display both upper and lower case. Line 32100 contains the following DATA statement:

```
32100 DATA 254,91,48,04,254,33,48,02,62,46
```

To make SUPERZAP display lower case letters (provided the proper hardware mod is installed) change the 91 to a 123.

To stop SUPERZAP from changing all spaces to periods, change the 33 in line 32100 to a 32.

One warning about the lower case change. If the upper-lower case switch is not in the lower case position, any lower case letters will appear as special characters.

There is also a major bug in SUPERZAP. I call it major because if it happens, a diskette could be ruined!

The DFS (Display File's Sectors) option of SUPERZAP allows the user to look at a file without playing around with the directory. To do this, SUPERZAP opens a random file. Suppose the user has completed all changes while in the DFS option. Hitting the "X" key returns control to the menu. Suppose no other options are to be run, so the break key is pressed to exit the program. Believe it or not, the file displayed with the DFS option is still open! If the diskette holding that file was switched with another, and some command which closes a file (such as RUN, CLOSE, etc.) was executed, the diskette would be ruined! It appears that the file is not closed unless another SUPERZAP command is executed.

The fix for this bug is simple. Just change line 25200 to the following:

```
25200 ON ERROR GOTO 0: CLOSE
```

Now when ever the menu is printed, any open files will be closed.

DIRECT STATEMENT IN FILE

By Ken Edwards

This problem crept up on me when I was converting some Model I software to run on the Model II TRS-80. The programs I was trying to convert had to be s ed in ASCII format, and many of the program lines were longer than 240 bytes. When you try to load the program , any line longer than 240 bytes will not load properly, and the computer lets you know this by giving you a "DIRECT STATEMENT IN FILE" error.

On my Model I TRS-80, I usually solve this problem with either SUPERZAP or Z80ZAP. But I don't have SUPERZAP for the Model II yet, so I had to come up with something different. Rather than manually reconstructing each and every program I needed to convert, I wrote this program.

The program will solve any "DIRECT STATEMENT IN FILE" errors encountered on the Model II, and will also work on the Model I for those persons not having SUPERZAP or Z80ZAP.

To use the program, you will first need to know the program line number that is causing the problems (the one with more than 240 bytes). To find this line, Model I users can just load the program. The computer will respond with the "DIRECT STATEMENT IN FILE" error message. Type and enter "LIST". The program will list up through the line with the error. Write down (or remember) this line number.

Model II users can find the problem line by attempting to load the program with the error. The Model II will try to load the program, and, not being able to, will hang up. When the red light on the disk drive goes out, press the "F1" key. The machine will then respond with "DS ERROR". Now, list the program. It will list up to the problem line, and you should make note of that line number.

Now, run my program. It will step through each line of your program, and ask you if that line is okay. If it is, just hit the ENTER key. When it comes to the line with the error, enter the new line, making sure that it does not exceed 240 bytes in length. You will either have to do some editing to the line, or chop some off and insert it later. If you decide to chop some of the line off, you might want to take notes on what you've deleted so that you can re-insert it later.

Sometimes a program will have more than one line that is longer than 240 bytes. When you first load the program , it will only load up to the first line that is too long. If there are others, you won't be able to tell. My program will test for these other lines. If it discovers any, it will let you know. It will also tell you the length of the problem line, so that you know how many bytes you have to chop off. This way, you'll only have to run the program once to take care of any and all problem lines.

PROGRAM LISTING

```
100 CLS
110 CLEAR 3000
120 INPUT "OLDFILE"; O$
130 INPUT "NEWFILE"; N$
140 OPEN"I",1,O$: OPEN"O",2,N$
150 LINEINPUT #1 ,A$
160 IF LEN(A$)>239 THEN PRINT
    "THIS LINE IS";LEN(A$);"BYTES LONG.  RE-ENTER."
170 CLS: PRINT "PRESS ENTER IF LINE IS OKAY,
    OTHERWISE ENTER THE NEW LINE."
180 PRINT A$
190 LINEINPUT C$
200 IF C$ <> "" THEN PRINT #2,C$:
    ELSE PRINT #2,A$
210 IF EOF(1) THEN CLOSE: END
220 GOTO 150
230 CLOSE
```

(Editor's note: When Ken first sent this article, we discussed the possibility of modifying it so that any line not over 239 bytes would automatically be rewritten to disk. Their is such a version in existence, however it does not handle certain 'Direct Statement' errors. Ironically ,one example was in the program and article Ken sent us (via RS-232/Modem). The article had a portion of the above program embedded in a large remark statement. Terminating each line was the standard 0DH, AKA a carriage return. In order to make the program run properly, we had to change the 0D's to 0A's (0AH=down arrow or line feed without a carriage return). For this reason, we left the program in the original form that Ken submitted.

For quick and dirty text editing of lines that only have the 'line too long' problem you can modify the program so that it will automatically write lines shorter than 240 bytes. Append line 160 to also read "ELSE GOTO 200". After the PRINT#2, C$ in line 610 add C$="" after the colon and before the ELSE. I think that'll do it.)

FROM THE SOURCE'S MOUTH

By Joni M. Kosloski

This is sort of our "mid-point" of our first year. It's went by much too quickly. Thought I'd take a peek at where we started, where we are, and where we're going.

Many of our original intentions have been altered. The primary stimulus we gauge many of our decisions on is your direct feedback. We entered with many assumptions, some undoubtedly wrong.

We had expected more response from users groups. We had assumed that many were structured similar to Central Michigan's TUG, i.e., a good mix of programmers, business folk, hobbiests, neophytes and the like who bounced around ideas, provided demonstrations of soft and hardware, including (gasp) other systems. The meeting offers opportunities for persons with similar interests to share learning. The meeting offers opportunities for persons with similar interests to share learning.

In our locale, we're finding that the "red tape" involved puts a strain on the mere existence of the group. Meeting places, printing costs and other factors have quadrupled membership costs (from $3.00 to $12.00). This was the original stimulation for our offer to help market "User Group Original Packages" on the last Bulletin Board.

If you haven't guessed by now, our response has been extremely minimal. My own suggestion would be for each group to appoint a publicity person to help you both locally and nationally. By all means don't make it your newsletter editor! He's got his hands full! I had once envisioned a "nationwide network" of user groups where, by some communication link, a problem once solved would no longer remain a problem. I had intended for TAS to partially supply that link. Should someone not take the initiative to promote your cl quite possibly you could succumb to apathy.

On Line Bulletin Boards could provide another link. We desire to start one in our area as soon as we can keep a system free long enough. Both Brian Allen and Dan Poorman have provided support to initially get the ball rolling. A lot of the local folks dropping by want to see programs run before they buy them, and we have a ton of data entry to perform, not to mention getting a newsletter ready, test software and play. Assuming the Model III maintains complete compatibility, maybe that will free one of our in-house systems. For now, we feel the above mentioned items rate higher.

Although I've seen a few exceptions, most Bulletin Boards are pretty much unorganized data bases which don't justify the long distance expenditure. Close to the source sources say the Source (ohhh...) is pretty slow. If they're using a prime computer they should read Kilobaud. Micro-Net is perhaps the best alternative. Even at $9.00 per hour, they're cheaper than ma Bell, and faster than the Post Office.

There are a few software packages it seems everyone has. We would like to elaborate on these packages. Ron Johnston has provided a demo article for his GSF sort for this issue. Did you know GSF has many other capabilities, such as writing data to tape without the obnoxious leader between every item, or block moves to move variables into protected memory to keep them from getting clobbered when you load another program? Lots of goodies there and that's only one package! Surely someone is using APL for some serious number crunching or the Structured Basic Translator to "structure" your BASIC programs. Quite probably you have much more capability than you realize--sitting right on the shelf--next to your Level I Blackjack! The wheel is already here if we can find someone who wants to market it!

Another implied original intent was to provide information in a real timely manner. This requires a distinction between "news" and "technical" information. With the exception of a couple of editorial-type columns The Alternate Source Magazine (yeah, I think I'll get risky and call it THAT.) will feature material of the technical vein. This is not to say we'll be cut and dried. I treasure the personalities exhibited by Dennis Kitsz, Bill Brown, Jesse Bob and some of our other authors. Memory performing properly, it seems that de-personalization was one of the big complaints about computers not so long ago. Here and now though, we're close to reaching a point where we have time to enter an article in a format usable by us, print out a proof copy and get verification that nothing was manipulated in the transportation. Should eliminate a few bugs that way.

Meanwhile, articles of a "newsy nature" will be delegated to our BTI (Between The Issues). This will include hot rumors, reviews or other matter of a timely nature. Quite possibly we'll also use this as an "overflow" value to trap omissions and bugs from the main issues, and maybe even throw in some semi-technical "meaty" stuff. Although the projected release date will be the "odd" months, we intend to use them in as timely a manner as possible so publication times and occurances will vary as the situation warrants. I think you'll see them expand as The Alternate Source has expanded the past few months. We intend for BTI to take a much more subjective stance than TAS in general. It could prove a good medium for airing complaints, commentary as it relates to the TRS-80 in particular, microcomputing in general or to critique some hot subject. Your letters are always welcome.

Ooops! I think I'm out of room.....

MENU POWER!

By C.W. Simpson

Some folks down the way are talking about starting a new organization for monitoring software. Their approval will equate the "Good Housekeeping Seal of Approval" for computer software. Rumor has it that any program to pass their criteria will contain nice user features like Z80 sort routines and will be menu driven.

Menus work very well with the "ON VARIABLE GOTO (or GOSUB) XXXX, YYYY, ZZZZ" etc. Perhaps another subtlety that would be appreciated is to "key" the menu options to "keywords" that provide clues to the option being selected. For example:

```
<A>DD
<C>HANGE
<D>ELETE
```

This provides a mneumonic which we end users can identify with, substantially more apropos than a numeric key, I think you'll agree.

Ordinarily, this type of routine would produce excessive code, most likely a series of IF statements testing the condition of the variable selected. The problem arises from irregular skips in the ASCII value of each valid input character.

The following could be an effective way to control input and maintain meaningful responses for the user:

```
100 REM ** DISPLAY OPTIONS **
110 CLS:I=5:PRINTTAB(30);"MENU:"
120 PRINTTAB(I);"<A>DD RECORDS"
130 PRINTTAB(I);"<C>HANGE RECORDS"
140 PRINTTAB(I);"<D>ELETE RECORDS"
150 PRINTTAB(I);"<R>EVIEW RECORDS"
160 PRINTTAB(I);"<S>ORT RECORDS"
170 A$=INKEY$:IFA$=""THEN170
    ELSEOP=INSTR("ACDRS",A$):
    ONOPGOSUBXXXX,YYYY,ZZZZ,AAAA,BBBB
180 GOTO110
```

The "ON GOSUB" statement will only execute a valid subroutine if OP (OP stands for OPtion--do you know why I didn't use the whole word?) is greater than zero or less than six. The only way OP can obtain one of these values is if a proper selection is made.

A beneficial side effect of this routine is that it will preserve the menu display if any invalid character is input, such as a down arrow or the clear key. If the operator does input a valid character, s/he is immediately rewarded by execution of the proper subroutine.

In order to modify the routine, there are three steps to consider. (A) Print the new option with its mneumonic letter emphasized. (B) Add the mneumonic letter IN THE PROPER POSITION to the INSTR literal parameter. (C) Provide a subroutine IN THE PROPER POSITION for the new option to branch.

Thanks to Allan Moluf for first exposing me to this technique. It was placed on paper especially for Brian Allen (of the popular Chicago Bulletin Board fame) as we were discussing interactive input recently.

****

(Here is a subroutine, submitted by Richard K. Riley of Augusta, ME that will POKE the date and time into TIME$, without going back to DOS or CMDing with NEWDOS. Very handy! ed.)

```
7500 CLS:INPUT"IS THE CURRENT DATE/TIME DATA CORRECT";DT$
7510 IF LEFT$(A$,1)="Y" RETURN
7520 INPUT"WHAT IS THE CORRECT DATE (MM/DD/YY)";DT$
7530 POKE16454,VAL(LEFT$(DT$,2))
7540 POKE16453,VAL(MID$(DT$,2))
7550 POKE16452,VAL(RIGHT$(DT$,2))
7560 INPUT"WHAT IS THE CORRECT HOUR (HH)";DT$
7570 POKE16451,VAL(DT$)
7580 INPUT"WHAT IS THE CORRECT MINUTE (MM)";DT$
7590 POKE16450,VAL(DT$)
7600 POKE16449,0: RETURN
```

****

(And here's yet another POKE subroutine submitted by Mr. Riley! These two lines will buzz the relay in the interface (a nasty shock!), and can be used for just about anything. Mr. Riley uses them for data entry errors and also to flag the end of a sort routine. ed.)

```
7700 FOR X=1 TO 10: POKE 14308,X: NEXT
7710 FOR X=1 TO 10: NEXT: RETURN
```

BOOT/SYS.WHO EXPLAINED

By Richard C. Vanderburgh

In The Alternate Source issue #3 on page 35, persons were urged to type in (from DOS mode) BOOT/SYS.WHO, hit the enter key, and while DOS was looking for the program, push the "2" and "6" keys down together. The resulting message is easily explained.

This DOS command loads in RAM starting at 4E00H, an encrypted text stored on disk as part of the BOOT system program, and starting at 4DD7 a short decoding routine. When control is passed to 4DD7, the decoder begins by copying the contents of keyboard RAM location 38FF to the B register. If the 2 and 6 keys are depressed, 44H gets put into B. HL is then initialized to point to the beginning of the encrypted message, and a loop entered which first XORs each coded byte with 44H (or whatever got put into B), then XORs it with the low-order byte of the address it comes from (pointed to by L). The decoded ASCII characters are then output to the screen, until the control character "3" is encountered, at which point an endless do-nothing loop freezes the display and locks out the keyboard.

There are three clever elements to this encryption scheme: first, the required keyboard input is not tested for (such as with a CP 44), eliminating a possible hint; secondly, the encrypted message will only decode properly if it is located at the beginning of a RAM page block (address beginning XX00); and thirdly, there are no obvious symmetries or repetitive patterns in the coded text. If Randy Cook had coded the decoder itself, and had it self-destruct following execution, I'd probably still be trying to figure it out!

Changing the message to one of your own choosing is not too difficult, since the exclusive or (XOR) function is reversible, and the order in which two successive XORs are performed makes no difference. The tricky part is getting your new message to the right place on disk, with the directory sector-linkage properly embedded. You'll find helpful a high memory monitor (such as RSM-2D), and an understanding of what Harv Pennington talks about in his "TRS-80 Disk..." book concerning directory structure and modifications to disk-stored code. The decoding routine starts at Sector 0, relative byte 2D7; the message at Sector 1, relative byte 4.

If there is sufficient interest, I'll get into more detail in a future issue of TAS.

Using VARPTR to Help Minimize BASIC String Compression Time

This is the second in a series of articles on the VARPTR function. The first, "What's Where VARPTR Points", appeared in TAS V1, N3 and covered basic introductory material. The present article assumes that you are familiar with the previous one. The last few paragraphs briefly talked about a method of using VARPTR to let you help BASIC manage string storage space. At the end of this article I will give a few short program segments that make use of that technique, but the major portion of the article will deal with BASIC's use of string space in general. Being familiar with that process can be helpful in a wider context than just using VARPTR, since there are other techniques that can also be employed to make more efficient use of strings and string space.

The situation where making use of VARPTR will help with the string space problem can be pretty much summarized as one where the program is:

1. using a lot of strings, and
2. moving string values around in memory by setting string variables equal to strings that already exist (the values of other string variables), rather than creating new ones.

Most programs that use strings operate on the strings in order to create new values. Only two general situations come to mind where there is a lot of "string swapping". One would be where you are sorting the string values in an array into some desired order so that they are easier to use, either by the program or by you when you print them (like sorting a name-list into alphabetical order). The other involves the need to insert new strings into, or delete them from an existing list that is in an order that you want to maintain (such as what an editor program has to do when you are inserting or deleting lines). To make it clear why there is a difference between "making" values and just "moving" them, let's take a look at BASIC's use and management of string space.

When your program starts to run, BASIC automatically sets aside 50 characters of string space to hold the values of the strings you use in the program. If you use no strings at all this 50 bytes of memory will not get used. If the total number of characters you are using at any one time for all of your strings is less than 51 characters, then the reserved string space will be adequate for your needs. When more bytes of string space are needed at one time than the amount that is reserved, BASIC will stop the program and give you an OUT OF STRING SPACE (OS) error message. It is necessary to use the CLEAR statement to reserve a larger amount of string space for such a program. If an attempt is made to reserve more string space than there is memory available, then you will get an OUT OF MEMORY (OM) error message when the CLEAR statement is executed. Since we usually do not know in advance just how much string space we are going to need, there is a certain amount of trial and error here.

The string space that is reserved, either by default or by your CLEAR statement, is always at the very top of memory , unless you have answered the MEMORY SIZE question with a value that reserves some memory at the very top. For simplicity, let's continue to refer to the "top of memory", although that may not be quite correct. (You may find the following discussion easier to follow if you refer to the memory map shown in Appendix D of the Level II Reference manual. Remember that even though we refer to 16K Level II machines, that the operating system really has to deal with 32K of addressable memory in order to use the 16K of user available memory. ROM and the part of RAM that is used for special purposes occupies approximately the first (low) 16K bytes of memory. Also, it may be helpful to keep in mind that "high" and "low" memory refer to the address values. Many diagram representations of memory, like Appendix D, show "low" memory at the top of the page and "high" memory at the bottom. It is sometimes easy to get confused about which end is up.) As far as the numbers are concerned then, your program will be loaded into memory starting at address location 17129 (42E9H), and will occupy as much memory as it needs for its storage. Immediately above the program is the area where information is stored about simple variables and arrays, and is the section of memory that is "looked at" by the VARPTR function.

After string space is reserved, and when your program gives the first string variable a value, the characters for that value will be stored at the top of string space (unless the value is a constant in a program statement or is READ from a DATA statement, both of which will cause VARPTR to point to someplace in the program storage area). If the variable is A$, then X in the following statement

```
70 X=PEEK(VARPTR(A$)+2) * 256 + PEEK(VARPTR(A$)+1):
   IF X > 32767 THEN X=X-65536
```

will give us the memory address of the first character of the string. BASIC keeps track of how much string space has been used, and when the next string variable is given a value, that value is stored immediately below the first. There is no separation in memory between where the two strings are stored; BASIC "remembers" where each is stored by storing the length of the string and its starting memory location (once again, this is the VARPTR information). After several string values have been set, we have the characters for the several values strung out end-to-end, starting at the top of string space and working down.

If the program now needs to change the value of a string variable, say the first one, it simply puts the new value in the next available string space (just like it was setting a value for the first time), resets the pointers for that variable to indicate the memory location of the new value, sets its new length, and abandons the data in the string space that held the original value. This process continues when string values are changed, until there is no more free string space. What results is a trail of "dead strings" that was referred to in the last article. When a new value needs to be stored and there is no room for it, BASIC stops executing your program and does "garbage collection" so that the memory that is occupied by the "dead" strings can be reused. During this process is when you get those pauses "for no apparent reason" in program execution and when the computer will not respond to the keyboard, not even the BREAK key.

The questions can be asked, "Why not reuse the space that was occupied by the old value when storing the new value?" or "Why not clean up (snug up) string space after each value is set, so that we don't get the long pauses?" The easy answer to both of these questions is, that is the way MICROSOFT decided to do it when they programmed BASIC. One other answer to the first question has to do with the fact that most of the time the two string values will not have the same number of characters, i.e., not be the same length. If the new value is longer than the old it will not fit into the same space, and splitting it into two pieces would require extra bookkeeping operations that would use more time and memory, and BASIC would have to be redesigned because that's not the way it does things now anyway. As for why it doesn't clean up after itself as it goes, I suspect that it has proven to be more efficient in the long run to do it all at once, and that MICROSOFT takes advantage of that. Whatever the case, let's take a quick look at the clean-up process.

Since the process of initially filling string space starts at the top of string space, it only seems reasonable that "string space compression" should start there also. Essentially what happens is that BASIC first searches all of the currently defined string variables to find the one that has its current value stored the highest in string space memory. When that is found, the string value for that variable or array element is copied into the storage locations at the very top of string space (highest memory address). The pointer for the variable is set to the new location, then BASIC searches for the string variable with the next highest memory pointer and copies the value for that one snug up against the first, resets its pointer and goes after the next one. This process continues until all of the currently defined strings have their values moved and there is no "dead" space separating any of them. BASIC then sets the pointer to the new "beginning of available string space" and returns to the execution of your program.

There is one other aspect of string space management that deserves an article of its own, and therefore will only be mentioned here. It has to do with the "temporary" string values that BASIC creates while executing your program, but that are not used directly as values for variables. For example, in the statement

```
10 IF LEFT$(A$,3) <> "END" THEN GOTO 40
```

requires that BASIC temporarily build the three character string (LEFT$(A$,3)) in order to check it against "END". After the operation of the IF is complete, BASIC deletes that string from string space; since it is at the very end of the "good" strings that is easy to do. This helps keep things tidy and cuts down on the frequency with which BASIC has to stop for string space compression.

There is one glitch here, however. If more than one temporary string is created in a single statement, only the last one is deleted. Any others contribute to the "trail of dead strings" (sounds like a good title of a science fiction movie). For example, in the statement

```
20 IF (LEFT$(A$,3)+LEFT$(B$,3)) <> "ABCXYZ" THEN GOTO 50
```

more than one temporary will be created but only one of them will be removed from active string space and the other(s) will be "dead". If you happen to get one of these situations inside a loop that repeats many times, string space availability can deteriorate rapidily. The general situation with temporaries is a good deal more complex then is even alluded to here. A general statement of the conditions under which temporaries are built but not deleted, would require more page space than I want to devote to it now. There are ways to compensate for this that require a bit of trickery, and another article to explain.

Before getting back to VARPTR let's take a short side trip to look at some general considerations about the way that string space is managed that may be helpful when writing programs.

The more string space that you make available to the program with the CLEAR statement, the longer it will take for the running program to fill the space, and the less often it will have to stop for string space compression. Providing more string space does not affect the amount of time it takes to do the compression; because of the way the process works, only the strings that have defined values are copied, and it takes the same amount of time to do that in 15K of string space as it does in 2K.

The time in seconds for a string compression is approximately S(T-N/2)/5000, where T is the total number of strings, S is the number of strings in the string space area (i.e., have not been READ from DATA statments or set as constants in the program), and N is the number of null strings. For example, in the following program

```
10 CLEAR 1000
20 DIM S$(799)
30 FOR J=0 TO 499
40  S$(J)=CHR$(32)
50 NEXT J
60 X=FRE(A$)
```

line 60, which forces a string compression (explained below), will take about 65 seconds to compress string space because S=500, T=800, N=300. If line 40 is changed to

```
40 S$(J)=" ": REM ONE SPACE
```

then line 60 will take almost no time because all the strings are constants and point into the program instead of into string space. (This example, courtesy A. Moluf)

It might be nice if BASIC operated such that it dynamically allocated string space to use all of available memory at any given time, but there are several complications that would be involved that give good reason for its not doing so. It is possible, however, for you to accomplish pretty much the same thing very simply. The CLEAR statement does two main things when it is executed: it deletes all variables (numeric and string) that are defined at the time, and it allocates the amount of string space you specify. The MEM function will tell you how much free memory is available. If you load a program and PRINT MEM before it is RUN then you will find out how much memory is available for variable names and their values, for the stack (see memory map) and for string space. You can figure out by trial and error how much variable and stack space is needed for a given program to run. Once you know that (at least approximately) then you can allocate the remaining memory to string space with a CLEAR statement. As you are writing a program its length changes and would require you to change the CLEAR allocation accordingly. However, if you have determined that a program needs 2000 bytes for the variables and stack, you can dynamically and "automatically" allocate string space, with respect to the changing length of the program, by putting the following statement at the beginning of the program.

```
10 CLEAR 0: CLEAR MEM-2000
```

The CLEAR 0 forces all variables to be deleted (if the program has been running previously during the time is has been loaded), and therefore allows MEM to be the maximum for the current length of the program.

It is possible to force a string space compression to be done any time you choose, by causing the FRE function to be used. This function will give you a numeric value that is the amount of FREe string space available when the function is executed. One of the things that the function does before calculating available space is to do a compression. If, for example, you come to a place in a program's execution where you have finished with the values that are assigned to the elements of a string array , then you could set the variables to null and force a string compression while there are fewer "active" strings to be copied. You can "exercise" the FRE function by simply setting a numeric variable that is not being used for something else, e.g., ZZ=FRE(A$).

Another general consideration with respect to the string management process is that it will take longer to do string compression when there are more strings. When string space is filled with many short strings, each one has to be searched for and copied separately. Each element of a string array counts as one string, and each must be dealt with individually. If there is an option in the programming process to use fewer long strings to hold the same information that might be stored in more short strings, the string compression will take less time. This difference can be as much as several minutes under certain circumstances. (If 1000 strings were changed to 100 longer strings , the compression time would drop from three minutes to less than two seconds. -- A. Moluf) But as always, nothing is free; I did this in a program recently, and found that the time required to unpack the short strings from the long ones consumed a fat chunk of the time I saved on string compression: now I have a new problem.

One last general consideration leads us back to VARPTR: When a program statement is executed that sets one string variable equal to another (A$=G$), a copy of the value is made so that the same value is stored in string space twice at the same time. If we immediately execute another statement that sets G$=C$, then one copy of the duplicated G$ string becomes a "dead" string (Dead G-strings are not something we need, whether we're playing guitar, writing programs or watching burlesque). If this is done repeatedly..., I need not dwell on the consequences.

Suppose we have a program that accepts the input of many strings from the keyboard, say a list of names, and stores them in the elements of the array N$. Once the input is complete, we want to sort the name list into alphabetical order. If NN is the count of the number of strings that have been entered, then the following program seqment will do a simple "bubble" sort of the names.

```
100 FOR K=1 TO NN-1
110 FOR J=1 TO NN-K
120 IF N$(J) > N$(J+1)
     THEN T$=N$(J): N$(J)=N$(J+1): N$(J+1)=T$
130 NEXT J
140 NEXT K
```

Without going into a lot of detail, a bubble sort works by making successive passes through the array, carrying the "heaviest" (highest valued) elements to the bottom, and gradually allowing the "lighter" elements to "bubble up" to the top. The bubble sort is not a very elegant or efficient sorting algorithm, and the above implementation can be made more efficient in several respects, but let's concentrate on VARPTR.

Notice in line 120 that each time an exchange of elements is made the value of three strings are changed: the duplicated copies remain momentarily, then one copy for each of the three variables becomes a "dead" string. If the list is long or string space is short,... Since this article has become more lengthy than I anticipated, I will not repeat an explanation of the VARPTR technique (outlined in the last article) that can be used to eliminate the "dead" strings in this process. If, in the example program statements that are given in the last few paragraphs of that article, you change all occurances of A$ to N$(J) and B$ to N$(J+1), then those statements will work in the above example.

To give a concrete example: Suppose we are sorting 100 names and that these strings use 80% of the available string space. Executing the statements in the loop (ignoring string compression) will take about a minute and a half with

NN=100. If the names require 2500 exchanges in line 120, we will generate 7500 garbage strings. Since about 25 strings will fit in the remaining 20% of space, BASIC will do string compression about 300 times. Using the formula above, with S=T=100 and N=0, we get about 2 seconds per string compression, or 10 minutes for all the string compression, and 12 minutes for the entire sort. Doubling the available string space would result in space for 150 garbage strings, 50 string compressions and less than three minutes for the entire sort. We can get a similar improvement using VARPTR with no additional string space; the additional statements needed will require more time to execute, but we will have reduced string compression time to zero. (A. Moluf)

Before we sign off for the night, let's set up one more general situation where this technique might come in handy. Suppose our name-list program has the ability to edit the list after it has been sorted, by deleting names from the list or inserting new ones into the list. When we delete, we want to shorten the list by moving all of the names following the deleted name up one array element, and reducing NN by 1. If DL is the array subscript value of the name to be deleted, then we can do this by

```
30 FOR K=DL TO NN-1: N$(K)=N$(K+1): NEXT: NN=NN-1
```

In which case we will create a lot of "dead" strings. Or we can use

```
30 FOR K=DL TO NN-1
40  POKE VARPTR(N$(K)), PEEK(VARPTR(N$(K+1)))
50  POKE VARPTR(N$(K))+1, PEEK(VARPTR(N$(K+1))+1)
60  POKE VARPTR(N$(K))+2, PEEK(VARPTR(N$(K+1))+2)
70 NEXT K
```

In which case we create one "dead" string, the one being deleted. A similar situation exists when we insert a name into the list, except that the direction of the FOR loop is changed to move everything down one to make room for the new name.

NOTE: For an example of VARPTR being used for a completely different purpose, see Dan Yerke's article in TAS V1, N2 entitled "No Frills".

(Appreciation to Allan Moluf, who supplied much of the "hard fact" presented above, or pointed me in the right direction so that I could find it myself.)

LINESET

By Victor J. Amor

Unless you've got Radio Shack's Line Printer I, aka Centronics 779, this program is probably not for you. Most people I know who do have printers for their TRS-80 Model I DO use this particular machine, so you're probably included.

But what does this little program or subroutine do, you ask yourself? Well, the Line Printer I is continuously adjustable from 10 to 16.5 characters per inch (CPI), BUT there are no marks nor clicks nor anything of consequence to tell you just how many CPI the machine is printing. So out of frustration was born this program.

The print density means three things: first, at 16.5 CPI, the characters are smallest in size (dots are closest together) and at 10 CPI, the largest (dots are farthest apart). Secondly, at 16.5 CPI, you can get more characters per line (CPL), thus more lines per page, than you can with 10 CPI. Thirdly, at 16.5 CPI you can print your text faster than at 10 CPI.

The problem lies in "wrap-around". On the CRT, when text cannot all fit onto one line, it is continued onto the next line, and you haven't lost it. But unless your line printer is set to 132 characters per 8 inch line, you will lose some of the text output to the machine if your program line length or string length is not matched with your CPL. If your maximum text is 80 characters long and you've got your machine set to print 80 CPL, you're fine...if not, you're in trouble.

I use multiple statement lines a lot, and most of the time I'll have well over 132 characters, so I set my printer to print 132 CPL. But when I print the names and addresses on our envelopes, I need a larger character size. In this instance, I have to set the printer for 90 CPL.

When the program is run and it asks you for the CPL you'd like, don't be hesitant. Play around with several possible replies and give them a try. The last two or three characters printed will be the number you requested, preceeded by the number of characters you requested LESS the two or three digits that are used as a "benchmark" (?). Just adjust until those digits are at the right hand margin you choose.

Do what you will with the program. It was a solution to one of my problems; I hope it can help you.

<PROGRAM LISTING NEXT PAGE>

LINESET PROGRAM LISTING

```
1000 ' "LINESET" BY VICTOR J. AMOR
1010 '      562 H PIIHOLO RD
1020 '     MAKAWAO, HI  96768
1040 '
1050 CLS: CLEAR300: GOSUB1060 : PRINT"ALL DONE!": CLEAR50: END
1060 PRINT@512,CHR$(31): INPUT"HOW MANY CHARACTERS WOULD
     YOU LIKE ME TO PRINT";HOWMANY
1070 IF HOWMANY<10 OR HOWMANY>255 PRINT@832,"BETWEEN 10
     AND 255...PLEASE TRY AGAIN": FOR DELAY=1 TO 640:
     NEXT DELAY: GOTO1060
1080 HOWMANY$ = MID$(STR$(HOWMANY),2): REM GET RID OF SIGN
1090 LARGA = LEN(HOWMANY$): NUMBER = HOWMANY-LARGA
1100 TEST$ = STRING$(NUMBER,"$") + HOWMANY$
1110 CLS: PRINT@512,"";: INPUT"HIT <ENTER> TO BEGIN";A$
1120 GOSUB1160
1130 CLS: PRINT@512,"HIT ANY KEY TO ESCAPE";: PRINT@576,
     "THE LAST 2 OR 3 DIGITS WILL BE THE NUMBER YOU
      REQUESTED.";
1140 LPRINT TEST$: FOR DELAY=1 TO 200: NEXT DELAY
1150 I$=INKEY$: IF I$="" THEN 1140 ELSE CLS: RETURN
1160 CLS: IF PEEK(14312) <> 63 THEN PRINT@512,
      "PRINTER NOT READY...";: FOR DELAY=1 TO 100:
     NEXT DELAY: GOTO 1160 ELSE RETURN
```

************* *****************

(Roy Scott from Newport, TN also has a Centronics Line Printer Patch users might find very helpful. ed.)

Trying to use the "STRING$(X,138)" function to skip lines on the Centronics Line Printer has caused me some difficulty. This function works fine, but when using it, only the first blank line so generated registers on the TRS-80 line counter that is used to calculate for form feeds.

Therefore, the POKE statements that you use at the beginning of your program to set the line counters for form feeds must be changed to take this into account. Where you would normally use the following form:

10 POKE 16424,67: POKE 16425,1 (For 8.5 by 11 inch paper)

you would need to use the following form:

10 POKE 16424,67: POKE 16425,(X-Y+1)

where X equals the total number of 138's used in the program and Y equals the number of STRING$(X,138) statements in the program. The one (1) merely accounts for the first printing line in the program.

I hope this information will be of use to some other readers.

LAST MINUTE ODDS & ENDS

OOOOPPS...Forgot a few things here and there:

We forgot to put in a subscription promo! Those desiring to subscribe should send $9.00 for six issues to The Alternate Source, 1806 Ada Street, Lansing, MI 48910. Phone is 517/485-0344 or 517/487-3358. Call anytime.

We should mention that this issue is available on diskette or cassette for $5.00, as are all issues. Send to address above.

We are now sponsoring a new service for DOS owners. We are not encouraging your business, although we are making it available as a last resort. With the help of our staff and some local programmers we will attempt recovery of damaged diskette directories and files. Minimum charge for recovery is $20, non-refundable and service is non-guaranteed. All monies goto the person who performs the actual recovery. While some diskettes can be reclaimed in just a few minutes, some take hours. Per our agreement, price includes a new diskette and your original returned exactly as you sent it. In some instances, only partial files can be recovered, sometimes to the detriment of other files. You should clearly state the file that is most important, type of file (data, BASIC, machine language, whatever), approximate length and any other information that will prove helpful in recovery. Please include your phone number. In certain instances prince may be more and the programmer will want to verify whether the file is worth the extra time. We aren't sure if this service is going to be used, but it is there should you lose an important file and have no one else to turn to. The one definite instance where recovery will cost more is when the entire directory is overwritten and the directory will have to be completely reconstructed -- not usually the situation. We hope you never have to call us.

We've added a brand new program to our library, but haven't had time to draw up ads yet. It's a program that will allow you to save machine language files under the MICRODOS operating system, and it sells for just $14.95. Any interested MICRODOS owners can contact us for information.

And that's all for this issue! Until next time (August 20th, 1980), may the source be with you!!

AFTERWARD FOR ISSUE 4

This was the kind of issue I had dreamed of putting together a few months back. On the darker side, we were feeling the pinch for Level II articles and software.

KILLER was a program Allan Moluf originally provided Central Michigan TRS-80 Users Group (CMTUG) members, but was kind enough to let us buy the publishing rights.

Regarding our issues on cassette or disk: They are causing many problems -- CLOADing for one. The mixture of BASIC, EDTASM SOURCE and OBJECT for another, and general complaints about remuneration to authors for yet another. Considering the amount of time making them, and the few sold, we're about ready to discontinue the practice -- even though I would have liked to have seen the project at least pay for itself.

Apparently no one is killing directories, or everyone has read "Disk and Other Mysteries" by this time. We received no response for our offer to reclaim diskettes.

Two Dollars
Volume 1, Number 5

THE ALTERNATE SOURCE

The magazine of advanced applications and software for the TRS-80.

In this issue:

Editorial RAMbling

By Charley Butler

Are you a hacker? Find out for sure by reading the August Psychology Today. Two great articles relating to some of our breed, The Hacker Papers and The Age of Indifference.

Just witnessed PerCom's new DBLDOS working with their double density board and Radio Shack's drives. It's a real fine feeling to see your meager 35 trackers come back with 120 grans when you do a "FREE". Forty track folks will have about 160 FREE grans. For us, this system appeared just in time -- our mailing lists were testing the limits of 35 track drives. I will mention that the sample demonstrated is an actual production model and NOT a prototype! Shipment of these boards, complete with DBLDOS, should be well underway by the time you read this. We have had a quantity of these boards on order for a couple months. If you would like more information, give me a call. The first two out of the box are reserved for our systems!

Sorry about all the lengthy articles this issue. Hope you find at least a couple that are suitable for your applications. Roxton Baker's PenRam and Jason Potter's Entry are part of larger systems, but each make excellent stand-alone applications. Al Domuret provides the first extensive review of NEWDOS-80, what is supposed to be the first in a three part series analyzing and comparing NEWDOS-80 with VTOS. Unfortunately, VTOS 4.0 has yet to arrive. The patches for NEWDOS-80 are the first we've seen...including the mythical ZAP sheets from Apparat. Thomas Frederick further elaborates on Bill Brown's string manipulation techniques -- a real time saver for those bulky applications. Dennis Kitsz continues his winning ways with "Let Your Fingers Do The...". Dennis is the winner of our issue #3 contest and wins the 25 bucks. For participation, Patricia Bryan, Douglas Clark and John E. Fisher are winners of $10.00 certificates. Don't forget to vote for your favorite author in this issue! A postcard will do.

TAS is published bimonthly by Charley Butler and Joni Kosloski at 1806 Ada Street, Lansing, MI 48910. Subscriptions are $9.00 per year, USA; $12.00 per year, CANADA and MEXICO; and $15.00 dollars per year FOREIGN. All monies must be in good ol' US currency. Subscriptions may be addressed to the above. We actively solicit meaningful articles and information relating to Tandy's TRS-80 and compensate for each article published. Call or write for more info. Entire contents is Copyright (c) 1980 by The Alternate Source unless copyright notice is declared by individual authors.

LET YOUR FINGERS DO THE.....

The Keyboard Scan of the TRS-80

By Dennis Bathory Kitsz

(You know, it's a little bit funny. We got a call from a fellow who was praising the quality of articles within our magazine, and he happened to mention that he was just a little bit tired of seeing so many articles on the keyboard scan. And what should we get in the mail the next day? You guessed it! This article, though, is the most comprehensive we've seen; it not only tells you about the scan, but also what you can do with it. We feel many readers will benefit from the discussion and ideas Dennis presents here. jmk)

Arrows? Control codes? Autorepeat? Whatever it is you would like that has to do with the keyboard, you can do with the TRS-80. The designers of the machine chose not to use an ASCII keyboard...one that outputs a code for each key pressed; instead, the keyboard is a matrix of switches. Because of this decision, the TRS-80 keyboard can be extremely versatile with a minimal body of software. In this issue, we will look at the TRS keyboard scanning routine and next time offer a few changes (improvements, I like to call them) to it.

First, we will have a look at the keyboard matrix itself; if you have been programming in machine language, or even relatively sophisticated BASIC, this map will be familiar:

Address								
3801	@	A	B	C	D	E	F	G
3802	H	I	J	K	L	M	N	O
3804	P	Q	R	S	T	U	V	W
3808	X	Y	Z					
3810	0	1	2	3	4	5	6	7
3820	8	9	:	;	,	-	.	/
3840	ENT	CLR	BRK	↑	↓	←	→	SPC
3880	SHIFT							
DATA ⟶	01	02	04	08	10	20	40	80

At first, the arrangement of the address lines and data information may seem unappealing. What is the use of having address and data information that doubles at each change? Why not just use 3801, 3802, 3803, and so on? The reasons will eventually become apparent in the ease and speed of the keyboard scan, but consider this binary for a moment....

```
00000001    00000010    00000100    00001000
00010000    00100000    01000000    10000000
```

...and there you have it. The bit is bumped along for each keyboard row and column, so that the presence of a single active bit instantly identifies any of the (potential) 64 keys.

The entrance to the keyboard scan is made at a jump from an address in the ROM's RAM switchboard, 4016 (unless otherwise noted, all data and address notations are in hex). Normally at power up, the jump address 03E3 is put in place, and the keyboard scan is entered at that point. Registers BC, DE, HL and A are used in the scanning process:

```
03E3     21 36 40     LD HL,4036
03E6     01 01 38     LD BC,3801
```

The HL register pair points to first RAM location at which the keystrokes will be stored. BC is set to look at the first row of the keyboard, whose memory map is 3801 to 3880, as noted earlier. Register D is set to zero, and it will become a "row counter"; the process begins at address 03EB:

```
03EB     0A           LD A,(BC)
03EC     5F           LD F,A
```

The accumulator reads the data at BC (recall at the outset it is pointing to the first keyboard row, 3801). The information it finds is stored in the memory location pointed to by HL (4036).

Okay so far. Now comes some of the interesting stuff that distinguishes this scan as an excellent piece of writing:

```
03ED     AE           XOR (HL)
03EE     73           LD (HL),E
03EF     A3           AND E
03F0     20 08        JR NZ,03FA
```

This short segment is responsible for the "rollover" capabilities of Level II. The contents of the accumulator (the keystroke, if found) is XORed with the previous contents of 4036. Recalling how the XOR function works, we discover that if the key pressed was the <u>same</u> as the previous one at this row, the accumulator will be "toggled" to zero. In any case, the current keystroke, whatever it is, is now saved in 4036 (so that next time 'round, it knows if a key is <u>still</u> pressed).

If the key was the same, AND E will be the result of A-toggled-to-zero AND the found keystroke...or zero. If there was no key pressed, the result will be A-XORed-with-HL (which is essentially irrelevant) AND E-which-is-zero...or zero. The test at 03F0 is for not zero. Under these conditions, it fails, so the program continues:

```
03F2      14            INC D
03F3      2C            INC L
03F4      CB 01         RLC C
03F6      F2 EB 03      JP P,03EB
03F9      C9            RET
```

The "row counter" (D register) is incremented, and the low-order byte of HL is incremented (to storage address 4037), and the low-order byte of BC is <u>rotated</u>. Recalling the keyboard matrix, we can see that this command to rotate moves us from 01 to 02, from 02 to 04, from 04 to 08, from 08 to 10, etc. That keeps track of the row that the scan is looking at, and as long as the result of the rotate is positive (bit 7 low), the loop will travel back to 03EB, where the next row will undergo the same testing as each previous one. When RLC C shifts the row pointer to 3880, then bit 7 will be high (10000000); this is "negative" in Z-80 architecture, and the loop falls through. Why does it fall through before checking the contents of address 3880? Because the only thing in this row is the shift key; it does not offer a decipherable code by itself, but merely modifies the information found when some <u>other</u> key is depressed. This explains why, among other peculiarities, INKEY$ does not acknowledge SHIFT.

When the loop falls through, the program encounters a RETurn from subroutine, which directs it immediately back to the rest of BASIC. The routine is remarkable, looping through just over 100 bytes when the keyboard is clear. Although not as time-efficient as obtaining input from a memory-mapped ASCII keyboard, it is quite speedy, and offers considerably better "rollover" than many encoded keyboards.

When a key is pressed, the program jumps to 03FA, and is able to provide upper/lower case ASCII codes, special functions, and, incredibly enough, all of the "missing" ASCII control codes (form feed, ring bell, etc.). Let us now follow it through:

```
03FA        5F              LD E,A
03FB        7A              LD A,D
03FC        07              RLCA
03FD        07              RLCA
03FE        07              RLCA
03FF        57              LD D,A
```

The position of the keystroke found has been stored in register E; recall that this is the "column" of the keystroke. The row itself is not yet accessible, but the row counter (register D) is crucial to determining it. After E is saved, the accumulator is loaded with the value in this row counter, and rotated to the left three times. For those shaky in their binary arithmetic, this is the effect: if a decimal number is 045, a left rotate makes it 450. This is multiplication by ten. If a binary number is 010 (decimal 2), a left rotate gives 100 (decimal 4)...in other words, multiplication by two. Therefore, three left rotates gives us 2x2x2, or multiplication by eight. That result is saved back in register D.

The purpose of this clever ploy will soon become clear:

```
0400        0E 01           LD C,01
0402        79              LD A,C
0403        A3              AND E
0404        20 05           JR NZ,040B
```

Here the C register is set to 1, sucked up by the accumulator, and ANDed with E (remember E still contains that keystroke-column byte). If the result is not zero (that is, if E equals 1), then the loop falls through and the program moves on. But have a look at what follows:

```
0406        14              INC D
0407        CB 01           RLC C
0409        18 F7           JR 0402
```

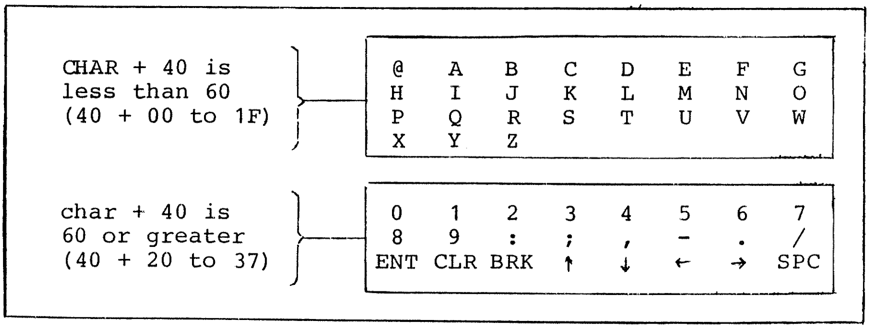
What is this about? Well, the D register, which contains 8 times the row value, is being incremented <u>each time</u> C is being rotated...making the lower three bits of D serve now as a <u>column</u> counter! Whoa, you say, back up there. Okay, here it is. The original value in D could have been 0 through 6, depending on the row in use. When shifted three times, the possible values become 00, 08, 10, 18, 20, 28, and 30. Each of <u>these</u> possible values, when incremented through all seven possible columns, might contain 00 to 07, 08 to 0F, 10 to 17, etc., up to 37. This gives us a complete, distinct value to represent each key.

Now a fairly crude process of byte-sized hunt-n-peck begins. The status of the SHIFT key is checked, and set aside in register B. The de-multiplexed keystroke value in register D is snapped back into the accumulator, and the comparisons take off:

```
040B        3A 80 38        LD A,(3880)
040E        47              LD B,A
040F        7A              LD A,D
```

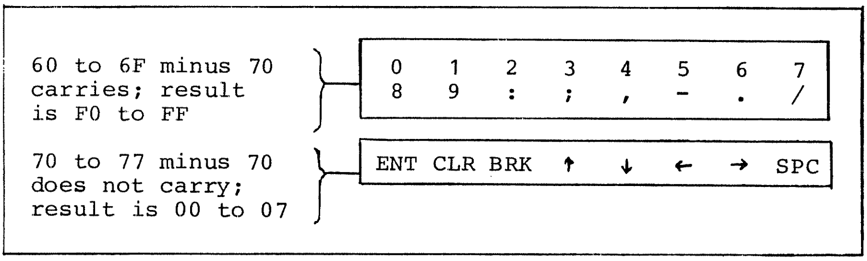
The character search can be followed through several branches; we will start with the most straightforward, and progress through some of the unique (and little publicized) aspects of TRS-80 keyboard output.

The program sets the character equal to character plus 40 (address 0410), and checks if the value is greater than or equal to 60 (0412).



If the compare finds a value less than 60, the routine rotates the SHIFT key value which had been saved in the B register (0416). If SHIFT is released, the value in B is zero, and hence the rotate resets the carry flag (0418). The program moves directly to the terminal steps at 044B (to be discussed later). At that point, the character contained in A would be in the range 00+40 to 1F+40, the ASCII values for upper case (@, A-Z, left bracket, separator, right bracket, carat, and cursor). This is the software routine that causes the bizarre "inverted" shift pattern on the TRS-80...no shift for upper case!

If the character test at 0412 returns a value greater than or equal to 60, then 70 is subtracted (0429). No carry is generated if the test value was greater than or equal to 70, so this further separates the keyboard. See the diagram below:


60 to 6F minus 70 carries; result is F0 to FF

70 to 77 minus 70 does not carry; result is 00 to 07

At address 043D, the value in the accumulator (00 to 07) is rotated left, producing the <u>even</u> values from 00 to 0E. The SHIFT byte in B is rotated right into the carry flag; if a carry is generated, the accumulator value is incremented (0442), providing the values 0+1, 2+1, 4+1, etc. -- in other words, the <u>odd</u> values from 01 to 0F.

What follows is a classic example of machine language table look-up. HL is set to 0050, the address of the table in ROM; BC will be used as an offset, with B set to 0 and C is made equal to A. When BC is added to HL, a resultant address (0050 to 005F) is produced, and the contents of that address are loaded up by the accumulator. Here is a look:

```
0443        21 50 00      LD HL,0050
0446        4F            LD C,A
0447        06 00         LD B,0
0449        09            ADD HL,BC
044A        7E            LD A,(HL)
044B        57            LD D,A
```

What do we find at 0050 to 005F? ASCII control codes! That result is stored in the D register (044B) before the termination sequence.

Address	Contents	TRS-80 Action	ASCII Description	Keyboard Entry
0050	0D	Carriage Return	Carriage Return	ENTER
0051	0D	Carriage Return	Carriage Return	SHIFT ENTER
0052	1F	Clear Screen	Unit Separator	CLEAR
0053	1F	Clear Screen	Unit Separator	SHIFT CLEAR
0054	01	Break	Start of Heading	BREAK
0055	01	Break	Start of Heading	SHIFT BREAK
0056	5B	Up Arrow	Left Bracket	Up Arrow
0057	1B	Edit Escape	Escape	SHIFT Up Arrow
0058	0A	Line Feed	Line Feed	Down Arrow
0059	1A	*See text	Substitute	SHIFT Down Arrow
005A	08	Backspace	Backspace	Left Arrow
005B	18	Backspace Line	Cancel	SHIFT Left Arrow
005C	09	Horizontal Tab	Horizontal Tab	Right Arrow
005D	19	32-Character Mode	End of Medium	SHIFT Right Arrow
005E	20	Space	Space	Space
005F	20	Space	Space	SHIFT Space

Alright, we have upper case ASCII and TRS-80 control functions. How about the rest? Back up now to the test for SHIFT, at 0416. If such a shift is present, the value in A (40 to 5F) is increased by 20 (60 to 7F). These are the ASCII codes for <u>lower</u> (@, a-z, left brace, separator, right brace, ~, delete). The resultant code, as usual, is saved in D.

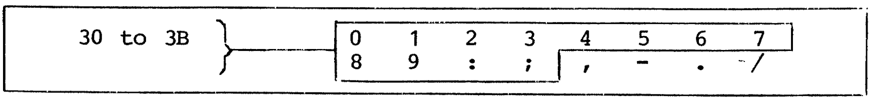
But what follows is curious:

```
041D        3A 40 38        LD A,(3840)
0420        E6 10           AND 10
0422        28 28           JR Z,0444C
```

The keyboard is tested again, this time at row 3840, data position 10 - the down arrow. If that key is not depressed, the program skitters right to the termination routine at 044C, with the lower case ASCII code ensconced in the D register.

Why the SHIFT/down arrow combination? If the down arrow is depressed, the value in D is retrieved and placed in the accumulator (60 to 7F), then reduced by 60, becoming ... aha! ... 00 to 1F. The program jumps to the end sequence, with the accumulator clutching one of the complete set of 32 ASCII control codes!

So where are we now? Upper and lower case, TRS-80 and ASCII control codes. We need numbers and figures, and so we shall have them. Recall the second diagram; at 042B, the command row was separated from the numbers, which were left as FO to FF. At 042D, 40 is added, resulting in possible values of 30 to 3F. A further separation is made via a comparison with 3C:



If the comparison is less than 3C, a carry is generated. The usual SHIFT test is made (at 0435), and if it fails, the value in A (30 to 3B) is maintained as the program moves into the end routine. These are the ASCII codes for numbers 0 to 9, colon and semicolon.

If the test value is 3C, 3D, 3E, or 3F, no carry would be generated at 042F, and these values are XORed with 10. This toggles the high nibble from 3 to 2, resulting in values from 2C to 2F (,-./). If a shift key was noted at 0437, the same toggle procedure is followed, changing values 30 to 3B into 20 to 2B (these would become space ! " # $ % & ' ( ) * =).

Here is how the code looks:

```
042D        C6 40          ADD A,40
042F        FE 3C          CP 3C
0431        38 02          JR C,0435
0433        EE 10          XOR 10
0435        CB 08          RRC B
0437        30 12          JR NC,044B
0439        EE 10          XOR 10
043B        18 0E          JR 044B
```

Thus, the coding is complete: control codes (00 to 1F), punctuation (20 to 2F), numbers and figures (30 to 3F), upper case (40 to 5F), and lower case (60 to 7F). Just as an aside, the terms lower and upper case are sometimes written small and large case; old-time printers would chuckle at that. The "case" referred to are printers cases, which, stacked one above the other, contained the capital and small letters. Thought you might like to know that.

Back to the routine, starting at the termination sequence (044C); the decoded character is saved in D, and that is the only information we need to preserve, since the bulk of the work is done.

```
044C        01 AC 0D       LD BC,0DAC
044F        CD 60 00       CALL 0600
0452        7A             LD A,D
```

A delay at 0600 is called, which was intended to wait through the normal bounce present in any type of mechanical contact -- but inappropriate for the easily dirtied switches on the TRS-80. This delay uses the accumulator, and when it is free, the value in D is restored to it. This value is compared to 01 (the BREAK code), and returns directly to the main routine (0455) with any code other than BREAK.

If BREAK is discovered, the program executes a call to 0028 (RST 28 -- more on that another time), finally returning to Level II.

The routine is quite efficient, and, in less than 64 keys, is capable of returning 128 values at a rate of better than 100 per second -- ten times the speed of the world's fastest typist!

Next time we will take a brief look at some of the improvements to this scan that can be patched in at address 4016, including a debounce which is better than Radio Shack's KBFIX (it keeps up with speed typists), and a very good autorepeat sequence.

ALPHA BYTE STORAGE

is wheelin' and dealin'!

Printers?

We stock 15 different kinds of dot matrix printers. We stock the 3 best letter quality printers — Dec, Qume and Diablo. Check our prices before you make your final decision!

Supplies?

We'll supply you with computer paper, print wheels, labels, forms, print thimbles, and much more!

Diskettes?

We're never undersold! Dig up the best price you can find on 5¼" Verbatims — then give us a call!

Software? Books? Miscellaneous?

We'd like the opportunity to chat with you about your current needs. And we'll help you find the best deal we can. Give us a call with your interests, or write for a free catalog!

Alpha
Byte
Storage

4636 Park Granada
Calabasas, CA 91302
1-213-883-8594

ENTRY
A Test Analysis and Grading Program

By Jason K. Potter

(Jason has played a major part in getting computers accepted in schools in the local mid-Michigan community. He quite frequently comes up with little gems and says: "Hey, see what one of my students did!". I mostly just sit back and burn with envy.

The following is a portion of a much larger data management system Jason has developed for maintaining and manipulating his own student records. The program stands quite well on its own. The part that really impresses me about the system is how subjective factors can be synthesized into the raw data. Too often we let the objective qualities of a computer rule the outcome of our projects, instead of simply using our machine as a tool.

This application may not be directly applicable to your needs, but Jason is using many techniques that could apply to your own data manipulation. cwb)

One of the most time consuming activities for the secondary teacher is the correction and scoring of various assignments, tests, and quizzes. Once the test has been corrected, the scores are often converted to percentage scores and a grade selected for converting scores to a letter grade. These grades or the percentage scores are then entered into the grade book in alphabetical order and later averaged with other grades to produce the final grade.

Although some educators and concerned persons have opposed the use of conventional grading systems for describing student progress, the practice continues for the following reasons: 1) Parents and students desire them, 2) they serve as a motivational factor, 3) and they provide a succinct method of communicating progress.

This article describes a program which I have used for the last three years, in a variety of forms. It has greatly reduced the time required to process and record the results of student assignments, tests and quizzes. I originally wrote the program as one module of a comprehensive record keeping system for the TI-59 programmable calculator. It has been rewritten for the TRS-80, both 16K cassette and for 32K with one disk drive. The program following this article is written specifically for a 32K one drive system; interested Level II users can write to myself or The Alternate Source for a modified listing. Persons interested in using GSF for the sort routine can also write to obtain the proper patches to the program listing.

Before discussing the program, it is necessary to describe some common methods of assigning grades to student work. One common method used with written or essay tests is to simply go by the overall impression and directly assign a letter grade to the test as a whole or to its parts. This method has the disadvantage that the grades given often are as much a result of the attitude and mental state of the teacher as a measurement of student performance. They also tend to be unreliable and difficult to defend to the questioning student or parent.

Probably the most commonly used method is to assign points for correct or partially correct answers, to total them and convert them to a percentage score. Many teachers then assign letter grades to the percentage scores according to a scale which has 90 to 103% equal to an "A", 80 to 89% equals a "B", and so on. This method has the advantage that the student knows the score needed to obtain a specific grade and may be motivated to greater effort by grades below what he considers adequate. The disadvantage of this method is that, while it seems to set an absolute standard, in reality the teacher usually manipulates the scale by trying to write tests so that student scores fall into an acceptable grade range. If performance on a test produces an unusually high number of "A" grades, the next test will tend to be a little harder.

A variation of the above method is to set an upper value which is equal to an "A" and a lower cutoff value which is equal to an "E" (or "F" depending on the school), and spread the grades out evenly between these values. This method has the advantage of allowing the teacher to compensate for the effects of an unexpectedly difficult test.

Another commonly used method of assigning grades is to assume that the statistical distribution of scores is approximately normal, and assign grades according to where the scores fall on the 'normal curve'. Two terms used in applying this method are 'mean' and 'standard deviation'. The mean of a set of scores is equal to the sum of the scores divided by their number. It gives the approximate center of distribution. (It is also known as the average.) The standard deviation of a set of numbers is a measurement of their variability or spread around the mean. It is numerically equal to the sum of the squared differences of the scores from the mean score. If a set of scores are normally distributed, about 70% of them will be within one standard deviation of the mean.

The 'curve' grades assigned more often in college courses are assigned according to a scale by which the average score receives a grade of "C" and the difference between letter grades is approximately equal to one standard deviation. Thus about 70% of the students will receive a grade of "B", "C", or "D". About 7.5% will receive a failing grade and 7.5% will receive an "A" grade. This method is not satisfactory in most cases, because most small classes are not normally distributed in ability or achievement, and the class may have a higher percentage of above average students or below average students. This would result in an injustice being done to one of the groups mentioned.

The above method may be made more fair by using some estimate of student ability, such as the grade point average, for the center of the scale and adjusting the spread around that central score to compensate for classes with more or less variability in ability. Information to do this may be obtained from student records or transcripts or may be estimated from past experience. A problem with using the 'curve' method of grading is that, in my experience, students tend to rely on other students' doing poorly enough to produce a low grading scale, rather than striving to meet a high standard. The result is a deterioration of performance over time.

It is possible to gain the advantages of both absolute scales and the 'curve' method by calculating the grades both ways and then averaging them. Thus, a student may have received a score of 65% on a very difficult test. According to the straight scale, this would be equal to a "D" grade. Because of the difficulty of the test, however, his was the highest score. According to most curve scales, he would receive a grade of "C+" or "B-". Thus he would get credit for having the best scsore in the class, but would also be penalized somewhat for getting such a low score on the test. The effect of using this method is to still require nearly a 90% score for an "A", but to spread the scale downward somewhat for especially difficult tests. It gives students a high standard to shoot for, but compensates for student ability and difficulty of tests.

It goes almost without saying that all of the above methods, especially the latter, require much computational effort and time. The microcomputer is ideally suited to solving this problem and making the latter method of grading practical.

The program "ENTRY" allows the following:

1. Names and scores may be entered in any order.
2. Descriptive statistics, such as the mean, standard deviation, highest score, lowest score, a reliability coefficient for the test, and a measurement of how far off a particular score may be from the "true" value (obtained from many tests over the same material), are calculated and printed out.
3. Choice of any of the above mentioned methods of grading.
4. Calculation of letter grades for all student scores according to the scale chosen.
5. Prints a list of names, scores, grades, and the grade scale used on the screen or on paper.
6. Saves the file produced on disk for later additions or for use with the program "RECORDS", which maintains student cumulative averages and keeps an updated average of each student's progress. This program is available through The Alternate Source.

In order to allow you to visualize the use of this program, let's go through a hypothetical session with it. The program will first ask if this is a new file. If it is, respond with "Y". The program will then ask for a description of the assignment or test being graded. This may be up to one line long and may contain commas, etc. It will then ask for a field heading five characters long for the grade list. The date will then be asked for, and instructions will be given for the entry of test scores.

Names are entered as shortened name codes, consisting of the first four letters of the last name, and the initial of the first name. This is to save time in data entry and to provide a standard format for use with the student cumulative records program (available separately). The score or number correct is then displayed. The last entry may be deleted by typing the word "DELETE" in response to the next "NAME" prompt.

When the last score has been entered, the word "END" is entered and the program goes on to calculate descriptive statistics for the test, which can then be used to set up a grade scale. These statistics are the mean score, the standard deviation, the highest score, the lowest score, the reliability coefficient, and the standard error of measurement. The reliability coefficient is how consistent the score would be on retesting with the same test. A score near 1.0 indicates a very high reliability, a score near zero indicates a very poor test. A typical value for a teacher made test would be 0.6; for a standardized achievement test it would be around 0.9. The standard error of measurement is an estimate of the expected amount of variation in an individual score on retesting with the same test.

The program then asks a series of questions which lead to the selection of a grading scale. It is possible to select any of the scales discussed above.

Once the desired scale is selected and the necessary information supplied, a grade scale will be calculated and printed and the teacher is given the chance to try another scale. The variable Z' stands for the average letter grade on a scale from 0 to 4.5 if this grade scale is used. The variable SZ' stands for the standard deviation of the letter grades if this scale is used. A typical value is seven tenths (.7).

When a scale is finally selected, it will be applied to all of the student scores. The list will then be sorted alphabetically and either printed out to the screen or printer. It may also be saved to diskette.

When saving the file to disk, the program will ask for a file name. It is possible to overwrite a previous file by giving the name used previously, or to write a new file by using a new name. The file can be retrieved for addition or printing by answering "N" to the quesiton "NEW FILE?" at the beginning of the program, and then responding with the filename in question when asked.

This program, as written for DOS users, requires 32K of memory and will handle up to 100 test scores. It may be redimensioned for larger numbers by making changes in lines 1000, 1260, and 1270. The program is also adaptable for use with a Level II 16K computer, with files being stored on tape. This involves a certain degree of unreliability and increased waiting time, but works just as well if the list is printed out immediately.

For further information on testing, grading and statistical analysis of test scores, the reader is referred to the excellent book "Testing and Evaluation for the Sciences in the Secondary School" by William Hedges, Wadsworth Publishing Company, Belmont, CA. For further information on Level II listings or GSF patches, my address is Jason Potter, Haslett High School, Haslett, MI, 48840.

```
                    CHEMISTRY TEST IONIC BONDS

                          MARCH 25, 1980

NUMBER      NAME CODE        RAW SCORE     % SCORE       GRADE
-----------------------------------------------------------------
1           AMWAY               29            64           1.45
2           ASIMI               38            84           3.15
3           BOHRN               34            76           2.47
4           BORNM               38            84           3.15
5           BURGT               31            69           1.87
6           GAUSS               40            89           3.57
7           JONES               36            80           2.81
8           KEPLJ               39            87           3.40
9           MENDR               35            78           2.64
10          MILLE               39            87           3.40
11          MILLR               33            73           2.21
12          NEWTI               38            84           3.15
13          OLSDR               36            80           2.81
14          PILSA               43            96           4.17
15          POPPM               36            80           2.81
16          ROENL               38            84           3.15
17          SWIFJ               28            62           1.28
18          TOBRU               25            56           0.77
19          WILSR               31            69           1.87
20          WOOLD               39            87           3.40
-----------------------------------------------------------------

SUMMARY:

AVERAGE SCORE = 78.45                 ST. DEVIATION = 10.1743
HIGHEST SCORE = 96                    LOWEST SCORE = 56
KR-21 RELIABILITY = .65155            ST. ER. OF MEAS. = 6.005

-----------------------------------------------------------------

GRADE SCALE CALCULATION

%AV = 78.45    %SD = 10.1743    Z' = 2.6766    SZ' = .86589

               4.5      =       99.875
               4.0      =       94.000
               3.5      =       88.125
               3.0      =       82.250
               2.5      =       76.375
               2.0      =       70.500
               1.5      =       64.625
               1.0      =       58.750
               0.5      =       52.875
               0.0      =       47.000
-----------------------------------------------------------------
```

"ENTRY" Program Listing

```
10 CLEAR 2000
20 REM ******** STUDENT RECORDS PROGRAM *********
30 REM
40 REM TITLE BLOCK
50 REM DRAW RECTANGULAR BLOCK
60 CLS:PRINT@ 69, STRING$(54,191)
70 PRINT@ 901, STRING$(54,191)
80 FOR I=133 TO 837 STEP 64: PRINT@ I, STRING$(4,191);:NEXT I
90 FOR I=183 TO 887 STEP 64: PRINT@ I, STRING$(4,191);:NEXT I
100 REM
110 REM PRINT BOX CONTENTS
120 PRINT@ 207,"CLASS RECORDS MAINTENANCE PROGRAM";
130 PRINT@ 341, "DATA ENTRY MODULE (2.1)";
140 PRINT@ 470, "COPYRIGHT FEB.9,1980";
150 PRINT@ 600, "JASON POTTER";
160 PRINT@ 725, "HASLETT HIGH SCHOOL";
170 PRINT@ 852, "PRESS ANY KEY TO CONTINUE";
180 REM
190 LET A$=INKEY$:IF A$="" THEN 190
200 CLS
210 '**************** DATA ENTRY MODULE *******************
220 '
230 '------------------INITIALIZATION------------------
240 CLEAR 2000
250 DEFINT C,E-K,M,W
260 LET N=1:LET SX=0:LET X2=0:LET XM=0:LET XL=100
270 DIM SN$(100),S(100),P(100),W$(100),W(100) ,L(100)
280 DIM D$(100),D(100)
290 '----------------DIRECTORY----------------------
300 CLS
310 INPUT "IS THIS A NEW FILE";A$
320 IF LEFT$(A$,1)="N" THEN GOSUB 2480
330 LINE INPUT "NAME OF FIELD - ";TN$
340 LINE INPUT"FIVE LETTER ABBREVIATION FOR FIELD HEADING";T1$
350 IF LEN(T1$)<>5 THEN 340
360 LINE INPUT "DATE - ";D$
370 INPUT "HIGHEST POSSIBLE SCORE";M
380 '----------------INPUT TEST RESULTS-----------
390 CLS
400 PRINT"
   TO ENTER SCORES, FIRST TYPE IN THE FIRST FOUR LETTERS OF
   THE LAST NAME AND THE FIRST LETTER OF THE FIRST NAME.
   E.G., BILL WILSON = WILSB"
410 PRINT"
   TO DELETE THE LAST ENTRY, TYPE 'DELETE'."
420 PRINT"
   WHEN YOU ARE DONE ENTERING SCORES, TYPE 'END'."
430 INPUT SN$(N)
440 IF SN$(N)="DELETE" THEN 570
450 IF SN$(N)="END" THEN 740       ' CALCULATE STATISTICS
```

(continued...)

```
460 IF LEN(SN$(N))<>5 THEN 390
470 INPUT "SCORE";S(N)  : PRINT CHR$(29)
480 IF S(N)>M THEN  PRINT "BAD SCORE-TRY AGAIN"
490 IF S(N)<0 THEN PRINT
     "NEGATIVE SCORES NOT ALLOWED-TRY AGAIN"
500 IF S(N)>M OR S(N)<0 THEN 470
510 LET P(N)=INT(S(N)/M*100+.5)
520 GOSUB 620
530 PRINT SN$(N),S(N),P(N);"%"
540 LET N=N+1
550 PRINT:PRINT "(";N;")";
560 PRINT "NAME?";:GOTO 430
570 REM ******** DELETE LAST ENTRY **********
580 LET N=N-1
590 GOSUB 700
600 GOTO 550
610 '------------------DESCRIPTIVE STATISTICS---------
620 REM ACCUMULATE INTERMEDIATE STATISTICS
630 REM -----------MEAN AND SD
640  LET SX=SX + P(N)
650 LET X2= X2 + P(N)[2
660 REM------MAX,MIN,RANGE
670 IF P(N) >XM THEN LET XM=P(N)
680 IF P(N) <XL THEN LET XL=P(N)
690 RETURN
700 REM--------------DELETE LAST X
710 LET SX=SX-P(N)
720 LET X2=X2-P(N)[2
730 RETURN
740 '------------------STATISTICS CALCULATIONS--------
750 LET N=N-1
760 REM--------------CALCULATE AVERAGE
770 LET AV=SX/N
780 REM--------------CALCULATE STANDARD DEVIATION
790 LET SD=SQR((X2-(SX[2)/N)/(N-1))
800 REM--------------CALCULATE KR-21 RELIABILITY
810 RA=AV/100*M:RS=SD/100*M
820 R=(M-RA)*RA
830 R=(R/M)/RS[2
840 R=1-R
850 R=R*(M/(M-1))
860 IF R>1 THEN LET R=1
870 REM-------------PRINT STATISTICAL SUMMARY
880 CLS
890 PRINT TAB(20);"ANALYSIS OF SCORES":PRINT
900 PRINT "AVERAGE SCORE =";TAB(35);USING "##.#";AV;:PRINT"%"
910 PRINT "STANDARD DEVIATION =";TAB(35);
     USING "##.#";SD;:PRINT"%"
930 PRINT "LOWEST SCORE =";TAB(34);USING"###.#";XL;:PRINT"%"
940 PRINT "RELIABILITY COEFFICIENT =";TAB(36);USING"#.##";R
950 PRINT "STANDARD ERROR OF MEASUREMENT =";TAB(35);
     USING "##.#";SQR(1-R)*SD;:PRINT"%"
```

(continued...)

```
960 '------------------GRADE CALCULATION--------------
970 PRINT
980 INPUT"DO YOU WANT TO USE AN ABSOLUTE GRADE SCALE?(Y/N)";A$
990 IF A$="N" THEN 1180
1000 INPUT "STRAIGHT SCALE; E.G. 90-100%=A,ETC. (Y/N)?";A$
1010 IF A$="N" THEN 1080
1020 ZA=2:SZ=1:X1=75:S2=10
1030  GOSUB 1300
1040 FOR I=1 TO N
1050 GOSUB 1700
1060 NEXT I
1070 GOTO 2850
1080 REM--------------OTHER SCALE
1090 INPUT "LOWER CUTOFF SCORE FOR 0 GRADE";XE
1100 INPUT "SCORE WHICH EQUALS A GRADE OF 4.0";XA
1110 GOSUB 1740
1120 LET X1=AV:S2=SD
1130 GOSUB 1300
1140 FOR I=1 TO N
1150 GOSUB 1700
1160 NEXT I
1170 GOTO 2850
1180 REM---------------GRADE ON CURVE OR COMBINATION--------
1190 INPUT" 'CURVE' GRADE (Y/N) ";A$
1200 IF A$="N" THEN 1480
1210 REM---------GRADE ON  NORMAL CURVE OR ADJUSTED FOR ABILITY
1220 INPUT "AVERAGE GRADE=2.0; STANDARD DEVIATION OF GRADE= 1?
     (Y/N)";A$
1230 IF A$="N" THEN 1390
1240 LET ZA=2:SZ=1:X1=AV:S2=SD
1250 GOSUB 1300
1260 FOR I=1 TO N
1270 GOSUB 1700
1280 NEXT I
1290 GOTO 2850
1300 REM------CALCULATE AND PRINT GRADE SCALE-------
1310 CLS
1320 PRINT TAB(25) "GRADE SCALE"
1330 ?:?"%AV=";AV,:?"%SD=";SD,:?"Z'=";ZA,:?"SZ=";SZ,:PRINT
1340 FOR Z=4.5 TO 0 STEP -.5
1350 LET X=(Z-ZA)/SZ*S2+X1
1360 PRINT Z,X
1370 NEXT Z
1380 INPUT "TRY ANOTHER SCALE?(Y/N)";A$:IF A$="Y"
     THEN 960  ELSE RETURN
1390 REM----------CURVE ADJUSTED TO DESIRED MEAN AND S.D.------
1400 INPUT "DESIRED AVERAGE GRADE THIS TEST";ZA
1410 INPUT "DESIRED STANDARD DEVIATION OF GRADE";SZ
1420 LET X1=AV:S2=SD
1430 GOSUB 1300
```

(continued...)

```
1440 FOR I=1 TO N
1450 GOSUB 1700
1460 NEXT I
1470 GOTO 2850
1480 REM ***  GRADE SCALE BASED ON AVERAGE OF CURVE
     AND STRAIGHT SCALE ***
1490 INPUT "ESTIMATED CLASS GPA";ZA
1500 INPUT "ESTIMATED S.D. OF GPA";SZ
1510 LET X=20        'CALCULATE LOWER CUTOFF VALUE FOR SCALE
1520 LET Z=ZA +SZ*(X-AV)/SD
1530 LET Z=(Z+ (2+((X-75)/10)))/2
1540 IF Z>=0 THEN 1560
1550 LET X=X+1: GOTO 1520
1560 LET XE=X
1570 LET X=110          ' CALCULATE SCORE EQUAL TO 4.0 GRADE
1580 LET Z=ZA+SZ*(X-AV)/SD
1590 LET Z=(Z+(2+((X-75)/10)))/2
1600 IF Z<=4 THEN 1620
1610 LET X=X-1 :GOTO 1580
1620 LET XA=X
1630 GOSUB 1740
1640 LET X1=AV:S2=SD
1650 GOSUB 1300
1660 FOR I=1 TO N
1670 GOSUB 1700
1680 NEXT I
1690 GOTO 2850
1700 REM LETTER GRADE CALCULATION SUBROUTINE
1710 L(I)=ZA+SZ*(P(I)-X1)/S2:L(I)=INT(L(I)*100+.5)/100
1720 IF L(I)<=0 THEN LET L(I)=.01
1730 RETURN
1740 REM ----------CALCULATE AVERAGE AND S.D. OF LETTER GRADES
1750 ZA=4*(AV-XE)/(XA-XE)
1760 SZ=4*SD/(XA-XE)
1770  RETURN
1780 REM-------------PRINT SORTED LIST----------------
1790 INPUT "DO YOU WANT THE LIST PRINTED IN THE ORIGINAL
     ORDER(Y/N)";A$
1800 IF A$="Y" THEN 2050
1810 INPUT "PRINT ALL(A),RANGE(R), OR SAVE TEST FILE";A$
1820 IF A$= "S" THEN 2300
1830 IF A$ ="A" THEN 1860
1840 IF A$= "R" THEN 1950
1850 GOTO 1810
1860 REM-------------PRINT ALL----------------
1870 CLS
1880 PRINT "RESULTS OF ";TN$:PRINT
1890 PRINT "NUMBER","NAME","%","GRADE"
1900 FOR I=1 TO G
1910 LET F=W(I)
1920 PRINT I,SN$(F),P(F),L(F)
```

(continued...)

```
1930 NEXT I
1940 INPUT "TRY ANOTHER SCALE?(Y/N)";A$:IF A$="Y"
     THEN 960  ELSE 1780
1950 REM------------PRINT RANGE---------------
1960 INPUT "LOWER BOUND OF RANGE";A
1970 INPUT "UPPER BOUND OF RANGE";B
1980 CLS:PRINT "RESULTS OF ";TN$:PRINT
1990 PRINT "NUMBER","NAME","%","GRADE"
2000 FOR I=A TO B
2010 F=W(I)
2020 PRINT I,SN$(F),P(F),L(F)
2030 NEXT I
2040 GOTO 1780
2050 REM------------PRINT IN ORIGINAL ORDER-----------
2060 INPUT "PRINT ALL(A) OR RANGE(R) ";A$
2070 IF A$="R" THEN 2200
2080 IF A$="A" THEN 2100
2090 GOTO 2060
2100 REM-----------PRINT ALL------------
2110 INPUT "WANT SUMMARY PRINTED ON LINE PRINTER";A$
2120 IF LEFT$(A$,1)="Y" THEN GOSUB 3150
2130 CLS
2140 PRINT "NUMBER","NAME","%","GRADE"
2150 PRINT
2160 FOR I=1 TO N
2170 PRINT I,SN$(I),P(I),L(I)
2180 NEXT I
2190 GOTO 1780
2200 REM -----------PRINT RANGE-----------
2210 INPUT "LOWER BOUND OF RANGE";A
2220 INPUT "UPPER BOUND OF RANGE";B
2230 CLS
2240 PRINT "NUMBER","NAME","%","GRADE"
2250 PRINT
2260 FOR I=A TO B
2270 PRINT I,SN$(I),P(I),L(I)
2280 NEXT I
2290 GOTO 1780
2300 REM--------------RECORD DATA FILE---------------
2310 LINEINPUT"OUTPUT FILE NAME? ";FI$: OPEN"O",1,FI$
2320 PRINT#1,TN$:PRINT#1,T1$:PRINT TN$,T1$:PRINT#1,D$:PRINT D$
2330 PRINT#1,N
2340 D(1)=X2:D(2)=AV:D(3)=R:D(4)=M:D(5)=N
2350 D(6)=RA:D(7)=RS:D(8)=S2:D(9)=SD:D(10)=SX
2360 D(11)=ZA:D(12)=SZ:D(13)=XL:D(14)=XM
2370 FOR I=1 TO 14:PRINT#1,D(I):NEXT
2380 REM----RECORD NAME LIST-----
2390 FOR I=1 TO N:PRINT#1,SN$(W(I)):NEXT
2400 REM -----RECORD RAW SCORE ARRAY-----
2410 FOR I=1 TO N:PRINT#1,S(W(I)):NEXT
2420 REM -----RECORD % SCORE ARRAY------
```

(continued...)

```
2430 FOR I=1 TO N:PRINT#1,P(W(I)):NEXT
2440 REM----RECORD LETTER GRADE ARRAY----
2450 FOR I=1 TO N:PRINT#1,L(W(I)):NEXT
2460 REM END OF FILE
2470 CLOSE: END
2480 REM-----RETRIEVE TEST DATA--------
2490 LINEINPUT"INPUT FILE NAME? ";FI$: OPEN "I",1,FI$
2500 INPUT "WHEN READY PUSH <ENTER> KEY";A$
2510 LINE INPUT#1,TN$:LINE INPUT#1,T1$:PRINT TN$,T1$:
     LINE INPUT#1,D$:PRINT D$
2520 INPUT "IS THIS THE CORRECT FILE";A$
2530 IF LEFT$(A$,1)="Y" THEN 2570
2540 PRINT"LOAD PROPER DISKETTE AND TRY AGAIN...";: GOTO2490
2550 GOTO 2510
2560  '
2570 INPUT#1,N
2580 FOR I=1 TO 14:INPUT#1,D(I):NEXT
2590 X2=D(1):AV=D(2):R=D(3):M=D(4):N=D(5)
2600 RA=D(6):RS=D(7):S2=D(8):SD=D(9):SX=D(10)
2610 ZA=D(11):SZ=D(12):XL=D(13):XM=D(14)
2620 REM------RETRIEVE NAME LIST------
2630 FOR I=1 TO N:INPUT#1,SN$(I):NEXT
2640 REM-----RETRIEVE RAW SCORES------
2650 FOR I=1 TO N:INPUT#1,S(I):NEXT
2660 REM-----RETRIEVE % SCORES--------
2670 FOR I=1 TO N:INPUT#1,P(I):NEXT
2680 REM-----RETREIVE LETTER GRADE ARRAY-----
2690 FOR I=1 TO N:INPUT#1,L(I):NEXT
2700 REM END OF FILE
2710 PRINT "ALL DATA RECOVERED": CLOSE
2720 REM---ALL DATA RECOVERED----
2730 REM---------PRINT RECOVERED DATA-------
2740 PRINT "NAME","RAW SC.","% SCORE","GRADE"
2750 FOR I=1 TO N
2760 PRINT SN$(I),S(I),P(I),L(I)
2770 NEXT
2780 INPUT "DO YOU WANT TO ADD MORE SCORES?(Y/N)";A$
2790 IF A$="Y" THEN LET N=N+1: GOTO 400
2800 INPUT "DO YOU WANT A LIST OF DESCRIPTIVE STATISTICS?";A$
2810 IF A$="Y" THEN 760
2820 INPUT "ASSIGN A DIFFERENT GRADE SCALE?(Y/N)";A$
2830 IF A$="Y" THEN 960
2840 RETURN
2850 REM----------------SORT BY NAME--------------------
2860 LET G=N
2870 FOR I=1 TO G
2880 W(I)=I:W$(I)=SN$(I)
2890 NEXT I
2900 GOSUB 2960
2910 REM--------PRINT OR RECORD ON TAPE---------
2920 INPUT "DO YOU WANT A PRINTOUT OF THE RESULTS?(Y/N)";A$
2930 IF A$="N" THEN 2300
2940 GOTO 1780
```

=======> Listing continued, pg. 65

Bit Kickin' with Jesse Bob

Nestled in the fertile plains of North Texas near the town of Carrollton, the Circle J Software Ranch occupies 180,000 microacres of prime bit grazing land. Owned and operated by the legendary Jesse Bob Overholt, the famed Circle J herd of over 90,000 bits produces fine software for TRS-80's all over the world.

In this issue we had hoped to print a picture of Jesse Bob in response to many requests. Unfortunately, he was on location for the shooting of 'Urban Bitboy' and we were unable to get a picture. It will be included in the next issue.

Here is the latest batch of questions received along with Jesse Bob's responses. Send your questions to Jesse Bob in care of The Alternate Source, 1806 Ada Street, Lansing, Michigan 48910.

---------- (J) ----------

Dear Jesse Bob,

The information that you gave 'Tense in Toledo' last issue for determining whether a disk was write-protected or not was great, and it really helped me a lot. I still have another problem, though, whenever I accidentally try to access a disk drive which doesn't have a disk in it. Usually, the drive motor turns off and I wind up in a 'Silent Death' situation. The only solution is to reboot. Do you have any tricks that can help me?

Also, can you tell me who shot J. R.?

Waiting in Wyoming

Dear Waiting,

The old 'missing diskette' problem can be a real killer alright. Here is a subroutine that Billy Fred, one of our top wranglers, wrote to determine the status of a disk drive before trying to access it. Just copy this routine into your program, set 'DN' to the number of the drive to be tested (0-3), and call the subroutine (via GOSUB) BEFORE trying to access the drive.

BILLY FRED'S SUBROUTINE

```
50000 'SUBROUTINE TO REPORT THE STATE OF THE DISK DRIVE
       WHOSE DRIVE NUMBER IS SPECIFIED IN VARIABLE 'DN'.
       IT RETURNS THE STATUS IN VARIABLE 'DS' WITH THE
```

```
        FOLLOWING VALUES:
50010 '     0 = THE DRIVE IS READY AND CONTAINS A DISK,
                WHICH MAY BE WRITTEN ON.
            1 = THE DRIVE IS READY AND CONTAINS A DISK,
                WHICH IS WRITE-PROTECTED.
50020 '     2 = THE DRIVE DOES NOT CONTAIN A DISKETTE.
            3 = THE DRIVE EITHER DOES NOT EXIST, OR IT
                CONTAINS A DISK AND THE DOOR IS OPEN.
50030 IF DN<0 OR DN>3 THEN DS=3
:                          RETURN
50040 SV% = 1
:     IF DN>0 THEN FOR X%=1 TO DN
:                       SV%=SV%+SV%
:                    NEXT X%
50050 POKE 14316,208
:     FOR X%=1 TO 100
:         POKE 14305,SV%
:     NEXT X%
:     PV%=PEEK(14316) AND 2
50060 FOR X%=1 TO 100
:         POKE 14305,SV%
:         CV%=PEEK(14316)
:         IF (CV% AND 2) <> PV% THEN 50080
50070 NEXT X%
:     DS=2
:     IF CV% AND 2 THEN 50090
      ELSE DS=3
:          GOTO 50090
50080 DS=0
:     IF CV% AND 64 THEN DS=1
50090  RETURN
```

Lines 50030 - 50040 validate the drive number and construct the select value (SV%) for that drive. Line 50050 resets the disk controller, starts the motor, and waits for the drive to come up to speed. It also saves the value of the index pulse bit in the disk controller status register (PV%). In line 50060 the drive is constantly reselected to keep the motor going. Each pass through the loop checks the status register for a change in the index pulse bit. If no change is detected, the loop runs out and falls into 50070. Here DS is set to 2 (no disk in drive). If the index pulse bit of the status register was constantly on we may deduce that there was no diskette blocking the sensor, so the drive is empty. Otherwise all we know is that the disk is either absent or contains a disk with the door open.

If the index pulse bit changes (meaning that a disk is in the drive and it is turning) we go to line 50080 where DS is updated to a value of 1 if this condition exists.

Unfortunately, I have no idea who shot J. R. Ewing.

Southfork, the Ewing ranch, is on up the road a good bit, on the other side of the Oleo Ranch (one of your cheaper spreads). My own personal theory is that J. R. is so mean that he shot himself!!

Jesse Bob

---------- (J) ----------

Dear Jesse Bob,

You are my last resort! For several months now my computer has been acting 'flakey'. It reboots from time to time and occasionally the memory forgets. I have gone through four sets of RAM chips trying to get it straightened out, but nothing seems to help. Radio Shack has checked both the keyboard and the Expansion Interface thoroughly and says they are fine.

The problem seems to be more serious on hot days when I run the air conditioner, but filtering the power line doesn't seem to help. Any suggestions?

Anxious in Angola

Dear Anxious,

Your problem is more common than you might think. In recent research here at the ranch, we found several of our Expansion Interfaces had improperly adjusted five-volt regulators. One system has less than four and a half volts on the five volt supply. This causes many of the circuits in the Interface, including the Disk Controller, to operate near to or outside of the tolerances they were designed for. Sudden memory drops in the line voltage (such as that caused by an air conditioner turning on) can cause chips to behave erratically.

The solution is simply to adjust the five volt power supply so that exactly five volts is present on the Expansion Interface bus. To do this you need a calibrated voltmeter, preferably either digital or high-impedance type. This type of equipment is usually owned only by hardware hackers (the guys with solder splashes on their shoes who draw schematic diagrams on napkins). If you are not able to get one, the best thing to do is have your local Radio Shack Computer Center check the voltage and adjust it for you.

Jesse Bob

---------- (J) ----------

Dear Readers,

In Issue #4 a Poke was given to 'Puzzled in Los Angeles' to solve the problem of printed output 'walking' down the page. Unfortunately, the value given in the POKE was wrong and would result in printout moving UP the page instead of down, which is hardly a solution. The correct statement should read "POKE 16424,66" instead of "POKE 16424,64". My deepest apologies for this error.

As you may know the weather in Texas has been unusuallly hot this Summer. Heavy use of air conditioners has resulted in frequent power fluctuations. The result has been that many of our "1" bits have been transformed into "0" bits, which are worthless. Those of you who live East of the Mississippi can help us out. Whenever you find yourself facing Southwest, how about flapping your Level II manual gently up and down. The resulting breeze may help cool us off a bit.

Until next time keep those cards and letters coming and remember -- Murphy's Law is only a theory. It has never been proven korrect!

Jesse Bob

THE INNOVATIVE PENGUIN

Presents

FRED

(the friendly robot educator)

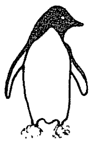
Appearing in

SKETCH-A-SOUND

A game that allows you to hear your drawings. What does the 1812 Overture sound like?

MUL-TI-SOUND

Teachers, Parents, Help at last! A painless multiplication drill!



Watch for Fred and the Penguin as they continue to bring you programs that make dreary memorization into fun. Sound makes the difference!

Check with your local computer store or send $14.95 for each 16K level 2 program to:

THE INNOVATIVE PENGUIN

2320 Hampton Dr., Harvey, La. 70058

Dealer and educator inquiries invited

META-TREK

This is NOT your average star-trek game! Meta-Trek requires 32K and one disk drive, and features several never-before-incorporated missions! You'll be dodging black holes and meteor storms, contemplating time warps and dealing with sensor-zapped quadrants! You'll have the help of your computer (which resembles a TRS-80 remarkably) and a detailed report on your standings, but this one still isn't easy! There is a FREEZE command though, to allow you to save your progress to disk. An excellent plot, with excellent graphics to match! Fast, too, with embedded Z-80 routines This one's **$19.95** and comes on diskette with an orientation manual well worth it! We've already enjoyed hours and hours, and have many more to go!

Order from The Alternate Source, 1806 Ada Street, Lansing, MI 48910 or by phoning (517) 485-0344 or 487-3358.

OUT OF SORTS??

By Thomas Frederick
ABS Suppliers

As many of you know, the Level II Interpreter has a peculiar way of handling strings. Of particular interest is what the 'garbage collection' routine does to affect sorting string array data. This is, according to Radio Shack's Microcomputing News (March/April 1980) "...is a computer controlled operation which allows the TRS-80 to recover previously used string space in memory. If you are sorting a large number of strings, 'garbage collection' can consume a significant portion of your (string) sorting time."

The Memory Management of Basic was covered in depth by T.R. Dettman in the 80-U.S. Journal (p. 28 Nov/Dec 1979) so I will avoid being redundant. Some indirect schemes have evolved to 'overcome' this obstacle to sorting strings. One of them uses the VARPTR statement to overcome the sudden death of your computer when sorting strings. Put simply, VARPTR is used to find the pointers of the strings and sort them -- rather than the strings themselves, to avoid the loss of your computer when the memory management routine occurs.

I have been using a slightly different method for some time now which I consider to be more elegant, easier to understand by all new folks, and faster than the VARPTR scheme -- which also has no loss in computer sort time.

To get a better handle on the problem, enter the following program EXACTLY:

```
10 CLEAR50: REM1 DIM Q$(2000), W$(2000)
20 FORJ=0TO9: INPUT A$(J): NEXT J
30 FORJ=0TO9: PRINT "A$(" J ")=" A$(J),: NEXT J
40 LO = PEEK(16598): HI = PEEK(16599)
50 PRINT: FOR J = 0 TO 9
60 Z$ = A$(J)
70 REM2 POKE 16598,LO: POKE 16599,HI
80 B$(J) = Z$
90 PRINT "B$(" J ")=" B$(J)
100 NEXT J
110 GOTO 50
```

When you first RUN this, enter two characters in response to the INPUT statement when it is encountered in line 20. You'll find that using an AM radio next to the keyboard will be invaluable in 'listening' to the CPU.

You'll also notice that the printed output on the video is a little 'jerky' while it's running. Now EDIT line number 10 and delete REM1. Running the program again produces a different result -- with the computer going into its sudden death or 'garbage collection' routine. By deleting the REM1 and including the DIM statement, we get our FIRST CLUE about what's going on.

If you edit line 10 again, so that you CLEAR 150 (instead of just 50), and run the program again, you'll find that the output is somewhat different (as dramatically evidenced by your AM radio. If you aren't using one, better get one quick! Otherwise you'll be missing the boat (actually it's the garbage truck, not the boat.)).

Change line 10 to read CLEAR 50 again. Also edit the size of the dimensions of the strings to 200 (instead of 2000) and run the program again. You'll notice a difference in program execution when you run it with a SMALLER string array size. CLUE NUMBER TWO just slipped under your nose!

It turns out that the size of the CLEARed space and the size of the string arrays are directly (or proportionately) related to the amount of time the computer will waste in its garbage collection routine. When Z$ is referenced in the program as an assignment, the free string space pointer (16598/16599) is updated to the next chunk of free memory. This occurs until it runs out of space set aside or CLEARed for strings. When this happens, what I believe occurs is, Basic goes to where the string arrays are stored and steps through them to determine where the free string space defined by the CLEAR statement starts. This is the space immediately below the STRINGS that are part of the STRING ARRAYS themselves. This is where the time is consumed -- by the Level II interpreter 'looking' at all the string pointers to determine where the start of free memory is.

Now when you're using a sorting procedure, say a Bubble Sort, you're testing or comparing a value in a list to the next value in the list (value + 1) to find out if value + 1 is less than value. If it isn't, then you shift your attention to the list and compare value + 1 to value + 2 and so forth. However, if value + 1 is lower (less than) value, you want to swap the contents of value with value + 1.

To achieve this swap, we need a temporary storage area for holding one of the values (just like Z$ in the accompanying program example) while we do the switch. This could take the following format: Z$ = value value = value + 1 value + 1 = Z$. Everytime we use Z$, depending on the length of the string temporarily stored (0 to 255 characters) the free string space pointer is decremented to point to the next free chunk of memory. In a sorting procedure, Z$ is referenced many times by assigning it different values -- with an accompanying decrementing of the free string space pointer. Larger strings in Z$ eat up CLEARed memory faster -- run the program inputting two characters, then run it again inputting four characters. Compare the effect. The AM radio tells all.

It is this reason the sorting algorithm using VARPTR is faster. It sorts the pointers to the strings instead of the strings themselves. This results in no free string space being eaten up, but is difficult to easily understand.

Get the values in line #10 back to their original ones (CLEAR 50, DIM (2000)) and RUN the program again. When you're satisfied that things are in a state of 'disfunction' in your computer, press the break key. You may have to hold it down; BREAK is NOT acknowledged while this 'garbage collection' routine is occuring.

Although the accompanying program does not contain an algorithm, it is intended to demonstrate what happens during a sorting procedure -- that being the rapid consumption of free string space by repeated reference (by assignment) to Z$. It is only intended to simulate the effect of sudden death brought on by a string sort.

Now we'll move into high gear. Take line number 70 and delete REM2. Run the program again, still entering two characters in response to the INPUT statement in line 20. You'll notice a DISTINCT IMPROVEMENT in program execution. What we have done here is 'reset' the free string space pointer after using Z$, and eliminated the need for garbage collection. The use of this method is not critical in its placement either, but a specific procedure must be followed to be successful.

This method for overcoming the string space compression routine in Basic is FASTER than using VARPTR (Poke occurs before VARPTR in the Level II Reserved Word Table), it also uses LESS MEMORY than the currently used VARPTR scheme, and finally, it's MUCH SIMPLER TO UNDERSTAND:

1. Insert a program line immediately before the for-next loop containing the sorting algorithm to save the contents of the pointers; similar to the following code: LO = PEEK(16598): HI = PEEK(16599).

2. Somewhere within the body of the for-next loop, insert the following code to reset the pointers: POKE 16598,LO: POKE 16599,HI.

PENRAM - A SCROLLING RAM EDITOR

Public Domain Software by Roxton Baker
56 South Rd. Ellington, Ct. 06029
(203) 875-2483

INTRODUCTION

I recently needed a program that would allow me to quickly examine large portions of RAM and easily change what I found there, working either in ASCII or hex. This ability is required when editing data or text (as opposed to machine code) in memory, because with these one has no idea at what address a particular byte may be. In my case I wished to scan and modify whole disk tracks which had been read in to RAM. Tracks never read the same way twice so the locations in memory of ID packs, sector boundaries, etc. are unpredictable.

The available monitors for the TRS-80 are, however, written with the intention of editing hex machine language programs, in which byte locations are well defined. These monitors require you to specify in advance the addresses to edit, and generally allow editing only in hex, and scanning only in one direction.

This is too clumsy when working with data. Two-directional scanning would improve matters, but one still must read off addresses of interest and re-enter them under the edit command. I greatly prefer the Electric Pencil's editing action. Not only does it provide true scrolling in both directions, but its blinking cursor can be positioned with the arrow keys and anything typed at the cursor location replaces what's there. But the Pencil is not used for editing RAM. It works on its own text files.

PENRAM, presented here, is a machine-language utility that provides Electric Pencil-like editing directly in RAM. Two-way scrolling can be done at low or high speed and the movable edit cursor allows direct input of hex bytes or ASCII characters (the mode can be toggled at a keystroke).

A continuous display is maintained of the edit cursor's "address" as well as it's displacement from a reference address that the user may fix. Users of the Electric Pencil should understand that PENRAM does not allow true insertion or deletion of bytes of code; it merely lets you write over what's already there. A compromise has been made between editing power on one hand and length (already 1100 bytes) and ease-of-use on the other.

PENRAM can be used as a stand-alone utility, called from Basic via USR or SYSTEM, or patched onto your favorite monitor as a replacement for its editing function. Instructions for appending it to RSM-2 and TBUG are given later. Furthermore, PENRAM is in the public domain. You may incorporate it into any program you write, commercial or otherwise, without royalty arrangements. Author credit is the only request.

CREATING THE PROGRAM

The source code for PENRAM is given in Fig. 1. By omitting most of the comments (and using the tab key to space right) the program will fit in a 16K machine using the Radio Shack tape Editor Assembler, or Microsoft's new EDTASM-PLUS. Or it may be entered into any of the disk Assemblers. To avoid the typing, contact The Alternate Source. They have kindly agreed to make the uncommented source code available on tape, at cost.

PENRAM may be assembled at any location by changing the STARTP address near the beginning. Allow a total of 1100 bytes for the code. PENRAM also uses a little of the stack space of the program that calls it; this is normally of no concern.

Once assembled PENRAM may be used by itself or it may be called by another program. The START address is also the entry point. When exited PENRAM executes a RETurn instruction so it should be called in such a manner that this return is meaningful. This will always be the case if PENRAM is called as a subroutine, or if it is entered from DOS. If PENRAM is called via the SYSTEM or USR commands in Basic, it may be preferable to jump on return to addresses 06CCH or 0A9AH, respectively. To achieve this, change the instruction at RETLBL near the beginning of the source code from:

```
from   RET Z
to     JP Z,nn
```

where nn might be 06CCH for example. This jump takes three bytes as opposed to the one byte RET, so omit the two NOP's immediately afterwards. The code will still be 1100 bytes long. You may of course add more code at this point to clear the screen before returning, etc. You should assume PENRAM modifies all registers except IY.

USING PENRAM

On entry, or whenever you press shift/LEFT-ARROW, PENRAM will request an address at which to start the display. Enter a four-digit hex address (or just press <ENTER> to quit). A hex display of the 256 bytes of memory beginning at that address will appear on the screen. You may change this to an ASCII display by pressing CLEAR. Another CLEAR takes you back to hex. The blinking edit cursor will be in the upper left corner. You move it using the four arrow keys -- any scrolling required is automatic. At any time you may enter data at the current cursor position. In the hex mode two hex characters (0-F) are required; in the ASCII mode any printable character may be entered (except arrows). The characters entered will replace the current byte at that location. You may BREAK after entering only one hex character in which case the original byte is restored. All data changes are seen instantly on the screen. Most keys repeat so that you may easily fill memory with a value, and high-speed scrolling is provided with the shift/UP-ARROW and shift/DOWN-ARROW keys.

On the right side you will see continuously displayed as:

> nn <

the actual address of the byte that the edit cursor is next to. This makes it easy to read off addresses of interest, without counting. Below that will be shown the value of the current reference address and (in decimal) the displacement of the edit cursor from it. The reference address is updated to the current edit cursor position whenever you press shift/BREAK. To understand the use of the reference address feature, imagine that you wish to move the cursor 287 (decimal) bytes beyond where it is. You would press shift/BREAK to set the reference address to the current position of the edit cursor, and then you would move the cursor downwards while watching the displacement value to see when you'd reached 287.

ATTACHING PENRAM TO A MONITOR

The addition of PENRAM will complement the editing features of a good monitor program such as RSM-2 from Small System Software. It will completely transform the editing of a simple monitor like T-BUG. Patching PENRAM to either of these is easily done. RSM-2 provides a "U" (user-definable) command that will access PENRAM. Under T-BUG the normal "M" editing command is replaced. In the following instructions all addresses and values are in hex.

The 48K version of RSM-2 is assumed. For 32K, subtract 4 from the first hex digit of each address or value marked "*". Thus *E7B3H for 48K becomes A7B3H for 32K. Similarly, *FFH becomes BFH. For 16K, subtract 8 instead of 4.

Assemble PENRAM at *E7B3H by setting STARTP in the source code to this value. Load the resulting object code into memory. Load and run RSM-2. Using the E-command of RSM-2, change the code at address:

```
*EEA6H   from   00             to   7F
*EEB1H   from   32  80  *FF    to   00  00  00
*FF80H   from   C9  00  00     to   C3  76  *E7
```

At this point the new monitor program PENRSM resides from *E7B3H to *FFFFH, with entry point *EE94H. You may write it out to tape with the P-command, or on a disk system you may go to DOS with G402D and use TAPEDISK or the DUMP command to put PENRSM/CMD on disk.

Adding PENRAM to T-BUG is also easy. A Level II non-disk machine is assumed. Assemble PENRAM at 4980H and load the object code into memory there. Load and run T-BUG. Use its M-command to carefully change the two bytes at address 440EH:

```
from   32 45
to     80 49
```

Use X to break from the editing mode. The new monitor program PENBUG now resides from 4380H to 4D09H, with entry point 43A0H. Immediately, before doing anything else, punch it out to tape with the P-command. Under PENBUG the M-command will access PENRAM; shift/LEFT-ARROW <ENTER> returns. It may also be necessary to press X on return to get the # prompt.

Attaching PENRAM to other monitors can be done as in the case of T-BUG. Locate the call or jump used when the edit command is invoked and replace it with a call to PENRAM.

HOW PENRAM WORKS

Those interested in modifying or extending PENRAM will need to know something of the program structure. Refer to the source code listing at the end of this article. PENRAM operates by keeping track of three important addresses, which will be referred to by the names of their storage locations (which are unimportant). This is not strictly correct, but it's easier to read:

1) HOMADD, the address that is displayed in the upper-left (home) corner.

2) EDCUR, the current location in video memory (3C00H-3FFFH) of the edit cursor.

3) BYTED, the actual address of the byte pointed to by the edit cursor.

The screen is initially filled with 256 bytes of memory starting at the hex address input by the user. This address is the first value of HOMADD. The edit cursor is positioned at the first byte displayed. From there it can be moved 15 spaces to the right and/or 15 lines down, without causing any scrolling. A count (stored in RGTCUR) is kept of how many spaces to the right the cursor is moved. A similar count (in DWNCUR) is kept of how many lines down it is moved. PENRAM begins each cycle of its operation at UPDATE by calculating, from RGTCUR and DWNCUR, the present address EDCUR of the edit cursor. It simultaneously calculates (from HOMADD, RGTCUR, and DWNCUR) the value of BYTED so that it knows to which byte in memory the edit cursor is pointing.

Once these values have been found, PENRAM displays the address stored at BYTED as the > nn < value mentioned earlier. It also displays the current reference address, called ATADD, and subtracts it from the BYTED address to find the present displacement. Available ROM routines are used to conver this displacement to decimal, and it too is displayed.

With this done, PENRAM goes into a keyboard scan loop at KBDSCN. It remains in this loop, blinking the edit cursor, until a key is pressed. The cursor blink rate is determined by BDELAY, and the graphics character used for the cursor is defined by CURCHR. Either of these may be changed before assembly.

When a key is pressed, PENRAM goes to KPRESS and takes action as follows:

1) If a shifted up- or down-arrow key, PENRAM immediately checks the value of DWNCUR and modifies it or scrolls the the screen, as appropriate. This is discussed in more detail below.

2) If the shift/LEFT-ARROW key PENRAM completely restarts by jumping back to its ENTRY point.

3) If the shift/BREAK key, PENRAM immediately sets ATADD equal to BYTED, thus updating the reference address, and returns to UPDATE so that the new value will be displayed.

4) If an unshifted arrow key, PENRAM enters a debounce delay loop and then processes the key by updating RGTCUR or DWNCUR and scrolling the screen if necessary. See below.

5) If the CLEAR key, PENRAM changes the type-of-display flag HATYPE from hex to ASCII or vice-versa, redraws the screen and restarts the cycle.

6) If a valid hex character and in the hex mode, PENRAM remembers the entry, delays briefly for debounce, and awaits the next hex character. When that is received PENRAM forms the new byte and writes it into BYTED. Then it steps the edit cursor right one space by jumping to the same place (DORT) that a right-arrow would have taken it.

7) If a valid ASCII alpha-numeric and in the ASCII mode, PENRAM writes it into BYTED and then steps the edit cursor right as is (6).

The delay values used for the debouncing are specified as KDELAY1 and KDELAY2. You may wish to increase them if you experience keybounce.

The cursor movement referred to above as resulting from the arrow keys is coded in routines DOUP, DODN, DOLF, and DORT. These can be generally described as follows. If the user is not attempting to move the cursor off-screen then PENRAM just translates the arrow keystrokes into appropriate changes in RGTCUR and DWNCUR, and returns to UPDATE to begin the next cycle.

When the cursor is moved beyond the edge of the screen, scrolling must take place. This is done by shifting the current contents of the screen up (or down) by one line, updating HOMADD, and writing one new line at the bottom (or top). RGTCUR and DWNCUR are also changed as required. PENRAM then returns to UPDATE to begin the next cycle.

The detailed comments in the source code may be referred to for further information.

PENRAM Listing

```
;*******************************************************
;
;               >>>>>  P E N R A M  <<<<<
;
;            A SCROLLING RAM EDITOR UTILITY
;
;             SOFTWARE IN THE PUBLIC DOMAIN
;
;*******************************************************
;
; BY ROXTON BAKER
; 56 SOUTH RD.  ELLINGTON, CT. 06029
; (203) 875-2483
;
; VERSION 2.0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - -
;
;COMMANDS:
;  'CLEAR' CHANGES TYPE OF DUMP (HEX/ASCII).
;  'BREAK' BREAKS FROM ERRONEOUS HEX ENTRY.
;  'SHIFT/BREAK' SETS REF. ADDRESS = CURRENT ADDRESS.
;  'SHIFT/UP- OR DOWN-ARROW SCROLLS RAPIDLY.
;  'SHIFT/LEFT-ARROW' ALLOWS NEW STARTING ADDRESS.
;      (REPLY  <ENTER>  TO LEAVE PENRAM)
;
;
;- - - - - - - - - - - - - - - - - - - - - - - - - - - -
; PENRAM CAN BE RELOCATED BY CHANGING THE VALUE OF
; 'STARTP' .    ALLOW 1100 (044CH) BYTES TOTAL.
;
STARTP  EQU     0E7B3H          ;FOR LINKING WITH RSM48.
;
;- - - - - - - - - - - - - - - - - - - - - - - - - - - -
;
; SYSTEM ROUTINES AND LOCATIONS ---
KI      EQU     002BH           ;INPUTS ASCII BYTE
CLS     EQU     01C9H           ;CLEARS SCREEN
CURPOS  EQU     4020H           ;POS'N OF STD. CURSOR
KIBUF   EQU     4036H           ;WORKSPACE FOR KI
LINEIN  EQU     05D9H           ;INPUTS WHOLE LINE
BYTDIS  EQU     0033H           ;PRINTS ASCII BYTE
DELAY   EQU     0060H           ;14.6 USEC PER COUNT BC
NTF2    EQU     0A9DH           ;DENOTES INTEGER VALUE
NUMSTR  EQU     0FBDH           ;CONVERTS NUM. TO STRING
;
; CONSTANTS ---
KDLAY1  EQU     2500H           ;MAIN INTER-KEY DELAY
KDLAY2  EQU     1200H           ;AUX DELAY REGULAR CHAR.
BDELAY  EQU     0A0H            ;SPEED OF CURSOR BLINK
BLANK   EQU     80H             ;BLANK GRAPHICS BLOCK
CURCHR  EQU     8AH             ;DEFINES CURSOR CHARACTER
```

(continued...)

```
00530 ; KEYBOARD INPUT VALUES ---
00540 CLEAR    EQU      1FH
00550  BREAK    EQU      4
00560 SHFBRK   EQU      5
00570 SHFTUP   EQU      9
00580 SHFTDN   EQU      17
00590 SHFLFT   EQU      21H
00600 UPAROW   EQU      5BH
00610 DNAROW   EQU      0AH
00620 LFAROW   EQU      08
00630 RTAROW   EQU      09
00640 ;
00650 ;KEYBOARD SCAN ADDRESSES ---
00660 KEYS1    EQU      3801H               ;@-G KEYS
00670 CNTL     EQU      3840H               ;DETECT CONTROL KEYS
00680 SHCNTL   EQU      38C0H               ;SAME BUT SHIFTED
00690 SHIFT    EQU      3880H               ;0 HERE MEANS NO SHIFT
00700 ;
00710 ; STORAGE LOCATIONS ---
00720 STRPLC   EQU      STARTP+2            ;STORES STRING TO PRINT
00730 STORAG   EQU      STARTP+66
00740 ADDR     EQU      STORAG              ;ADDRESS TO PRINT NEXT
00750 HOMADD   EQU      STORAG+2            ;ADDRESS AT TOP LEFT
00760 EDCUR    EQU      STORAG+4            ;LOC'N OF EDIT CURSOR
00770 BYTED    EQU      STORAG+6            ;BYTE AT EDIT CURSOR
00780 RGTCUR   EQU      STORAG+8            ;# TIMES MOVED RIGHT
00790 DWNCUR   EQU      STORAG+9            ;# TIMES MOVED DOWN
00800 HATYPE   EQU      STORAG+10           ;TYPE OF DUMP FLAG
00810 KEY      EQU      STORAG+11           ;KEY INPUT STORAGE
00820 KEYH     EQU      STORAG+12           ;USED IN HEX ENTRY
00830 LSTARW   EQU      STORAG+13           ;DIRECTION OF LAST SCROLL
00840 ATADD    EQU      STORAG+14           ;HOLDS REFERENCE ADDRESS
00850 ;
00860  ; VIDEO LOCATIONS ---
00870 LINE01   EQU      3C00H
00880 LINE02   EQU      3C40H
00890 LINE05   EQU      3D00H
00900 LINE16   EQU      3FC0H
00910 VIDFST   EQU      3C00H
00920 VIDLST   EQU      3FFFH
00930 ;
00940 ;
00950 ;- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
00960 ;START OF MAIN PENRAM ROUTINE
00970 ;
00980          ORG      STARTP
00990          JR       ENTRY
01000 ;
01010          DEFS     80                  ;WORK AND STORAGE SPACE.
01020 ;
01030 ENTRY    DI
```

<u>(continued...)</u>

```
              CALL     CLS
              LD       HL,3E38H          ;PROMPT FOR ADDRESS.
              LD       (CURPOS),HL
              LD       HL,INMSG
              CALL     OUTSTR            ;DISPLAY IT.
      ;
      ;       TAKE IN 4-DIGIT HEX ADDRESS.  BREAK FROM PENRAM
      ;       IF ONLY <ENTER> WAS PRESSED.
      ENTRY2  LD       HL,3E78H          ;CLEAR ANY PREV. ADDRESS
              LD       (CURPOS),HL       ;  AND TAKE IN NEW ONE.
              LD       A,1EH             ;ERASE TO END OF LINE.
              CALL     BYTDIS
              LD       B,4               ;ALLOW ONLY 4 DIGITS
              LD       HL,STRPLC
              CALL     LINEIN
              LD       A,B               ;B HAS # CHAR. TYPED.
              AND      A
      ;
      RETLBL  RET      Z                 ;LEAVE PENRAM IF NO CHAR.
              NOP                        ;  ENTERED.  CHANGE THIS
              NOP                        ;  TO A JUMP IF DESIRED.
      ;
              CP       4
              JR       NZ,ENTRY2         ;DO AGAIN IF < 4 CHAR.
      ;
      ;       IS INPUT VALID HEX?  IF NOT, BACK TO PROMPT.
              LD       HL,STRPLC+3
              CALL     FRMBYT            ;MAKES ASCII CHAR AT HL
              JR       C,ENTRY2          ;  INTO VALID HEX IN A.
              LD       (HOMADD),A        ;  RETURNS CARRY SET IF
              CALL     FRMBYT            ;  NOT VALID HEX.
              JR       C,ENTRY2
              LD       (HOMADD+1),A
      ;
      ;       STARTING ADDRESS IS AT HOMADD.  NOW INITIALIZE
      ;       SCROLLING DUMP ROUTINE AND BEGIN.
              XOR      A
              LD       (HATYPE),A        ;ASSUME HEX DUMP.
      ;
              LD       (DWNCUR),A        ;EDIT CURSOR IS AT
              LD       (RGTCUR),A        ;  TOP LEFT
      ;
              LD       HL,(HOMADD)       ;THIS IS FIRST REFER-
              LD       (ATADD),HL        ;  ENCE ADDRESS.
      ;
              LD       HL,3C78H          ;SET UP CURSOR AND
              LD       (CURPOS),HL       ;  PRINT LOGO AND
              LD       HL,PENMSG         ;  OTHER MARKS.
              CALL     OUTSTR
              LD       A,'>'
              LD       (3CF8H),A
```

(continued...)

```
               LD       A,'<'
               LD       (3CFFH),A
               LD       A,'='
               LD       (3D78H),A
               LD       A,'H'
               LD       (3D7EH),A
      ;
      REINIT   LD       HL,(HOMADD)       ;START WITH REQUESTED
               LD       (ADDR),HL         ;  ADDRESS.
               LD       B,16              ;FOR 16 LINES.
               LD       HL,VIDFST-1       ;INITIALIZE CURSOR.
               LD       (CURPOS),HL
      ;
      INITLP   EXX                        ;REPAINT SCREEN BY PRINT-
               LD       HL,(CURPOS)       ;  ING 16 LINES.
               INC      HL
               LD       (CURPOS),HL
               LD       HL,(ADDR)
               CALL     MAKSTR            ;CREATES DUMP STRING FROM
               LD       HL,STRPLC         ;  ADDR AND THE 16 BYTES
               CALL     OUTSTR            ;  THERE.  PRINT STRING.
               LD       BC,16
               CALL     AADDR             ;POINT TO NEXT ADDRESS.
               EXX
               DJNZ     INITLP
      ;
               LD       A,1               ;CURSOR MOVED DOWN LAST.
               LD       (LSTARW),A
      ;
      UPDATE   LD       HL,(EDCUR)        ;ERASE OLD EDIT CURSOR.
               LD       (HL),BLANK
      ;
      ; NOW CALCULATE NEW EDIT CURSOR POSITION (EDCUR) AND EDIT
      ; BYTE ADDRESS (BYTED) FROM NEW VALUES OF DWNCUR AND
      ; RGTCUR.
               LD       A,(DWNCUR)        ;INITIALIZE FOR BEING
               INC      A                 ;  COUNTED.
               LD       B,A
               LD       HL,VIDFST-61
               LD       DE,64
               EXX                        ;COUNT ADDRESSES IN ALT
               LD       HL,(HOMADD)       ;  REGISTERS.
               LD       BC,17
               XOR      A
               SBC      HL,BC
               LD       DE,16
               EXX
      ;
      LINEDN   ADD      HL,DE             ;FOR EVERY LINE DOWN, ADD
               EXX                        ;  64 TO EDCUR AND 16
```

(continued...)

```
02050          ADD     HL,DE          ;  TO BYTED.
02060          EXX
02070          DJNZ    LINEDN
02080 ;
02090          LD      A,(RGTCUR)
02100          INC     A
02110          LD      B,A
02120          LD      DE,3
02130 ;
02140 SPCRGT   ADD     HL,DE          ;FOR EVERY SPACE RIGHT,
02150          EXX                    ;  ADD 3 TO EDCUR AND 1
02160          INC     HL             ;  TO BYTED.
02170          EXX
02180          DJNZ    SPCRGT
02190 ;
02200          LD      (EDCUR),HL     ;UPDATE EDIT CURSOR POS'N
02210          LD      (HL),CURCHR    ;  AND PRINT CURSOR.
02220          EXX
02230          LD      (BYTED),HL     ;UPDATE EDIT BYTE POS'N
02240          EXX
02250 ;
02260 ;        DISPLAY THE CURRENT EDIT BYTE ADDRESS ALONG
02270 ;        WITH THE REFERENCE ADDRESS AND OFFSET FROM IT.
02280          LD      HL,3CFAH       ;WILL PRINT HERE FIRST.
02290          LD      (CURPOS),HL
02300          LD      HL,STRPLC+4    ;PUT IN TERMINATOR FOR
02310          LD      (HL),3         ;  STRINGS BELOW.
02320 ;
02330          DEC     HL             ;MAKE BYTE THAT EDIT
02340          LD      BC,(BYTED)     ;  CURSOR POINTS TO INTO
02350          CALL    BSTRNG         ;  AN ASCII STRING AND
02360          CALL    OUTSTR         ;  PRINT IT.
02370 ;
02380          LD      HL,3D7AH       ;PRINT REF. ADDRESS SIM-
02390          LD      (CURPOS),HL    ;  ILARLY.
02400          LD      HL,STRPLC+3
02410          LD      BC,(ATADD)
02420          CALL    BSTRNG
02430          CALL    OUTSTR
02440  ;
02450          LD      HL,3DB8H       ;TO PRINT OFFSET, FIRST
02460          LD      (CURPOS),HL    ;  CLEAR THE LINE.
02470          LD      A,1EH
02480          CALL    BYTDIS
02490          LD      HL,(BYTED)     ;NOW CALC. OFFSET.
02500          LD      DE,(ATADD)
02510          XOR     A              ;RESET CARRY FLAG.
02520          SBC     HL,DE          ;GET OFFSET IN HL.
02530          LD      (4121H),HL     ;GIVE IT TO THE ROM
```

(continued...)

```
        CALL    NTF2            ;  AS AN INTEGER.
        CALL    NUMSTR          ;ROM MAKES IT A STRING.
        CALL    OUTSTR          ;WHICH WE PRINT.
        LD      A,'D'           ;FOLLOW WITH D FOR DEC.
        LD      (3DBEH),A
        LD      HL,3DB8H        ;NOW SEE IF IT WAS A
        LD      A,(HL)          ;  NEGATIVE NUMBER.
        CP      '-'
        JR      Z,KBDSCN
        LD      (HL),'+'        ;IF NOT, MARK WITH +.
;
;       THE EDIT CURSOR POSITION AND ALL POINTERS HAVE
;       BEEN COMPLETELY UPDATED AT THIS POINT.  ALL
;       LOCATION INFORMATION HAS BEEN PRINTED.  NOW
;       BLINK THE CURSOR WHILE AWAITING KEYBOARD INPUT.
KBDSCN  LD      B,BDELAY        ;BLINK ON COUNTDOWN TO 0.
HALFCK  LD      A,(SHIFT)       ;SEE IF SHIFT PRESSED.
        AND     A
        JR      Z,REGKI         ;GO IF NO SHIFT
        LD      A,(SHCNTL)      ;ELSE CHECK FOR RETURN TO
        CP      SHFLFT          ;  ADDRESS PROMPT OR
        JP      Z,ENTRY2        ;  FOR SHIFTED UP/DOWN
        CP      SHFTUP          ;  ARROWS.  IF UP OR DOWN
        JR      Z,DOUP          ;  ARROW, SCROLL WITHOUT
        CP      SHFTDN          ;  DELAY.
        JR      Z,DODN
        CP      SHFBRK          ;IF SHIFT/BREAK, REPLACE
        JR      NZ,REGKI        ;  CURRENT REF. ADDRESS
        LD      HL,(BYTED)      ;  WITH THE BYTE POINTED
        LD      (ATADD),HL      ;  TO BY THE CURSOR.
        JR      JUPDAT
;
REGKI   CALL    KIR             ;SCAN REGULAR KEYS WITH
                                ;  REPEAT.
        JR      NZ,KPRESS       ;GO IF ANY KEY DOWN.
        DJNZ    HALFCK          ;BACK UNLESS BLINK TIME.
;
        LD      HL,(EDCUR)
        LD      A,(HL)          ; FLIP-FLOP CURSOR CHAR-
        XOR     0AH             ;  ACTER AND GO BACK TO
        LD      (HL),A          ;  BLINK IT.
        JR      KBDSCN
;
KPRESS  LD      (KEY),A         ;A KEY IS DOWN. SAVE IT.
        CP      CLEAR
        JR      NZ,DBNC         ;GO IF NOT CLEAR KEY.
        LD      A,(HATYPE)      ;ELSE ON CLEAR ONLY, FLIP
        XOR     1               ;  -FLOP TYPE OF DUMP.
        LD      (HATYPE),A
        JP      REINIT          ;AND REPAINT SCREEN.
;
```

(continued...)

```
03050 DBNC      LD      BC,KDLAY1       ;DELAY TO AVOID BOUNCE.
03060           CALL    DELAY
03070 ;
03080           LD      A,(KEY)         ;RETRIEVE KEY ENTERED.
03090 ;
03100 ;         NOW CHECK TO SEE IF THE KEY WAS AN ARROW KEY.
03110 ;         GO TO THE APPROPRIATE PROCESSING IF IT WAS.
03120 CKUP      CP      UPAROW
03130           JR      Z,DOUP
03140           JR      CKDN
03150 DOUP      CALL    CURSUP          ;SCROLLING UP IS EASY -
03160           JR      JUPDAT          ;  CURSUP DOES IT ALL.
03170 ;
03180 CKDN      CP      DNAROW
03190           JR      Z,DODN
03200           JR      CKLF
03210 DODN      CALL    CURSDN          ;SCROLLING DOWN IS EASY -
03220           JR      JUPDAT          ;  CURSDN DOES IT ALL.
03230 ;
03240 CKLF      CP      LFAROW
03250           JR      Z,DOLF
03260           JR      CKRT
03270 DOLF      LD      A,(RGTCUR)      ;LEFT ARROW MAY REQUIRE
03280           DEC     A               ;  MOVING CURSOR UP ONE
03290           CP      0FFH            ;  IF AT EXTREME LEFT.
03300           LD      (RGTCUR),A
03310           JR      NZ,JUPDAT       ;GO IF NOT AT FAR LEFT.
03320           LD      A,15            ;ELSE MOVE TO FAR RIGHT
03330           LD      (RGTCUR),A      ;  AND SCROLL CURSOR UP
03340           CALL    CURSUP          ;  ONE ROW.
03350           JR      JUPDAT
03360 ;
03370 CKRT      CP      RTAROW
03380           JR      Z,DORT
03390           JR      KEYIN
03400 DORT      LD      A,(RGTCUR)      ;RIGHT ARROW MAY REQUIRE
03410           INC     A               ;  MOVING CURSOR DOWN ONE
03420           CP      16              ;  IF AT EXTREME RIGHT.
03430           LD      (RGTCUR),A
03440           JR      NZ,JUPDAT       ;GO IF NOT AT FAR RIGHT.
03450           XOR     A               ;ELSE MOVE TO FAR LEFT
03460           LD      (RGTCUR),A      ;  AND SCROLL CURSOR DOWN
03470           CALL    CURSDN          ;  ONE ROW.
03480 ;
03490 JUPDAT    JP      UPDATE          ;ARROW KEY PROCESSED.
03500 ;
03510 ;
03520 ;         ARRIVE HERE IF KEY PRESSED WAS NOT A CONTROL
03530 ;         KEY.  THAT SIGNIFIES AN ATTEMPT TO ENTER DATA.
03540 ;         IF IN THE HEX MODE, CHECK THE DATA FOR 0-F.  IF
03550 ;         IN THE ASCII MODE, JUST ENTER AND DISPLAY IT.
```

(continued...)

```
KEYIN   LD      BC,KDLAY2
        CALL    DELAY           ;DELAY FOR DEBOUNCE.
        LD      A,(HATYPE)      ;CHECK TYPE OF DUMP.
        AND     A
        JR      NZ,ASCIN        ;GO IF ASCII.
;
        LD      A,(KEY)         ;IN HEX MODE. CHECK KEY.
        CALL    VALHEX          ;RETURNS CARRY SET IF KEY
        JP      C,UPDATE        ;  IS INVALID HEX.
        RLCA                    ;ELSE RETURNS VALID HEX
        RLCA                    ;  IN A.
        RLCA                    ;PUT HEX 0-F IN HIGH
        RLCA                    ;  NYBBLE OF A.
        LD      (KEYH),A        ;SAVE IT.
;
        LD      HL,(EDCUR)      ;INCIDENTALLY REDRAW
        LD      A,CURCHR        ;  EDIT CURSOR SINCE IT
        LD      (HL),A          ;  MAY BE OFF.
;
        LD      A,(KEY)         ;RETRIEVE KEY AND DISPLAY
        CALL    CURINC          ;  IT AT EDCUR+1.
        CALL    BYTDIS
;
WAITNX  LD      A,(CNTL)        ;NOW WAIT FOR NEXT VALID
        CP      BREAK           ;  HEX ENTRY.  BREAK BACK
        JP      Z,REINIT        ;  TO MAIN ROUTINE IF
        CALL    KIR             ;  REQUIRED.
        CALL    VALHEX          ;CHECK WHATEVER CAME IN,
        JR      C,WAITNX        ;  EVEN IF NOTHING DID.
;
        LD      C,A             ;HAVE A VALID HEX CHAR!
        LD      A,(KEYH)        ;MERGE IT WITH LAST CHAR.
        OR      C
        LD      HL,(BYTED)      ;STORE NEW BYTE AT BYTED.
        LD      (HL),A
;
        LD      A,(HL)          ;RETRIEVE AND DISPLAY IT
        CALL    CURINC          ;  AT THE EDIT CURSOR.
        LD      C,A             ;  THIS VERIFIES THE EDIT
        CALL    ASCII           ;  ACTION.
        EX      DE,HL
        LD      A,H
        CALL    BYTDIS
        LD      A,L
        CALL    BYTDIS
;
        LD      BC,KDLAY1+KDLAY2  ;DEBOUNCE DELAY HERE
        CALL    DELAY           ;  LIKE OTHER PATHS.
        JP      DORT            ;AND GO ACT JUST AS IF
                                ;  RIGHT ARROW PRESSED.
```

(continued...)

```
04060 ;
04070 ASCIN   LD      A,(KEY)         ;KEY IN SHOULD BE VALID
04080         LD      C,A             ;  ASCII, BUT CHECK IT
04090         CALL    VALASC          ;  TO BE SURE...
04100         JR      NC,JUPDAT       ;GO IF BAD.
04110         LD      HL,(BYTED)      ;ELSE ENTER INTO RAM AND
04120         LD      (HL),A          ;  READ IT BACK TO VERIFY
04130         LD      A,(HL)          ;  THE EDIT ACTION.
04140         LD      C,A             ;CHECK AGAIN FOR ASCII
04150         CALL    VALASC          ;  IN CASE WE'RE IN ROM.
04160         JR      NC,ENDASC       ;GO IF BAD.
04170         CALL    CURINC
04180         CALL    BYTDIS          ;ELSE DISPLAY IT.
04190 ;
04200 ENDASC  JP      DORT            ;PRETEND WAS RIGHT-ARROW.
04210 ;
04220 ;
04230 ; END OF MAIN PENRAM ROUTINE.
04240 ;- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
04250 ;
04260 ; SUBROUTINES ---
04270 ;
04280 ; VALHEX TAKES AN ASCII CODE IN  A  AND CHECKS TO SEE IF
04290 ; IT IS A VALID HEX CHARACTER (0-F).  IF IT IS NOT,
04300 ; VALHEX RETURNS WITH THE CARRY FLAG SET.  IF IT IS A
04310 ; HEX VALUE, THE  A  REGISTER RETURNS WITH THAT HEX
04320 ; VALUE (0-F) IN IT, AND THE CARRY FLAG RESET TO 0.
04330 VALHEX  LD      (KEY),A
04340         LD      C,A
04350         LD      A,'F'
04360         CP      C
04370         JR      C,NOTHEX        ;C SET IF CHAR > F.
04380         LD      A,'/'
04390         CP      C
04400         JR      NC,NOTHEX       ;C SET IF CHAR > /.
04410         LD      A,'@'
04420         CP      C
04430         JR      C,ATHRUF        ;C SET IF CHAR > @.
04440         LD      A,C
04450         CP      ':'
04460         JR      C,ZTHRU9        ;C SET IF CHAR < :
04470 NOTHEX  SCF                     ;TO FLAG NOT-HEX.
04480         RET
04490 ZTHRU9  SUB     30H             ;CONVERT TO HEX.
04500         RET                     ;CARRY WON'T BE SET FOR
04510 ATHRUF  LD      A,(KEY)         ;  EITHER OF THESE.
04520         SUB     37H             ;  RETURNS ON VALID HEX.
04530         RET
04540 ;
04550 ;
```

(continued...)

```
; SUBROUTINE VALASC TAKES A BYTE IN C AND CHECKS TO SEE
; IF IT A PRINTABLE ASCII CHARACTER - EITHER UPPER OR
; LOWER CASE.  IF ALPHANUMERIC IT IS SIMPLY RETURNED
; IN A, WITH THE CARRY FLAG SET.  IF NON-PRINTABLE, AN
; UNDERSCORE IS SUBSTITUTED AND THE CARRY FLAG IS RESET.
VALASC  LD      A,7FH
        CP      C
        JR      C,NOTASC        ;HAVE CARRY IF CHAR > 7F
        LD      A,1FH           ;SEE IF ITS TOO LOW TO
        CP      C               ;  BE ASCII.
        LD      A,C
        RET     C               ;RETURN IF PRINTABLE.
;
NOTASC  LD      A,5FH           ;NOT ASCII.  PUT IN
        AND     A               ;  UNDERSCORE AND
        RET                     ;  RESET CARRY.
;
;
; FRMBYT LOOKS AT (HL), (HL+1) AND IF THE ASCII CODES
; THERE ARE BOTH FOR HEX CHAR 0-F, THEN THE HEX BYTE
; THEY DEFINE IS FORMED IN  A  AND RETURNED, WITH THE
; CARRY FLAG RESET TO 0.  IF EITHER OF THE ASCII CHAR
; IS NOT 0-F, THEN A RETURN IS MADE WITH THE CARRY FLAG
; SET.
FRMBYT  LD      A,(HL)          ;GET FIRST CHAR.
        CALL    VALHEX          ;CHECK IT FOR 0-F.
        RET     C               ;RETURN IF NOT 0-F.
        LD      E,A             ;WAS 0:F.  KEEP ITS
        DEC     HL              ;  VALUE IN E.
        LD      A,(HL)          ;GET SECOND CHAR.
        CALL    VALHEX          ;IS IT ALSO HEX?
        RET     C               ;RETURN IF NOT.
        RLCA                    ;BOTH ARE 0-F.  FORM
        RLCA                    ;  THE HEX BYTE IN A.
        RLCA
        RLCA
        OR      E
        DEC     HL              ;PREPARE FOR NEXT CALL.
        RET                     ;RETURN W/HEX IN A.
;
;
; CURINC POSITIONS THE STANDARD CURSOR AT THE LOCATION
; OF THE EDIT CURSOR PLUS ONE.
CURINC  LD      HL,(EDCUR)
        INC     HL
        LD      (CURPOS),HL
        RET
;
;
; KIR SCANS THE KEYBOARD AND RETURNS IN A THE ASCII VALUE
; OF ANY KEY THAT IS PRESSED AT THAT INSTANT.  IT
```

(continued...)

```
; ALSO CLEARS THE KEYBOARD INPUT ROUTINE BUFFER AT
; 4036-403C.  THIS MAKES THE KI ROUTINE THINK THAT
; THE LAST KEY PRESSED WAS SUBSEQUENTLY RELEASED, SO THAT
; IT WILL SCAN AGAIN, THUS GIVING THE EFFECT OF REPEATING
; KEYS.
KIR     CALL    KI                  ;GET KEY IF PRESSED.
        LD      D,A
        LD      E,7                 ;WILL PUT 7 ZEROS IN
        LD      HL,KIBUF            ;  BUFFER.
        XOR     A
RPT     LD      (HL),A              ;CLEAR ONE BYTE.
        INC     HL
        DEC     E
        JR      NZ,RPT
        LD      A,D                 ;RETRIEVE CHAR.
        AND     A                   ;SET FLAG IF ZERO.
        RET
;
 ;
; CURSUP HANDLES ALL MOVEMENT OF THE CURSOR UPWARDS.
; IT CHECKS FIRST TO SEE IF THE TOP OF THE SCREEN HAS
; BEEN REACHED (DWNCUR=0).  IF NOT, IT SIMPLY DECREMENTS
; DWNCUR AND RETURNS TO THE UPDATE.  IF AT THE TOP OF
; SCREEN, THE TOP 15 LINES MUST BE MOVED DOWN ONE, AND
; THE NEW DUMP LINE (AS FORMED BY MAKSTR) PRINTED ON THE
; TOP LINE.  SINCE MAKSTR NEEDS AN ADDRESS ADDR TO FORM
; THE DUMP STRING, CURSUP MUST ALSO DETERMINE WHETHER THE
; LAST SCREEN SCROLLING WAS DUE TO THE CURSOR MOVING UP
; (IN WHICH CASE THIS IS MERELY A CONTINUATION AND ADDR
; IS ALREADY CORRECT) OR DOWN (IN WHICH CASE THE VALUE OF
; ADDR WAS LEFT BY CURSDN AT 272 BYTES MORE THAN WHAT IS
; WANTED HERE).
; BEFORE RETURNING, CURSUP SUBTRACTS 16 FROM BOTH ADDR
; (ANTICIPATING ANOTHER CURSOR-UP SCROLL) AND FROM
; HOMADD, BECAUSE THE ADDRESS DISPLAYED AT THE TOP LEFT
; HAS ALSO DECREASED BY 16 BYTES.
; NOTE THAT ONLY THE FIRST 55 CHAR. OF EACH LINE ARE
; SCROLLED;  THE LAST 9 CHAR. ARE UNTOUCHED.
CURSUP  LD      A,(DWNCUR)
        AND     A
        JR      Z,CONTU1            ;GO IF TOP OF SCREEN.
        DEC     A                   ;ELSE MOVE EDIT CURSOR UP
        LD      (DWNCUR),A          ;AND RETURN TO UPDATE.
        RET
;
CONTU1  LD      A,(LSTARW)          ;SEE IF LAST SCROLL WAS
        AND     A                   ;DUE TO CURSOR DOWN.
        JR      Z,CONTUP            ;GO IF NOT.
        LD      BC,272              ;MUST FIX ADDR.
        CALL    SADDR               ;SUBTRACT 272 FROM IT.
```

(continued...)

```
;
CONTUP  LD      HL,(EDCUR)      ;ADDR IS CORRECT HERE.
        LD      (HL),BLANK      ;REMOVE OLD CURSOR.
        LD      HL,(ADDR)       ;GET ADDR TO DUMP.
        CALL    MAKSTR          ;MAKE DUMP STRING OF IT.
;
        LD      A,15            ;SCROLL TOP 15 LINES
        LD      HL,3F80H        ;  DOWN, MOVING ONLY THE
        LD      DE,3FC0H        ;  LEFT 55 CHARACTERS.
MVLOPS  LD      BC,37H
        LDIR
        LD      BC,-119
        ADD     HL,BC           ;POINT TO NEXT LINE.
        EX      DE,HL
        ADD     HL,BC           ;AND WHERE IT WILL GO.
        EX      DE,HL
        DEC     A
        JR      NZ,MVLOPS       ;BACK IF 15 NOT DONE.
;
        LD      HL,LINE01       ;POINT TO FIRST LINE.
        LD      (CURPOS),HL
        LD      HL,STRPLC
        CALL    OUTSTR          ;PRINT DUMP STRING THERE.
;
        LD      BC,16           ;ADD 16 TO ADDR, HOMADD.
        CALL    SADDR
        CALL    SHOMAD
        XOR     A               ;NOTE SCROLL CAUSED BY
        LD      (LSTARW),A      ;  UP-ARROW.
        RET
;
;
; CURSDN IS THE MIRROR IMAGE OF CURSUP IN THAT ALL
; THE SAME ACTIONS ARE INVOLVED, BUT THE CURSOR IS
; MOVED DOWNWARD.  IF IT IS ALREADY AT THE BOTTOM OF
; THE SCREEN, THE BOTTOM 15 LINES ARE SCROLLED UP AND
; THE NEW LINE FORMED BY MAKSTR IS PRINTED ON THE
; BOTTOM.  ADDR IS CORRECTED IF THE LAST SCROLL WAS DUE
; TO AN UP-ARROW, AND IS LEFT ANTICIPATING ANOTHER DOWN-
; ARROW COMMAND.  THAT IS, IT POINTS 16 BYTES BEYOND THE
; ADDRESS DISPLAYED AT THE BOTTOM LINE.  HOMADD IS
; UPDATED TO REFLECT THE NEW ADDRESS DISPLAYED IN THE
; TOP LEFT CORNER.
CURSDN  LD      A,(DWNCUR)
        CP      15
        JR      Z,CONTD1        ;GO IF BOTTOM OF SCREEN.
        INC     A               ;ELSE MOVE EDCURSOR DOWN.
        LD      (DWNCUR),A      ;AND RETURN TO UPDATE.
        RET
;
```

<<<< LISTING CONTINUED, PG. 66 >>>>

......60 Cycle Sine Wave U.P.S.

(Uninterruptibe Power Supply)

●●●●for those systems that need 60 cycle sine wave keeps computer & disk systems on when the power goes out **For less than $500.00***

●●●●Rated 250 Watt continuous operation

●●●●will provide 400 Watts for 5 minutes (Model II TRS-80 with 4 disk drives)

* Standard MAYDAYS available starting at $195.00 for 150 Watt

Mayday™

from●●● *Sun Research, Inc.*
Box 210 New Durham, NH 03855
(603) 859-7110 TWX 510-297-4444

FROM THE SOURCE'S MOUTH

By Joni M. Kosloski

I'd like to take this page or so and share with you the two major complaints our 'magazine' has received since it's conception. Both have resulted in subscription cancellations, the only two we've experienced. (This technique sure makes an impression on us! You might try it elsewhere.)

The first is that we're "only" bimonthly. Apparently bimonthly doesn't qualify for the valuable periodical category. We have several defenses regarding this:

1. We are priced to accommodate a bimonthly schedule. In all humbleness (and we all know it's hard to be humble!) we feel we give close, if not the best, dollar value per issue available.

2. BTI, our subjective sister, is evolving quite rapidly. If the next couple issues are well received we will probably announce this as generally available. BTI is published on months opposite from TAS.

3. There is no question that we could generate enough articles to go monthly, but the more articles we solicit, and the more articles we have to publish, the less we can pick and choose. We also like to work with authors, trying to develop viable subjects and quality articles. This can evolve into a time consuming project.

We like to think that we exist for more than the sake of publishing. If articles don't generate a minimum amount of favorable feedback and discussion then the job just won't be fun anymore, and I don't think you'd enjoy each issue as much as (I hope) you do now.

Actually, what you should do is subscribe to both us and 80-U.S. and get 12 months of really diverse exposure! (I wonder if Mike Schmidt has these problems?)

As a bottom line to this first complaint, becoming a monthly publication is something that will almost definitely happen -- it's just a matter of REALLY being ready for it. We'll know when we are, and we'll let you know, too.

Our second major complaint is regarding our heavy emphasis on DOS and a lack of material for a Level II, 16K configuration. We've received a small handful of letters with comments like "If you're concentrating on SERIOUS applications, then forget Level II!" or "You mean there's STILL people using TAPE?" Well, I personally know quite a few people who will get very defensive about their Level II systems. And I can't argue with them. I don't think it's a lack of money for upgrading, either. These people are quite satisfied with their under-a-thousand dollar machines; who are we to judge them? If they can get a Level II TRS-80 to accomplish what they want it to, I think that's commendable.

I also know, from software orders and various feedback, that a large portion of our subscribers have disk systems. I know the figure is over half; I would hesitate only briefly to claim a 75% figure.

As far as articles for either configuration, it seems that there is a growing number of authors coming from the DOS portion of the TRS-80 audience. But we're trying to satisfy both sections. Recent feedback seems to indicate we're keeping both sides satisfied. It's important to let us know your wants and needs -- keep us informed! More than one reader has indicated that we should address articles more clearly towards either Level II or DOS. We really thought we were doing this. Speaking generally, if one or the other is not mentioned in the first paragraph, the reader can safely assume that the topic is applicable for both systems.

Another reader writes a more lengthy tale -- can you actually believe we were criticized for lack of software ads in issue 4? I was really surprised. His basis for comments is that no publication can survive on subscriptions alone (amen!). They actually have two choices -- they can help support the publication through numerous advertisements, or through software sales. At this point, TAS is staying alive and well thanks to the many subscribers that also choose to do their software business with us as well.

However, I have heard many complaints about other so-called magazines actually being software catalogs for the company involved. I tend to agree, and try not to swamp you folks with listings of everything we have in stock. I extend my appreciation to those of you who support our software staff; let's suffice it to say that we handle a multitude of software and would be pleased to service your needs. We'll continue to run a minimum of ads for those of you who are not in regular contact with us -- it's our means of letting you know about new and exciting products.

NEWDOS-80

An Appraisal, by Al Domuret

Well, here it is. After long, anxious, and curious anticipation, NEWDOS-80 has finally arrived. And an impressive package it is! But that is my opinion, and everyone may not agree. After wading through the following paragraphs, the reader should have a better idea of what NEWDOS-80 is all about.

What I hope to accomplish with this appraisal is twofold: to provide for those of you who have not yet purchased NEWDOS-80 enough information, both pro and con, so you can determine whether or not you want to invest in it, and also to provide the current and potential users with a bit of operational guidance and some fixes for some of the minor difficulties I encountered. Table 1 at the end of this article contains some zaps that should also prove useful.

Upon receipt of the NEWDOS-80 package (hereinafter also referred to as N80), the first surprise was the quantity of documentation. A quick count revealed 96 pages, printed on both sides and bound in an attractive brown notebook. I was pleased at this breakthrough -- up to this point, Apparat was known for the quality of their software, not necessarily the documentation. While it is my impression that the documentation is complete, well done and understandable, I have heard complaints that it still isn't clear enough. Perhaps this is understandable, since NEWDOS-80 is a complex and somewhat advanced disk operating system, and there is a lot to it in terms of both quality and quantity. The distributed diskette has ZERO free space!

The following quote has been extracted from Page 1-1 of the N80 documentation to give potential buyers some idea of what to expect:

> "NEWDOS-80 is not a simple system. ...the user...should spend one to two hours studying the documentation before doing anything with the NEWDOS-80 diskette."

Had I written Apparat's N80 documentation, I would have stated it differently: "N80 will take time to learn, just as any complex piece of new software would. It is not a game that will simply 'load and go'. However, once the power and utility of N80 is fully digested and understood, it is no more difficult to use than any other system." When first getting NEWDOS-80, the novice will find many of the functions easy to use and understand. The COPY and BACKUP functions of N80 are slightly different and more advanced than TRSDOS, but simple to use. The psuedo-experts will find plenty of new and complex stuff to keep them busy for quite some time, too.

Okay, but aside from the usual DOS features carried forward from TRSDOS 2.3 and NEWDOS 2.1, just what does N80 offer that is new and different? Well, first, it permits the DOS itself to be "customized". This one feature can be as powerful as the user wants it to be -- you set up the DOS according to your exact hardware configuration and your exact needs. For example, it is possible, using the new "PDRIVE" command, to define your DOS for diskettes having 01 to 96 tracks, almost any combination of 5 or 8 inch drives, single or double density. (Since someone is bound to ask, this magic can only occur if you have the proper hardware attachments -- N80 provides only the necessary software.)

It is also possible to define either the OMIKRON eight inch drive system or the LOBO expansion interface to N80. Provisions are made for future hardware adaptations by leaving space for additional customizing code.. For those of you having 77 track (or more) drives, you've probably had occasions when the standard one-track directory doesn't have enough capacity for the numerous files on the diskette. NEWDOS-80's directory can be expanded to up to three tracks for this purpose. Additionally, the directory track can be relocated elsewhere on the diskette, if desired. I often wondered if having the directory on track one rather than 17 (decimal) would be more efficient in terms of reducing disk head travel. Now I can find out!

Further, it is possible to use the new DOS "SYSTEM" command (which is not the same as Basic's SYSTEM command used for loading machine language tapes) to customize certain DOS functions. Some of the more interesting ones are: passwords can be enabled or disabled; there's a BASIC RUN-ONLY mode to protect BASIC code from unauthorized access by computer operators which can be enabled or disabled; lower case as installed or not installed; BREAK key enabled or disabled; DEBUG enabled or disabled; and many more. No manual ZAPS are required. Just follow instructions for DOS "SYSTEM" command and everything else is pretty much automatic. I initially experienced some difficulty getting my DOS SYSTEM command to work. The solution is to enter the command as follows:

```
SYSTEM,USD:1 AA=N          (and so on...)
```

Evidently, the USD (Use System Disk) serves as a password until such time as they are disabled by the appropriate SYSTEM command. Unless I was initially doing something wrong, this was not immediately obvious to me.

MINI-DOS and DOS commands via the 2.1 "CMD" functions are beauties. Specifically, MINI-DOS and "CMD" are two ways of bringing up DOS commands.

The operational distinctions between DOS and BASIC become blurred as it becomes possible to execute most DOS library commands at almost any time, and from almost any program, whether in BASIC or otherwise. (Library commands are the usual commands such as DIR, DUMP, FREE, CLOCK, etc. Not included are the commands that cause a program to execute.) Let's first take a look at the "CMD" function.

The BASIC "CMD" function still works as it did in NEWDOS 2.1, but it has been expanded to allow more DOS operations as long as the DOS command doesn't clobber the resident BASIC program and its variables. N80 performs some checking routines to help prevent this disaster. For example, both SUPERZAP (which is now written in machine language, by the way, and is it fast!) and DIRCHECK can be exercised from BASIC (one at a time) without clobbering BASIC operations, variables or programs. Unless, of course, you do so intentionally. Here is an interesting trick that can be worked in BASIC:

```
10 CLS
20 CMD"SUPERZAP":REM ** DIRCHECK also possible **
30 CLS
40 PRINT"Finished"
```

When finished with SUPERZAP (or DIRCHECK), exit with SUPERZAP's EXIT command or DIRCHECK's character "N". Although both SUPERZAP and DIRCHECK imply that they will exit to DOS, both will return to BASIC if called up from BASIC, like the mini program above demonstrates. There are more possibilities, but there is not enough space here to cover them all. Certainly, many unique applications can be found for "CMD" operations from BASIC.

Now let's consider MINI-DOS. Perhaps the N80 documentation best explains its purpose:

"There are many times when, during the execution of a main program, the operator would like to interrupt the main program, execute one or more of the DOS library commands, and then resume main program execution without any change having occurred in the main program's state during the interruption. To execute MINI-DOS, simultaneously press the 'DFG' keys (but not during disk I/O), and execute the DOS command. Any DOS library command can be issued except APPEND, CHAIN, COPY, FORMAT, PDRIVE, and SYSTEM. Single file copy can, however, be executed with the MDCOPY command."

(See? That bit of documentation wasn't too difficult to read, was it?) What exactly can MINI-DOS do? How about calling up a DIRectory from SCRIPSIT?! Simple. Move the cursor to the end of the text file (a precautionary measure to prevent the 'DFG' keys from overwriting SCRIPSIT text when pressed) or to the command position by using the BREAK key (preferred over the former method). Now press the 'DFG' keys to bring up MINI-DOS, then enter "DIR :1". When finished, just enter "MDRET" to return to SCRIPSIT. Yes, Tandy, Apparat bailed you out again! A major SCRIPSIT weakness, the inability to manipulate directory files, has been compensated for. For users who, like myself, have become addicted to ST80D or ST80III (by Lance Micklus), its inability to call a directory is no longer a problem either. Use MINI-DOS to perform a DIR, KILL, or whatever -- without disturbing the main program.

Want to check diskette free space before writing to it? Or kill a file to release space on a full diskette? How about renaming a file? No problem, either. Want to exit SCRIPSIT without reaching back for the reset button? Enter MINI-DOS, then enter "BOOT". It's great!

There are some things to understand in order to maximize accessibility to MINI-DOS. Toward the end of this article, some instructions are provided on how to achieve MINI-DOS operations with the Electric Pencil (Shrayer Software) and on what to watch out for in other programs that might defeat MINI-DOS, especially those written in machine language.

Another subtle but potentially powerful feature has been added for bringing up DEBUG: simultaneous depression of the '123' keys will bring up DEBUG at any time. This is a handy way of getting to the innards of a BASIC program that disables the BREAK key to prevent the program from being listed. Similarly, DEBUG can be activated in the midst of a machine language program, as long memory requirements of the program and DEBUG don't conflict and the target program doesn't conflict with the keyboard Device Control Block (DCB) at 4016 hex.

The CHAINing command will exercise a series of DOS or BASIC commands, or it can make automatic inputs to a BASIC program. For those who are familiar with the BOOTSTRAP or COMPROC programs (Practical Applications and RACET computes), the functions are similar. One important difference, though, is that BASIC can implement a CMD "CHAIN" instruction which will then activate a predetermined collection of inputs to BASIC. Think of it as automating the keyboard.

A nice feature of the N80 CHAIN command is that a wasteful five-sector (one gran) file is not necessarily created for every individual CHAIN file. To avoid this, you can append a number of CHAIN command strings into one file, then access only the one that is needed.

For machine language enthusiasts, the documentation provides a wealth of technical information for making use of N80's internal instruction code in personal programs. As only one example, a "DOS CALL" machine language routine is provided which very easily allows the incorporation of DOS commands into your programs. Want to use RSM2D to 'LOAD' a file into RAM? It's possible with about a dozen bytes of code. How about creating your own SYS20/SYS file? The documentation explains how. Apparat's 'no secrets' marketing approach is genuinely refreshing.

Disk BASIC, like N80, is a complete rewrite. BASIC still works as it always did, and there appear to be no BASIC program incompatibilites. But Disk BASIC has been upgraded with a number of fascinating enhancements.

In addition to being able to RENUMber a BASIC file, it is now possible to move program lines around within the program. REF (variable and line number cross referencing), improved scrolling, and "CMD" (described above), are still available and operate as implemented in NEWDOS 2.1. In BASIC RUN-ONLY mode, the CLEAR and BREAK keys are disabled and will not accept direct statements from the operator. In business applications, this keeps the computer operator from gaining access to unauthorized files and from manipulating the program itself. A nice touch is that Apparat thoughtfully made it possible for a BASIC menu program to LOAD or RUN another BASIC program. In other words, it is possible for a BASIC program to call up another program or data file with embedded commands, but the computer operator cannot do this directly from the keyboard. At power-up, the computer boots up and directly executes the selected program. The operator has no control other than to operate the program as intended by the boss. Of course, DEBUG and MINI-DOS can also be locked out to further frustrate the operator.

Now comes the part that I am afraid will leave the reader somewhat unfulfilled. A number of wondrous new BASIC Disk Input/Output (I/O) enhancements are implemented, but there is not enough space to go into all the details here.

Briefly, disk record lengths can now be up to 4096 bytes long instead of the old 256 byte maximum. Disk files can be created and accessed in a variety of ways that allows manipulation of file data in almost any conceivable format, from a single byte to a 4096 byte record, using fixed length files or variable length files. These new disk I/O options are available via five new BASIC file types which are classified into two major groupings: fixed item files and marked item files.

But the reader is cautioned. Apparat introduces new and esoteric terminology which makes things a bit difficult for the novice. To simplify things as much as possible, N80 includes a sample program which is accompanied by a tutorial chapter in the documentation to help the new user along. This beginners approach, plus an alphabetized glossary, eases the operator gradually and painlessly into a very powerful set of disk I/O functions.

In the tutorial, the operator is walked through the procedures, step by step, for creating and using the new BASIC files. To further simplify things, analogies are made to the more traditional sequential and random file operations. With a little patience and practice, a whole new world of disk I/O manipulations becomes possible. One might ask, "Are my old sequential and random disk BASIC files still compatible? And can I still create them with N80's enhanced BASIC?" The answer is, absolutely. Your old BASIC files are compatible with N80, and it is still possible to create the "standard" sequential and random files.

AREAS OF INCOMPATIBILITY

For the benefit of readers who have not yet learned a fundamental fact about newly introduced software, particularly when the softwaree is a new Disk Operating System, there is no way in the world to maintain total compatibility with existing software. This is especially true when machine language programs directly access "non-standard" subroutines within the guts of the DOS software itself. It may sound trivial to make such an obvious statement, but it has become very clear to me that even well known and experienced personalities in this microcomputer business can get upset; indeed, have gotten upset, about the few and minor incompatibilites in NEWDOS-80.

Apparat acknowledges known incompatibilites and they even provide the necessary zaps or guidance, where applicable, to compensate for them. If and when new problems arise, I feel certain that fixes will become available. At any rate, it appears at the present time that N80's incompatibilites are trivial.

Enough rationalizing. The following is a brief description of real or potential incompatibilites as listed in the N80 documentation.

1. User routines which are driven by the 25ms TRSDOS or NEWDOS 2.1 interrupt must be modified to work with N80. The N80 documentation explains the patches necessary to correct potential problems. To my knowledge, about the only commercial programs that might experience interrupt problems are spoolers and despoolers.

There is a compatibility problem with ST80III and with the Microsoft FORTRAN package, including the MACRO-80 assembler, but Apparat thoughtfully provides the necessary zaps to make these programs compatible with N80.

2. Enabling or disabling the BREAK key was formerly accomplished by changing the address at 4313 hex. The new address to accomplish this is 4369 hex. The procedure is explained in the N80 documentation.

3. The same NEWDOS 2.1 "incompatibility" involving 'NEXT' and 'EOF' in the FCB (File Control Block) continues with NEWDOS-80. Typically, this has been inconsequential for most users.

In this same context, SCRIPSIT owners have no doubt learned by now that its disk I/O works properly only with TRSDOS. But Apparat provides the necessary zaps to make SCRIPSIT work with N80, and with NEWDOS 2.1 also.

It should be evident that these minor incompatibilites will not concern the average user unless one of the spoolers marketed by other distributors is involved. Machine language pros should have no difficulty in coping. Again, the N80 documentation lists old and new interrupt addresses to aid machine language hackers in making repairs.

Virtually all popular DOS routines are still accessible with the standard DOS address calls; for instance, Open, 4424H; Read, 4436H; Display ASCII text, 4467H; and so on. This applies to BASIC functions as well. The traditional DOS call addresses, plus some new ones, are conveniently listed in the N80 documentation.

As was true with NEWDOS 2.1 when first introduced, and as is now true with NEWDOS-80, Apparat has done an exceptional job of maintaining compatibility among DOS systems. This is especially notable when one considers that N80 is a complete rewrite. Not only has Apparat given us the best available DOS system for the TRS-80 (NEWDOS 2.1, and now NEWDOS-80), they have also managed to do so with a minimum of inconvenience to the user.

I found two other problems. Both the Electric Pencil and the MISOSYS Editor Assembler (get this one if you haven't already. It has some great improvements over the Apparat disk EDTASM, and its disk files are mutually compatible with Apparat's EDTASM. It's called DISK*MOD and available through TAS) will not properly read a disk directory. The fixes for both of these programs are provided in Table 1.

For TRS-80's with a CPU clock speedup board installed, the zaps to make N80 function at faster clock speeds are also provided in Table 1. However, let me suggest this: If your TRS-80 is modified for the 2.66 Mhz CPU clock, first try N80 without my fast clock zaps; they may not be required. Since I do not have a 2.66 Mhz system, I have no way of knowing if the unmodified N80 will work at the 50 percent speedup.

Those of you who are running at a 100 percent CPU clock speedup (3.54 Mhz) will no doubt have to install my zaps. I am running at 4.0 Mhz and N80 requires my zaps to function at this clock speed. If my zaps are installed, you should experience no problems running N80 at either the normal 1.77 Mhz or any souped up CPU clock speed, from 2.66 to 4.0 Mhz.

SOME HINTS ON MINI-DOS

In order for MINI-DOS to function, the current program (the program this is up and running) must allow for enabled interrupts and it must not take away the keyboard device control block at memory location 4016 hex.

For example, PENCIL changes the keyboard DCB, although it does not make use of the DCB after it makes the change -- it uses its own keyboard scan routine. Perhaps PENCIL modifies the DCB to keep DEBUG from nosing around or to prevent user access to the PENCIL RAM area. At any rate, the fix is a simple one, and I have experienced no problems as a result. The zaps are listed in Table 1.

By teaching PENCIL to leave the keyboard DCB alone, it is possible to use MINI-DOS, and even DEBUG, without crashing PENCIL or its text. This is useful for renaming PENCIL files, doing a "FREE" on the diskette, or using DEBUG (call it up by pressing '123') to examine the text directly in memory, perhaps to determine how much memory space is still available.

By the way, if DEBUG is called up to play around with PENCIL, 5C6F hex is a good re-entry address. Just enter from DEBUG, 'G 5C6F'. PENCIL's text file will not be disturbed.

If my zaps are not made to PENCIL it will still be possible to call up MINI-DOS or DEBUG, but they will be useless because they will have no access to the keyboard. In fact, your system will hang. Need I say more?

Another thing to watch out for is the block move routines (the LDIR or LDDR instructions) which move programs or subroutines around in RAM. Normally, these block move routines, including those created by Apparat's LMOFFSET, disable interrupts with the DI machine language code as the first instruction. Then, unless the program to be run reactivates interrupts, MINI-DOS or DEBUG will not function.

The solution is to patch each block move routine as necessary with an enable interrupt command (hex code 'FB') before the jump to the program's execution address. Here is what it should look like:

```
DI                               ;disables interrupts
LD          HL,ssss              ;source address
LD          DE,dddd              ;destination address
LD          BC,bbbb              ;byte count
LDIR                             ;move it
EI (new)                         ;enable interrupts (added)
JP          eeee                 ;JP to execution address
```

With the EI properly inserted, interrupts are re-enabled and, if the program does not permanently disable them again (as RSM2D and some others do), MINI-DOS will function normally.

If you have not patched SCRIPSIT with one of these block move routines (as I did with PENCIL), only one ZAP is required to re-enable its interrupts. It is provided in N80's documentation.

Well, there you have it. There are many additional new and useful features in NEWDOS-80, but I tried to limit myself to the most interesting ones. Hopefully there is enough information here to ease the transition into NEWDOS-80 and to give potential buyers some idea of what to expect.

Let me part with one last suggestion. If you have an original NEWDOS 2.1 and are eligible for Apparat's N80 upgrade special deal (submit your registration number and pay the cost differential of $50.00), and if you are considering buying one of the commercially available spoolers for the typical price of $40.00 or more, pay the $50.00 for the NEWDOS-80 upgrade. You get a spooler for free with NEWDOS-80 (remember routine to printer and display?). And it is probably better than most of the others currently available.

TABLE 1
NEWDOS-80 ZAPS

For the following zaps, the purpose is first explained, followed by the relative file's sector in hex, then the relative byte in hex. The format is as follows:

01/60 = relative sector one, relative byte 60

The use of hex is emphasized because the new SUPERZAP/CMD will accept decimal inputs as well as hex.

1. ZAP to make PENCIL read a directory properly:

```
05/60          change:  58 23 22
                   to:  58 00 22
```

2. ZAP to make the MISOSYS EDTASM read a directory properly:

```
08/39          change:  5D 13 ED
                   to:  5D 00 ED
```

3. ZAP to make PENCIL accept a MINI-DOS request:

```
00/61          change:  54 22 16 40 21
                   to:  54 00 00 00 21
```

Also, verify that 00/C5 reads C1 FB C9. Likewise for 01/57; should read C9 FB 21. (Some users zeroed out the FB, which is an enable interrupt code, because of old TRSDOS 2.1 problems).

4. FAST CLOCK ZAPS (Caution: these zaps use DOS RAM areas that appear to be unused, but may turn out to be used by some routine I haven't found yet. Use with caution, and if you develop a problem I would appreciate hearing about it):

```
SYS0/SYS:  03/54    change:  C5 0A 08 F5 E5 E1 E5 E1 F3
                        to:  C5 C3 F5 4C E3 E3 E3 E3 F3
```

(NOTE: the E3's may already be in your SYS0/SYS. If so, leave them, but it will still be necessary to put in the C3 F5 4C.)

```
SYS0/SYS:  04/7E    change:  01 00 80 DC
                        to:  01 00 00 DC


SYS0/SYS:  04/89    verify:  C9 E3 E3 E3 E3 3A
     (The E3's here were already in place in my copy.)


SYS0/SYS:  04/A0    change:  11 00 24 1B
                        to:  11 00 52 1B


SYS0/SYS:  0A/10    change:  00 00 00 00 00 00
                             00 00 00 00 00 A5
                        to:  E3 E3 E3 E3 0A 08
                             F5 C3 59 46 00 A5


SYS6/SYS:  04/C7    change:  01 00 80
                        to:  01 00 00


SYS6/SYS:  0C/20    verify:  E3 E3 E3 E3 0A
     (The E3's were already in place in my copy.)


0C/27:  change:  5E CB 4E C2 37 5E CB 4E 20 3A CB 4E 20
                 36 CB 4E 20 32 CB 4E 20 2E CB 4E 20 2A
                 CB 4E 20 26 D9 C9 CB 4E C2 3F 5E CB 4E
                 20 23 CB 4E 20 1F CB 4E 20 1B CB 4E 20
                 17 CB 4E 20 13 CB 4E 20 0F D9  (Stop)

            to:  5E C5 06 12 CB 4E 20 05 10 FA C1 D9 C9
                 C1 C3 37 5E 00 00 00 00 00 00 00 00 00
                 00 00 00 00 00 00 C5 06 12 CB 4E 20 05
                 10 FA C1 D9 C9 C1 C3 3F 5E 00 00 00 00
                 00 00 00 00 00 00 00 00 00 D9  (Stop)


0C/9F:  change:  0A CB 4E C2 48 5E CB 4E 20 D3 CB 4E 20
                 CF CB 4E 20 CB CB 4E 20 C7 CB 4E 20 C3
                 CB 4E 20 BF CB 4E 20 BB 08   (Stop)
```

```
          to:  0A C5 06 16 CB 4E 20 06 10 FA C1 C3 8D
               5E C1 C3 48 5E 00 00 00 00 00 00 00 00
               00 00 00 00 00 00 00 00 08   (Stop)
```

5. Optional ZAP to make the BREAK key functional for the 'JKL' screen print option. (Caution: these zaps use DOS RAM areas that appear to be unused, but may turn out to be used by some routine I haven't found yet. Use with caution, and if you develop a problem I would appreciate hearing about it):

```
SYS0/SYS:  00/B9   change:  All zeros (Note -- there is
                            a 01 at 00/B1.  Sector/Byte
                            corresponds to RAM memory
                            location 43A6).

                       to:  F5 E5 3A 7F 38 FE 04 28 08 E1
                            F1 CB 74 23 C3 B2 45 3E 0D CD
                            3B 00 E1 F1 C3 66 45 00 (End)


SYS0/SYS:  02/A9   change:  7E CB 74 23 20
                       to:  7E C3 A6 43 20
```

6. Optional ZAP to the BASIC/CMD file to implement BASIC's unused 'NAME' command. After setting this ZAP, entering 'NAME' from BASIC will cause a jump to the address specified in the NAME jump vector. For example, LOAD RSM2D48/CMD into high memory with an appropriate memory size protect, using either the (4049) entry or answer BASIC's 'MEMORY SIZE?' request. RSM2D's execution address (for a 48K version) is EE94 hex. Make the ZAPs to the BASIC/CMD file in the usual way.

```
BASIC/CMD:  16/72    change:  C3 4A 1E C3
                         to:  C3 94 EE C3
```

With RSM2D (48K version) in high memory, entering 'NAME' will jump to RSM2D. Now here is the interesing part. To get back to BASIC gracefully from RSM2D, enter 'G 0072' (Go to address 72 hex). BASIC's "READY" and its prompt will appear and the resident BASIC program will still be intact! Give you any ideas??

Note: This can also be done with NEWDOS 2.1 or TRSDOS 2.3. Just find the C3 1E 4A (for the 'NAME' function) in the BASIC/CMD file and change it in the same way.

FIRST OFFERING:

TRAKCESS by Roxton Baker

After much ado, we have decided to offer what is perhaps the most powerful utility yet for TRS-80 disk maintenance — TRAKCESS. TRAKCESS was written with special goals:

- To bring under direct control, with as much software assistance as is practical, every capability of the TRS-80's 1771 Floppy Disk Controller (FDC).
- To combine these capabilities into powerful "intelligent" functions whose only limitations are those of the machine in general, or of the FDC in particular.

What does it all mean? Good question. With TRAKCESS, you may:

- Protect diskettes . . . and guess what else? TRAKCESS is perfect for 'breaking down' protected diskettes — now you can back them up!!
- Recover damaged diskettes; read, write, edit, or create ANY track or sector, standard or not!
- Utilize a super 'Electric Pencil' type editing!
- Gain access to previously unaccessible information embedded in diskettes!
- And more . . .

TRAKCESS is **$24.95**, but to make it even more affordable, Roxton has agreed to let you and your friends share the purchase. For every penny you add (up to five), you will be licensed to make one copy of the diskette and documentation! Two of you can buy it for **$12.48 each!** Five of you will pay **just $5.00 each!** This is a special arrangement with particular advantages for the end user in mind. Appreciate it? Make it a success! Order your copy today, from:

THE ALTERNATE SOURCE 1806 Ada Street Lansing, MI 48910
Phone: 517/485-0344

"ENTRY" Program listing, continued from page 23:

```
2950 REM---------------SORT ROUTINE----------------
2960 C=G:C=C+1:E=1:F=1
2970 PRINT:PRINT "***NOW SORTING***":PRINT
2980 C=C/2:C=INT(C)
2990 IF C=0 THEN GOTO 3090
3000 D=G-C
3010 FOR K=1 TO D
3020 E=K+C:W$=W$(E):W=W(E):F=K
3030 IF W$(F)<=W$ THEN GOTO 3060
3040 E=F+C:W$(E)=W$(F):W(E)=W(F):F=F-C
3050 IF F>1 THEN 3030
3060 LET E=F+C:W$(E)=W$:W(E)=W
3070 NEXT K
3080 GOTO 2980
3090 RETURN
3100  FOR I=1 TO 6
3110 INPUT#-1,H$
3120 PRINT H$
3130 NEXT I
3140 END
3150 REM-----PRINT CLASS LIST AND SUMMARY ON LINE PRINTER-----
3160 PRINT "TURN ON LINE PRINTER AND HIT THE 'P' KEY."
3170 LET A$=INKEY$:IF A$="" THEN 3170
3180 IF A$<>"P" THEN RETURN
3190 LPRINT TAB((95-LEN(TN$))/2) TN$ :P=9
3200 LPRINT TAB((95-LEN(D$))/2) D$
3210 LPRINT " "
3220 LPRINT "NUMBER"; TAB(10) "NAME CODE";TAB(25) "RAW SCORE";
3230 LPRINT  TAB(40) "% SCORE"; TAB(60) "GRADE"
3240 LPRINT STRING$(100,"*")
3250 LET P=P+3
3260 FOR I=1 TO G
3270 LPRINT I; TAB(10) SN$(I);
3280 LPRINT TAB(25);USING"###";S(I);
3290 LPRINT TAB(40);USING"###";P(I);
3300 LPRINT TAB(60);USING"#.##";L(I):P=P+1
3310 IF P>50  THEN GOSUB 3550
3320 NEXT I
3330 REM PRINT DESCRIPTIVE STATS
3340 IF P>42 THEN GOSUB 3550
3350  LPRINT " ":LPRINT "SUMMARY"
3360 LPRINT " "
3370 LPRINT "AVERAGE SCORE =";:LPRINT AV,
3380 LPRINT "ST. DEVIATION =";:LPRINT SD
3390 LPRINT "HIGHEST SCORE =";:LPRINT XM,
3400 LPRINT "LOWEST SCORE =";:LPRINT XL
3410 LPRINT "KR-21 RELIABILITY =";:LPRINT R,
3420 LPRINT "ST. ERROR OF MEASUREMENT =";:LPRINT SQR(1-R)*SD
3430 P=P+5:IF P>42 THEN GOSUB 3550
3440 REM PRINT GRADE SCALE
3450 LPRINT " ":LPRINT "GRADE SCALE CALCULATION"
3460 LPRINT " "
```

(continued...)

```
3470 LPRINT "%AV=";X1, "%SD=";S2, "Z'=";ZA, "SZ'=";SZ
3480 LPRINT " "
3490 FOR Z=4.5TO 0 STEP -.5
3500  LETX=(Z-ZA)/SZ*S2+X1:LPRINTTAB(15)Z;TAB(25)"=";TAB(30)X
3510 NEXT Z
3520 LET P=P+15
3530 GOSUB 3550
3540 RETURN
3550 REM END OF PAGE
3560 FOR J=1 TO 72-P :LPRINT " ":NEXT J
3570 LET P=8
3580 RETURN
```

"PENRAM" Program listing, continued from page 48:

```
06070 CONTD1   LD      A,(LSTARW)      ;SEE IF LAST SCROLL WAS
06080          AND     A               ;DUE TO CURSOR UP.
06090          JR      NZ,CONTD2       ;GO IF NOT.
06100          LD      BC,272          ;MUST FIX ADDR.
06110          CALL    AADDR           ;ADD 272 TO IT.
06120 ;
06130 CONTD2   LD      HL,(EDCUR)      ;ADDR IS CORRECT HERE.
06140          LD      (HL),BLANK      ;REMOVE OLD CURSOR.
06150          LD      HL,(ADDR)       ;GET ADDR TO DUMP.
06160          CALL    MAKSTR          ;MAKE DUMP STRING OF IT.
06170 ;
06180          LD      A,15            ;SCROLL BOTTOM 15 LINES
06190          LD      HL,3C40H        ;  UP, MOVING ONLY THE
06200          LD      DE,3C00H        ;  LEFT 55 CHARACTERS.
06210 MVLOPA   LD      BC,37H
06220          LDIR
06230          LD      BC,9
06240          ADD     HL,BC           ;POINT TO NEXT LINE.
06250          EX      DE,HL
06260          ADD     HL,BC           ;AND WHERE IT WILL GO.
06270          EX      DE,HL
06280          DEC     A
06290          JR      NZ,MVLOPA       ;BACK IF 15 NOT DONE.
06300 ;
06310          LD      HL,LINE16       ;POINT TO LAST LINE.
06320          LD      (CURPOS),HL
06330          LD      HL,STRPLC
06340          CALL    OUTSTR          ;PRINT DUMP STRING THERE.
06350          LD      BC,16           ;SUB 16 FROM ADDR, HOMADD
06360          CALL    AADDR
06370          CALL    AHOMAD
```

(continued...)

```
06380          LD      A,1               ;NOTE SCROLL CAUSED BY
06390          LD      (LSTARW),A        ;  DOWN-ARROW.
06400          RET
06410 ;
06420 ;
06430 ; SUBROUTINE MAKSTR EXPECTS TO RECEIVE AN ADDRESS IN HL.
06440 ; IT FORMS THIS ADDRESS AND THE 16 BYTES OF DATA STARTING
06450 ; THERE INTO AN ASCII STRING AT  STRPLC .  IF THE TYPE OF
06460 ; DUMP REQUIRED IS HEX, THEN EACH CHARACTER BLOCK (THERE
06470 ; ARE 16 OF THESE AFTER THE ADDRESS STRING) WILL CONTAIN
06480 ; THE ASCII REPRESENTATION OF THE BYTE IN MEMORY. IF THE
06490 ; TYPE OF DUMP IS ASCII, THEN EACH BLOCK WILL SIMPLY
06500 ; CONTAIN THE ASCII CODE OF THE BYTE IN MEMORY.  SPACES
06510 ; ARE ADDED SO THAT EITHER WAY EACH CHARACTER BLOCK TAKES
06520 ; UP THREE POSITIONS.  A TOTAL OF 64 BYTES IS REQUIRED TO
06530 ; HOLD THIS STRING AND ITS '03' TERMINATOR.
06540 MAKSTR PUSH    HL                ;SAVE ADDRESS.
06550          PUSH    HL
06560          POP     IX                ;PUT IT IN IX.
06570          POP     BC                ;GET MSBYTE OF ADDRESS.
06580          LD      HL,STRPLC+3       ;LAST DIGIT OF ADDRESS.
06590          CALL    BSTRNG            ;BC INTO ASCII AT HL- .
06600 ;
06610          LD      HL,STRPLC+4       ;POINT AFTER ADDRESS.
06620          LD      (HL),':'          ;PUT : AFTER ADDRESS.
06630          INC     HL
06640          LD      (HL),' '          ;PUT SPACE AFTER ADDRESS.
06650 ;
06660          INC     HL                ;POINTS TO 1ST CHAR. BLK.
06670          LD      B,16              ;WILL DO 16 CHAR. BLOCKS.
06680 CHRBLK LD      (HL),' '          ;EACH START WITH BLANK.
06690          INC     HL
06700          LD      C,(IX+0)          ;GET BYTE AT ADDRESS.
06710          LD      A,(HATYPE)        ;CHECK DUMP TYPE.
06720          AND     A
06730          JR      NZ,ACHAR          ;GO IF ASCII DUMP.
06740          CALL    ASCII             ;HEX DUMP REQ'D.
06750          LD      (HL),D            ;FIRST DIGIT OF BYTE.
06760          INC     HL
06770          LD      (HL),E            ;SECOND DIGIT OF BYTE.
06780          JR      CHRDN             ;ONE BLOCK DONE.
06790 ;
06800 ACHAR  CALL    VALASC            ;ASCII DUMP.  CHECK CHAR.
06810          LD      (HL),A            ;PUT FINAL CHAR IN STRING
06820          INC     HL
06830          LD      (HL),' '          ;FOLLOW WITH BLANK.
06840 ;
06850 CHRDN  INC     IX                ;DONE WITH CHAR. BLOCK.
06860          INC     HL                ;POINT TO START NEXT BLK.
06870          DJNZ    CHRBLK            ;DO NEXT ONE.
06880 ;
```

(continued...)

```
06890          LD      (HL),0AAH          ;ONE CHAR. FOR BORDER.
06900          INC     HL
06910           LD      B,8                ;EIGHT CURSOR-ADVANCES
06920 TRALER   LD      (HL),19H           ;AT END RESULT IN STRING
06930          INC     HL                 ;  PRINTING TO END OF
06940          DJNZ    TRALER             ;  LINE MINUS ONE.
06950 ;
06960          LD      (HL),03H           ;TERMINATOR OF STRING.
06970          RET                        ;END OF MAKSTR.
06980 ;
06990 ;
07000 ; SUBROUTINE BSTRNG CONVERTS THE TWO BYTES IN BC INTO AN
07010 ; ASCII STRING OF FOUR HEX DIGITS WHICH IT LOADS INTO
07020 ; HL-3, HL-2, HL-1, HL.
07030 BSTRNG   CALL    ASCII              ;CONVERT BYTE IN C.
07040          LD      (HL),E
07050          DEC     HL
07060          LD      (HL),D
07070           DEC     HL
07080          LD      C,B                ;NOW CONVERT B.
07090          CALL    ASCII
07100          LD      (HL),E
07110          DEC     HL
07120          LD      (HL),D
07130          RET
07140 ;
07150 ;
07160 ; SUBROUTINE ASCII TAKES A BYTE IN C AND RETURNS, IN DE,
07170 ; THE ASCII CODES REPRESENTING ITS TWO DIGITS.
07180 ASCII    LD      A,C
07190          CALL    SBRASC             ;DO FIRST 4 BITS.
07200          LD      E,A
07210          LD      A,C
07220          RRCA                       ;BRING NEXT 4 OVER.
07230          RRCA
07240          RRCA
07250          RRCA
07260          CALL    SBRASC             ;CONVERT THEM TOO.
07270          LD      D,A
07280          RET
07290 ;
07300 SBRASC   AND     0FH                ;CHANGE LOWER 4 BITS OF
07310          OR      30H                ;  A TO ASCII BYTE.
07320          CP      3AH
07330          RET     C
07340          ADD     A,07
07350          RET
07360 ;
07370 ;
07380 ; SUBROUTINE OUTSTR CALLS BYTDIS TO PRINT A STRING
07390 ; POINTED TO BY HL AT THE CURRENT CURSOR POSITION.
07400 ; IT RECOGNIZES 00, 01, 02, OR 03 AS A TERMINATOR.
```

(continued...)

```
07410 OUTSTR  LD      A,(HL)          ;GET CHAR. FROM STRING.
07420         CP      04              ;ONE OF THE TERMINATORS?
07430         RET     C               ;RETURN IF SO (0-3).
07440         CALL    BYTDIS          ;ELSE PRINT IT.
07450         INC     HL
07460         JR      OUTSTR          ;GO DO NEXT BYTE.
07470 ;
07480 ;
07490 ; AADDR - A UTILITY THAT ADDS BC TO ADDR.
07500 AADDR   LD      HL,(ADDR)
07510         ADD     HL,BC
07520         LD      (ADDR),HL
07530         RET
07540 ;
07550 ;
07560 ; AHOMAD - A UTILITY THAT ADDS BC TO HOMADD.
07570 AHOMAD  LD      HL,(HOMADD)
07580         ADD     HL,BC
07590         LD      (HOMADD),HL
07600         RET
07610 ;
07620 ;
07630 ; SADDR - A UTILITY THAT SUBTRACTS BC FROM ADDR.
07640 SADDR   LD      HL,(ADDR)
07650         XOR     A
07660         SBC     HL,BC
07670         LD      (ADDR),HL
07680         RET
07690 ;
07700 ;
07710 ; SHOMAD - A UTILITY THAT SUBTRACTS BC FROM HOMADD.
07720 SHOMAD  LD      HL,(HOMADD)
07730         XOR     A
07740         SBC     HL,BC
07750         LD      (HOMADD),HL
07760         RET
07770 ;
07780 ;- - - - - - - - - - - - - - - - - - - - - - - - - - - -
07790 ;     MESSAGES
07800 ;
07810 INMSG   DEFM    'ADDRESS?'
07820         DEFB    03
07830 ;
07840 PENMSG  DEFM    '-PENRAM-'
07850         DEFB    03
07860 ;
07870 ;
07880         END     STARTP
07890 ;- - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

BULLETIN BOARD

NEW PRODUCTS

DISK KEYPLUS is a powerful collection of utilities that can be turned on or off in just two keystrokes. DISK KEYPLUS supports auto-repeat, lowercase video software, restoration of lost Basic programs, single keystroke user definable strings, Basic shorthand, direct graphic character input, lowercase without shift, and more! DISK KEYPLUS is available for $19.95, non-disk KEYPLUS for $14.95. Write to: SJW, Inc., Box 438, Huntington Valley, PA, 19006 or phone (215) 947-2057.

PROgrammer is an excellent utility for Level II users who are tired of writing programs the hard way: it's a small machine language routine that, once loaded, will provide renumber functions, block moves, appending additional routines from tape, packing of the program, and block deletes. PROgrammer also includes a keyboard debounce routine. For simplified usage, a single keystroke is all that is needed to invoke PROgrammer. Available now from Rational Software, 963 E. California Blvd, Pasadena, CA, 91106, for just $25.00.

TERMCOM hardware allows Level II users to utilize timesharing systems without Radio Shack's Expansion Interface and RS-232 board. The hardware may also be connected to Expansion Interfaces. Modems, serial printers or other peripherals can also be connected to the system. Software includes full paging, scrolling, automatic memory buffer overflow protection, uploading and downloading files from disk and variable loading rates. For tabular material, automatic left or right justification may be specified to keep charts readable. Software is available on cassette or diskette and may be purchased independently of hardware. More information, including complete documentation ($10) is available from STATCOM Corp., 5758 Balcones Drive, Suite 202, Austin, TX, 78731.

SUPER DIRECTORY will allow disk users to simply type "D" and enter to access a menu of commands. Another single keystroke will allow the user to: RUN any program by entering the index number displayed by the program name, even machine language programs. If the program name has a /BAS extension, BASIC will be loaded first. KILL a file by entering the program index number while in KILL mode. FREE, to determine number of free grans and files. PRINT to print out the directory on your parallel printer. SUPER DIRECTORY is a fast acting machine language program for TRSDOS or NEWDOS users, and is available for only $9.95 from Mediamix, Box 8775, Universal City, CA, 91608.

HELP!

"I have not been able to get my Radio Shack screen printer to display material in a machine language environment. It will just take off and print a screen full of garbage. Best I can tell, it loses its line synch and everything gets mixed up royally. If anyone is aware of a program or patch, it sure would be nice!" HELP can be sent to TAS, 1806 Ada Street, Lansing, MI 48910 and will be appreciated.

"I've got one of those pet peeves -- I hate it when my screen shrinks because of a power surge! I've tried several things to take care of the problem, but none have worked. There are several others to try, but I'd go broke. Are there any readers who have tried something that REALLY works?" HELP can be sent to TAS, 1806 Ada Street, Lansing, MI, 48910 and will be appreciated.

PACKER PATCH

TAS recently received notification of an error in Packer's memory size. The label of the 48K version should read MEM SIZE = 61315, then type SYSTEM and respond to the prompt with a /61316. If any users are experiencing problems, this could be the solution.

NETWORK ORDERING

The Alternate Source is now a member of both The Source and MicroNET. We log onto both systems daily to check the mail. Feel free to send software orders, comments, suggestions, complaints, survey responses, etc. via this method. Our MicroNET User ID is 70150,255. To reach us through The Source, mail to TCH565.

LAST MINUTE ODDS & END

The next issue of BTI and TAS are currently under production -- look forward to some real exciting articles, tidbits and discussions. If you're a subscriber, your mailing label will show a letter on the same line as your name. This letter indicates the last issue of TAS that your $9.00 paid for. An "A" means your subscription ends with issue 6, "B" for issue 7, "C" for issue 8, etc. If any subscribers wish to renew their subscription BEFORE it expires, please indicate this in your correspondence to help avoid confusion and double-mailings. Those persons desiring first class mailings of TAS should enclose an extra $3.00 to help cover postage.

AFTERWARD FOR ISSUE 5

Our biggest issue yet! Unfortunately, we have accumulated some very prolific writers! Our title page is looking bare.

A lot of exciting things are happening with this issue: NEWDOS/80, VTOS 4.0 and the Doubler make their appearance. Roxton Baker's article "PENRAM" was one stage in the development of TRAKCESS and Texas was experiencing its worst heat wave in years. Jesse Bob sends his thanks for helping out.

THE ALTERNATE SOURCE

The magazine of advanced applications and software for the TRS-80™

IN THIS ISSUE

Editorial RAMbling...

By Charles Butler

'Other activities' have kept me from being involved with the Alternate Source for the past month or so. In some ways,this has been good. The view obtained from being within something can be substantially different from the view obtained outside the entity. (Hail, Bucky Fuller!) My position from the peripheral of TAS undoubtedly has colored the contents of this issue.

********** ********** **********

We're pleased to announce the winner of the best article for issue #4 -- Allan Moluf's KILLER. Allan wins $25.00 in addition to his remuneration for writing the article. For participating in our contest, Alan Abrahamson, Vaughn Jupe and Ted Kehoe win certificates that are worth $10.00 on anything we sell. Thanks for participating. Don't forget to get votes in for issue five and for this issue. Please put votes on a separate piece of paper and include your complete name and address so we may notify you if you are a winner!

********** ********** **********

Patches are now available for making Percoms new double density board compatible with both VTOS 4.0 and NEWDOS/80. From all indications they have a winner. Many people are purchasing the board in lieu of more disk drives. Unfortunately, our complete shipment was sold out before we received it so we stilldon't have one! Expecting more though, any day now!

********** ********** **********

Regarding our issues on cassette or diskette: We've had a substantial number of problems with loading on both media. Probably most stem from the fact that we intersperse various types of programs -- assembly source and object code, BASIC and even some text. In addition, some authors have requested the right to retain the copyright on their material and prefer that it not be included in magnetic issues. For these two primary reasons, we are going to discontinue offering them. Sorry for any inconvenience this may cause.

********** ********** **********

One gentleman has written Jason Potter about his student rating system in the last issue. He is using the system to rate his employees! Sounds great, and is exactly what we like to see done with good ideas.

********** ********** **********

THROUGH THE INS AND OUTS OF TAPE

Dennis Bathory Kitsz
Roxbury, Vermont 05669

One of the most maligned aspects of the TRS-80 is its cassette loading procedure. Interestingly, it is a lengthy and well thought out piece of coding, a victim of a combination of poor hardware (an inexpensive casette recorder), the inclination personal computer owners have to purchase the least expensive tapes they can find, and the lack of foresight on the part of the engineers designing the routines. But there's no question that with a good tape recorder and reasonable tape, it works well. Here's how.

The routine to read and accept serial information is fairly convoluted, collapsed to about a dozen major CALLs. We will start with the SYSTEM command; since BASIC programs have other bytes to juggle (looking for out of memory errors, etc.), we won't tackle its major routines.

The SYSTEM module

The SYSTEM command is evaluated by the BASIC interpreter, and its control routine is entered at 02B2. If you don't want to know how this command gets to work, then skip right to the tape loading routine two paragraphs below. An initial CALL is executed to DOS link 41E2, which in Level II merely executes a RETurn. The stack is set up at 4288, and another CALL executed to 20FE, which checks the DOS link at 41C1, picks up the "device type" (video, tape, or printer - video at this time), displays a carriage return, checks and saves port FF status (32- or 64-character mode and cassette state), clears the accumulator, and returns. This is preparatory housekeeping.

The accumulator is set up with a star, it is displayed (with more housekeeping), and the line-input routine is CALLed from location 1BB3. This is the same routine used for INPUT statements, and it displays a question mark, evaluates the input line, discards everything after certain punctuation, and returns the evaluated line to the CALLing program. If a BREAK is discovered, the program returns to READY. Spaces, line feeds, tabs, etc., are cleaned out, and a syntax error is declared if no alphanumeric characters are found. If a slash (/) is found, the SYSTEM program jumps past its loading routines, picks up the start address from 40DF (more about that later), cleans out blanks again, and evaluates the string after the slash as an integer (a CALL to 1E5A). The whole business starts over if a non-numeric string is found. If, at last, the program does discover that a number was input, the SYSTEM module is executed from the starting address stored at 40DF.

Build-A-Byte

The first major loading call is to 0293, which searches for a sync byte. Since this will eventually call the "build-a-byte" routine, let's move there first. It begins at 0241; BC and AF registers are saved, then:

```
0243     DB FF          IN      A,(FF)
0245     17             RLA
0246     30 FB          JR      NC,FB
0248     06 41          LD      B,41
024A     10 FE          DJNZ    FE
```

Port FF is checked repeatedly by inputting the value to the accumulator and rotating that value into the carry flag. If no carry is found - i.e., no "one" bit has yet triggered port FF - the program loops back to 0243. Once a bit is found, the B register is loaded with 41, and a "waste time" loop is executed at 024A (a total of just under 500 microseconds). A CALL is then executed to 021E. Let's have a look at that:

```
021E     21 00 FF       LD      HL,FF00
0221     3A 3D 40       LD      A,(403D)
0224     A4             AND     H
0225     B5             OR      L
0226     D3 FF          OUT     (FF),A
0228     32 3D 40       LD      (403D),A
022B     C9             RET
```

This curious subroutine seems to stumble through checking port FF for its video state, then resetting the OUTSIG flip-flop (see the Technical Reference Handbook for details on this circuitry). Isn't a byte ANDed with FF and ORed with 00 merely itself? True enough, but since this is also called as a subroutine entering at 0221, with a different value for HL, the complex AND/OR strategy makes sense.

So at this point we have picked up a bit from tape, delayed, and reset the flip flop, readying it for the next bit to trigger it. Another delay loop follows (over 850 microseconds), and a byte is input to A from port FF:

```
0253     DB FF          IN      A,(FF)
0255     47             LD      B,A
0256     F1             POP     AF
```

```
0257    CB 10          RL      B
0259    17             RLA
025A    F5             PUSH    AF
```

The input byte is saved in the B register, and the previously saved value of A is restored from the stack. Here is a wonderful piece of serial-to-parallel conversion - a sort of software shift register. Bit 7 of port FF was input to A and saved in B, and is then rotated left into the carry flag. Then the accumulator is rotated left, bringing the state of the carry flag into bit 0 of A. The accumulator is then saved onse more on the stack. Another CALL to 021E resets the port FF flip-flop, both registers are restored, and the subroutine returns to the calling program.

You'll notice that at this point we only have one bit saved in the accumulator. An eight-iteration loop would be necessary to create a whole byte...and it will be done. But for the moment let's see how this routine is used in the initial syncing program, which we were about to enter at 0293.

The routine's first action is to CALL 01FE. This is a detailed routine to determine the drive number and other parts of the syntax, the state of port FF (again), select the drive and get it moving. Examining the code will show that it also uses the routine entered at 0221, but with a value of FF04 in HL; this routine won't be covered here, but it is worth looking at it.

The find-sync-byte routine thus turns on the tape, saves the HL register, clears the accumulator, and calls the "build-a-byte" routine at 0241. Since this is the synchronization process, no loop value is specified:

```
0297    AF             XOR     A
0298    CD 41 02       CALL    0241
029B    FE A5          CP      A5
029D    20 F9          JR      NZ,F9
```

It continually seeks bits, endlessly rotating the accumulator until it assembles a serial stream which matches A5 (i.e., binary 10100101 - nice and symmetrical). This routine is so accurate, in fact, that whenever tape motor start-up is not a consideration, the leader consisting of zero bytes would be unnecessary. The leading "1" of A5 serves as a kind of serial "start bit" - and the routine at 0241 handles it from there.

Any kind of a match to sync byte A5 might be found, though, since the serial stream coming in from the tape does not distinguish start and end of byte. For example, the byte pattern "DD 28" also contains an "A5" embedded in it. As a serial stream, DD 28 is -

```
1101110100101000
.....10100101...
```

- with the A5 appearing at the junction of DD and 28. So once the matching A5 is found, a return is executed to the main SYSTEM loading module. That module then CALLs a subroutine at 0235, which is a gussied up bit-reader. BC and HL are saved, then:

```
0237    06 08       LD      B,08
0239    CD 41 02    CALL    0241
023C    10 FB       DJNZ    FB
```

There's the byte read... read a bit with eight iterations. HL and BC registers are restored, and the subroutine returns to the main program.

Loading the Code

The SYSTEM module now compares the byte it created with the value 55, the code assigned to machine-language programs. It loops until it finds that code, then proceeds:

```
02D8    06 06       LD      B,6
02DA    7E          LD      A,(HL)
02DB    B7          OR      A
02DC    28 09       JR      Z,09
02DE    CD 35 02    CALL    0235
02E1    BE          CP      (HL)
02E2    20 ED       JR      NZ,ED
```

Above, the B register is loaded with the number of characters to be found in the SYSTEM program's name. The accumulator is set up with the first character of the name as entered on the *? command line. The accumulator is tested for zero, and skips out of the loop when the end of the entered name is found. Each character following the name is read into the accumulator (CALL 0235) and compared with each letter of the entered name. If at any point the entered name does not match the name on tape, the program goes back to searching for hex value 55 (machine program indicator), and the name search process begins again.

There is a minor flaw in this process. Let's look at the succeeding lines of code:

```
02E4    23          INC     HL
```

```
          02E5     10 F3            DJNZ     F3
```

This coding increments the HL register to the next character and loops back, looking for a total of six letters in the name. But what if the machine program code (55) is found, and one or more characters of the name match, but the rest do not match? There is no provision in this routine to decrement the HL register pair ... which means that, if only part of a correct name has been found, the program will begin its search anew until it finds a program that matches only the *last part* of the entered name! This is the reason the SYSTEM routine is not always able to search until it finds the correct program, the way the BASIC load does.

Let's assume the best - that a machine program was found with the name as entered from the keyboard. A CALL is then made to 022C, where the star or space at 3C3F is toggled (XORed) with 0A. Star XOR 0A is a space, and space XOR 0A is a star; easily done.

The SYSTEM module continues -

```
          02EA     CD 35 02         CALL     0235
          02ED     FE 78            CP       78
          02EF     28 B8            JR       Z,B8
          02F1     FE 3C            CP       3C
          02F3     20 F5            JR       NZ,F5
```

- searching for either 78 (end of program code) or 3C (beginning of data block code). If 78 is found, the program skips back to 02A9, where a CALL is executed to 0314. This subroutine merely reads the last two bytes on tape into the HL register, preparing the start address. This is saved at 40DF, the cassette recorder is turned off (CALL 01F8), and the SYSTEM module is re-entered *from the start* at 02B2. This module is a continuous loop, allowing a group of machine language programs to be entered sequentially. Only the presence of the slash-start address combination will break out of the loop.

If a 3C is found, the beginning of a block of machine code is assumed. (If neither is found, the program loops until it finds one or the other). Here's a snippet of code:

```
          02F5     CD 35 02         CALL     0235
          02F8     47               LD       B,A
          02F9     CD 14 03         CALL     0314
          02FC     85               ADD      A,L
          02FD     4F               LD       C,A
```

A byte is read and saved in B. At 0314, two bytes are read and saved, respectively, in the HL register pair. These three bytes are, first, the number of bytes to read, and second, the two-byte starting address of the block. The 0314 subroutine leaves the value transferred to H in the accumulator; to it is added the value in L, and this number, sans carry value, is saved in the C register. The C register will be used to calculate the checksum for the block being read.

Curious Checksum

Each succeeding byte is read from tape and placed at the address now specified by HL. That byte is also added to the C register to update the simple checksum. HL is incremented to the next contiguous address, and the loop is iterated until B (the number of bytes to read in the block) reaches zero.

When the block is fully read, another byte is read from tape. This is the checksum byte, and should match the last updated value in the C register. If it does match, the program loops back, toggles the star, and begins the search for end-of-program (78) or block header (3C) anew.

A correct checksum byte, curiously enough, is not a necessary element of the SYSTEM module. If the checksum is incorrect, the program will display a "C" at video location 3C3E, and loop back regardless to continue reading the program from tape. I first noticed this action when a gentleman from New Hampshire called; he had been using my tape-duplication routines to make a corrected copy of a machine language program. He had loaded the tape, returned to BASIC, then POKEd in a few byte changes. He then continued with the duplication. When he loaded the tape later on, he got a "C" error message on the screen ... but the program continued to load and did execute properly. The checksum was wrong because of the byte changes he had made, but the program, checksum notwithstanding, was read and loaded completely.

Let's take a look at that final portion of code:

```
02FE    CD 35 02     CALL    0235
0301    77           LD      (HL),A
0302    23           INC     HL
0303    81           ADD     A,C
0304    4F           LD      C,A
0305    10 F7        DJNZ    F7
0307    CD 35 02     CALL    0235
030A    B9           CP      C
030B    28 DA        JR      Z,DA
030D    3E 43        LD      A,43
```

```
030F    32 3E 3C        LD      (3C3E),A
0312    18 D6           JR      D6
```

Overall, these routines give the appearance of being reasonable and reliable, and they should be. What, then, gives rise to the tape problems? Mostly the timing loop in the 0235/0241 subroutine. The values placed in the B register at 0248 and 024F are too short for low-grade audio processing. Details of the audio processing will be presented in an upcoming issue of 80 Microcomputing (forgive me, Charley), but simply stated, the audio waveform coming in from tape "rises" too slowly for the fast bit-check loop at 0251 to catch. A "one" might come through, but it comes through too laggardly for port FF to have flipped into place.

Special Loaders

This was initially one of the mysteries of TRS-80 operation. Microchess was produced with a loader, then others quickly followed, mysteriously taking control of the machine and locking it up completely.

Let's now take a look at some of these special loaders, which will be designated Loaders A, B, C, D, and E in order to help then continue to do the job they were supposed to - protect software.

Loader A sets up a stack at 5000, clears the accumulator, and calls ROM to turn on the tape recorder and find the sync byte. It places a star on the bottom of the screen, sets up the HL register to receive the program, and prepares register C to perform simple checksum. A byte is read, it is saved in memory, and the checksum is created as in the SYSTEM mode. Then:

```
4D25    7C              LD      A,H
4D26    1F              RRA
4D27    23              INC     HL
4D28    3E 2A           LD      A,2A
4D2A    DA 2F 4D        JP      C,4D2F
4D2D    3E 20           LD      A,20
4D2F    32 FD 3F        LD      (3FFD),A
4D32    3E 4C           LD      A,4C
4D34    BC              CP      H
4D35    C2 1F 4D        JP      NZ,4D1F
4D38    3E FF           LD      A,FF
4D3A    BD              CP      L
4D3B    C2 1F 4D        JP      NZ,4D1F
4D3E    B9              CP      C
4D3F    C2 00 00        JP      NZ,0000
```

```
4D42    CD F8 01        CALL    01F8
4D45    C3 80 47        JP      4780
```

The strange appearance of RRA has nothing to do with rotating incoming bits. Rather, since the accumulator contains the H register value, each page (256 bytes) of information will change the high page value by one. Consequently, the high page will alternate between odd and even values, and the least significant bit, rotated into the carry flag, will trigger the display-star or display-space routines at 4D2F.

Finally, this somewhat awkward loader does a pair of compares to see if it has yet reached 4CFF, the end of the program load. If not, it loops back and continues; if so, it examines the checksum in C. Amazingly enough, it goes back to MEMORY SIZE? if there is a checksum error! There's no tampering with this program. A successful load is followed by a jump to the program's beginning at 4780.

Loader B is virtually identical to Loader A, except that the beginning of the program is found at 41FD instead of 4780.

Loader C is of a more interesting variety. It is written entirely without calls to ROM, because it is capable of loading into a Level I or Level II TRS-80. Less fortunately, the ROM timing errors are not corrected, so the chances of loading this program on a marginal machine are not at all improved. The stack is prepared, and a block of memory is cleared from 5800 to the end of potential RAM at FFFF. My only guess as to the reason for this is that the authors wish to wipe out any programs such as monitors or disassemblers, as the clearing byte (A5) does not strike me as otherwise meaningful.

The tape is then turned on, and a pattern of three asymmetrical and two symmetrical sync bytes is found (B1, 83, 79, 5A, 00). Again, the choice strikes me as arbitrary, and may be the authors' way of identifying their own code. If these bytes are found, the program continues; if not, the entire five-byte pattern is sought again.

As in the other loaders, register C is set to zero for use as a checksum byte. The program load point is set high in memory (747F), and a byte is read. Here is a part of the code:

```
433D    CD 8F 43        CALL    438F
4340    77              LD      (HL),A
4341    32 3F 3C        LD      (3C3F),A
4344    81              ADD     A,C
```

```
4345    4F                  LD      C,A
4346    2B                  DEC     HL
4347    7D                  LD      A,L
4348    3C                  INC     A
4349    C2 53 43            JP      NZ,4353
434C    CD 8F 43            CALL    438F
434F    B9                  CP      C
4350    C2 66 43            JP      NZ,4366
```

The secret to this portion of code rests in address 4346. Unlike most other loaders, this one loads (and displays) the *last* byte of code first, moving backwards through memory. (438F is the location of the byte read subroutine). When the page is crossed (4346-4348), the checksum is evaluated; if the checksum is incorrect the program jumps to 4366, where an error message is displayed and the machine locks up.

The user's display is worth noting:

```
4353    7C                  LD      A,H
4354    32 3E 3C            LD      (3C3E),A
```

This loader actually displays the ASCII equivalent of the page of memory being loaded with data ... and it looks like an alphanumeric countdown as the program is fit into place.

Finally, Loader C does a comparison for the end of the first major load block, changes the value of H, and loads the next block. It then overwrites critical portions of the load routine, effectively obscuring the loading and entry point of the program. Interrupts are disabled, and the process moves out of the loader into the main program. Interestingly, the authors forgot to turn the tape recorder off.

Loader D's byte read is RAM-based to correct the timing errors in the original ROMs. (The new ROMs, by the way, have corrected these problems, the engineers finally admitting what many of us had claimed all along - that it wasn't the fault of "the other brand" of tape instead of expensive "certified" cassettes.) This altered code looks instead like this:

```
4337    DB FF               IN      A,(FF)
4339    17                  RLA
433A    30 F6               JR      NC,F6
433C    06 41               LD      B,41
433E    10 FE               DJNZ    FE
4340    CD 1E 02            CALL    021E
4343    06 50               LD      B,50
4345    10 FE               DJNZ    FE
```

```
4347     05 14          LD       B,14
4349     DB FF          IN       A,(FF)
434B     10 FC          DJNZ     FC
434D     47             LD       B,A
434E     F1             POP      AF
434F     CB 10          RL       B
4351     17             RLA
4352     F5             PUSH     AF
```

A comparison of this with the Microsoft loader in ROM reveals not only a change in the timings (at Loader D 4343, ROM 024F), but also a loop where port FF is loaded redundantly into A twenty times (4347-434B). The timing for this (as contrasted with the timing in ROM at 024F to 0253) is about 900 microseconds, a slightly longer wait for the audio tape to trigger the INSIG flip-flop. The presence of all the input data on the screen is psychologically reassuring.

Finally, Loader E is of an entirely different sort. First, some code:

```
BEFE     3E 04          LD       A,4
BF00     D3 FF          OUT      (FF),A
BF02     DB FF          IN       A,(FF)
BF04     17             RLA
BF05     30 FB          JR       NC,FB
BF07     06 XX          LD       B,XX
BF09     10 FE          DJNZ     FE
BF0B     06 09          LD       B,9
BF0D     3E 04          LD       A,4
BF0F     D3 FF          OUT      (FF),A
BF11     DB FF          IN       A,(FF)
BF13     17             RLA
BF14     00             NOP
BF15     38 0C          JR       C,0C
BF17     23             INC      HL
BF18     2B             DEC      HL
BF19     10 F6          DJNZ     F6
```

This remarkable loader is written for high-speed operation, setting up the output ports (BEFE and BF0D), clocking itself with start bits (BF0D), and then reading a nine-bit serial stream. Careful timing and self-clocking are essential in high-speed data I/O, and this routine is capable of reading and writing on ordinary audio cassettes, with excellent reliability, at better than 2000 baud. The only point to the instructions at BF17 and BF18, for example, is the delay introduced by executing them; yet that timing is very important. The actual timing value at BF07 has been dropped for a measure of protection of this author's fine software.

Conclusion

In sum, the tape read/write routines for the TRS-80 are efficient and, especially now with special loaders and a corrected ROM, quite reliable. Different levels of user prompts, particularly those used by the reverse-loading module described above, are probably more satisfactory than flashing stars. A checksum process for BASIC similar to the SYSTEM module would have been valuable. Finally, by careful attention to clocking details, a reliable higher-speed loader could have been included in the TRS-80.

For those especially interested in high-speed loaders, I recommend examining the Exatron Stringy-Floppy operating system, which shows what can be done with equipment designed for digital operation. It is capable of reliable loading and saving at rates exceeding 11,000 baud.

With the exception of Loader D (it is mine), the special loaders presented are from popular commercial programs. A bouquet of used typewriter ribbons and cassette tape, plus a free copy of Keepit 3.1, to the first reader who can identify the programs using those special loaders.

Leftovers

In the last issue I promised some improvements to the TRS-80 keyboard scan, including an updated debounce program to keep up with fast typists. Those changes will appear next time, along with some polemics about relocatable programs and the validity of providing source code with commercial programs. I would appreciate comments from any readers on the advantages and disadvantages to both users and vendors who provide source code for their programs. Address correspondence to me at Roxbury, VT 05669.

With B17 Tape Operating System (ver.2) you can save/load Basic or System tapes almost 4X faster. Save/load Data Arrays 165X faster. Has Automatic Preloader to let you easily use it. Load-&-Go System programs are standard. You can back-up your System tapes with this feature. It comes with a 20-page *easy-to-understand* Operating Instruction manual. Licensing B17 preloader gives your programs being sold *copy-proof protection* plus an edge on the competition with a program which loads much faster. B17 comes with a 15-day UNCONDITIONAL money-back guarantee for only $20 + $3 first-class postage & handling. Check or Money Order only. We also Mailorder new hardware, etc. which saves you $. 40-track 5" drive (bare) is under $255. Many other items at affordable prices. SASE brings free information.

ABS SUPPLIERS
POST OFFICE BOX 8297
ANN ARBOR, MICHIGAN 48107
(313) 971-1404

ACCEL: from England, a compiler for Level II TRS-80 BASIC. Compiles integer statements and functions to fast Z80 code, resolves dictionary search at compile time, more. Graphics can be 3000% faster $44.95

ALLEN GELDER SOFTWARE
Box 11721 Main Post Office
San Francisco, CA 94101

TRS 80 tm Radio Shack/Tandy Corp

$24.95 on diskette

2 drives & 48k Req.

THE ALTERNATE SOURCE

TAS PRESENTS . . .

TRAKCESS

Written By: Roxton Baker

DOS USERS LOOK HERE

After much ado, we have decided to offer what is perhaps *The Most Powerful Utility Yet For TRS-80 Disk Maintenance. . .*

TRAKCESS was written with two very special goals:

A. To bring under direct user control, *almost every practical capability* of the 1771 Floppy Disk Controller.

B. To combine these capabilities into super powerful *'intelligent'* functions whose *only limitations are those of the user or machine.*

'What does it all mean?' You ask, Good Question. . .

WITH **TRAKCESS**, YOU MAY:

- Protect your own diskettes with special format variations *you create!*
- Utilize a potent 'pencil type'editor.
- Recover damaged diskettes; read, write, edit or create any track or sector, *standard or not!*
- **TRAKCESS** is capable of duplcation of any sector, of any track including *PROTECTED DISKETTES!*
- Smart DUPLICATE command analyzes every track! (2 drives)
- *PLUS MUCH, MUCH MORE*

*** * YOU CAN LEGALLY MAKE COPIES * ***

TRAKCESS IS $24.95. For every penny you add (up to five) you will be *given a limited licence* to make one copy of the diskette and documentation. This is the first in a new line of software to be marketed in this manner. The intent of this concept is to allow you and your friends to share the cost of adding to your software library.

This is a new concept. Please comment on it.

517/485-0344 .... *517/487-3358*

☐ *YES I'LL BYTE!! Send me trakcess for $24.95 plus ___ pennies for my licence to make copies. I've included .75¢for shipping.*

☐ *My check's enclosed.* *Please bill my MC card* ☐
☐ *Please send COD* *Please bill my Visa card* ☐

THE ALTERNATE SOURCE
1806 ADA STREET
LANSING, MI 48910

NAME ________________
ADDRESS ________________
CITY, ST, ZIP ________________
CARD # ____________ **EXP.** ___

ANOTHER WAY TO INSTALL MACHINE LANGUAGE PROGRAMS

By Jesse Bob Overholt

One night a few weeks ago some of the boys and I were sitting on the back porch of the Circle J Ranch house sipping Dr. Peppers and listening to Willie Nelson on the radio. The topic of conversation was simple ways to load short machine language programs while in BASIC. Billy Fred said his favorite method was to code the program in DATA statements and then POKE it into protected RAM. Buck and I maintained that the best way was to POKE it into a dummy string, a method previously described in The Alternate Source. Then Roy, the Weird Wrangler, spoke up.

We call ol' Roy the Weird Wrangler because ever since he got kicked in the head by a 48 bit word, his pilot light's been out. "The best way,", Roy said, "is to use the old Phantom Line Trick." Needless to say, none of us had ever heard of that one. It took us most of the night and another six quarts of Dr. Pepper to get him to explain, but we did it and here it is.

To understand how to use the Phantom Line Trick, it is helpful to know how BASIC program lines are stored in RAM. Each line of a program has a four-byte header in front of it. The first two bytes are the address in RAM of the next line of the program. Following this is the line number in two bytes as a binary number. Then comes the rest of the line, pretty much as typed in except that all BASIC keywords have been converted into a single-byte "token". A binary zero marks the end of the line. By following the chain of pointers (known to buzzword freaks as a "linked list") BASIC can rapidly locate line numbers referred to by the user program. A special pointer at 40A4-5 (16548-9 decimal) is the head of the chain and always points to the first line in the program. The last line in the program is followed by an end-of-program mark consisting of three zero bytes. To get a better idea how all this works together, let's look at a simple example. Consider the following program which, incidentally, was the one that Lucretia Grunge, our local school marm', used to teach us kids how to count:

```
10 FOR I = 1 TO 10
20 PRINT I
30 NEXT I
```

Now let's pry the lids off those 4116's and see how this program looks in memory. Since it is really hard to get much detail by looking into a memory chip, the boys and I have taken Crayolas in hand and produced the following diagram (note that the contents of 40A4-5 may vary with different DOS's, number of files, etc.):

```
        ADDRESS                  CONTENTS

         40A4                      BA 68
                          I------I
                          I
         68BA             I--->C8 68 0A 00 01 20 49 D5 31 20
                      I----------I
                      I
         68C8         I------->D0 68 14 00 B2 20 49 00
                          I------I
                          I
         68D0             I--->D8 68 1E 00 87 20 49 00
                      I----------I
                      I
         68D8         I------->00 00 00
```

It was at about this point in the discussion that Willy Bob nodded off and rocked off the porch. Don't let that happen to you! By following the diagram (graphics by Harlow "Wrong-way" Nurn, our local surveyor), it is easy to see how the program lines link together. All addresses in the chain are in the standard Z-80 format of least significant byte first. Notice that the key to this whole scheme is the head of the chain at 40A4. Without it, BASIC would be unable to find any of the program. It is here that we can work our magic. By changing the head of the chain, we can remove one or more links. If we remove only the first line, the diagram will look like this:

```
        ADDRESS                  CONTENTS

         40A4                    C8 68
                          I------I
                          I
         68BA             I      C8 68 0A 00 01 20 49 D5 31 20
                          I            BD 31 30 00
                          I
         68C8             I--->D0 68 14 00 B2 20 49 00
                      I----------I
                      I
         68D0         I-------->D8 68 1E 00 07 20 49 00
                          I-------I
                          I
         68D8             I---->00 00 00
```

Notice how the line at 68BA has been "cut free". That's the trick! All of the RAM from 68BA to 68C7 is now protected from use by BASIC and is available for use by the user. By changing the size of the first line we can control the amount of RAM to be allocated. A REM statement is ideal for this purpose, since it will take up as many bytes as there are characters following 'REM' plus six. PEEK and POKE provide the tools for manipulating the program. Try the following example:

```
10 REM - THIS LINE WILL SOON DISAPPEAR!
20 LIST
30 L%=PEEK(16548):H%=PEEK(16549):A%=L%+H%*256
40 POKE 16548,PEEK(A%):POKE 16549,PEEK(A%+1)
50 LIST
```

Key this program in and RUN it. It will LIST, showing five lines. Then it will LIST again, but only four lines (20-50) are listed on the screen. Line 10 has vanished into thin air! Variable A% now points to the memory area occupied by line 10, and may be used to POKE in a machine language program.

There are several advantages to this method. First, it does not require any memory to be reserved via the "MEMORY SIZE" question and is, therefore, less error-prone to use. Second, it is address independent and will never get in the way of any other machine language programs that a user might need, such as drivers, etc. Finally, the memory allocation can be accomplished quickly and efficiently with a few lines in BASIC.

However, as Ptomaine Tony, our ranch cook, likes to say, "Every biscuit has two sides." (I'm not sure what that means, since Tony's biscuits are a lot like diskettes -- black and flat.) But the Phantom Program Line Trick does have a couple of drawbacks. First, it works best on short programs. Long ones will take forever to POKE in and require too many DATA statements. Another big disadvantage is that it requires the machine language program to be position-independent. That's a $4 computer expression which means a program which can run correctly anywhere in memory. To do this the assembly language programmer must either play some complex address-manipulation games or avoid direct references to addresses within his program. (An article on some of these techniques is forthcoming.)

Well, there it is. Should you use it? You'll just have to weigh the pros and cons and decide for yourself. In the meantime, if you want to play with the technique, the boys and I have whomped up a little program for you. It's a hexadecimal-decimal converter that converts hex to decimal and vice-versa. While the program is written for Disk Basic, Level II users not yet blessed with disk can still use it by changing it where marked. This program loads the converter and installs it as USR9. When the function is called with a string argument containing a hex number, it returns a decimal result. If called with a decimal integer argument it returns a string argument containing a hex equivalent. No error checking is done in hex to decimal conversion -- decimal numbers greater than 32767 must be entered as negative numbers for proper conversion. This may be accomplished by subtracting 65536 from the number. Hex numbers from 0000 to FFFF are returned as decimal negative numbers -32768 to -1. To get the true decimal value, add the negative number to 65536.

License-free use of this program is granted to any individual who promises to answer every "Where'd ya get that?" question loudly and clearly with "I got it from Jesse Bob and The Alternate Source." Violators will be restricted to use of Level I BASIC for 90 days.

Jesse Bob's FREE Example Program

```
10 'HEXADEC BY THE CIRCLE J SOFTWARE RANCH
     * CONVERTS HEXADECIMAL TO DECIMAL
     * CONVERTS DECIMAL TO HEXADECIMAL
     * INSTALLED AS USR9 (USR FOR LEVEL II USE)
     * WRITTEN IN MACHINE LANGUAGE FOR SPEED
20 CLEAR 500
:  CLS
:  PRINT "NOW INSTALLING 'HEXADEC'"
:  PRINT "BY THE CIRCLE J SOFTWARE RANCH"
30 L%=PEEK(16548)
:  H%=PEEK(16549)
:  A%=L%+H%*256
:  POKE 16548,PEEK(A%)
:  POKE 16549,PEEK(A%+1)
40 FOR I%=0 TO 102
:      READ D%
:      POKE A%+I%,D%
:  NEXT I%
:  PRINT "'HEXADEC' IS INSTALLED!"
:  PRINT
50 DEFUSR9=A%              <------- Disk BASIC only
50 POKE 16526,L%           <------- Level II only
:  POKE 16527,H%
60 DATA 231,40,54,205,127,10,62,4,
        205,87,40,33,33,65,71,175
70 DATA 237,111,35,237,111,43,246,48,
        254,58,56,2,198,7,18,19
80 DATA 16,237,62,3,50,175,64,79,
        42,179,64,34,33,65,235,33
90 DATA 211,64,237,176,235,34,179,64,
        201,42,33,65,70,35,94,35
```

```
100 DATA 86,33,33,65,54,0,35,54,
         0,26,214,48,56,19,254,10          110          DATA
56,6,214,7,254,16,48,9,
         43,237,111,35,237,111,19,16        120          DATA
232,62,2,50,175,64,201
```

Examples of Use

```
PRINT USR9("FF")            <-----Displays 255

PRINT USR9(255)             <-----Displays 00FF

A$=USR9(-1)                 <-----A$ contains FFFF

AD%=USR9("BAD #")           <-----AD% contains 2989

Z=USR9("DEAF")              <-----Z contains -8529

PRINT USR9(64000-65535)     <-----Displays FA00

PRINT USR9("C000")+65535    <-----Displays 49152
```

Save programming time and effort —

with PROgrammer

• PACK • RENUMBER

• APPEND • MOVE

Powerful, easy-to-use programming tools
for Level II Basic users from:

Rational Software

963 East California Blvd., Pasadena, California 91106

Cassette, $25. California residents add 6% ($1.50) sales tax.
Specify the memory size of your computer when ordering.

DIRBASIC PROGRAM

By Bruce G. Hansen

This article describes a machine language program to print directories and free space of diskettes while in BASIC or DOS.

This program is self-relocating. For example, no modifications need be done on the program for it to work with a 16K, 32K or 48K machine. The program moves itself up to high memory.

Another feature is that it is completely compatible with other machine language programs you may wish to use. While under DOS, just load any other routines and then load DIRBASIC (one important note - DIRBASIC must be the last program loaded or it will not work.)

(I couldn't get past this point without wondering what would happen if I wanted to use TWO relocatable programs. So I had to try it. First, I loaded my lower case driver program. It told me I had to protect memory at 65110. Then I executed the BOSS program, which is relocatable. It asked what memory size I would like to protect, so I responded with the above figure. BOSS then told me to protect memory at 62310. But I also wanted to have DIRBASIC loaded and ready, so I executed it. It responded by telling me to answer the memory size question with 61459. Thinking that everything had gone smoothly, I proceeded to venture into Basic, and reserved memory at 61459, as instructed. I then attempted to get a directory listing of both drives zero and one, and encountered no problems. So I loaded a Basic program, with the intention of single-stepping through it. BOSS, however, could not be summoned. A second attempt at getting a directory of drive zero also failed. Hmmm. Now that authors are being considerate enough to provide us with relocatable code, it looks like our next problem will be figuring out how to make relocatable programs compatible with each other! Any takers? jmk)

EXECUTING DIRBASIC

The instructions for DIRBASIC are:

From DOS type in:

```
DIRBASIC
```

With NEWDOS 2.1 and TRSDOS 2.1 a memory size will need to be set if going to BASIC. The program will figure the memory size question for you and print it on the screen when the program is first loaded. With TRSDOS 2.2 and 2.3 no memory size need be set when going to BASIC, as the program protects itself. To display a directory and free space of a diskette type in:

SHIFT, UP ARROW and the drive number holding the target diskette, simultaneously.

Since the SHIFT UP ARROW combination is used by NEWDOS to print the first line of a program, the ELECTRIC PENCIL control key can be depressed instead of the SHIFT UP ARROW combination provided the proper hardware modification has been installed. For example, hitting SHIFT, UP ARROW and the 0 (zero) key will display the directory and free space of drive number 0 (as will the PENCIL control key and the 0 key).

The program does use some ROM routines to do data conversions from integer format to single precision format, along with some ROM arithmetic functions. I refer you to the book INSIDE LEVEL II (published by MumfordMicro and available through TAS) for more information on that subject.

I would suggest not typing in DIRBASIC twice without pushing the reset button. The program will still work fine but the second load of DIRBASIC will put the program below the first one, thus wasting memory.

PROGRAM LOGIC

When "DIRBASIC" is entered from DOS it loads at 5200H. The first part of DIRBASIC, the relocatable loader and patch routine, then take over. The program first changes the memory size to accommodate DIRBASIC. Then all addresses of jumps and calls are changed to their new high memory value. Next, the keyboard driver address is saved and then replaced with the starting address of DIRBASIC. Now whenever the computer asks for keyboard input, it passes through DIRBASIC first. The last function of the loader part of the program is to display the memory size required if going to BASIC. After that is displayed, the program returns to DOS. DIRBASIC is executed every time a keyboard response is inquired by the computer. The registers are saved and a check is made to see if the SHIFT key or ELECTRIC PENCIL control key is depressed. If neither are depressed, the program restores the registers and jumps to the previously saved keyboard driver address. If the SHIFT (and UP ARROW) key or EP control key is depressed then a check is made for which number key, if any, are pushed. If none are pushed, the program jumps to the keyboard driver routine. If a correct key combination occurred then the drive select bytes at 4308H and 4309H are set. Next the current cursor position is saved.

The cursor position is saved since a SHIFTED BACK SPACE is sent out after the free space and directory are printed. This is done because the SHIFT, UP ARROW, ZERO key combination is sometimes picked up as a valid keyboard input. For example, if you typed in "RU" from BASIC and typed SHIFT, UP ARROW and ZERO to jump to DIRBASIC, the key combination required for DIRBASIC might be tacked onto the "RU" typed in from BASIC. This problem is taken care of by saving the cursor position, restoring the horizontal cursor position after DIRBASIC is through and returning a SHIFTED BACK SPACE when exiting DIRBASIC.

The next step is to determine if the selected drive is available. The call to CKDRV does this by playing with the floppy disk controller chip registers. Look at the program listing for more information.

If the selected drive is a valid one, the GAT sector (track 11H, sector 0) is read in via a call to 46DDH. The free space of that drive is found by summing the number of grans currently not allocated. The drive number, name and date are displayed followed by the number of free grans. All directory sectors are then read in, one at a time. Each is checked for existing files. DIRBASIC is normally displayed as a "DIR (I)". Refer to the program listing for instructions on how to change this if desired.

The first byte of a directory entry will be greater than 0FH and less than 0FFH if that entry holds an existing file. If the first byte of the existing file entry is greater than 7FH, the entry is a FXDE (File's extended directory entry).

After all appropriate file names have been displayed, the cursor horizontal cursor position is restored and a character 18H is sent back to the calling program (18H is a SHIFTED BACK SPACE).

Some of the routines used in DIRBASIC can be useful in other programs. They are divided into sections.

DOS CALLS

4409H - DSKERR
(Display disk error message)

This routine will display the DOS error stored in the A register. Whenever a disk I/O routine is called the Z flag is set if no error occurred. If there was an error, the error code is in the A register on return from the call. See chapter 6 of the TRSDOS manual for a list of error codes.

This call will normally return to DOS. To inhibit this and return to the calling program instead, OR the A register with 80H before the call.

This routine will normally print an extended error message when called. To stop this extra information from being printed, OR the A register with 40H before the call. If the extended error message is desired, make sure the file is closed or the message will be unreadable.

4308H - DSKNMB
(Current drive number)

This byte contains the current drive number. It should contain a 0 for drive 0, 1 for drive 1, etc.

4309H - DRVBIT
(Current drive select bit)

This byte also contains the current drive number . The byte here has the same bit set as the drive number. Bit 0 is set for drive 0, bit 1 set for drive 1, etc. The resulting bytes are 1 for drive 0, 2 for drive 1, 4 for drive 2 and 8 for drive 3.

4467H - DSPMSG
(Display a message)

This is a very useful DOS routine which displays a message on the screen.

To use this routine load the HL register pair with the starting address of the message to be printed. The message must be terminated by a 03H byte or a carriage return (0D byte). Then CALL 4467H. For the message to have embedded carriage returns use a 0A in place of a 0D. This way the DSPMSG routine can print on many lines from just one call.

402DH - DOSJMP
(Jump back to DOS)

This is not a CALL, but a JUMP address.

To return to the DOS READY state, do a JP 402DH. This is the normal return address. The abnormal or error return address is 4030H.

46DDH - DSKRD
(Read a disk sector)

This is an extremely useful DOS routine to read any disk sector.

To use this call, load the HL register pair with the starting address of a 256 byte file buffer, the D register with the track number to be read, the E register with the sector number to be read, the C register with the drive number to be read and the B register with the logical record length.

The bytes at 4308H and 4309H must also be set before reading a sector as described above.

To use the routine CALL 46DDH.

If no error occurs, the Z flag will be set. If an error occurs, the A register will contain the error code. If a directory sector was read in, an error will occur and the error code will be 6. A check should be made when reading in a directory sector.

ROM CALLS

033AH - WRCHR
(Display a character on the screen)

This ROM routine displays the byte in the A register on the screen at the current cursor position.

To use this routine place the byte to be printed in the A register and CALL 033AH. This routine is an upgrade from the display routine at 0033H. The 0033H routine destroys the DE register pair when displaying, while the 033AH call leaves all registers in tact.

OACCH - ITOS
(Convert integer data format to single precision)

This routine will take the integer number stored at 4121-4122H and convert it to the TRS-80's single precision format. There are also other types of conversions in ROM. I again refer you to the book INSIDE LEVEL II for more information on using ROM calls for arithmetic purposes.

OAOCH - SNGCP
(Compare two single precision numbers)

This routine will compare two numbers stored in the single precision format. The first number is stored at 4121-4124H and the second in the BCDE register quad.

This call does not affect the two numbers in question, only the Z-80 flags. If the numbers are equal, the Z flag is set. If the first number is smaller, the S and C flags will be cleared.

0716H - SNGADD
(Add two single precision numbers)

This routine will add two numbers stored in the single precision format. The first number is stored at 4121-4124H and the second in the BCDE register quad. After the call the result is stored at 4121-4124H.

Care must be taken so an overflow does not occur. This results in control being sent to the LEVEL II monitor.

OFBDH - SNGSTR
(Convert a single precision number to a string of characters)

This routine will convert a number stored in the single precision format to a string of ASCII characters. The number to be converted must be stored at 4121-4124H. After the call the string of characters starts at 4130H and ends with a 00H byte.

FURTHER COMMENTS

I suggest reading the comments on the program listing for a more in-depth look at how the program works.

DIRBASIC may just become one of your most useful utilities!

<PROGRAM LISTING BEGINS ON PAGE 26>

```
                ;-----------------------------------------------------------
                ;
                ;       DIRBASIC        VERSION 1.87  SEP 24,1980  22:43
                ;
                ;               BY BRUCE G. HANSEN
                ;
                ;THIS PROGRAM WILL PRINT A DIRECTORY OF ANY DRIVE BY
                ;SIMULTANEOUSLY HITTING THE SHIFT KEY, THE UP ARROW KEY AND
                ;THE NUMBER OF THE DRIVE TO PRINT THE DIRECTORY (0-3).
                ;(OR HIT THE ELECTRIC PENCIL CONTROL KEY INSTEAD OF THE SHIFT
                ;UP ARROW COMBINATION).
                ;
                ;THE PROGRAM WILL DISPLAY THE DRIVE NUMBER, NAME, DATE, AND
                ;FREE GRANS OF THE SELECTED DRIVE.
                ;
                ;THE PROGRAM IS COMPATIBLE WITH DVR AND OTHER SUCH TYPES
                ;OF PROGRAMS IF "DIRBASIC" IS LOADED LAST.
                ;
                ;
                ;-----------------------------------------------------------
                ;
                ;
5200                    ORG     5200H           ;LOAD PROGRAM HERE
                ;
                ;
                ;       DOS ROUTINES
                ;
                ;
4409            DSKERR  EQU     4409H           ;DOS ERROR PRINTING ROUTINE
4308            CURDRV  EQU     4308H           ;CURRENT DRIVE BYTE
4309            DRVBIT  EQU     4309H           ;DRIVE SELECT BIT
4467            DSPMSG  EQU     4467H           ;DOS ROUTINE TO DISPLAY A MESSAGE
402D            DOSJMP  EQU     402DH           ;JUMP ADDRESS FOR A RETURN TO DOS
4030            DOSERR  EQU     4030H           ;RETURN TO DOS AFTER ERROR
46DD            DSKRD   EQU     46DDH           ;READ A DISK SECTOR: HL->BUFFER
                                                ;D->TRACK#, E->SECTOR#, B->LRL
                                                ;C->DRIVE#
                ;
                ;
                ;       MISCELLANEOUS EQUATES
                ;
                ;
4049            TOPMEM  EQU     4049H           ;TOP OF MEMORY
000D            CR      EQU     0DH             ;CARRIAGE RETURN
4121            ARTHBF  EQU     4121H           ;RAM BUFFER USED BY LEVEL II ROM
                                                ;TO DO ARITHMETIC IN
40AF            TYPNUM  EQU     40AFH           ;TYPE OF NUMBER ROM IS TO PERFORM
                                                ;AN ARITHMETIC OPERATION ON.
                                                ;2 FOR INTEGER, 4 FOR SINGLE
                                                ;PRECISION, 8 FOR DOUBLE AND
                                                ;3 FOR STRING
                ;
                ;
                ;       ROM ROUTINES
                ;
                ;
033A            WRCHR   EQU     033AH           ;ROM ROUTINE TO WRITE A CHARACTER
                                                ;AND SAVE DE REGISTER PAIR
0ACC            ITOS    EQU     0ACCH           ;CONVERT FROM INTEGER TO SINGLE
                                                ;PRECISION FORMAT
0A0C            SNGCP   EQU     0A0CH           ;COMPARE TWO SINGLE PRECISION NUMBERS
0716            SNGADD  EQU     0716H           ;ADD TWO SINGLE PRECISION NUMBERS
0FBD            SNGSTR  EQU     0FBDH           ;CONVERT A SINGLE PRECISION NUMBER
                                                ;TO A STRING OF ASCII CHARACTERS
                ;
                ;
                ;       BEGINNING OF RELOCATABLE LOADER MODULE
                ;
                ;
5200            BEGIN   EQU     $               ;BEGINNING OF PROGRAM
                ;
                ;       THIS PART OF THE PROGRAM (UP TO LABEL "START") IS A
                ;       LOADER FOR PATCHING THE MAIN PROGRAM INTO THE KEYBOARD
                ;       DRIVER AND RELOCATING IT.
                ;
5200 3E0D               LD      A,CR            ;LOAD THE A REGISTER WITH A
                                                ;CARRIAGE RETURN
5202 CD3A03             CALL    WRCHR           ;WRITE IT
5205 115103             LD      DE,ENDADD-START ;GET THE LENGTH OF THE PROGRAM
5208 2A4940             LD      HL,(TOPMEM)     ;GET TOP OF MEMORY BYTES
520B AF                 XOR     A               ;CLEAR THE CARRY FLAG
520C ED52               SBC     HL,DE           ;GET NEW TOP OF MEMORY
520E 2B                 DEC     HL              ;MAKE THE MEM SIZE ONE LESS THAN PRGM START
520F 224940             LD      (TOPMEM),HL     ;SAVE NEW MEMORY SIZE
5212 23                 INC     HL              ;INCREMENT TO START OF PROGRAM ADDRESS
5213 E5                 PUSH    HL              ;SAVE IT
5214 115002             LD      DE,SECNUM-START ;GET NUMBER OF BYTES FROM START
                                                ;OF PROGRAM TO LABEL "SECNUM"
5217 19                 ADD     HL,DE           ;CALCULATE NEW LABEL ADDRESS
```

```
5218 223753        LD    (ADD01+1),HL      ;SAVE IT IN THE PROGRAM
521B 22F753        LD    (ADD02+1),HL
521E 228554        LD    (ADD03+1),HL
5221 228954        LD    (ADD04+1),HL
5224 114E02        LD    DE,JMPADD-START   ;GET NUMBER OF BYTES JMPADD IS
                                           ;FROM START OF PROGRAM
5227 E1            POP   HL                ;GET STARTING ADDRESS OF PROGRAM
5228 E5            PUSH  HL                ;SAVE IT AGAIN
5229 19            ADD   HL,DE             ;CALCULATE NEW ADDRESS FOR
                                           ;LABEL "JMPADD"
522A 22B854        LD    (JUMP01+1),HL     ;SAVE IT IN THE PROGRAM
522D 115102        LD    DE,DISKBF-START   ;FIND NUMBER OF BYTES FROM START
                                           ;OF PROGRAM TO LABEL "DISKBF"
5230 E1            POP   HL                ;GET START OF PROGRAM ADDRESS
5231 E5            PUSH  HL                ;SAVE IT AGAIN
5232 19            ADD   HL,DE             ;CALCULATE NEW ADDRESS FOR
                                           ;LABEL "DISKBF"
5233 224653        LD    (CONT1+1),HL      ;SAVE IT IN THE PROGRAM
5236 225A53        LD    (DAD001+1),HL
5239 229253        LD    (CONT2+1),HL
523C 22EE53        LD    (SETSEC+1),HL
523F 220954        LD    (DAD002+2),HL
5242 227954        LD    (CONT4+1),HL
5245 E1            POP   HL                ;GET STARTING ADDRESS OF PROGRAM
5246 E5            PUSH  HL                ;SAVE IT AGAIN
5247 113502        LD    DE,TEXT1-START    ;GET NUMBER OF BYTES FROM START OF
                                           ;PROGRAM TO LABEL "TEXT1"
524A 19            ADD   HL,DE             ;CALCULATE NEW ADDRESS FOR
                                           ;LABEL "TEXT1"
524B 22DF53        LD    (TX001+1),HL      ;SAVE IT IN THE PROGRAM
524E 114602        LD    DE,TEXT2-START    ;FIND NUMBER OF BYTES FROM START
                                           ;OF PROGRAM TO LABEL "TEXT2"
5251 E1            POP   HL                ;GET STARTING ADDRESS OF PROGRAM
5252 E5            PUSH  HL                ;SAVE IT AGAIN
5253 19            ADD   HL,DE             ;CALCULATE NEW ADDRESS FOR
                                           ;LABEL "TEXT2"
5254 227753        LD    (T001+1),HL       ;SAVE IT IN THE PROGRAM
5257 11ED01        LD    DE,CKDRV-START    ;FIND NUMBER OF BYTES FROM START
                                           ;TO LABEL "CKDRV"
525A E1            POP   HL                ;GET STARTING ADDRESS OF PROGRAM
525B E5            PUSH  HL                ;SAVE IT AGAIN
525C 19            ADD   HL,DE             ;CALCULATE NEW ADDRESS FOR
                                           ;LABEL "CKDRV"
525D 223F53        LD    (CK001+1),HL      ;SAVE IT IN THE PROGRAM
5260 112502        LD    DE,TEXT3-START    ;FIND NUMBER OF BYTES FROM START
                                           ;TO LABEL "TEXT3"
5263 E1            POP   HL                ;GET START OF PROGRAM
5264 E5            PUSH  HL                ;SAVE IT AGAIN
5265 19            ADD   HL,DE             ;CALCULATE NEW ADDRESS FOR
                                           ;LABEL "TEXT3"
5266 22FA54        LD    (CKERR+1),HL      ;SAVE IT IN THE PROGRAM
5269 11B101        LD    DE,ALTRET-START   ;FIND NUMBER OF BYTES FROM
                                           ;START TO LABEL "ALTRET"
526C E1            POP   HL                ;GET START OF PROGRAM
526D E5            PUSH  HL                ;SAVE IT AGAIN
526E 19            ADD   HL,DE             ;CALCULATE NEW ADDRESS FOR
                                           ;LABEL "ALTRET"
526F 220155        LD    (CK002+1),HL      ;SAVE IT IN THE PROGRAM
5272 2A1640        LD    HL,(4016H)        ;GET KEYBOARD DRIVER JUMP ADDRESS
5275 222C55        LD    (JMPADD),HL       ;SAVE IT AT JMPADD
5278 DDE1          POP   IX                ;GET START OF PROGRAM ADDRESS
527A DDE5          PUSH  IX                ;SAVE IT AGAIN
527C DD221640      LD    (4016H),IX        ;MAKE KEYBOARD DRIVER ADDRESS
                                           ;JUMP TO "DIRBASIC"
5280 21DE52        LD    HL,START          ;GET START OF PROGRAM
5283 D1            POP   DE                ;GET STARTING ADDRESS OF
                                           ;PROGRAM WHEN IN HIGH MEMORY
5284 015103        LD    BC,ENDADD-START   ;NUMBER OF BYTES IN PROGRAM
5287 EDB0          LDIR                    ;MOVE IT TO HIGH MEMORY
5289 21C952        LD    HL,MEMSIZ         ;LOAD HL WITH START OF MEMSIZ MESSAGE
528C CD6744        CALL  DSPMSG            ;PRINT IT ON THE SCREEN
528F DD2B          DEC   IX                ;SET IX TO MEMORY SIZE
5291 DD222141      LD    (ARTHBF),IX       ;LOAD RAM ARITHMETIC BUFFER
                                           ;WITH MEMORY SIZE
5295 3E02          LD    A,2               ;LOAD A WITH TYPE OF OPERAND
                                           ;2=INTEGER, 4=SINGLE, 8=DOUBLE, 3=STRING
5297 32AF40        LD    (TYPNUM),A        ;SAVE IT IN TYPE BYTE
529A CDCC0A        CALL  ITOS              ;CONVERT THE MEMORY SIZE FROM
                                           ;INTEGER FORMAT TO SINGLE PRECISION
                                           ;FORMAT.  THE ROM ROUTINE
                                           ;AT ITOS DOES THIS TASK.
529D 110000        LD    DE,0              ;LD THE DEBC REGISTER QUAD WITH
52A0 010090        LD    BC,9000H          ;A SINGLE PRECISION 32768
```

```
52A3 CD0C0A                CALL    SNGCP          ;COMPARE THE NUMBER IN SINGLE PRECISION
                                                  ;FORMAT AT ARTHBF TO THE ONE STORED
                                                  ;IN THE BCDE REGISTER QUAD
52A6 F2B252                JP      P,SKPADD       ;JUMP TO SKPADD IF THE NUMBER
                                                  ;IN BCDE WAS GREATER THAN THE ONE
                                                  ;AT ARTHBF
52A9 110000                LD      DE,0           ;LOAD THE BCDE REGISTER QUAD WITH
52AC 010091                LD      BC,9100H       ;65536 IN SINGLE PRECISION FORMAT
52AF CD1607                CALL    SNGADD         ;ADD THE SINGLE PRECISION NUMBER
                                                  ;IN BCDE TO THE ONE AT ARTHBF.
                                                  ;
                ;          SINCE A NUMBER FROM 32768-65535 IS A NEGATIVE NUMBER
                ;          IN INTEGER FORMAT, THE MEMORY SIZE MUST HAVE 65536
                ;          ADDED TO IT SO IT APPEARS TO BE A NUMBER FROM 32768-65535
                ;
52B2 CDBD0F      SKPADD    CALL    SNGSTR         ;CONVERT THE SINGLE PRECISION NUMBER
                                                  ;AT ARTHBF TO A STRING OF ASCII CHARACTERS
52B5 213041                LD      HL,4130H       ;START OF ASCII STRING OF MEM SIZE
52B8 0606                  LD      B,6            ;NUMBER OF CHARACTERS TO PRINT
52BA 7E          WRLOOP    LD      A,(HL)         ;GET A CHARACTER
52BB 23                    INC     HL             ;POINT TO NEXT CHARACTER
52BC CD3A03                CALL    WRCHR          ;WRITE IT
52BF 10F9                  DJNZ    WRLOOP         ;WRITE ANOTHER IF NOT THROUGH
52C1 3E0D                  LD      A,CR           ;LOAD A WITH A CARRIAGE RETURN
52C3 CD3A03                CALL    WRCHR          ;WRITE IT ON THE SCREEN
52C6 C32D40                JP      DOSJMP         ;RETURN TO DOS
52C9 55          MEMSIZ    DEFM    'USE A MEMORY SIZE OF'  ;MEMORY SIZE MESSAGE
     53 45 20 41 20 4D 45 4D
     4F 52 59 20 53 49 5A 45
     20 4F 46
52DD 03                    DEFB    03             ;TERMINATING BYTE OF DSPMSG ROUTINE
                ;
                ;
                ;          THIS IS THE START OF THE ACTUAL "DIRBASIC" PROGRAM
                ;
                ;          THE FIRST PART IS JUST A RELOCATABLE LOADER FOR THIS
                ;          PROGRAM.
                ;
                ;
52DE             START     EQU     $              ;STARTING ADDRESS OF PROGRAM
52DE D9                    EXX                    ;EXCHANGE REGISTERS (HL,DE,BC,HL',DE',BC')
52DF 3A8038                LD      A,(3880H)      ;GET ROW OF SHIFT KEY
52E2 B7                    OR      A              ;COMPARE IT TO ITSELF
52E3 285E                  JR      Z,P001         ;IF ZERO THEN JUMP TO P001
52E5 FE10                  CP      10H            ;IS THE PENCIL CONTROL KEY PRESSED
52E7 2807                  JR      Z,PCNTL        ;IF SO JUMP TO PCNTL
52E9 3A4038                LD      A,(3840H)      ;GET THE UP ARROW KEY ROW
52EC CB5F                  BIT     3,A            ;IS THE UP ARROW KEY DEPRESSED?
52EE 2853                  JR      Z,P001         ;IF NOT JUMP TO P001
52F0 3A1038      PCNTL     LD      A,(3810H)      ;GET 0-3 KEY ROW
52F3 FE00                  CP      0              ;ANY KEYS PRESSED?
52F5 384C                  JR      C,P001         ;IF NOT JUMP TO P001
52F7 FE09                  CP      9              ;ANYTHING ABOVE THE 3 KEY PRESSED?
52F9 3048                  JR      NC,P001        ;IF SO JUMP TO P001
52FB FE01                  CP      1              ;IS THE 0 KEY PRESSED?
52FD 2009                  JR      NZ,NOTDR0      ;IF NOT JUMP TO NOTDR0
52FF 320943                LD      (DRVBIT),A     ;LOAD DRVBIT WITH A
5302 AF                    XOR     A              ;LOAD A WITH THE DRIVE#
5303 320843                LD      (CURDRV),A     ;SAVE IT
5306 1828                  JR      SETUP          ;JUMP TO SETUP
5308 FE02        NOTDR0    CP      2              ;IS THE 1 KEY PRESSED
530A 200A                  JR      NZ,NOTDR1      ;IF NOT JUMP TO NOTDR1
530C 320943                LD      (DRVBIT),A     ;SET THE DRVBIT
530F 3E01                  LD      A,1            ;SELECT DRIVE# 1
5311 320843                LD      (CURDRV),A     ;SAVE IT
5314 181A                  JR      SETUP          ;JUMP TO SETUP
5316 FE04        NOTDR1    CP      4              ;IS THE 2 KEY PRESSED
5318 200A                  JR      NZ,NOTDR2      ;IF NOT JUMP TO NOTDR2
531A 320943                LD      (DRVBIT),A     ;SAVE THE DRVBIT
531D 3E02                  LD      A,2            ;SELECT DRIVE# 2
531F 320843                LD      (CURDRV),A     ;SAVE IT
5322 180C                  JR      SETUP          ;JUMP TO SETUP
5324 FE08        NOTDR2    CP      8              ;IS THE 3 KEY PRESSED
5326 201B                  JR      NZ,P001        ;IF NOT JUMP TO P001
5328 320943                LD      (DRVBIT),A     ;SAVE THE DRVBIT
532B 3E03                  LD      A,3            ;SELECT DRIVE# 3
532D 320843                LD      (CURDRV),A     ;SAVE IT
5330             SETUP     EQU     $
5330 FD2A2040              LD      IY,(4020H)     ;LOAD IY WITH THE CURRENT CURSOR POSITION
5334 3E02                  LD      A,2            ;LOAD A WITH THE STARTING SECTOR
                                                  ;NUMBER TO READ
5336 322E55      ADD01     LD      (SECNUM),A     ;SAVE IT
5339 3E0D                  LD      A,CR           ;LOAD A WITH A CARRIAGE RETURN
533B CD3A03                CALL    WRCHR          ;WRITE IT
533E CDCB54      CK001     CALL    CKDRV          ;CHECK IF THE SELECTED DRIVE IS AVAILABLE
5341 1802                  JR      CONT1          ;JUMP TO CONT1
5343 184A        P001      JR      P002           ;JUMP TO P002
5345 212F55      CONT1     LD      HL,DISKBF      ;LOAD HL WITH THE START OF THE
                                                  ;DISK BUFFER
5348 1611                  LD      D,11H          ;SET D TO TRACK 11H
```

```
534A 1E00                 LD      E,0             ;SET E TO SECTOR 0
534C 0600                 LD      B,0             ;B= LOGICAL RECORD LENGTH
534E 3A0843               LD      A,(CURDRV)      ;LOAD A WITH THE CURRENT DRIVE#
5351 4F                   LD      C,A             ;PUT IT IN C
5352 CDDD46               CALL    DSKRD           ;READ A DISK SECTOR
5355 FE06                 CP      6               ;IS THE ERROR CODE 6?
                                                  ;(IT SHOULD BE FOR A DIRECTORY SECTOR)
5357 2043                 JR      NZ,E001         ;IF NOT JUMP TO E001
5359 212F55       DAD001  LD      HL,DISKBF       ;LOAD HL WITH THE START OF THE DISK BUFFER
535C 0650                 LD      B,50H           ;LOAD B WITH THE NUMBER OF BYTES
                                                  ;TO COUNT
535E 110000               LD      DE,0            ;SET DE TO ZERO
5361 7E           FRESPC  LD      A,(HL)          ;GET A FILE ALLOCATION BYTE
                  ;
                  ;       THESE BYTES TELL WHICH GRANS ARE ALLOCATED AND WHICH AREN'T
                  ;
                  ;       FC= 2 FREE GRANS
                  ;       FD= 1 FREE GRAN
                  ;       FE= 1 FREE GRAN
                  ;       FF= 0 FREE GRANS
                  ;
5362 FEFE                 CP      0FEH            ;IS IT 0FEH
5364 2001                 JR      NZ,LBL001       ;IF NOT JUMP TO LBL001
5366 13                   INC     DE              ;ADD ONE TO FREE SPACE COUNT
5367 FEFD         LBL001  CP      0FDH            ;IS IT 0FDH
5369 2001                 JR      NZ,LBL002       ;IF NOT JUMP TO LBL002
536B 13                   INC     DE              ;ADD ONE TO FREE SPACE COUNT
536C FEFC         LBL002  CP      0FCH            ;IS IT 0FCH?
536E 2002                 JR      NZ,NEXTFR       ;IF NOT JUMP TO NEXTFR
5370 13                   INC     DE              ;ADD TWO TO THE FREE SPACE COUNT
5371 13                   INC     DE
5372 23           NEXTFR  INC     HL              ;POINT TO NEXT GRAN ALLOCATION BYTE
5373 10EC                 DJNZ    FRESPC          ;CHECK ANOTHER ONE IF NOT THROUGH
5375 D5                   PUSH    DE              ;SAVE FREE SPACE COUNT
5376 212455       T001    LD      HL,TEXT2        ;LOAD HL WITH THE START OF THE
                                                  ;"DRIVE#" MESSAGE
5379 CD6744               CALL    DSPMSG          ;DISPLAY IT
537C 3A0843               LD      A,(CURDRV)      ;GET THE CURRENT DRIVE#
537F C630                 ADD     A,30H           ;MAKE IT ASCII
5381 CD3A03               CALL    WRCHR           ;DISPLAY IT ON THE SCREEN
5384 0608                 LD      B,8             ;NUMBER OF SPACES TO WRITE
5386 3E20                 LD      A,' '           ;LOAD A WITH A SPACE
5388 CD3A03       TABOVR  CALL    WRCHR           ;WRITE IT
538B 10FB                 DJNZ    TABOVR          ;WRITE ANOTHER IF NOT DONE
538D 1802                 JR      CONT2           ;JUMP TO CONT2
538F 185A         P002    JR      P003            ;JUMP TO P003
5391 212F55       CONT2   LD      HL,DISKBF       ;LOAD HL WITH THE START OF THE DISK BUFFER
5394 11D000               LD      DE,0D0H         ;SET DE TO POINT TO DRIVE NAME
5397 19                   ADD     HL,DE           ;POINT TO IT
5398 0608                 LD      B,8             ;NUMBER OF CHARACTERS TO WRITE
539A 1802                 JR      WRNAME          ;JUMP TO WRNAME
539C 1854         E001    JR      E002            ;JUMP TO E002
                  ;
                  ;       WRITE THE DISK NAME ON THE SCREEN
                  ;
539E 7E           WRNAME  LD      A,(HL)          ;GET A CHARACTER
539F 23                   INC     HL              ;POINT TO THE NEXT ONE
53A0 CD3A03               CALL    WRCHR           ;WRITE THE CHARACTER
53A3 10F9                 DJNZ    WRNAME          ;WRITE ANOTHER IF NOT DONE
53A5 0608                 LD      B,8             ;NUMBER OF CHARACTERS TO WRITE
53A7 3E20                 LD      A,' '           ;LOAD A WITH A SPACE
53A9 CD3A03       WRSPC   CALL    WRCHR           ;WRITE IT
53AC 10FB                 DJNZ    WRSPC           ;WRITE ANOTHER IF NOT THROUGH
53AE 0608                 LD      B,8             ;NUMBER OF CHARACTERS TO WRITE
                  ;
                  ;       WRITE THE DISK DATE ON THE SCREEN
                  ;
53B0 7E           WRDATE  LD      A,(HL)          ;GET A CHARACTER
53B1 23                   INC     HL              ;POINT TO NEXT CHARACTER
53B2 CD3A03               CALL    WRCHR           ;WRITE IT
53B5 10F9                 DJNZ    WRDATE          ;WRITE ANOTHER IF NOT THROUGH
53B7 0608                 LD      B,8             ;NUMBER OF CHARACTER TO WRITE
53B9 3E20                 LD      A,' '           ;LOAD A WITH A SPACE
53BB CD3A03       WRSPC2  CALL    WRCHR           ;WRITE IT
53BE 10FB                 DJNZ    WRSPC2          ;WRITE ANOTHER IF NOT THROUGH
53C0 E1                   POP     HL              ;GET NUMBER OF GRANS FREE
53C1 222141               LD      (ARTHBF),HL     ;SAVE IT IN THE ROM ARITHMETIC BUFFER
53C4 3E02                 LD      A,2             ;LOAD A WITH 2 - INTEGER FORMAT
53C6 32AF40               LD      (TYPNUM),A      ;SAVE IT
53C9 CDBD0F               CALL    SNGSTR          ;CONVERT IT TO A STRING OF ASCII
                                                  ;CHARACTERS
53CC 3A3341               LD      A,(4133H)       ;GET THE HUNDREDS DIGIT
53CF CD3A03               CALL    WRCHR           ;WRITE IT
                  ;
                  ;       I WRITE OUT THE HUNDREDS DIGIT BECAUSE SOME PEOPLE
                  ;       MAY HAVE 77 TRACK DRIVES AND/OR PERCOM DATA DOUBLER
                  ;
53D2 3A3441               LD      A,(4134H)       ;GET THE TENS DIGIT
53D5 CD3A03               CALL    WRCHR           ;WRITE IT
53D8 3A3541               LD      A,(4135H)       ;GET THE ONES DIGIT
53DB CD3A03               CALL    WRCHR           ;WRITE IT
53DE 211355       TX001   LD      HL,TEXT1        ;LOAD HL WITH THE START OF TEXT1
53E1 CD6744               CALL    DSPMSG          ;DISPLAY THE MESSAGE
```

```
53E4 3E0D                 LD      A,CR            ;LOAD A WITH A CARRIAGE RETURN
53E6 CD3A03               CALL    WRCHR           ;WRITE IT
53E9 1802                 JR      SETSEC          ;JUMP TO SETSEC
53EB 1850        P003     JR      P004            ;JUMP TO P004
53ED 212F55      SETSEC   LD      HL,DISKBF       ;LOAD HL WITH THE START OF THE DISK BUFFER
53F0 1802                 JR      ECONT1          ;JUMP TO ECONT1
53F2 1852        E002     JR      E003            ;JUMP TO E003
53F4 1611        ECONT1   LD      D,11H           ;SET D TO TRACK 11H
53F6 3A2E55      ADD02    LD      A,(SECNUM)      ;GET THE SECTOR#
53F9 5F                   LD      E,A             ;PUT IT IN E
53FA 0600                 LD      B,0             ;LRL=256
53FC 3A0843               LD      A,(CURDRV)      ;GET DRIVE#
53FF 4F                   LD      C,A             ;PUT IT IN C
5400 CDDD46               CALL    DSKRD           ;READ A SECTOR
5403 FE06                 CP      6               ;IS THE ERROR CODE 6?
5405 203F                 JR      NZ,E003         ;IF NOT JUMP TO E003 (ERROR HAS OCCURRED)
5407 DD212F55    DAD002   LD      IX,DISKBF       ;LOAD IX WITH THE START OF THE DISK BUFFER
540B 1802                 JR      GETONE          ;JUMP TO GETONE
540D 18DE        S002     JR      SETSEC          ;JUMP TO SETSEC
540F DD7E00      GETONE   LD      A,(IX+0)        ;GET A FILE'S STATUS
5412 FE20                 CP      20H             ;HOW DOES IT COMPARE TO 20H
                 ;
                 ;        THIS PROGRAM NORMALLY DISPLAYS LIKE A "DIR (I)".
                 ;
                 ;        TO MAKE IT DISPLAY LIKE "DIR" CHANGE THE 20H TO 18H.
                 ;
                 ;        TO MAKE IT DISPLAY LIKE "DIR (I,S)" CHANGE THE 20H TO 7FH.
                 ;
                 ;
5414 3050                 JR      NC,NXTONE       ;IF GREATER THAN 20H THEN JUMP TO NXTONE
5416 FE10                 CP      10H             ;HOW DOES IT COMPARE TO 10H?
5418 384C                 JR      C,NXTONE        ;IF LESS THAN 10H JUMP TO NXTONE
541A DDE5                 PUSH    IX              ;SAVE IX
541C E1                   POP     HL              ;PUT IT IN HL
541D 110500               LD      DE,5            ;SET DE TO POINT TO FILE NAME
5420 19                   ADD     HL,DE           ;SET HL TO START OF FILE NAME
5421 E5                   PUSH    HL              ;SAVE IT
5422 0608                 LD      B,8             ;MAX NUMBER OF CHARACTERS TO WRITE
5424 7E          WLOOP1   LD      A,(HL)          ;GET A CHARACTER
5425 FE20                 CP      20H             ;IS IT A SPACE?
5427 2806                 JR      Z,CNTUM         ;IF SO JUMP TO CNTUM
5429 CD3A03               CALL    WRCHR           ;WRITE IT
542C 23                   INC     HL              ;POINT TO NEXT CHARACTER
542D 10F5                 DJNZ    WLOOP1          ;CHECK ANOTHER CHARACTER IF NOT THROUGH
542F 48          CNTUM    LD      C,B             ;NUMBER OF SPACES TO PAD FILE NAME
5430 E1                   POP     HL              ;RESTORE FILE NAME START
5431 E5                   PUSH    HL              ;SAVE IT AGAIN
5432 110800               LD      DE,8            ;SET DE TO POINT TO FILE EXTENSION
5435 19                   ADD     HL,DE           ;HL POINTS TO FILE NAME EXTENT
5436 7E                   LD      A,(HL)          ;GET A CHARACTER
5437 FE20                 CP      20H             ;IS IT A SPACE?
5439 281D                 JR      Z,WLOOP3        ;IF SO JUMP TO WLOOP3
543B 1802                 JR      CONT3           ;IF NOT JUMP TO CONT3
543D 1837        P004     JR      P005            ;JUMP TO P005
543F 3E2F        CONT3    LD      A,'/'           ;LOAD A WITH A "/"
5441 CD3A03               CALL    WRCHR           ;WRITE IT
5444 1802                 JR      ECONT2          ;JUMP TO ECONT2
5446 1823        E003     JR      E004            ;JUMP TO E004
5448 0603        ECONT2   LD      B,3             ;MAX NUMBER OF CHARACTER TO WRITE
544A 7E          WLOOP2   LD      A,(HL)          ;GET A CHARACTER
544B 1802                 JR      SCONT1          ;JUMP TO SCONT1
544D 18BE        S001     JR      S002            ;JUMP TO S002
544F CD3A03      SCONT1   CALL    WRCHR           ;WRITE THE CHARACTER
5452 23                   INC     HL              ;POINT TO NEXT CHARACTER
5453 10F5                 DJNZ    WLOOP2          ;CHECK ANOTHER IF NOT THROUGH
5455 79                   LD      A,C             ;COPY PAD SPACES TO A
5456 1803                 JR      WLOOP4          ;JUMP TO WLOOP4
5458 79          WLOOP3   LD      A,C             ;ADD TO NUMBER OF SPACES TO WRITE
5459 C604                 ADD     A,4             ;SUM IT
545B C604        WLOOP4   ADD     A,4             ;ADD AN AUTOMATIC PAD OF 4 SPACES
545D 47                   LD      B,A             ;SUM IT
545E 3E20                 LD      A,' '           ;LOAD A WITH A SPACE
5460 CD3A03      WLOOP5   CALL    WRCHR           ;WRITE A SPACE
5463 10FB                 DJNZ    WLOOP5          ;WRITE ANOTHER IF NOT THROUGH
5465 E1                   POP     HL              ;RESTORE FILE NAME POINTER
5466 DDE5        NXTONE   PUSH    IX              ;SAVE FILE'S STATUS BYTE
5468 E1                   POP     HL              ;PUT IT IN HL
5469 1802                 JR      ECONT3          ;JUMP TO ECONT3
546B 184F        E004     JR      ERROR           ;JUMP TO ERROR
546D 112000      ECONT3   LD      DE,20H          ;LOAD DE TO POINT TO NEXT FILE ENTRY
5470 19                   ADD     HL,DE           ;ADD IT TO HL
5471 E5                   PUSH    HL              ;SAVE NEW FILE POINTER
5472 DDE1                 POP     IX              ;COPY IT TO IX
5474 1802                 JR      CONT4           ;JUMP TO CONT4
5476 183D        P005     JR      RETURN          ;JUMP TO RETURN
5478 212F55      CONT4    LD      HL,DISKBF       ;LOAD HL WITH THE START OF THE
                                                  ;DISK BUFFER
547B 24                   INC     H               ;POINT TO 256 BYTES PAST BUFFER
547C DDE5                 PUSH    IX              ;SAVE NEW FILE STATUS POINTER
547E D1                   POP     DE              ;COPY IT TO DE
```

```
547F AF         04520           XOR     A               ;CLEAR THE CARRY FLAG
5480 ED52       04530           SBC     HL,DE           ;SUBTRACT DE FROM HL
5482 208B       04540           JR      NZ,GETONE       ;IF RESULT NOT EQUAL TO ZERO
                04550                                   ;THEN THIS DIRECTORY SECTOR
                04560                                   ;IS NOT USED UP YET.
5484 3A2E55     04570 ADD03     LD      A,(SECNUM)      ;GET CURRENT SECTOR#
5487 3C         04580           INC     A               ;POINT TO NEXT SECTOR#
5488 322E55     04590 ADD04     LD      (SECNUM),A      ;SAVE IT AGAIN
548B FE0A       04600           CP      0AH             ;HOW DOES IT COMPARE TO 0AH?
548D 38BE       04610           JR      C,S001          ;IF LESS THAN 0AH JUMP TO S001
548F 3E0D       04620 ALTRET    LD      A,CR            ;LOAD A WITH A CARRIAGE RETURN
5491 CD3A03     04630           CALL    WRCHR           ;WRITE IT
5494 CD3A03     04640           CALL    WRCHR           ;WRITE IT AGAIN
5497 2A2040     04650           LD      HL,(4020H)      ;LOAD HL WITH THE CURRENT CURSOR POSITION
549A 3620       04660           LD      (HL),20H        ;PUT A SPACE THERE
549C FDE5       04670           PUSH    IY              ;SAVE CURSOR POSTION BEFORE "DIRBASIC"
                04680                                   ;STARTED OPERATION
549E D1         04690           POP     DE              ;COPY IT TO DE
549F 7B         04700           LD      A,E             ;LOAD A WITH E (BITS 0-5 OF E
                04710                                   ;CONTAINS THE HORIZONTAL POSITION
                04720                                   ;OF THE CURSOR).
54A0 E63F       04730           AND     3FH             ;ISOLATE THE HORIZONTAL POSITION
54A2 57         04740           LD      D,A             ;SAVE IT IN D
54A3 7D         04750           LD      A,L             ;LOAD A WITH THE OLD HORIZONTAL POSITION
54A4 E6C0       04760           AND     0C0H            ;MAKE THE HORIZONTAL POSITION ZERO
54A6 82         04770           ADD     A,D             ;CALCULATE THE NEW CURSOR POSITION
                04780                                   ;(HORIZONTAL)
54A7 6F         04790           LD      L,A             ;SAVE IT IN L
54A8 2B         04800           DEC     HL              ;BACK UP CURSOR ONE POSITION
54A9 222040     04810           LD      (4020H),HL      ;SAVE NEW CURSOR POSITION
54AC 3E20       04820           LD      A,20H           ;LOAD A WITH A SPACE
54AE CD3A03     04830           CALL    WRCHR           ;WRITE IT
54B1 3E18       04840           LD      A,18H           ;LOAD A WITH A SHIFTED BACKSPACE
54B3 B7         04850           OR      A               ;SET THE FLAGS
54B4 C9         04860           RET                     ;RETURN TO THE CALLING PROGRAM
54B5 D9         04870 RETURN    EXX                     ;EXCHANGE REGISTERS BACK
54B6 E5         04880           PUSH    HL              ;SAVE HL
54B7 2A2C55     04890 JUMP01    LD      HL,(JMPADD)     ;LOAD HL WITH THE RETURN ADDRESS
54BA E3         04900           EX      (SP),HL         ;EXCHANGE THEM
54BB C9         04910           RET                     ;JUMP TO THE KEYBOARD ROUTINE
                04920 ;
                04930 ;         SUPPORT ROUTINES
                04940 ;
54BC F6C0       04950 ERROR     OR      0C0H            ;DISPLAY A DOS ERROR MESSAGE
54BE F5         04960           PUSH    AF              ;SAVE AF
54BF 3E0D       04970           LD      A,CR            ;LOAD A WITH A CARRIAGE RETURN
54C1 CD3A03     04980           CALL    WRCHR           ;WRITE IT
54C4 F1         04990           POP     AF              ;RESTORE AF
54C5 CD0944     05000           CALL    DSKERR          ;DISPLAY THE DISK ERROR
54C8 C33040     05010           JP      DOSERR          ;RETURN TO DOS - ERROR HAS OCCURED
                05020 ;
                05030 ;         CKDRV - THIS ROUTINE CHECKS TO SEE IF THE DRIVE SELECTED
                05040 ;                 IN THE 4309H DRIVE SELECT BIT BYTE IS AVAILABLE
                05050 ;
54CB 3ED0       05060 CKDRV     LD      A,0D0H          ;LOAD A WITH DISK CONTROLLER RESET BYTE
54CD 32EC37     05070           LD      (37ECH),A       ;STORE IT
54D0 010010     05080           LD      BC,1000H        ;GET A LOOP NUMBER
54D3 3A0943     05090 CKLP01    LD      A,(DRVBIT)      ;GET THE DRIVE SELECT BYTE
54D6 32E137     05100           LD      (37E1H),A       ;START THE DISK MOTOR AND SELECT
                05110                                   ;THE DRIVE
54D9 0B         05120           DEC     BC              ;DECREMENT THE STALL COUNTER
54DA 78         05130           LD      A,B             ;ARE BOTH B AND C EQUAL TO ZERO?
54DB B1         05140           OR      C               ;(BC=0?)
54DC 20F5       05150           JR      NZ,CKLP01       ;IF NOT JUMP BACK AGAIN
54DE 3AEC37     05160           LD      A,(37ECH)       ;GET INDEX PULSE BIT FROM
                05170                                   ;FROM THE DISK CONTROLLER STATUS
                05180                                   ;REGISTER
54E1 E602       05190           AND     2
54E3 57         05200           LD      D,A             ;SAVE IPB IN D REGISTER
54E4 010010     05210           LD      BC,1000H        ;SET LOOP COUNTER AGAIN
54E7 3A0943     05220 CKLP02    LD      A,(DRVBIT)      ;GET DRIVE SELECT BIT
54EA 32E137     05230           LD      (37E1H),A       ;SELECT THE DRIVE
54ED 3AEC37     05240           LD      A,(37ECH)       ;GET THE IPB
54F0 E602       05250           AND     2
54F2 BA         05260           CP      D               ;HAS IT CHANGED?
54F3 C0         05270           RET     NZ              ;IF SO RETURN TO CALLING INSTRUCTION
54F4 0B         05280           DEC     BC              ;IF NOT DECREMENT THE COUNTER
54F5 78         05290           LD      A,B             ;AND KEEP SELECTING THE DRIVE
54F6 B1         05300           OR      C
54F7 20EE       05310           JR      NZ,CKLP02
54F9 210355     05320 CKERR     LD      HL,TEXT3        ;LOAD HL WITH START OF MESSAGE
54FC CD6744     05330           CALL    DSPMSG          ;DISPLAY IT
54FF E1         05340           POP     HL              ;GET RETURN ADDRESS FROM CALL
5500 C38F54     05350 CK002     JP      ALTRET          ;JUMP TO ALTRET
5503 44         05360 TEXT3     DEFM    'DRIVE NOT READY'       ;MESSAGE TEXT
     52 49 56 45 20 4E 4F 54
     20 52 45 41 44 59
5512 03         05370           DEFB    03H
5513 20         05380 TEXT1     DEFM    ' GRANS FREE            ;MESSAGE TEXT
     47 52 41 4E 53 20 46 52
```

```
      45 45 20 20 20 20 20
5523 03        05390          DEFB    03              ;TERMINATING BYTE
5524 44        05400 TEXT2    DEFM    'DRIVE# '       ;MESSAGE TEXT
      52 49 56 45 23 20
552B 03        05410          DEFB    03              ;TERMINATING BYTE
552C 0000      05420 JMPADD   DEFW    0               ;RETURN ADDRESS TO THE KEYBOARD
               05430                                  ;ROUTINE BEING USED JUST BEFORE
               05440                                  ;"DIRBASIC" WAS LOADED
552E 00        05450 SECNUM   DEFB    0               ;CURRENT SECTOR#
0100           05460 DISKBF   DEFS    100H            ;256 BYTE DISK BUFFER
562F           05470 ENDADD   EQU     $               ;ENDING ADDRESS OF THE PROGRAM
5200           05480          END     BEGIN           ;STARTING ADDRESS OF THE PROGRAM IS BEGIN
00000 TOTAL ERRORS
```

EDAS > **An editor and assembler for the Model I and III**

WITH DISK FEATURES SIMILAR TO EDTASM-PLUS:

1. A symbol table buffer area of over thirty thousand bytes, with text buffer area equal to your drive capacity.
2. "LOAD" and "WRITE" text buffers from/to disk, as well as assembled object code.
3. Block move features for rearranging text lines.
4. Ability to locate designated strings within text.
5. Global replace throughout user designated range of lines.
6. Execute DOS commands without leaving EDAS.
7. The editor maintains command syntax identical to Basic editor, and adds line insert, line replace and line renumbering abilities.
8. "ASSEMBLE" with numerous switches to allow direct assemble to memory. Debug before leaving EDAS.
9. "BRANCH" allows you to execute your program that has been assembled to RAM, then return to EDAS.
10. "USAGE" will give you the number of free text buffer bytes, the number of bytes used, and the address of the first free byte.
11. "Hardcopy" to print all or part of text; "Type" to print without line numbers.
12. Functional CLEAR key; scroll text with ARROWS, upper and lower case.

$79.95 MAIL TO:

THE ALTERNATE SOURCE

THE ALTERNATE SOURCE
1806 ADA STREET
LANSING, MICHIGAN 48910

NAME ______________________________
ADDRESS ______________________________
CITY ____________________ **ST** _____ **ZIP** __________
CARD # ____________________ **EXP.** __________

MAKING YOUR MACHINE LANGUAGE PROGRAMS RELOCATABLE

By Jack Decker

How would you like to be able to create machine language programs that don't require you to pre-set the MEMORY SIZE, don't wipe out other machine language programs that are already in high memory, and automatically locate themselves in the highest available memory? Not only is all of this possible, it isn't terribly difficult to do, if you will follow the steps outlined below.

Step one: Write the program in the normal way, with two exceptions. The first exception is that the first three lines of your assembly language program should be these:

```
                 ORG          nnnn
      OFFSET     EQU          0
      START      (first instruction of your program)
```

(There is an exception to the exception: If your program has an initialization segment, for example a piece of code to patch the program into the USR function, it should be inserted immediately following the ORG statement. The reason is that you will have to delete it later. You will be writing a new initialization segment during step three, but you still may want to write a temporary one now for testing and debugging purposes).

The other exception is that whenever you make an absolute reference to another portion of your program (with a JP or CALL instruction, for example), use a label with -OFFSET added. For example, suppose you wanted to CALL a subroutine in your program named KEYTST. You must write the instruction like this:

```
                 CALL         KEYTST-OFFSET
```

Note that at this stage of the game, subtracting the OFFSET will not make any difference in the assembled instruction, because OFFSET is set equal to zero at the beginning of the program. You must write all instructions that reference other locations in your program in this manner, if the instruction assembles to a two-byte absolute address. You must NOT write relative jump instructions (JR or DJNZ) in this manner. And you must NOT write instructions that reference locations outside the program (calls to ROM, etc.) in this manner. Only instructions that directly reference an address within your program must be written this way.

To save yourself some effort later, you should write your program to use relative jump (JR) instructions whenever possible, and in any event your program must not be position- dependant (in other words, you should be able to change the ORG address without running into problems).

Write your program, test it, debug it, and make the changes and improvements you feel are needed.

When your program runs perfectly, you are ready to make it relocatable. To begin step two, change the ORG address of your program to one - that's right, numero uno - and assemble it. Either use the SHIFT-@ key to stop the assembled listing every few lines, or, if you are fortunate enough to have a printer, dump the assembled listing to hardcopy. What you are looking for is any instruction that references an address within your program. These instructions should be easy to spot because they should all have the -OFFSET suffix. Because we have assembled the program starting at address 1, the address field of the assembly listing will show us the number of bytes from the start of the program to each label. Confused? Here's an example. Suppose the beginning of our assembler listing looks like this:

```
0001              100             ORG     0001H
0001              110   OFFSET    EQU     0
0001    3C        120   START     INC     A
0002    CAnnnn    130             JP      Z,ELSWHR-OFFSET
```

Note that the address 0002 is at the beginning of the line containing a JP instruction to elsewhere in the program. You want to make a note of this address (02H in this case) for this and any other lines containing the -OFFSET suffix, because you will need it later.

Why are we doing this? Well, go back to the example above, and count the byte displacement from the label START to the address field indicated by the label ELSWHR-OFFSET. The INC A instruction has zero displacement (it is located at the address indicated by START). The JP instruction is located one byte away, and the address for the JP instruction - which will have to be changed when the program is relocated - is two bytes away from START. And our address field says 0002! Well, why NOT let the computer do our counting for us?

Note that this method is accurate when there is an instruction in front of the address field. If you happen to have a table of addresses within the program (using the DEFW instruction, for example), you could change the ORG to zero to get the correct byte count for those addresses.

But before you change the ORG address, we need one more bit of information while it is still set to one. We need to know the total number of bytes in the program, and the easiest way to get this information is to check the address field at the beginning of the last instruction in the program. Don't forget to allow for a multi-byte instruction. For example, if the last line of our program (ignoring the END statement) is

```
   0136    C9        990          RET
```

your program is 136H bytes long. But suppose it is

```
   0136    C3CC06    990          JP           6CCH
```

in that case, your program is 138H bytes long.

At this point you should have a list of byte-displacements for all addresses that will have to be changed when the program is relocated, and the length of the program (in hexadecimal).

Now we come to step three - this is the good part. First, delete the ORG address line in your program (as well as any temporary initialization lines that may have followed it). Then change the next line so that it reads:

```
              OFFSET     EQU          $
```

The first two lines of your program should now be:

```
              OFFSET     EQU          $
              START      (first instruction of your program)
```

Next, using the EDITOR-ASSEMBLER "N" command, make a lot of room at the beginning of your program. I suggest typing N5000,10 to get everything well out of the way.

We are now going to add some code to the start of your program. These instructions will only be used to relocate your program - they are not saved for later use, so we needn't be too concerned with making the code as short as possible (but we won't make it overly sloppy, either).

The first added code will be the following lines, which should be placed at the very start of your program:

```
          ORG     5800H
 INTLZE   LD      A,0C9H                ;Plug DOS vector with
          LD      (41BBH),A             ;  a RET instruction
          LD      HL,(40B1H)            ;Get old MEMORY SIZE
          LD      DE, program length obtained during step two
```

```
        XOR    A                      ;Clear carry flag
        SBC    HL,DE                  ;HL=new MEMORY SIZE
        LD     (40B1H),HL             ;Put back new MEM SIZE
        LD     DE,32H                 ;"CLEAR 50"
        CALL   1E83H                  ;& reset other pointers
        LD     BC, program length obtained during step 2
        LD     DE, START-1+ program length from step 2
        LD     HL, (40B1H)            ;Get new prgrm location
        ADD    HL,BC                  ;HL=New program end
        EX     DE,HL                  ;HL=present, DE=new end
        LDDR                          ;Move the program
        INC   DE                      ;DE=Actual start of
                                      ;   program address
```

The above code will protect memory for the program and move it (don't forget to put the H after the program length in the two locations that call for it, unless you have converted it to decimal since step two). If you have managed to make your program completely relocatable (it has NO absolute jumps, calls, etc.) you can skip the next instructions, and also omit the REPLCE subroutine from your program. Otherwise, take your list of address displacements (from step two), and insert the following code for each one:

```
        LD     HL, address offset (don't forget the "H")
        CALL   REPLCE
```

Going back to our example in step 2, for that program we would have to insert the following:

```
        LD          HL,02H
        CALL        REPLCE
```

Repeat this code for each displacement on the list, inserting the displacements after the LD HL, instruction. Hopefully you won't have very many addresses to change, but if you do you may want to try and write a table lookup routine or some such thing. If you do, bear in mind that the contents of the DE registers must be saved, and be careful not to locate the table beyond the instruction labeled OFFSET (this is the start of your program).

Once you have inserted the relocator routine and the LD HL,nnnn/CALL REPLCE sequence as many times as is necessary, you can do whatever initialization your program requires. DE contains the address of the first instruction of your relocated program (the instruction labelled START). If the entry point to your program is elsewhere, you'll have to count the offset from START first. You could then use the sequence

```
        LD      HL,entry point offset
        ADD     HL,DE
```

to get the entry point into the HL registers. For now, let's assume that the entry point is the same as the START address, which is stored in the DE registers. If your program is to be called by the Level II USR function, you would insert the following instruction:

```
        LD      (16526D),DE
```

Another method of access to your program might be through an unused Disk BASIC command. For example, to patch into the "NAME" command (so that you could call your program by "NAME" or using NAME in a BASIC program), you could use this instruction sequence:

```
        LD      HL,418EH                ;NAME command vector
        LD      (HL),0C3H               ;JP instruction
        INC     HL
        LD      (HL),E                  ;LSB of START address
        INC     HL
        LD      (HL),D                  ;MSB of START address
```

Now we come to the end of our relocator program (except for the REPLCE subroutine). Terminate the routine according to what you want to have happen when initialization is completed. If you want to go back to BASIC (the "READY" prompt), do it like this:

```
        JP      6CCH
```

On the other hand, if you want to jump right into your relocated machine language program, you can do this:

```
        EX      DE,HL       ;Get start address in HL
        JP      (HL)
```

Finally, we need the REPLCE subroutine (unless, of course, there were't any absolute jumps, calls, etc. in the program). The REPLCE subroutine looks like this:

```
REPLCE  ADD     HL,DE           ;Get address of label
        LD      C,(HL)          ;Get byte displacement
        INC     HL              ;  from program and
        LD      B,(HL)          ;  put in BC
        DEC     HL              ;HL=address of label
        PUSH    DE              ;Save new START address
        EX      DE,HL           ;HL=new START address
        ADD     HL,BC           ;HL=new address for label
        EX      DE,HL           ;  now put in DE
        LD      (HL),E          ;New address calculated-
        INC     HL                      ;  now write it into
        LD      (HL),D                  ;  label (address field)
        POP     DE                      ;Restore new START
                                        ;        address
        RET                             ;Back to main routine
```

Immediately following the REPLCE subroutine, your main program should begin. If you have followed directions this far, the first two instructions of your main program will be labeled OFFSET and START. Drop down to the end of your program, and change the END statement to READ

```
        END     INTLZE
```

Your relocatable program is now complete.

When your program is first loaded into the computer under the SYSTEM command, it occupies memory starting at 5800H. This address was chosen to allow room for the DOS if you have a disk system, and yet be low enough to allow several programs to be loaded into high memory. If this isn't suitable for your application, change the ORG address.

You will find that your programs will be much easier to use when they are relocatable (just remember not to reserve any memory before loading, since the program will avoid already-protected memory like the plague). Also, if you have any thoughts about marketing your programs, you may find that they will be easier to sell since one program will fit any memory size (and distributors will like them better since they won't have to stock three or four versions of the same program). If you learn how to use the routines in this article, your programs can be the ones that are "smarter than the average."

From The Source's Mouth

By Joni M. Kosloski

The number of programs available for the TRS-80 today is staggering. And while most cities can boast at least one local software source, that source or those sources probably provide you with only a small glimpse of all that's available. The problem, of course, is that no store can afford to stock the wide variety of software now available for your perusal. And your perusal is important; otherwise, how are you to know which word processor is best for you? Or which DOS? Or which data base manager?

Little needs to be said about the local stores that provide a preview service -- their importance in helping the consumer decide is unquestionable. But even so, there are problems with timing, having the right inventory to solve your needs, and the trained personnel to demonstrate the effectiveness of any given program.

At TAS, about 90 to 95% of our sales are through the mail; programs and products purchased sight unseen. And I don't think I have to tell you that's not always the best way to buy. Sometimes you expect more of the program. Or don't realize that it won't work with your operating system. For whatever reason, there are times when almost EVERYONE will be unhappy with a software purchase.

The problem isn't always the program either. There are programs that I love that some people wouldn't even look at twice -- and visa versa. So what do you (and I) do?

At TAS we have had an implied policy that software can be returned if the user (that's us!) finds it unacceptable. The intention of our column this month is to make that policy explicit. Unfortunately we simply don't have the personnel to adaquately judge all software from respective viewpoints; by formally adopting a return policy the pressure will be on us to look cautiously at a program that has had several returns. In this era of the entrapeauner, ripoffs (both intentional and unintentional) are too frequently overlooked. We are definitely not the first company to offer a money-back guarantee on software, but we applaud the efforts of those who have taken the initial steps to promote this activity.

We do ask that you begin testing a package immediately and inform us of your intentions within 30 days.

May the Source be with you!

ALPHA BYTE STORAGE

is wheelin' and dealin'!

Printers?

We stock 15 different kinds of dot matrix printers. We stock the 3 best letter quality printers — Dec, Qume and Diablo. Check our prices before you make your final decision!

Supplies?

We'll supply you with computer paper, print wheels, labels, forms, print thimbles, and much more!

Diskettes?

We're never undersold! Dig up the best price you can find on 5¼″ Verbatims — then give us a call!

Software? Books? Miscellaneous?

We'd like the opportunity to chat with you about your current needs. And we'll help you find the best deal we can. Give us a call with your interests, or write for a free catalog!

Alpha
Byte
Storage

4636 Park Granada
Calabasas, CA 91302
1-213-883-8594

```
**********************************
*  Bit Kickin' With Jesse Bob    *
**********************************
```

Copyright (c) 1980*

Dear Readers,

Well, here I am, back from making my first movie, 'Urban Bitboy'. It was a lot of work, but I really had a good time doing it. I know that I said I'd run my picture in this issue, but I had so many publicity photos taken that I sprained my grin. Next issue for sure, I promise.

Recently a letter was received containing the disparaging suggestion that Jesse Bob Overholt is a mythical person concocted by Joni and Charley. Boy Howdy, that's the dang-foolest statement I've ever laid eyes on! Why, Joni thinks Willy Nelson is one of Ozzie and Harriet's kids and Charley says that dieing with your boots on is what happens when you push RESET with no disk in drive zero! To set the record straight, let me say here and now that Jesse Bob Overholt does exist and the following is a list of people that he positively ain't:

Charley Butler
Joni Kosloski
Randy Cook
J.R. Ewing
Wayne Green

--------------- (J) ---------------

Dear Jesse Bob,

I've been writing more and more machine language software lately. While I try to write programs which are position dependent, I find it very difficult to perform the address relocation without knowing "where I am" in the code. Is there some way for a machine language program to find out where it is?

Perplexed in Paw Paw

Dear Perplexed,

Some processors have the ability to address memory using offsets to the current value in the program counter register (called program-counter-relative addressing). It is unfortunate that the Z-80 does not have this form of addressing, but it is still possible for a machine language program to find out for itself "Where am I?".

*Copyrighted by the Circle J Software Ranch

The way to do this is by accessing the Program Counter register. No hardware instruction exists to perform this directly. It is possible, however, to get it indirectly using the stack. The Program Counter is placed on the stack by the CALL instructions, the RST instructions, and by interrupt processing. Once the Program Counter is on the stack, it can easily be transferred to another register using POP. A routine exists in the Level II ROM at address 000BH for this express purpose. It is used by issuing a CALL to 0BH. The routine will return with the HL register pair containing the address of the instruction following the CALL. In the following program, when the program gets to HERE the HL register pair will contain the address of HERE.

```
        CALL    0BH         ;Where am I?
HERE    EQU     $
```

Another easy way of finding out "where I am" is built into the Level II USR calling sequence. BASIC does not actually CALL your USR function. Instead it loads the BC register pair with the address of your routine, PUSHes it on the stack, and RETurns to your subroutine. When your program receives control the address at which it was entered will be contained in the BC register pair. A word of caution, however. This is not the case if the USR function was called using a string parameter. In that event, the string handling necessary for parameter passing destroys the BC registers before passing control to you.

Jesse Bob

--------------- (J) ---------------

Dear Jesse Bob,

How the heck do you do a CMD"T" in an assembly language program? I wrote me a super little cassette utility, but if I forget to turn off the clock before I use it, the blasted thing hangs up and I have to reboot.

Troubled in Tuscon

Dear Troubled,

Stay cool, your troubles are over. Add the following code to your utility and you can do a CMD"T" whenever you want.

```
CMDT    LD      A,0C9H      ;A=RET COMMAND
        JR      CMDX        ;GO PROCESS IT
CMDR    LD      A,0C3H      ;A=JP COMMAND
CMDX    LD      (4012H),A   ;PATCH INT VECTOR
```

```
                EI                      ;ALLOW INTERRUPTS
                RET                     ;EXIT SUBROUTINE
```

All you have to do is CALL CMDT to turn off the clock and CALL CMDR to restart it. What CMDT does is to patch the JP instruction which receives control on interrupt to a RET. This causes the interrupted program to resume and leaves the interrupt enable flags set to disable future interrupts. The CMDR routine restores the JP instruction so that interrupts may be processed normally.

Incidentally, one thing many people don't know about the "heartbeat" interrupt in DOS is that it takes nearly one full millisecond (one thousandth of a second) to process each interrupt. Interrupts occur once every twenty-five milliseconds. That means that four percent of the total available processing time is spent handling the interrupts. In very long programs, this can add up. So if you don't need the clock, doing a CMD"T" may save you a little time.

Jesse Bob

--------------- (J) ---------------

Dear Jesse Bob,

Okay, here's my six bucks. Now tell me what the HALT instruction does in Z80 code.

Wondering in Waco

Dear Wondering,

I'm returning your money -- this one's on me. The HALT instruction does just that. It stops the Z80 processor so that something else can use the system bus. Another use would be to synchronize the Z80 with an external event, since once the processor is halted it doesn't start up again unless triggered by an interrupt. That is how the HALT instruction works normally. That is NOT how it works in the TRS-80.

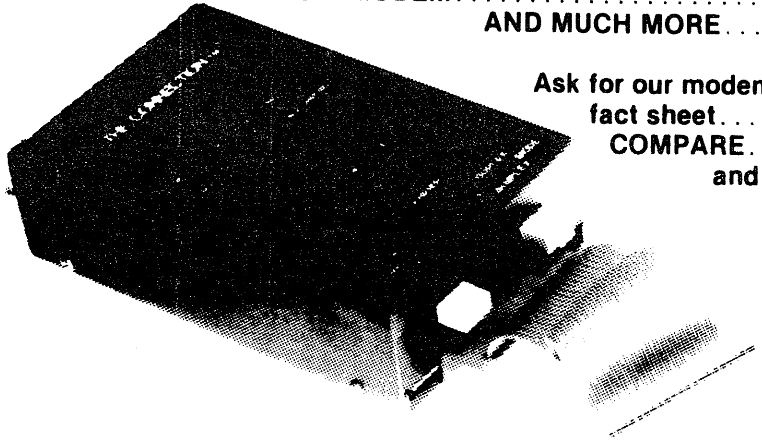
One of the things that HALT does is send out a signal that tells the external devices it is halted and they may use the bus. The sneaky devils in Fort Worth tied this line to the non-maskable interrupt pin of the processor. This, in effect, makes the HALT instruction a kind of software RESET, since the effect will be the same as if the RESET button were pushed. That's why sometimes when a machine language program "gets lost" and gallops through memory the system reboots. The runaway program found a HALT.

Jesse Bob

SAVE 50% AND

GET MICRO CONNECTED Plugs directly between ALL S80'S (Level I,4K to MOD II,64K) and the telephone network. To compliment the DIRECT CONNECT MODEM, a seperate RS-232 jack ALSO operates your serial printer on OR off line. With THE MICRO CONNECTION™ you have accurate 300 baud two-way and simplex communications between you and the computer data base of your choice. In ADDITION, you have radio communications and extensive software options and packages. **NO NEED** for the expansion interface, acoustical coupling modem, RS-232 board or cableing. That is why **THE MICRO CONNECTION™ IS A MODEM**.....................

AND MUCH MORE...!



Ask for our modem fact sheet....
COMPARE..
and.

***Then* get your MICRO CONNECTION™ for only $249⁰⁰!**

the microperipheral corporation

PO Box 529 ■ Mercer Island ■ WA 98040
206 454 3303

VISA

Round 2: VTOS 4.0

The competition heats up.

The last issue of TAS (Aug., 1980) featured my review of NEWDOS-80 in which I expressed considerable enthusiam about its new features and capabilities. I am also enthusiastic about VTOS 4.0 (VTOS = Virtual Technology, Inc.), but my enthusiasm is a bit dampened, as it was with the original VTOS 3.0, by the somewhat inadequate documentation. The documentation consists of a skimpy 5 X 7 manual with 42 pages, plus 2 errata sheets. For an advanced and complex DOS with many unique and new features, this little green manual just doesn't do the job. Admittedly, most --but not all-- of the necessary operational guidance is crammed into this guidebook. Nevertheless, I had to spend many hours experimenting with various commands to figure out what the guidebook was trying to tell me.

A complaint often shared among college students concerns the professor who is so educated that he has difficulty communicating with his students at their own level. Perhaps this is what we are up against with Mr. Randy Cook, the VTOS author. He is obviously an extremely competent programmer, but he is weak in communicating the application of his DOS software to those of us who are less experienced.

I hasten to add, however, that Mr. Cook promises a comprehensive "Master Reference Manual," possibly sometime in November (optional at about $30). I urge the purchase of this manual, even for experienced users. I have not seen the expanded manual, but hopefully it will be more illuminating than the little green reference manual.

So much for my bickering. Now, what does the much heralded VTOS 4.0 system do? Briefly, the most significant capabilities can be summarized as follows:

1. VTOS is "device independent," meaning that data can be routed very easily between devices. Devices are defined as keyboard, display, printer, job log, and more. Additional devices can be defined by the user. A communications line (*CL) is one example.

2. It employs a unique method of CHAINing, which can be described as a third mode of programming (DOS would be the first mode and BASIC the second).

3. It can be configured to handle virtually any combination of disk systems, from thirty-five track, single density, to

80 track, double sided, double density. It can also handle hard disk drives for those of you who need massive amounts of data storage.

4. It can be configured to handle a variety of different computer systems and user preferences. For example, lower case, blinking cursor, etc.

5. A number of new utility features are added, some of which are carried forward from the original VTOS 3.0.

Now for some details.

COPY PROTECTION

The original VTOS 3.0 used a "master" disk with a "bad" sector (four) which prevented making DOS copies by traditional means (e.g., TRSDOS, NEWDOS, or SUPERZAP). An operational "system" disk was made from the "master" disk, and it too had a "bad" sector four. This was Mr. Cook's way of preventing the pirating of his software: the system disk would not back itself up and additional system disks could only be created by the owner's master disk.

Conflicting rumors preceded the release of VTOS 4.0: did it, or did it not, have the a sector as a form of copy protection? The answer straight from the shoulder is, "yes and no." VTOS 4.0 still has a bad sector four (it is actually numbered as sector '7C') which prevents making backups by normal means. Only a VTOS 4.0 disk can create another VTOS 4.0 disk. But this time there is a difference from the original VTOS 3.0. Any number of duplicate disks can be created with the original disk, and additional copies can be created with these second generation copies. In other words, if I chose to give someone a copied VTOS 4.0 disk (which of course I won't), that individual would still have the capability to make additional copies to give to a someone else, and so on.

I can't understand the reason for this unusual disk formatting because it serves only to pain my backside. It certainly does not prevent unauthorized copying or pirating. My "yes and no" answer above should now be clear: yes, the "bad" sector four is still there, but no, it does not prevent making backups. Is it merely an unnecessary irritant?

DEVICE INDEPENDENCE

The concept of device independence was introduced to

TRS-80 owners in the original VTOS 3.0, so it is not new to VTOS 4.0. VTOS 3.0 owners are no doubt aware of the power and usefullness of this unique feature, but for the benefit of newcomers to the VTOS system, here is an example of how it can be used. The command: <ROUTE *PR TO *DO>

will direct any program data intended for the printer (*PR) to the display (*DO). It is also possible to: <ROUTE *PR to filespec>

where 'filespec' is the user's selected name for a disk file. The benefits of this capability should be obvious. If your printer is out of commission, it is a simple matter to route all program printer output to either the display or to a disk file for later printout. If printer output is wanted on both the display and a disk file simultaneously, the following command does the trick:

<LINK *DO to filespec> followed by:

<ROUTE *PR to *DO>

The symbols *PR and *DO are called "device specs" and can be used in virtually identical fashion as the more familiar disk "filespec." Almost anything done with a filespec can also be done with a device spec.

The above ROUTE and LINK commands can also be employed with both the Electric Pencil (by Michael Shrayer Software) and SCRIPSIT (by Radio Shack). Patching routines for both word processing programs are provided in VTOS 4.0 to enable the device-independent capabilities. The patches are easily installed following instructions provided in the little green book.

It was mentioned above that the user can define special device specs. For example:

<SET *CL TO RS232 (BAUD=300)>

In this example, a device spec is user-defined as *CL (CL = Commmunications Line), and it utilizes a disk file called RS232/DVR (included on the VTOS 4.0 disk. DVR is an abbreviation for Driver, and 300 BAUD specifies the data transmit rate to the *CL). After this device spec is defined, any programming reference to *CL will automatically utilize the RS232 program for its functional guidance.

For the user who desires a more extensive explanation of "Device Indepence," recent issues in PROG-80 (published by TSE - The Software Exchange) contain some useful explanations by Lance Micklus. I do not have these copies, so the reader will have to search 1980 back issues for these reviews.

CHAINing

New and expanded CHAINing capabilities provide a unique third mode of programming capability. By a third mode of programming, I am suggesting that several logical and specialized programming commands can be implemented in a CHAIN file, which essentially makes the file into an executable program. As an example, part of an actual CHAIN file from the VTOS 4.0 disk is reproduced in figure one. I added some line numbers to ease the task of following the logic of the CHAIN file and explaining how it works.

Line 1 is simply a file title. Line 2 with the two slash marks (//) is a decision making or functional command for the system. If the command received is a BACKUP, the comments on lines 3, 4,and 5, preceded with a period, are displayed on the screen. Line 5 initiates the actual BACKUP from drive zero.

If, instead, the SCRIPSIT PATCH routine was selected, the CHAIN operation will bypass the BACKUP instruction and continue down until it finds the "// IF SCRIP" logic as found on line 23. Everything prior is ignored. Again, decision logic is represented by the double slashes, comments to the screen are preceded by a period, and the actual operation to be performed is invoked by the command line which is not preceded by any character. In this case the executed command will be:

'PATCH SCRIPSIT/LC SCRIPSIT/FIX.

The SCRIPSIT patch does several things. High Memory will be honored, hardcopy printout will be via the VTOS DCB (Device Control Block) rather than via SCRIPSIT's own printer driver code, and SCRIPSIT becomes capable of Device Independence as described above. PENCIL can be patched in the same way. Many additional logical expressions are available for creating different CHAIN files. Some self-evident examples include IF, ELSE; SET or RESET a symbol or variable; FLASH a message to the screen; DELAY, perhaps for user response; AND, OR, and NOR; and KEYIN, which waits for a user keyboard input.

Perhaps it is clear now why I referred to the CHAIN function as a "third mode" of programming. CHAIN programs similar to the one described here can be developed to perhaps control machinery with computer reference to clock times, introduction of delays, and invoking specialized control programs.

There is more potential here than I can cover in a limited space, so experiment and see what happens.

SYSTEM CONFIGURATION

A 'SYSTEM' command enables the user to customize the VTOS disk with several options: fast or slow CPU clock (kits are available to speed up the Z80 CPU clock from several sources); large or small cursor, blinking or non-blinking; type-ahead, on or off (very useful - keyboard input can be done at any time, even during disk I/O); lower case driver, on or off; disable or enable any disk drive; 'JKL' screen print option, same as that implemented by Apparat's NEWDOS; variable disk head step rate; and disk motor delay time before disk I/O begins.

These options can be made into a permanent file with the command - 'SYSTEM (SYSGEN)' - which, after it is created, loads automatically when a BOOT or system RESET occurs. This command actually creates a file named CONFIG/SYS which executes automatically after the initial BOOT or RESET and it patches the DOS resident code to activate the user-selected options. The automatic CONFIG/SYS file can be defeated during the BOOT process by pressing the CLEAR key. Also, a different SYSTEM configuration can be re-defined at any time.

The unique thing about this feature is that the system customizing is done via a separate program file without a requirement for making more permanent changes (or ZAPS as we have come to know them) to the disk SYSTEM files. Not only does this method retain the integrity of the disk SYSTEM files, it also makes it very simple for the novice programmer to implement the customizing features. Consequently, there is less chance of accidently ZAPping erroneous codes to the disk SYSTEM files.

OTHER NEW FEATURES

VTOS 4.0 boasts what is called a "symbiont" spooler. The word symbiont derives from "symbiosis," meaning "...the intimate living together of two dissimilar organisms in a mutually beneficial relationship." Obviously, there are no organisms involved in VTOS. Rather, the implication is that the spooler is symbiotic with VTOS by virtue of its integrated design. In short, there are no incompatibility problems as might be encountered with the purchase of a spooler sold by another distributor.

For the benefit of newcomers, a spooler permits running the computer's printer simultaneously with the running of another program, even allowing keyboard input while the printer is doing its thing.

A "wild card" feature permits copying, killing, or getting a DIRectory of files by class, such as /BAS, /SYS, and /PCL. The file reference need not be only in the filespec extension. The class might be, say, INV, which would reference all files with INV imbedded, like INV1, INV2 and so on.

A MEMORY command can be implemented from DOS to define the high memory limit available to any program. This is similar to the NEWODS-80 HIMEM command.

DISK BASIC is not provided on the VTOS 4.0 disk. Rather, the user must provide TRSDOS 2.3 BASIC, which, in turn, is PATCHed for use with VTOS. Simple copying and PATCHing instructions are provided in the documentation.

New BASIC features include: DOS commands which can be issued by entering a CMD "command" (for example, CMD"DIR" will call up a directory from BASIC. NEWDOS users had this capability when NEWDOS 2.1 was first released some time ago); screen printing is supported; an OPEN"E" allows appending to an existing file (another NEWDOS innovation); variable length file support which makes BASIC file I/O considerably easier to mmanage; "Run only" BASIC program protection; and BASIC line renumbering.

SOME PROBLEM AREAS

Password Protection: VTOS 4.0 provides no simple method of defeating the system's password protection. The VTOS disk master password is given as "PASSWORD," and this enables access to most VTOS 4.0 disk files. But attempting access to another disk's password protected file just gets "FILE ACCESS DENIED" printed on the screen. For those who are interested, the VTOS disk password checking can be defeated by using SUPERZAP (available from Apparat, the producers of NEWDOS) to ZAP SYS2/SYS as follows:

```
SYS2/SYS:  01/C9  (relative sector one, byte C9):
           change: E1 28 2D
               to: E1 18 2D
```

MEMORY (High$) Protect: VTOS 4.0 uses high memory for many of its functions, but the user can define memory's upper limit (similar to a BASIC MEMORY SIZE statement) so as to protect his own programs which might also reside at the top. However, there is a problem. If you have a program, printer driver, or whatever which occupies top memory, it will conflict with the SYSTEM (SYSGEN) configuration file described above.

For example, suppose it is desired to configure a SYSTEM with a small, blinking cursor and a lower case driver. The CONFIG/SYS file which is created to implement these selections will automatically load to high memory everytime the TRS-80 BOOTs up. If I intend to put a program of my own in high memory, I will have a problem because after the BOOT or RESET, the VTOS lower case driver is already up there. When I load my program, it will crash the whole system because it clobbers the lower case driver.

The VTOS 4.0 documentation implies, I think, that I should be able to overcome this problem by somehow including a HIGH MEMORY statement within the SYSTEM configuration defined above. But so far, I have not been able to do this successfully. The procedure is either inadequately covered in the documentation, or it is a VTOS defect. Consequently, if I intend to load any programs into top memory, it will be necessary to bypass the automatic loading of the SYSTEM configuration file by holding down the CLEAR key during the BOOT or RESET. Subsequently, HIGH MEMORY is manually set via the MEMORY command and my program in top memory will be protected. If I then want the VTOS small, blinking cursor, and its lower case driver, it will be necessary for me to manually reconfigure with another SYSTEM configuration operation. After setting up this new configuration, the <SYSTEM (SYSGEN)> command will not create the new CONFIG/SYS file as it should. Instead, the disk drive simply sits there and spins continuously.

The VTOS documentation also says, and I quote, "User may SYSGEN a custom VTOS system configuration containing special I/O drivers, device LINKing and ROUTEing, SPOOLing, and DEBUG tasks, etc. which will be automatically loaded during the BOOT process." Sounds good, but the documemtation stops there; it does not explain how to do this with the SYSGEN command. (The documentation does, however, describe how to do this with a CHAIN operation which is different than the SYSGEN operation).

That covers most of the special VTOS 4.0 features. There are a few more goodies, and of course there are the standard DOS commands with which we are all familiar.

It is my hope that the NEWDOS-80 and VTOS 4.0 reviews presented in TAS will give readers some guidance on what to expect from these systems. But keep in mind that most likely I have either missed something useful or something problematical. Readers are encouraged to let the rest of us know about your discoveries, both good and bad. Send your comments in to TAS so that we all may share. Thanks.

FIGURE 1

```
 ... VTOS 4.0 AUTO-INITIALIZED CHAINING FILE
 //IF BACKUP
.THIS IS YOUR  M-A-S-T-E-R  VTOS 4.0 DISK,
.LET'S MAKE A  S-Y-S-T-E-M  DISK TO RUN FROM,
.     OK? ...
BACKUP :0
//STOP
//END
//IF BASIC
.HAVE YOU LOADED YOUR OWN COPY OF BASIC 2.2 OR 2.3?
//PAUSE  <ENTER>  WHEN READY
PATCH BASIC/CMD.BASIC BASIC/FIX
.OK, YOU ARE ALL FIXED UP NOW.
.ALSO, READ 'BASIC/DOC' FOR INFO ON TWO NEW BASIC
FEATURES
//EXIT
//END
//IF PENCIL
.HAVE YOU LOADED YOUR OWN COPY OF PENCIL ?
//PAUSE  <ENTER> WHEN READY
PATCH PENCIL/CMD PENCIL/FIX
//EXIT
//END
//IF SCRIP
.HAVE YOU LOADED YOUR OWN COPY OF SCRIPSIT ?
.DO YOU WISH TO PATCH SCRIPSIT/UC OR SCRIPSIT/LC ?
//KEYIN  1= UC,   2= LC
//1
PATCH SCRIPSIT/UC SCRIPSIT/FIX
//EXIT
//2
PATCH SCRIPSIT/LC SCRIPSIT/FIX
//EXIT
//END
```

********** ********** ********** **********

The letter on your mailing label is indicative of which issue your subscription expires with. "A" means this issue will be your last, "B" means issue 7 will be your last, and so forth. Persons wishing to can renew at any time; persons expiring with issue #6 (this one!) will be receiving notice and a renewal order blank within the next 3 to 4 weeks. When renewing, it is best to indicate such, therefore avoiding double mailings, etc. It would help the girls if you let them know when you expire, too. Thanks!!

Undocumented Z-80 Opcodes

By Daniel R. Lunsford

This is a complete list of the undocumented opcodes executed by the Zilog Z-80 microprocessor. Use of these opcodes can speed software development by making previously cumbersome operations relatively easy.

The first group of instructions is represented by the mnemonic "SLS", for which I am indebted to Allan Ashley of Pasadena, California. The action of this instruction is to shift its operand left and then set the least significant bit. Alternatively, the operand is multiplied by 2 and incremented. Any shift out of the high bit is put into the Carry flag, and sign and parity are affected in the expected manner.

Mnemonic	Hex Code	Octal Code
SLS A	CB 37	313 067
SLS B	CB 30	313 060
SLS C	CB 31	313 061
SLS D	CB 32	313 062
SLS E	CB 33	313 063
SLS H	CB 34	313 064
SLS L	CB 35	313 065
SLS (HL)	CB 36	313 066
SLS (IX+d)	DD CB dd 36	335 313 ddd 066
SLS (IY+d)	FD CB dd 36	375 313 ddd 066

The next major groupings involve the Z-80's twin index registers, IX and IY. These registers are 16-bit pointer registers and can be thought of as extensions of the HL register pair in capabilities. However, unlike the HL pair, the official documentation from Zilog did not divide the index registers into two 8-bit registers each, with the result that many desirable operations with the index registers became remarkably clumsy to implement. It has been discovered, however that the index registers are indeed analogous to the HL pair, and are divided into two 8-bit high and low registers. Opcodes are formed by attaching a precursor byte (0DDH for an IX operation, and 0FDH for an IY operation) to an opcode belonging to the 8080 compatible subset of the Z-80's instruction set.

In the following examples, HX refers to the register formed by the upper 8 bits of the IX register, LX to the lower 8 bits and similarly for HY and LY. The pattern of generation should be obvious.

Desired instruction	Object code	Standard assembler
LD HX,A	DD 67	DEFB 0DDH LD H,A
XOR LY	FD AD	DEFB 0FDH XOR L
INC HY	FD 24	DEFB 0FDH INC H
ADD A,LX	DD 85	DEFB 0DDH ADD A,L
LD HX,LX	DD 65	DEFB 0DDH LD H,L

For those interested in timing, these extra instructions are precisely four (4) T-states longer than the corresponding H or L codes; e.g., timing for ADD A,L is 4 T-states, while for ADD A,LX is 8 T-states.

The following is a complete list of the extended opcodes guaranteed to run on any Z-80. Like all micro chips, the Z-80 has "ghost" instructions which do one thing on one chip, and another, radically different thing on another. These extra index register instructions do not fall into this class.

ADC A,HX	ADD A,HX	AND HX	CP HX
ADC A,LX	ADD A,LX	AND LX	CP LX
ADC A,HY	ADD A,HY	AND HY	CP HY
ADC A,LY	ADD A,LY	AND LY	CP LY
DEC HX	INC HX	OR HX	SBC A,HX
DEC LX	INC LX	OR LX	SBC A,LX
DEC HY	INC HY	OR HY	SBC A,HY
DEC LY	INC LY	OR HY	SBC A,LY
	SUB HX	XOR HX	
	SUB LX	XOR LX	
	SUB HY	XOR HY	
	SUB LY	XOR LY	

The following is the LOAD group for the new registers. Note the H and L cannot be loaded from or to the new registers, and that things such as LD HX,LY are not permissible.

LD HX,A	LD HY,A	LD A,HX	LD A,HY
LD HX,B	LD HY,B	LD B,HX	LD B,HY
LD HX,C	LD HY,C	LD C,HX	LD C,HY
LD HX,D	LD HY,D	LD D,HX	LD D,HY
LD HX,E	LD HY,E	LD E,HX	LD E,HY

```
LD LX,A       LD LY,A       LD A,LX       LD A,LY
LD LX,B       LD LY,B       LD B,LX       LD B,LY
LD LX,C       LD LY,C       LD C,LX       LD C,LY
LD LX,D       LD LY,D       LD D,LX       LD D,LY
LD LX,E       LD LY,E       LD E,LX       LD E,LY

          LD HX,N           LD HX,LX
          LD LX,N           LD LX,HX
          LD HY,N           LD HY,LY
          LD LY,N           LD LY,HY
```

(Editor's Note: The Microsoft Macro-80 assembler allows generation of all of the above missing index registers. Dan has a macro available in listing form, with complete documentation and a list of the instruction sets covered, available for $10.00. Interested persons should write to DarkStar Microsystems, 8725 La Riviera Drive, No. 68, Sacramento, California, 95826 to obtain the listing.)

WE'RE HAVING A CONTEST!!

Big prizes!
Lots of excitement!
(Maybe...)

...it all depends on you.
We had a problem deciding on a good theme for our contest. Bill Brown finally came up with this idea:

Who can come up with
THE BEST CONTEST IDEA???

The rules? The contest must involve the TRS-80.
The prize? Not a trip to Petersborough. Nope.
The big one is a lifetime subscription to TAS!
The deadline? December 31, 1980.

Submit your entries to:

THE ALTERNATE SOURCE
1806 Ada Street
Lansing, Michigan 48910

THE ALTERNATE SOURCE

Introducing....

......60 Cycle Sine Wave U.P.S.

(Uninterruptibe Power Supply)

....for those systems that need 60 cycle sine wave keeps computer & disk systems on when the power goes out **For less than $500.00***

....Rated 250 Watt continuous operation

.... will provide 400 Watts for 5 minutes (Model II TRS-80 with 4 disk drives)

Mayday™

* Standard MAYDAYS available starting at $195.00 for 150 Watt

from...

Sun Research, Inc.
Box 210 New Durham, NH 03855
(603) 859-7110 TWX 510-297-4444

Floating Point USR

By Mark T. Longley-Cook

Floating point variables can be transferred between BASIC and machine language programs. The purpose of this article is to point out two ways of doing this -- one fairly obvious and one not so obvious. Finally, a simple method of transferring 7 bytes (less than one bit) of information will be identified.

The more obvious method of transferring variables is to use USR(VARPTR(X)). The low-memory byte address of X can then be accessed into HL in the machine language program by CALLing 0A7FH. Saving HL will permit storing a new floating point variable as X prior to RETurning to BASIC. This method can be used to advantage when X is an array element, as then other elements in the array can be accessed simply. There are some shortcomings. The returned variable must be the same type as that passed; X cannot be an expression; the value of the USR function will be useless if it has no significance; VARPTR must not be forgotten when typing the program; and the machine language program is complicated by not having the variable where it is most useful for floating point operations, that is, in locations 411DH to 4124H. These locations serve as a floating point accumulator for ROM routines.

A new, and in many ways better, method for transferring floating point variables is based on the normal entry points for transferring integers into and out of USR routines (0A7FH and 0A9AH). The first location is also the entry for the BASIC function CINT. The second location is also the entry for a subroutine to load the floating point accumulator with an integer in HL. Try the following two-liner and you will have an idea of what I mean.

```
10 POKE 16526,67: POKE 16527,0  or  DEFUSR = 67
20 INPUT X: PRINT USR(X): GOTO 20
```

Location 67 (43H) in ROM contains 0C9H, an unconditional RETurn instruction. Inputting any value will result in the computer parrotting that value back. The machine language program can be as complex as desired. After accessing X in the floating point accumulator, it can perform ROM or non-ROM floating point operations. It can also return a floating point variable to BASIC by storing it in the accumulator, setting the type flags in location 40AFH if necessary. For example, here is a quick and simple (no overflow checks) multiply/divide-by-two routine for single or double precision variables.

```
          LD      HL,4124H          ;EXPONENT BYTE ADDRESS
          LD      A,(HL)            ;EXPONENT TO A
          OR      A                 ;TEST FOR ZERO
          RET     Z                 ;  AND IF SO, RETURN
          INC     (HL)              ;MULTIPLY BY TWO
(or...)   DEC     (HL)              ;DIVIDE BY TWO
          RET
```

Of course, *2 or /2 is simpler, but not as quick!

Finally, there is an alternate way of transferring several bytes of information to the machine language program. The normal method is to POKE the information into RAM, but an easier way is as follows. Let I, J, K, L, M, N, O be seven numbers between 0 and 255 ("O" can only go to 127). Then set Z# = 36028797018963968. USR(Z#+I+256*(J+256*(K+256*(L+ ...)))) places the following information in the floating point accumulator:

LOCATION	CONTENTS
411DH	I
411EH	J
411FH	K
4120H	L
etc.	etc.

The variable "O" is restricted to 127 because the high order bit in location 4123H is the sign bit of the floating point number. Location 4124H will contain a constant exponent of 184. If less than seven bytes is required, the inner parenthesis will collapse, making typing the expression simpler. If the ranges required exceed 255, the change is simple and can best be shown by example. USR(Z#+I+65536*(J+ ...)) will reserve 411DH and 411EH for I (in reversed LSB/MSB order of course). Here 65536 is the square of 256. What is Z#? It is two to the 55th power, and 55 is 7*8-1 or the number of bits that are transferred. But don't try to calculate Z# in the program; the BASIC routines are not set up to handle that sort of precision. Z# forces alignment of integers, like "I" above, with its least significant bit for easy decoding. Transferring information back to BASIC using this scheme is not quite so simple, but it can be done. The machine language program should set the type flags (40AFH) to 08H, set the exponent (4124H) to 184 (0B8H) and reset the high order bit location 4123H. The BASIC program then subtracts Z# from the USR function. The following statements are typical of what is required:

```
10 DEFINT I-O: DEFDBL T-Z: Y=USR( ... )-36028797018963968
20 T =FIX(Y/256): I =Y-256*T: Y =FIX(T/256): J =T-256*Y
```

Alternatively, a FOR-NEXT loop might be used (with an extra statement Y=T) assigning the bytes to an integer array.

In summary, a new, straightforward method of passing floating point numbers between BASIC and machine language exists. What is done with the number once in the machine language program depends on your manipulation of floating point ROM routines -- a subject too large and too complex to cover here. The only disadvantage the procedure has is that more than one floating point number cannot be passed at once, as arrays indirectly can with VARPTR. Additionally, a multiple-byte transfer scheme has also been described. In specific applications, these methods can be extended as desired.

SPEEDING UP A SEQUENTIAL SEARCH

By Joni M. Kosloski

(Before we begin, proper credit should be given to Donald Knuth. The techniques described within this article were obtained from reading Volume 3, Chapter 6 of "The Art of Computer Programming," Sorting and Searching, which is written by Donald Knuth and published by Addison Wesley.)

"Start at the beginning, test each record till you find a match, then stop." In the English language, that just about sums up a sequential search. And it's relatively easy in the Basic language, too, except that some methods of sequential searching are faster than others. Since searching is the most time consuming part of many programs, the substitution of a good search method for a bad one can often lead to a substantial increase in speed. This article will describe three different methods, discuss differences and attempt to explain the process of speeding up the search.

PROGRAM ONE

In all of the following examples, we are assuming that we have a set of data records stored in the array "R", which are labeled R(1), R(2), R(3)....R(N) where "N" is the total number of data records available. These records, in turn, each have associated with them a set of unique keys, which we will use to locate the correct record. The keys are K(1), K(2), K(3)....K(N). The variable "I" will be used as a counter, the variable "S" as a switch to indicate whether or not we have found the record for which we are searching and the variable "K" is the key value for which a match is being sought.

With that in mind, the most obvious search algorithm is as follows:

```
100 I = 1
110 IF K = K(I) THEN S = 1: PRINT "FOUND IT!": END
120 I = I + 1
130 IF I <= N THEN GOTO 100
    ELSE S = 0: PRINT "IT'S NOT HERE!": END
```

Note that the program can end in one of two ways -- by finding the right key and terminating, or by searching the entire file, deciding that the right key is missing, and terminating unsuccessfully. If the program finds the match it's looking for, "S" will be equal to "1"; if not, then zero.

The running time of this program depends on two things; the number of comparisons being performed and whether the search was successful. Referring to Donald Knuth, the formula for computing the processing time of the above program is:

$$5 * C - 2 * S + 3$$

where "C" is equal to the number of comparisons being made and "S" is either zero or one, depending on whether a match is found. In a successful search, where "K" was found to be equal to "K(I)", "I" is the number indicating the record position within the file and will also be equal to the number of comparisons being made. So we can use "I" in place of "C" in the above formula. "S" will be "1" because we did find the record we were looking for. Knowing this, the total units of time can be determined to be:

$$5 * I - 2 * 1 + 3$$

or

$$5I + 1 \text{ (units of time)}$$

If the search was NOT successful, "I" would equal "N", the total number of records (having searched them all), and "S" would be equal to "0". Using the same algorithm, an unsuccessful search would take:

$$5 * N - 2 * 0 + 3$$

or

$$5N + 3 \text{ (units of time)}$$

Let's see if we can't speed up this 'searching program' just a little. Noting the original formula, it's easy to determine that the number of comparisons being made has a great deal to do with how much time is being consumed.

PROGRAM TWO

The only difference between Program One and Program Two is that we have inserted a 'dummy' record at the end of our file. Note the following code, then I'll explain the purpose of the 'dummy' record:

```
100 I = 1: K(N+1) = K
110 IF K = K(I) THEN 130
120 I = I + 1: GOTO 110
130 IF I <= N THEN S = 1: PRINT "FOUND IT!": END
    ELSE S = 0: PRINT "IT'S NOT HERE!": END
```

```
        ELSE
           S = 0: PRINT "IT'S NOT HERE!": END
```

If you go back and look at Program One, you'll find that TWO comparisons are made for each record being searched. The first comparison will determine if it is the record we are looking for, the second comparison will determine whether or not it's the last record in the file.

Now note that in Program Two, only ONE "IF" statement is being executed for each record in the file. What we have accomplished is cutting in half the number of comparisons being performed. The only problem with arranging the statements in this order is that we must FORCE at match when we reach the end of the file, otherwise we'd be hung up in an endless loop of execution or the program would terminate with some kind of error. This is where the dummy record comes in. If we have 20 records in our file, then we make the the 21st record our dummy record. "N" would still equal 20, the actual number of records available. So when we set "K(N+1)" equal to "K" in line 100, what we're doing is forcing a match in line 110 when we reach record 21 (end of file).

Cutting the number of comparisons in half makes use of an important "speed-up" principle: when an inner loop of a program tests for two or more conditions, an attempt should be made to reduce it to just one condition.

Making use of the same variables "C" and "S" from Program One, the new formula for determining the processing time is as follows:

$$4 * C - 4 * S + 10$$

Again, assuming our search was successful, "C" will be equal to "I" and "S" will be equal to "1". Total elapsed time for a successful search using Program Two computes at:

$$4 * I - 4 * 1 + 10$$

or

$$4I - 6 \text{ (units of time)}$$

Can we speed it up even more?

PROGRAM THREE

Let's take a look at the following program listing, which still assumes the 'dummy' record at end-of-file:

```
100 K(N+1) = K: I = -1
```

```
110 I = I + 2
120 IF K = K(I) THEN 140
130 IF K <> K(I+1) THEN 110: I = I + 1
140 IF I <= N
       THEN S = 1: PRINT "FOUND IT!": END
    ELSE
       S = 0: PRINT "IT'S NOT HERE!": END
```

In line 100, we again force our dummy record to be equal to "K", thus terminating the routine at the end of the file. We also set "I" equal to "-1" because we're going to be increasing each step of our loop by 2 and we want to begin the search with record number one. Line 120 will test record one for a match; if found, it drops to line 140 which terminates the routine. If it is not a match, line 130 will test record "I+1", or record three in this instance. (Please note that line 130 is NOT increasing "I", it is just adding "1" for testing purposes!) If line 130 doesn't find a match, then back to line 110 for an increment. If it does find a match, then the value of "I" must be increased by one to reflect the actual record being tested before proceeding to line 140 for termination.

By using this set of instructions, we have duplicated the inner loop. In doing this, we have eliminated one half of our "I=I+1" instructions. You'll probably notice that there are two "IF" statements in the above coding, but each loop is testing TWO records, not just one.

So what modifications have we made to our original Program One? In Program Two, we eliminated one half of our "IF" statements. In Program Three, we eliminated one half of our "I=I+1" instructions. These two changes have resulted in a saving of a full 30% processing time when compared to the original Program One!

****

TAS is published by Joni M. Kosloski and Charley Butler at 1806 Ada Street, Lansing, MI, 48910. Subscriptions are $9.00 per year USA; $12.00 per year Canada and First Class; and $15.00 per year Foreign. All monies must be in US currency. Subscriptions may be addressed to the above, as well as advertising rate inquiries. We actively solicit meaningful articles and information relating to Tandy's TRS-80, and compensate for each article published. Call or write for more info. The entire contents of this issue are Copyright (c) 1980 by The Alternate Source unless copyright notice is declared by individual authors.

****

AFTERWARD FOR ISSUE 6

I had never seen any reference to undocumented Z-80 opcodes prior to Dan Lunsford's article in this issue, thus I believe it to be a first.

This is our last issue printed by a "Quick Printer", mainly because of inconsistent print quality.

We have had follow up articles to the "Make Your Machine Language Programs Relocatable". The subject sparked quite a bit of interest -- and criticism.

I guess my all time goal is to get William Barden and Jesse Bob Overholt to collaborate on an article. "Bit Kickin" doesn't bring out Jesse Bob the way he "really is". Another Way to Install Machine Language" has got to be one of my all time favorites. If they ever do team up, watch out world!

$14.95

THE ALTERNATE SOURCE
1806 ADA STREET
LANSING, MICHIGAN 48910
TELEPHONE (517) 487-3358