# TRSTIMES

Volume 2. No. 3. - May/Jun 1989 - $4.00

**Reverse video for Model III**

**Transfer MacIntosh Graphics to the TRS-80**

**Reviews - CP/M column & much more**

# LITTLE ORPHAN EIGHTY

Rather than getting on a soapbox to rant and rave about something or other in the TRS-80 world, let me sit back, relax, and reminisce about the early days spent with the Tandy machines.

It all started when I retired from professional soccer in 1972 and began playing music for a living. Having to work only a 4-5 hours per night left much time to cultivate alternate interests. While some of the guys in the different bands I was with over the years spend their time with alcohol and drugs, I got hooked on calculators. As soon as a new model came out, just a little better than my current one, I bought it.

When the early computer kits appeared in the electronic stores, you guessed it, I brought one home. However, not being blessed with an abundance of talent holding screwdrivers, soldering guns, etc., I eventually managed to blow it up. This cooled me off electronics until 1978.

I left the music business in late 1977, moving to Tampa, Fla. to run motels. Tampa had just gotten a franchise in the North American Soccer League (NASL) and, being a fan, I bought season tickets. It didn't take me long to develop a 'game-day' routine. I would leave home three hours before the game, park across from the stadium in the shopping mall where I would have dinner. As luck would have it, next to the restaurant was a Radio Shack. After dinner I would look at all the neat gadgets, but for some time I stayed on the wagon. I went to the game without buying anything.

Then one day I saw the biggest calculator of them all. The salesman called it a TRS-80. He explained that this was a real computer; it could do thousands of calculations in a matter of seconds, it could play games and, best of all, it could be programmed to do almost anything imaginable. The salesman was so courteous, so friendly, so knowledgable.

No, I didn't buy the computer right then. Instead, I went to the game, but spent the next few weeks thinking, dreaming and drooling about it. Each 'game-day' found me back in the Radio Shack store, learning a couple of things about this new wonder.

I finally broke down. My thirst for calculators had been re-awakened and would be denied no longer, so I wrote a check for what was to become the Model I. The next month was spent completely absorbed in this little silver and black monster, sleeping only when I absolutely no longer could stay awake, learning as much as I could, as fast as I could. This was heaven.

I had many, many questions which resulted in several trips back to the store to pick the brains of my friendly salesman. Each time back, it seemed as though he got a little less friendly and a little less knowledgable. Also, being busy enticing other poor souls with the wonders of the TRS-80, he did not have much time to answer my questions. However, my problems worked themselves out and I kept on computing into the wee hours of the mornings.

I was now programming in Basic, actually making the computer do things. My first attempt at writing a real program ended in disaster. It was a bookkeeping program that would have worked nicely, except for one thing. Somewhere in the middle of the program was a routine that closed the motel income accounts for the day, transferring all monies to a variable that was named, of all things: NEWBALANCE.

I had not yet saved my program to tape and before doing so, I proceeded to view my masterpiece. Data after data was input, everything working just fine. *Hey, I really had something here*. Then came the closing sequence with the cursed variable - POOF - everything gone. Suffice to say, I have never used long variable names since.

More months passed. Still plenty of questions, but by now I had figured out that Radio Shack just was not the place to get the answers. My Basic programming was coming along nicely; by trial and lots of error, I could now write programs without consulting the manual - some of them even worked!

In the early eighties I moved to Fresno, Ca. to set up the new corporate offices of the motel chain. There the Model I was sold, replaced with a Model III and a Color Computer. Working with a DOS that didn't crash at regular intervals was enjoyable, and the CoÇo held my interest long enough to write a game called 'CHANGO' which I submitted to 80-US. Amazingly, they bought it. I was on cloud 9. Imagine, somebody was paying me for writing code.

I immediately wrote another 10 or 20 programs and sent them to the various magazines. My euphoria was quickly stopped. Each came back with a rejection slip. Well, so much for the CoCo. It was eventually put on a shelf and my efforts were now solely geared to the Model III. The money made on 'CHANGO' was spent buying Radio Shack's Editor/Assembler 'EDTASM'. I wanted to learn Assembly language. However, being out of space, this story will have to wait for a future issue.

And now.............Welcome to TRSTimes 2.3

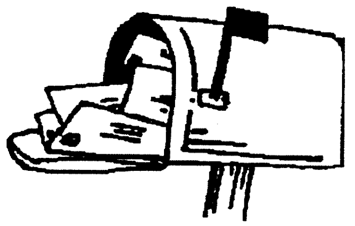
*Lance W.*

# TRSTimes - Volume 2. No. 3. May/Jun 1989

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

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

# THE MAIL ROOM



**CPM**

I'm very happy that TRSTimes is going to be published for another year, and thrilled with the great look that laser printing has given it. It is absolutely first rate.

One minor problem that I notice is the misplacing of Roy Beck's CP/M column. I'm sure that it was forced out of the January issue only because of the wealth of material you felt had to be published, and, of couse, it will return with the very next issue.

CP/M has given my trusty TRS-80 Model 4P a new lease on life. I've been able to get languages for CP/M that run without modification on my Tandy. Turbo Pascal, APL, Modula-2, and, soon now, ADA. Vast resources of public domain exists to play with. New software is STILL being written, albeit slowly.

There is a whole world besides the LSDOS, LDOS, TRSDOS operating systems for the Model 4 family of computers. I don't mean to disparage these fine DOS's. Rather I mean to point out the rest of the resources of software that are still available. Please, please, please put the CP/M column back in. It is much enjoyed and greatly missed.

Ted Seidler
Aurora, CO.

*You are right. The CP/M column was missing from the January issue, only because we ran out of room, plain and simple. As you can see from the March issue, as well as this one, Roy is back. This time he is tackling the CP/M disk and directory formats, a subject largely ignored in the CP/M world.*

*Ed.*

---

**Model 4 Scripsit tips revisited**

In looking through back issues of TRSTimes, I notice the item "Model 4 Scripsit tips" (Volume 1, No.4 - July 1988, Page 5). Barry K. Morley of Pudsey, England warned about SYSGENing the SETKI with minimum RATE and WAIT settings.

You didn't mention it, but as you probably have determined from the manual, you can issue the command:

**SETKI (@** [for QUERY?]

and the system will prompt for

**Wait=**

(a smaller number will speed the response - minimum is 10), and

**Rate=**

(a larger number will slow the repeat rate down)

showing you the current setting after each prompt and allowing you to enter your choice of setting (within limits allowed by the system).

Then, you can just SYSGEN again and the NEW settings will be saved in your configuration file. I hope the above information is helpful, without being too wordy.

Patrick H. Larkin
Bedford, TX.

*Thanks, Pat. This kind of tip is how we learn to use our software more productively. Sure, it might be in the manual, but quite a few of us miss many of the finer points, so it is, indeed, very helpful.*

*Ed.*

---

**LISTER/BAS**

For the record, LISTER/BAS (TRSTimes 2.1.) was first found in 80 Micro February 1987 in the reader forum section, submitted by Kenneth M. Frith of Baton Rouge, LA. The May 1987 issue, same magazine, same section, had enhancements by M.H Briggs of Walla Walla, WA. Thanks for your help on this, and for beginning the Assembly language tutorial. The new laser printer is a big improvement.

Jim Savage
Clinton, MS.

*A tip of the hat to Kenneth Frith and M.H. Briggs fromTRSTimes. Good writing.*

*Ed.*

---

**More on LISTER/BAS**

Congratulations on the birthday and on the success of TRSTimes! May the publication and you manage to survive for years to come! My purpose in writing is to comment on the LISTER program. It is something I could use, so I took a closer look at it. In such cases I practically always go through a program to see how it works and what I might want to do to customize it for something I in particular might like to have it do. I almost always pick up a good programming idea or two as well.

LISTER has some clever methods in it, for example the search for the end of the BASIC line and breaking it into two or more lines for the printout if necessary. However, unless I have made a wrong turn somewhere in the program flow, there can be a problem.

Lines 240 through 330 neatly breaks up a long BASIC line and print it in segments - that is, if the printout hasn't yet reached the end of the page. If, however, there is still a remnant of the BASIC line left in I$ when line 330 is reached, the program goes on to skip to the next page. If single-sheet printout has been selected, lines 400 through 420 create a pause for sheet changing and they do so by waiting for a keystroke.

This keystroke puts some character into I$ and the program continues. The problem appears to be that if any portion of an original BASIC line happens to be left in I$ when line 55 is reached, it will get replaced with the keystroke character, whatever it may be. Then line 210 will read in a new line from the disk and the remainder of the old line will not get printed.

The problem isn't too likely to show up, of course, since most BASIC lines will fit onto a single printout line and the probability of a longer line's occurring exactly at the end of a printout page is probably small.

Maybe I'm missing omething, but since you worked through the program for the reader, perhaps you can spot my error, if any. If you agree with my diagnosis, however, I think it would be good to print the very simple fix: merely replace the variable in line 400 with any other that isn't used in the program, such as A$.

Thanks for your efforts.

Dick Houston
Durango, CO.

*The TRSTimes 'sharp-eye award' goes out to Dick. Line 400 will, indeed, cause the error condition EXACTLY as described. The fix, also as described, is:*

```
400 A$=INKEY$:IF A$="" THEN 400
```

*Another error is found in line 180. There we have:*

```
180 IF O)1 THEN....
```

*this should have been:*

```
180 IF O>1 THEN ....
```

*Dumb errors like the above have plagued computer programs from day one, and they will surely continue until programs are no longer written by humans. Meanwhile, it is good that we have people out there who care enough to take the code apart and share the fix. Frankly, that's why the TRS-80's have lasted as long as they have.*

*Ed.*

---

**Assembly 101**

I am a new subscriber and have received issues 1 and 2 of TRSTimes. Thank you for the prompt shipment. TRSTimes is a fine publication; I'm glad I subscribed. The "Assembly 101" articles where assembly language is compared to equivalent Basic statements is what I hoped I would find somewhere. I have RS's EDTASM manual, cassette (from my uncle some years ago who has a M3; he also obtained a M1 disk for it and sent it to me). I've read the EDTASM manual from cover to cover 3 times (and realize I will have to read it many more times). The registers reminded me of file buffers for sequential and random files. This was a little erroneous in that those buffers are much larger than 1 or 2 bytes. Your comparison of the registers to Basic variables is certainly a better one. I have wondered for some time what Basic's INKEY$ and INPUT statements were changed to by the Interpreter (or the source code equivilant). So I will certainly be looking forward to the next issue.

David K. Berg
Fabens, TX.

*Thank you for the kind words. Assembly language is tough, but we WILL try to bring it down to earth, as much as possible.*

*Ed.*

---

**MULTIDOS**

"Heck, we may even play around with MULTIDOS."

This statement from TRSTimes 2.2. page 6 has really aroused my interest and curiosity. I have seen very little information other than an occasional review on this DOS over the years. I use it in Models I, III & 4 versions. On the Model I, I moved up to ULTRADOS from TRSDOS 2.3. I used ULTRADOS happily until I installed Radio Shack's double density board. I then bought LDOS as Radio Shack was selling it. After a while, though, I also purchased MULTIDOS. Both DOS's have their advantages and I have used both since.

I enjoy programming in BASIC and felt that MULTIDOS had the advantage for this purpose. LDOS 5.3. and LS-DOS 6.3. have easier to use BASIC than previous versions. Therefore, I would really enjoy seeing someone comment on, make suggestions for, add to, etc. MULTIDOS. My Model I is still in use, as are the Model 4, 100, COCO2, and PC1. I have just added a 1000SX to my collection and am now enjoying working my way through MS-DOS.

Clifton N. Duval
Star Lake, NY.

*ULTRADOS!!! You've been around for a while. We have been gently twisting the arms of a couple of devoted MULTIDOS users, hoping they will share some of their expertise. As time permits, I will disassemble portions of the DOS to see if I can discover a thing or two. As yet nothing is firm, but we will definitely do something with this fine DOS.*

*Ed.*

---

**TRSDOS 1.3.**

Keep up the good work! We want to see more articles on TRSDOS 1.3.- 1.3.- 1.3. - 1.3. - 13.!

I would like to be able to enter some of your Assembly language programs, but don't know how. Do I need a disk with an Assembly language "translator" on it? There ought to be some textbook type books out on the subject that are easy to understand. What little I've read, I have to already know the subject before I can understand what the book is saying.

Thanks for the "PATCH" commands. When I go back to school in January, I'll try them out. Have you seen the TRSDOS 1.3. PATRCH UTILITY PROGRAM from computer news 80. It costs $10.00, I'm thinking about ordering it.

Now about BASIC V.1.3. Do you know where I can get a disk that will let me RENUMBER the statement line numbers? Do you know where I can get a disk that will let me copy BASIC files from one disk to another? (similar to MS-DOS: COPY *.* B:) Do you know if there is a TRS-80 user group in Houston? Radio Shack doesn't know.

I understand that the CONVERT utility program converts files from Model 1 to TRSDOS 1.3. Will it convert from Ver. 6 to Ver. 1.3.?

Well, I guess I,ve bent your ear long enough, if you can help me on any of these, please let me know.

Maurice Superville
Bellaire, TX.

*Let's take each question one at a time. Assembly language programs are entered into an editor/assembler. See the "Assembly 101" article in TRSTimes 2.1. for a detailed description on how to do this. Regarding books on the subject, I agree with you 100 percent. I have not seen one book on Assembly language that REALLY catered to the beginner.*

*The TRSDOS 1.3. patch disk from CN80 was put together by Henry Herrdegen. It is a very fine compilation of various patches and, at $10.00, it is a bargain. I recommend it to all TRSDOS 1.3. users.*

*You do not need a special disk to RENUMBER a BASIC program. You have a command you can use directly from BASIC that will do exactly what you want. This command is called NAME and here is how it works:*

NAME newline,startline,increment

**newline** *is the new line number of the first line to be renumbered. If omitted, 10 is used as the default.*

**startline** *is the line number in the original program where renumbering will start. If omitted, the entire program will be renumbered.*

**increment** *is the increment to be used between each successive line number. If omitted, 10 is used as the default.*

*In other words, from BASIC you can type:*

NAME
*this will renumber all lines with an increment of 10.*

NAME 6000,5000,100
*this will renumber all lines numbered 5000 up; the first renumbered line will be become 6000, and the following lines will be incremented by 100. All line references within your program will be renumbered also.*

*As far copying files from one disk to another, you have the BACKUP utility. This, of course, copies the entire disk. There are a couple of 'shell' programs available that will let you tag a series of files, and then perform a mass copy. The one that comes to mind is George Fischer's DOS-TAMER. At this point, TRSDOS 1.3. is not capable of performing 'wild-card' copying as in MS-DOS. However, maybe we can persuade Gary Campbell of GRL software to write such a utility. (how about it, Gary!!) Incidentally, do check out the articles written by Gary AND also check out the ads for TRSDOS 1.4. and 1.5. Gary has done amazing things for all of us 1.3. lovers.*

*Unfortunately, I am not aware of a TRS-80 group in the Houston area. The closest one I know is the MID CITIES TRS-80 USERS GROUP. P.O. BOX 171566. ARLINGTON, TX. 76003. Drop them a note, maybe they will know of a group in your area. Meanwhile, can any of you Texas people help?*

*Finally, CONVERT will copy non-system Model I files over to Model III. It will NOT copy Model 4 files to Mod 3. You can do this by FORMATing a single density disk from Mod 4 and then copy the desired files to it. Then, use the Model III CONVERT utility to transfer the files to Mod 3. This is the hard way of doing things. Gary Campbell, once again, came to the rescue. His TRSDOS 1.5. has a utility that will let you copy files DIRECTLY from TRSDOS-LSDOS 6, DOSPLUS and LDOS to Model III TRSDOS. He also shared a scaled down version with us in his 'CPY' article (TRSTimes 2.1.)*

*Ed.*

---

**GAMES**

Many years ago I subscribed to SOFTSIDE because I really enjoyed typing in the games. I know that this kind of software is frowned upon today, but please... how about it... more games!!

Arnie Setzer
Wheeling, WV

*I have known a few people who insisted games were 'frivolous and a waste of time'. Guess what!! When I looked through their collection, they all had complete sets of the Big Five disks (well worn). You just know that when nobody's looking, they knock off a couple of rounds of METEOR MISSION II or SUPER NOVA.*

*As one who also subscribed to SOFTSIDE because of the games, I promise that we will never be so stuffy that we forget to have fun. I am particularly addicted to board games, so this issue brings ROTATE/BAS for Model I & III and next issue will feature MAXIT/BAS for Model 4.*

*We try to please.*

*Ed.*

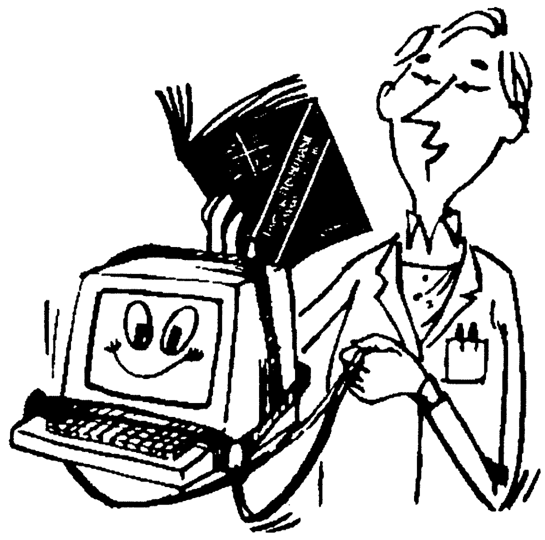
---

*More PEEKING & POKING Model 4*

# Hidden Video Fun

## Reverse Video in the III Mode of the Model 4

by Donald G. Shelton



You can get reverse video in the III mode of your model 4. This may sound simple enough, but I was told by many very knowledgeable people that it was not possible. That was a tremendous disappointment to me when I brought my new 4p to the office and set it next to my dependable Model III. I had heard about the 80 x 24 column screen, internal sound, and reverse video, and eagerly waited to snazz up my programs.

My heart sank when I found out that most of these features were only available in the model 4 mode. I considered converting my programs, but I had made extensive use of the fact that model III video was memory-mapped; since the model 4 was not, I did not see a way to convert those programs (not until the "Hunting for Buried Treasure" article in the November TRSTIMES. I sure could have used that article 5 years ago!)

One by one ingenious TRS-80 users have discovered ways to incorporate model 4 features in the III mode: LDOS 5.1.4 and 5.3 take advantage of the faster clock speed, TRSTimes had an article showing how to use the 80 x 24 mode, MISOSYS developed the Model 4 interface kit so we could fully use the keyboard, and now - reverse video.

Familiarity with the article on using 80 x 24 mode (September '88, page 26) is helpful, because we will manipulate the same video port. Bit 3 of port 132 (decimal) toggles the reverse video. In other words, OUT 132,8, will do the trick. You may be manipulating other bits in this port, such as bit 2 for page 1 of 80 x 24 video. If so, just add 8 to the figure you are sending to the port. In this case, OUT 132,12 gives you both reverse video and page 1 of 80 x 24 video.

However, it isn't quite that simple. The reverse video characters replace the "special characters" - ASCII values above 128. The special characters can be toggled with space compression characters. However, space compression characters cannot be replaced by the reverse video characters. Whew! In other words, before we manipulate port 132 to force reverse video, we must make sure that special characters/compression characters are forced to special characters. We do that by turning bit 1 of byte 16420 (decimal) on:

```
POKE 16420, PEEK (16420) OR 1
```

To review:

```
POKE 16420 -special chars/compression chars
           |               |                   |
OUT 132 -   reverse chars/special chars
```

Type the following for an impressive demonstration from basic:

**POKE 16420,PEEK(16420) OR 1** <ENTER>
**OUT 132,8** <ENTER>

Now, if you are using LDOS with the KI/DVR active, hold down the <CLEAR> key and hit some letter keys. The letters will appear in reverse video. (Ta-da).

If you are using LDOS without the KI/DVR, or any other Model III operating system type:

**PRINT CHR$(193)** <ENTER>

This will display letter 'A' in reverse video on the screen. You are adding 128 to the normal ASCII value of the letter (65 + 128). CHR$(194) will give you a 'B', etc.

Now we have to integrate this into our programs. We've just looked at the direct method; you can actually type a line like A$ = " " and fill the quotes with something you type while holding the <CLEAR> key down (typing this way is not very easy, but can be done). This is a great way for dressing up your displays. *(this works only with LDOS with KI/DVR active).* However, you may wish to reverse a part of the screen, such as highlighting a menu choice. This requires a subroutine.

My MENUDEM/BAS program shows two methods for doing this. The first method is done entirely from BASIC and is in the subroutine at line 3500. Basically all you are doing is telling the routine a starting position on the screen (PO%) and number of characters to reverse (A1%). The routine then adds 128 to each character in that range in a FOR/NEXT loop. This does the job easily, but it can be slow on anything but a small area to reverse. The program uses the string pointing technique I demonstrated in "Hidden Memory Fun" (TRSTIMES Nov. 88) to store what was at that location before it was reversed, so that you can call it back quickly when the highlight is moved.

The second method is a small machine language routine embedded in S$. You have to do a few things to set up for this, but the speed is very rewarding. The top line of the screen is reversed using this method.

Line 50 DEFines a function (SI%) that expresses numbers in signed integer format. You don't really need to understand this, just know that it is necessary housekeeping for finding our machine languague program.

Line 60 creates a machine language program inside of S$. It is a 15 byte routine that does essentially the same thing as a FOR/NEXT loop that adds 128 to each character in a defined range, but MUCH faster.

In line 920 we POKE &H4050 with 80. 80 is the number of bytes we want reversed. This can be any number up to 256. I chose &H4050 because this location appears to be unused by most DOS's, but any free location will do (you would have to change the machine language routine to look in the new location).

The subroutine is at 3600.

Line 3610 looks messy, but actually all it is doing is finding S$ in memory, which is where our routine is.

Line 3620 changes the location to a signed integer format to avoid errors.

Line 3630 points USR0 at S$.

Line 3640 calls the routine. The variable in the parenthesis was defined in 930 and is the place for the routine to start reversing. 15360 is decimal for the beginning of video memory, so you just add the PRINT@ position to 15360 to get the number to go in the ( ) after the J = USR0.

Play with it a little bit, and you will find that it is easy to use, even if you don't fully understand all the specifics of variable pointers, signed integers and imbedded machine language (I again recommend Lewis Rosenfelder's "Basic Faster & Better" for the lowdown on those subjects).

You can also see that the program is non-functional, but it would be easy for you to get it working with your own program choices in the menu. This type of program sure makes life easier, and reverse video makes the program look professional.

## MENUDEM/BAS

```
0 'REVERSE PROG "MENUDEM/BAS" demonstrating
reverse video 1989 DONALD G. SHELTON
5 CLS:OUT 132,140:CLS:
PRINT@100,"USE ARROWS TO MOVE HIGHLIGHT"
' print msg on bottom of screen
10 CLEAR 1000
15 SG$=STRING$(80,"-"):DEFSTR F
20 OUT 132,12 'puts in 80 x 24 reverse video
25 POKE 16420,PEEK(16420) OR 1
' special characters
30 DATA 320,400,480,560,640,720,800,880,960,346,426,
506,586,666,746,826,906,986,372,452,532,612,692,772,
852,932,1012
40 DIM PO(26):FOR X=1 TO 26:READ P:PO(X)=P:
NEXT X
50 DEFFNS I%(S!)=-((S!I>32767)*(S!-65536))-
((S!<32768) *S!)
'necessary for signing the integer of varptr(s$) to avoid
overflow error
60 S$=CHR$(205)+CHR$(127)+CHR$(10)+
CHR$(58)+ CHR$(80)+CHR$(64)+CHR$(71)+
CHR$(126) + CHR$(198)+CHR$(128)+CHR$(119)+
CHR$(35)+ CHR$(16)+CHR$(249)+CHR$(201)
70 'line 60 imbeds a short mach lang routine in s$
900 'video print routine
910 CLS:EM$="Master Command Console"+
STRING$(50,".") +LEFT$(TIME$,8)
920 PRINT@80,EM$:POKE &H4050,80
'80 is # of bytes we want to reverse
930 XT=15360+80:GOSUB 3600
1000 SS=80:PRINT@0,SG$;:PRINT@160,SG$;:
PRINT@3*SS,CHR$(30);:
PRINT@4*SS,"<A> CALCULATOR";
1002 PRINT@5*SS,"<B> CALENDAR";
1004 PRINT@6*SS,"<C> DIRECTORY";
1006 PRINT@7*SS,"<D> MAILBOX";
1008 PRINT@8*SS,"<E> MULT FILE KILL";
1010 PRINT@9*SS,"<F> TREASURER";
1012 PRINT@10*SS,"<G> GRAPHS";
1014 PRINT@11*SS,"<(H> GAMES MENU";
1016 PRINT@12*SS,"<I> INVOICE";
1018 PRINT@4*SS+26,"<J> RESET TIMER";
1020 PRINT@5*SS+26,"<K> TAX YIELD COMPARE";
1022 PRINT@6*SS+26,"<L> LDOS";
1024 PRINT@7*SS+26,"<M> MORTGAGE";
1026 PRINT@8*SS+26,"<N> PRINTER CODES";
1028 PRINT@9*SS+26,"<O> MILEAGE RECORDS";
1030 PRINT@10*SS+26,"<P> CREATE MAILBOX
FILE";
1032 PRINT@11*SS+26,"<Q> TYPEWRITER";
1034 PRINT@12*SS+26,"<R> LEXBASE";
1035 PRINT@4*SS+52,"<S> SCRIPSIT";
1038 PRINT@5*SS+52,"<T> TYPE-AHEAD";
1040 PRINT@6*SS+52,"<U> FILE SORTING";
1042 PRINT@7*SS+52,"<V> HELP (DOS)";
1044 PRINT@8*SS+52,"<W> WORD CHECKER";
```

```
1046 PRINT@9*SS+52,"<X> PRINTER FILTER";
1048 PRINT@10*SS+52,"<Y> LESCRIPT";
1050 PRINT@11*SS+52,"<Z> MORE CHOICES";
1093 PS=1:PO%=PO(PS):A1%=26:GOSUB 40070:
EM$=AN$: TM$=AN$:GOSUB 40015:
PRINT@PO(PS),EM$;
1095 GOSUB 40500:IF A$=CHR$(91) OR A$=
CHR$(10) OR A$=CHR$(13) OR A$=CHR$(8) OR
A$=CHR$(9) THEN GOTO 1500
1096 GOTO 1095
'here is where you would call programs
1500 'move the location of the highlight
1505 IF A$=CHR$(91) AND PS=1 THEN GOTO 1095
1506 IF A$=CHR$(10) AND PS=26 THEN GOTO 1095
1510 IF A$=CHR$(91) THEN PRINT@PO(PS),TM$;:
PS=PS-1:GOTO 1550
1515 IF A$=CHR$(10) THEN PRINT@PO(PS),TM$;:
PS=PS+1:GOTO 1550
1517 IF A$=CHR$(8) THEN IF PS-9<1 THEN 1095
ELSE PRINT@PO(PS),TM$;:PS=PS-9:GOTO 1550
1519 IF A$=CHR$(9) THEN IF PS+9>26 THEN 1095
ELSE PRINT@PO(PS),TM$;:PS=PS+9: GOTO 1550
1520 IF A$=CHR$(13) THEN A$=CHR$(PS+64):
CLS:RUN
1550 A1%=26:PO%=PO(PS):GOSUB 40070:
EM$=AN$: TM$=AN$:PRINT@PO(PS),EM$;:
GOSUB 3500:GOTO 1095
3500 'reverse video
3510 FOR XX=PO%+15360 TO PO%+15360+A1%-1
3520 POKE XX,PEEK(XX)+128
3540 NEXT XX
3550 RETURN
3600 'reverse video ii
3610 A!=PEEK(VARPTR(S$)+1)+256*PEEK
(VARPTR(S$)+2)
'where is s$?
3620 A!=FNSI%(A!)
'avoid overflow error
3630 DEFUSR0=A!
3640 J=USR0(XT)
'val in () is memory loc for beg of screen+ position of
area to reverse
3650 RETURN
40015 B1%=0:FOR XX=1 TO LEN(EM$):
A$=CHR$(ASC(MID$(EM$,XX,1))+128):
MID$(EM$,XX,1)=A$:IF A$=CHR$(160) THEN
B1%=B1%+1:GOTO 40016 ELSE B1%=0:
NEXT XX:RETURN
40016 IF B1%=1 THEN NEXT XX:RETURN
40017 EM$=LEFT$(EM$,XX):EM$=EM$+
STRING$(26-XX,160):RETURN
40070 AN$=" ":POKE VARPTR(AN$),A1%:
POKE VARPTR(AN$)+2,INT(PO%/256)+60:
POKE VARPTR(AN$)+1,PO%-INT(PO%/256)*256:
RETURN
40500 A$=INKEY$:IF A$=""THEN 40500:ELSE
RETURN
```

## TRS-80 Software from Hypersoft.

### NEW ! PC-Three TRS-80 Model III Emulator !

**PC-Three**, new program from Hypersoft, lets you run LDOS 5.1-5.3, TRSDOS 1.3, NEWDOS/80 V2, DOS-Plus 3.5 & MultiDOS on a PC, XT, AT or compatible. **PC-Three** emulates a TRS-80 M3 with its Z80 Microprocessor and 64K memory. It supports the printer and serial ports and most of the functions of the floppy disk controller. To use it you must be the legal owner of a TRS-80 M3 DOS and either a copy of the MODELA/III file (on TRSDOS 6.2) or a working TRS-80 M3 or 4.
**Runs on PC, XT, AT & compatibles and laptops with at least 384K of memory. ONLY emulates TRS-80 Model III.**
Comes with a special version of PCXZ to transfer your disks to MSDOS. Depending on the type of drives on your PC you may need access to a working TRS-80.
**Price:: (Includes 1 free upgrade) Order #PC3....................................$109.95**

### Run Model 4 Software on a PC with PC-Four !

**Run your favorite TRS-80 Model 4 programs on a PC!**
**PC-Four**, a program making your PC or Compatible act like a 128K TRS-80 M4 complete with operating system, Z80 microprocessor, can run many true M4 programs: **ALDS, ALLWRITE, BASCOM, BASIC, C, COBOL, EDAS, ELECTRIC WEBSTER, FED, FORTRAN, HARTForth, Little Brother, MZAL, MULTI-BASIC, PFS FILE, PASCAL, Payroll, Power-Mail, PROFILE, SUPERSCRIPSIT, TASMON, VISICALC, ZEUS**, etc..
Runs on PC, PS/2, compatibles & laptops with at least 384K memory. ONLY emulates M4 mode of M4. To use it you must transfer your old files to MSDOS disks using PCXZ or Hypercross.
**Prices: Order #PC4 $79.95 alone, #PC4H $104.95 with Hypercross. SX3PCM4, #PC4Z $119.95 with PCXZ. Available on 3.5" disk format**

### PCXZ reads TRS-80 disks on a PC, XT or AT

PC Cross-Zap (PCXZ), a utility to copy files to or from BASIC **automatically, no need to save in ASCII first.** Also format & copy disks, explore, read & write sector data, repair bad directories and much more. **Supports:** all double density M1, 3 & 4 formats. **Requires:** PC, XT, AT or compatible. **You must have at least one 5-1/4" regular or high density drive and 256K memory. Not for PS/2. Order:#PCXZ..................................................$79.95**

### READ CP/M, CoCO & PC disks on your TRS-80

Use **HYPERCROSS** to **COPY** files between TRS-80 disks & those from many CP/M and IBM-PC type computers on your TRS-80 1, 3 or 4/4P. FORMAT alien disks, read their directories, copy files to & from them, copy directly from one alien disk to another. Converts TRS-80 BASIC to MSDOS or CP/M as it copies, no need to save in ASCII first. **Formats supported:** IBM-PC and MS-DOS inc. DOS 1.1, 2.0 - 3.2, Tandy 2000, single & double sided. 3.5 & 5 inch. CP/M from Aardvark to Zorba. CoCo format on XT+ version. **HyperCross 3.0 PC** reads popular MSDOS 1.1-3.2 formats. **Order SX3PCM1, SX3PCM3 or SXPCM4...........................$49.95**
**HyperCross XT/3.0** reads 90 different CP/M and PC formats. **Order SX3XTM1, SX3XTM3 or SX3XTM4...................................................$89.95**
**HyperCross XT/3.0-Plus.** Reads over 220 formats, including CoCo. **Order SX3XTM1+, SX3XTM3+ OR SX3XTM4+.......................................$129.95**
Specify TRS-80 M1 (needs doubler), 3, 4/4P or MAX-80. Dual model versions e.g. Mod 3/4 on one disk add $10 extra.

### Other TRS-80 Programs

**HYPERZAP 3.2G.** Our ever popular TRS-80 utility for analyzing, copying, repairing and creating floppy disks of all kinds....................................**$49.95**
**MULTIDOS 2.1.** New for 1988 for 1 or 3. **$79.** 64/80 for Mod 4(3)...**$89.00**
**Mysterious Adventures** - Set of 10 for M1, 3 or 4 (3) complete.........**$49.95**
**TASMON** debug trace disassemble TASM1 TASM3 or TASM4.....**$49.95**
**TMDD** Memory Disk Drive for NewDOS 80/Model 4 users.............**$39.95**
**XAS68K** 68000 Cross Assembler. Specify Mod 1, 3 or 4........................**$49.95**
**ZEUS Z80** editor/Assembler for Model 1, 3 or 4.................................**$74.00**
**ZIPLOAD** fast load ROM image. DOS & RAMDISK on your 4P......**$29.95**

We have more ! Write or call for complete catalog.

**HYPERSOFT**
**PO Box 51155, Raleigh, NC. 27609**
Orders: 919 847-4779 8am-6pm. Support: 919 846-1637 6pm-11pm EST
MasterCard, VISA, COD, Checks, POs $3 for shipping. $5 - 2nd day

# GET HIGH PERFORMANCE POWER WITH SUPERCHARGED TRSDOS SYSTEM 1.5

## Bob Rose reviews TRSDOS System 1.5. for Model III/4

*There comes a time when even the most diehard NEWDOS/80 user, such as myself, will find a Disk Operating System that is superior to that which they are currently using.*

Gary Campbell of GRL Software, Kelowna, British Columbia, Canada has released an upgrade to TRSDOS 1.3. It is called TRSDOS System 1.5. for Model III and 4 (in M3 mode). He has managed to take the much maligned TRSDOS 1.3. system, rewriting, modifying and upgrading it to such an extent that it is now a pleasure to use.

### INSTALLING AND USING THE SYSTEM

The software package comes on two disks. One contains the System 1.5 data which will perform the upgrade from TRSDOS 1.3. to SYSTEM 1.5.; the other is a documentation disk containing files which, when sent to the printer, will produce a neat 37 page USER MANUAL, complete with a table of contents.

Also included in the package is a brief three page "*UPGRADING TO 1.5*" installation manual. As a bonus, GRL Software sends along 2 disks filled to the rim with a variety of good public domain software - a nice touch!

The installation manual, unfortunately, is somewhat vague. An experienced user should not have problems, but it could cause a new TRS-80 user needless frustrations, diminishing the pleasures to come. It starts by telling you what you will need to install the system:

*A backup copy of a virgin TRSDOS 1.3 copied from drive 0 to drive 1.*

*A data or system disk containing Basic/CMD.*

*A formatted disk that you will label System 1.5 Backup.*

| | |
|---|---|
| **Easy to use:** | * * * |
| **Good docs:** | * * |
| **Bug free:** | * * * * |
| **Does the job:** | * * * * * |

You are then instructed to place the System 1.5 disk in drive 1 and type "UPGRADE/NOW".

Once you've gotten this far the upgrade process is quite nice, as it basically takes over from here.

### SYSTEM 1.5 FEATURES

Listed below are just a few of the many enhancements found in SYSTEM 1.5. Some are modifications to existing commands, others are brandnew features adding the dimension and depth sorely lacking in TRSDOS 1.3.

**BACKUP** - long the nemesis of TRS-80 users wanting to back up their disks, this command has been changed to NOT REQUIRING A MASTER PASSWORD and NOT VERIFYING SECTORS during FORMAT. This results in a much easier and speedier process.

*Backup does require the double sided driver (DBSIDE) to be installed , or it will not work.*

**BOOT** - SYSTEM 1.5 has added the BOOT command found on other DOSes. Now you can 'software reboot' directly from DOS or Basic.

**CAT** - many programs displaying short directories have been written, taking up disk space. These are now passe. SYSTEM 1.5. includes CAT as a library command. It displays all files, including system files.

**DBSIDE** - a new library command that extends DOS' power to take advantage of double sided drives. TRSDOS was never able to use anything other than a single sided disk. This enhancement, providing you have double sided drives, allows you to use the back side of a disk as a separate drive, thus giving you the possibillity of having 8 single sided drives hooked on to the system - more storage than ever before.

I did find a slight problem with this feature in that, if side two of drive 0 is not formatted, the system will sit on the back side of drive 0 (called drive 4) for a while, and then abort with an error message. This will occur every time you are loading a file that does not exist on drive 0, 1, 2, or 3. In all fairness, any other operating system will do the exact same thing if it encounters an unformatted diskette in its search for a program to load. Formatting the back sides of your diskettes is something you will just have to remember. But then, isn't the fact that you can use the back sides of the diskettes why you are interested in the system in the first place?

**DEBUG** - never before did this utility allow access to the DOS portion of RAM. It has now been improved to display and modify all RAM addresses

**DIR** - this much used command is changed to prompt you to hit < ENTER > when the screen is filled. The syntax has been relaxed, making the delimeter (the colon) optional, thus making it compatible with most of the other Model III and 4 DOSes.
Example: DIR 1
*(I found this to be one of the best time saving features in the system.)*

**LIB** - has been expanded to include BOOT - CAT - DBSIDE - SWAP AND SYSTEM.

**CLS** - the command to erase the screen from DOS has been removed to make space for some of the enhancements. Who needs it when we have the < CLEAR > key.

**SWAP** - another new command that is very useful. On occassion certain software, such as SCRIPSIT, will balk if you tell it that your data is stored on a drive numbered higher than 3. The swap command allows you to bypass this limitation by swapping one drive number for another. For example, typing the following

```
SWAP (0,4) <ENTER>
SWAP (1,5) <ENTER>
SWAP (2,6) <ENTER>
SWAP (3,7) <ENTER>
```

will make DOS recognize the back sides of the disks as being in drive 0,1,2, and 3. The front sides will be recognized as drive 4,5,6, and 7. With a little ingenuity you can have SCRIPSIT in drive 0 along with 7 data disks, giving you almost one megabyte of on-line storage. Now you have no excuse for not finishing that book.

Before covering the SYSTEM command, let me briefly mention some other extremely nice features:
Upper and lower case are now supported in DOS
*(a very good addition - having to remember always to be in upper case was annoying.)*

The periods have been removed from DOS Ready
*(not an earthshaking change.)*

PURGE no longer requires a password
*(very nice, but potentially dangerous for new users.)*

FORMAT now adds DIR/SYS and BOOT/SYS to disk and asks you whether or not you want to verify sectors
*(good - I like having choices.)*

Errors are now displayed as actual error messages rather than numbers
*(very good - never could remember the numbers.)*

Bank storage is now available for Mod 4 users; also 4mhz speed is supported
*(Mod 4 users running SYSTEM 1.5. in Mod III mode can zip right along. You will be amazed how fast TRSDOS is now.)*

The date and time are now displayed as mm/dd/yy rather then just mm/yy
*(I could have lived without that, but more directory information is like chicken soup - it doesn't hurt.)*

The most powerful aspect of the program is the SYSTEM Driver. This, not unlike those of NEWDOS/80, LDOS and TRSDOS 6., will allow you to customize the DOS and then save the preferences with the SYSTEM CONFIG command. Once this is done, your config file will load every time you boot up. If you do not wish to load the config file, opting for the systems defaults, you can hold down the right shift key at boot up to disable it.
The SYSTEM command has many options that will make life easier for most users. My favorite is the SYSTEM (DATE = Y/N) option. Here you can decide whether or not you wish to answer a date prompt at boot-up. Entering the date in TRSDOS 1.3. was mandatory.
The ability to copy files from other DOSes back into TRSDOS is available with SYSTEM (CPY parameters) command.
The cursor character can be defined with SYSTEM (CURSOR = "hexnum').
The keyboard driver has many new and exciting options, such as key click, the execution of macros from DOS, entering graphic characters directly, special control keys and much more.
The new printer driver also gives added abilities. Not only do you now have a spooler, you can also add or ignore line feeds, set top and bottom margins, set number of lines per page, set page width, enable or disable type-ahead, and set page numbers, all directly from the DOS command line.
Another feature which must be mentioned, is HLPGEN. This converts an ASCII file to TRSDOS 6. /HLP file format. The HLPGEN P parameter is immensely useful. It will actually print the ASCII file as a manual, complete with page headings, page numbers, margins and formatted text.

## CONCLUSION

SYSTEM 1.5. is an exceptional piece of software. It really makes using TRSDOS worthwhile. *(Not a mean feat to make a NEWDOS/80 fan admit to this.)* My enthusiasm for the program, however, stops with the written documentation. The manual is only adequate, briefly covering installation and usage. I think the author assumes that if you are using a TRS-80, you can probably figure out how to use 1.5. without the manual.
The bottom line: I think System 1.5 deserves a spot in any TRS-80 users software library.

# ROTATE
## a game for Model I & III
**by Lance Wolstrup**

I have always enjoyed games and puzzles. As a kid I played Monopoly and put together Jigsaw puzzles. As an adult, it is Chess and the biggest puzzle of them all, computer programming. These days most of my programs are of the business variety or utilities, but I still manage to throw together an occassional game or two.

ROTATE is a combination game and puzzle, a simulation of the old plastic toy with sliding pieces numbered 1 to 15 or lettered A-P. In the old days you could buy this plastic game at your neighborhood 7/11, or in the junk section of any toy store, for about 25 cents. Now you can type in the program and run it on your $1000 computer (isn't technology wonderful?)

Seriously, ROTATE is fun and mentally stimulating. It certainly isn't easy. The object of the game is to place the letters A-P, which have been positioned randomly on a 4x4 board, in alphabetical order by rotating any four letters clockwise one position. The board may look something like this:

```
C F M O
A B D I
H E G P
N J K L
```

Now if you can imagine the board positions as being

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

you can see that rotating position 5 would move A up to position 1, C to position 2, F to position 6, and B to position 5. The board would now look like this:

```
A C M O
B F D I
H E G P
N J K L
```

Note that legal moves are 1,2,3,5,6,7,9,10 & 11. In other words, you cannot rotate 4,8,12,13,14,15, & 16

To make the game a little easier to solve, you are allowed one special move per game. This move will exchange two adjacent pieces and the legal moves are 1,2,3,5,6,7,9,10,11,13,14 & 15

You can abort a game in progress at any time by pressing <Q> <ENTER>. This will bring you back to the main menu from where you can choose to play a new game or quit.

ROTATE, written in Basic, employs a 31 byte machine language routine to speedily jazz up parts of the program. The routine simulates an advanced version of the Basic STRING$ command; that is, it will display a character a number of times horizontally. The advanced part is that it will also display the character a number of times *vertically*, lightning fast!

To make it more versatile, the routine is written to accept five parameters before being called. Variable V holds the vertical position of the cursor, variable H is the horizontal position, variable CH contains the character to be displayed, variable L holds the number of times the character in CH should be displayed horizontally, and finally, variable W determines how many times the character from CH should be displayed vertically. Since this routine is used often, the best way to explain it is to go through the program line by line.

Line 5 jumps over the subroutines to line 100, which is the beginning of the actual program.

Line 100 clears string space and sets all variables, except A, as integers.

Line 110 dimensions two arrays, B() and B1(). Each will hold subscript 1 through 16 (I hardly ever use subscript 0). B() will contain the playing pieces and B1() will make sure we don't produce duplicate playing pieces.

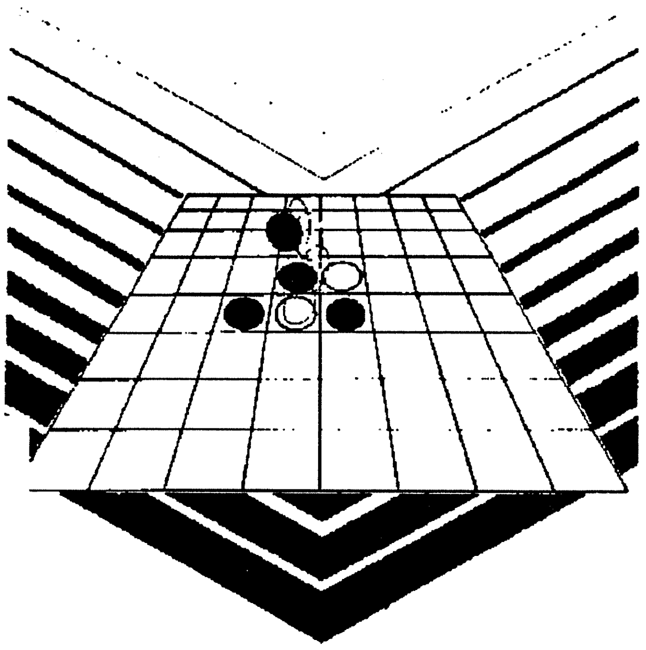
Line 120 goes to the subroutine in line 20 where the machine language routine is POKEd into ML$.

Line 130 uses the ML routine to completely white-out the screen. Variables V and H are both set to 0 (0,0 is the top left corner of the screen), L=64 (horizontal length of the screen), W=16 (vertical width of the screen) and CH=191 which is the all-white graphic block. Going to the subroutine in line 40, V and H are converted into a number representing the actual video RAM memory location. This number is further broken down into two numbers (LSB and MSB) and stored in variables A1 and A2. A quick excursion to the subroutine in line 30 finds the actual memory location of the first byte of our machine language routine. This value is stored in variable A. Now knowing where the routine is, we return to the line 40 subroutine and POKE the LSB of the cursor position into memory location A+5, the MSB of the cursor position into memory location A+6, the vertical width into memory location A+11, the horizontal length into memory location A+15 and the value of the character we wish to replicate into memory location A+17. Having done that, we tell the program that our ML routine starts at memory location A and proceed to execute it. Then we return from the subroutine.

Line 140 sets variable S=0. This is done to create a split screen which displays the program title and credits.

Since we will later display this at a different place on the screen, it is written as a subroutine (line 60), with variable S as the offset to the horizontal position of the cursor.

Lines 150-170 sets up the menu. All text to be displayed is stored in A$, then routed to the subroutine in line 50 where the screen location is determined according to V and H and then sent to the the screen with the PRINT@ statement.



Line 180 sets up all the parameters, except CH, to create a flashy display on the right hand half of the screen.

Line 190 waits for a keystroke to choose one of the menu items. As long as an appropriate key is not hit, variable CH is given a random graphic character value and the ML routine is executed. Not only does the graphic display jazz up the menu, since we execute the RND statement an uncountable amout of times, it also insures that the game will be truly random.

Line 200 checks if we want to quit. If so, a rather unique method of clearing the screen is used. Again, this is done by employing the ML routine.

Line 210 checks if we want to see the instructions. If so, we go to the subroutine in line 680 and do just that.

Line 220 checks if we have chosen to play the game, in which case we jump to line 230. Otherwise the program goes back to the INKEY$ routine in line 180.

Line 230 initializes the moves to 0, sets two flags handling the exchange move (EF & FF), then erases the left half of the display and moves the title & credits over to the right side.

Lines 240-280 shuffles the playing pieces. Line 270 makes sure that there will be no duplicates.

Lines 290-340 displays the playing pieces on the screen.

Line 350 makes a quick detour to the subroutine in line 8 0 where a flag (FL) is set if the pieces are in alphabetical order. If the flag is set upon return, the game starts out in order (very unlikely, but theoretically it could happen), so we go back to line 240 for a re-shuffle.

Lines 360-410 sets up the scoreboard, the mini-help screen and the prompt to make a move.

Line 420 sets the position of the cursor, the length and width of the allowable input, and then erases any potential characters at that location. We then proceed to the INKEY$ subroutine in line 70 which returns with the character(s) in I$.

Line 430 checks if we have chosen to quit or exchange. If quit, we go to the exit routine in line 130. If exchange, we go to line 440. If neither, the program converts the string to a numeric and jumps to line 520.

Line 440 checks if the exchange flag is set. If so, we allow an exchange; otherwise go back and prompt for an input.

Line 450 turns off the exchange flag to indicate no further exchanges are allowed. The secondary exchange flag is set to indicate that only two pieces are to be moved. The prompt is replaced with the last line of numbers that can be exchanged.

Line 460 displays the prompt to make the exchange.

Line 470 erases any potential characters at the input field, and then goes on to the INKEY$ subroutine.

Line 480 makes checks to make sure the chosen exchange is legal.

Line 490 increments and displays the moves. The exchange is then made.

Line 500 erases the exchange prompt, as well as the last line of allowable exchange moves. The regular prompt is then displayed.

Line 510 jumps over the regular rotate routine to 550.

Line 520 is the normal rotate routine. We check if the input is legal. If not, we go back to line 420.

Line 530 increments and displays the moves.

Line 540 rotates the pieces.

Lines 550-570 determines the row(s) of the moved (or exchanged) pieces. The first piece is then displayed.

Line 580 displays the second piece. If the secondary exchange flag is set, we skip displaying the third and fouth piece (they are already displayed) by jumping up to 610.

Lines 590-600 displays the third and fourth pieces of the rotate move.

Line 610 checks the subroutine in line 80 to see if the pieces are now in alphabetical order.

Line 620 determines if we have won. If the flag (FL) returns from the subroutine set, we go on to the win routine in line 630; otherwise we go back to line 420 to make the next move.

Lines 630-670 contain the win routine. A graphic display is enabled, and we are prompted to play another game.

Lines 690-940 hold the game instructions.

## ROTATE/BAS

```
1 ' ROTATE - MODEL I & III
2 '(c) 1989 Lance Wolstrup
3 ' a puzzle game
4 '
5 GOTO 100
10 DATA 33,1,60,45,17,1,1,21,29,25,6,1,197,229,6,1,54,1,
35,16,251,225,17,64,1,21,25,193, 16,238,201
19 ' ML$ has 31 periods
20 ML$="..............................."
21 GOSUB 30:FOR X=0 TO 30:READ A0:POKE
A+X,A0:NEXT:RETURN
30 A=PEEK(VARPTR(ML$)+2)*256+PEEK(VARPTR(ML$)+1):
IF A>32767 THEN A=A-65536
31 RETURN
40 A0=V*64+H+257:A2=INT(A0/256):A1=A0-A2*256:
GOSUB 30:POKE A+5,A1:POKE A+6,A2:POKE A+11,W:
POKE A+15,L:POKE A+17,CH:DEF USR0=A:A3=USR0(0):
RETURN
50 LO=V*64+H
51 PRINT@LO,A$;:RETURN
60 V=1:H=S+1:L=30:W=14:CH=32:GOSUB 40
61 V=2:H=S+7:A$="TRSTimes Presents:":GOSUB 50
62 V=3:H=S+10:A$="R O T A T E":GOSUB 50
63 V=4:H=S+5:A$="A game of mental skill":GOSUB 50
64 V=5:H=S+4:A$="(c) 1989 Lance Wolstrup":GOSUB 50
65 V=6:H=S+1:L=30:W=1:CH=131:GOSUB 40:RETURN
70 A$="":CH=46:GOSUB 40:PO=V*64+H:LE=0
71 I$=INKEY$:IF I$="" OR I$=CHR$(31) THEN 71
72 IF I$=CHR$(13) THEN I$=A$:RETURN
73 IF I$=CHR$(9) OR I$=CHR$(10) THEN 71
74 IF I$=CHR$(8) AND LE=0 THEN 71
75 IF I$=CHR$(8) THEN LE=LE-1:PRINT@PO+LE,CHR$(46);:
A$=LEFT$(A$,LE):GOTO 71
76 IF L=LE THEN 71
77 PRINT@PO+LE,I$;:A$=A$+I$:LE=LE+1:GOTO 71
80 FL=1:FOR X=65 TO 80
81 IF B(X-64)<>X THEN FL=0:X=80
82 NEXT:RETURN
100 CLEAR 2500:DEFINT B-Z
110 DIM B(16),B1(16)
120 GOSUB 20
130 V=0:H=0:L=64:W=16:CH=191:GOSUB 40
140 S=0:GOSUB 60
150 V=8:H=6:A$="(P) = Play game":GOSUB 50
160 V=10:A$="(I) = Instructions":GOSUB 50
170 V=12:A$="(Q) = Quit":GOSUB 50
180 V=1:H=33:L=30:W=14
190 I$=INKEY$:IF I$="" THEN CH=RND(63)+129:GOSUB 40:
GOTO 190
200 IF I$="Q" OR I$="q" THEN H=32:L=32:W=1:CH=32:
FOR V=15 TO 0 STEP-1:GOSUB 40:NEXT:V=0:L=1:W=16:
FOR H=0 TO 31:GOSUB 40:NEXT:END
210 IF I$="I" OR I$="i" THEN V=8:H=1:L=30:W=6:CH=32:
GOSUB 40:V=1:H=33:W=14:GOSUB 40:GOSUB 680:
GOTO 150
220 IF I$="P" OR I$="p" THEN 230 ELSE 180
230 M=0:EF=1:FF=0:V=1:H=1:L=30:W=14:CH=32:
GOSUB 40:S=32:GOSUB 60
240 V=2:H=10:A$="Shuffling...":GOSUB 50
250 FOR X=1 TO 16:B1(X)=1:NEXT
260 FOR X=1 TO 16
270 N=RND(16)+64:IF B1(N-64)=1 THEN B(X)=N:
B1(N-64)=0 ELSE 270
280 NEXT
290 V=2:H=1:L=30:W=1:CH=32:GOSUB 40
300 L=2:W=2:X=1
310 FOR V=3 TO 12 STEP 3
320 FOR H=7 TO 22 STEP 5
330 CH=B(X):GOSUB 40:X=X+1
340 NEXT:NEXT
350 GOSUB 80:IF FL THEN 240
360 V=7:H=42:A$="Moves: ":GOSUB 50:
PRINT USING"###";M;
370 V=9:H=34:A$="Legal moves:  1  2  3":GOSUB 50
380 V=10:H=48:A$="5  6  7":GOSUB 50
390 V=11:A$="9 10 11":GOSUB 50
400 V=12:H=34:A$="Exchange  =  X":GOSUB 50
410 V=14:H=34:A$="Enter your move, please: ":GOSUB 50
420 V=14:H=59:L=2:W=1:CH=32:GOSUB 40:GOSUB 70
430 IF INSTR(I$,"Q") OR INSTR(I$,"q") THEN 130 ELSE IF
INSTR(I$,"X") OR INSTR(I$,"x") THEN 440 ELSE I=VAL(I$):
GOTO 520
440 IF EF THEN 450 ELSE 420
450 EF=0:FF=1:V=12:H=34:L=12:W=1:CH=32:GOSUB 40:
H=47:A$="13 14 15":GOSUB 50:V=14:H=34:L=27:
GOSUB 40
460 V=14:H=34:A$="Enter your exchange: ":GOSUB 50
470 V=14:H=55:L=2:W=1:CH=32:GOSUB 40:GOSUB 70
480 I=VAL(I$):IF I<1OR I>15 OR I/4=INT(I/4) THEN 480
490 M=M+1:V=7:H=49:A$="":GOSUB 50:PRINT
USING"###";M;:TP=B(I):B(I)=B(I+1):B(I+1)=TP
500 V=12:H=34:L=29:W=1:CH=32:GOSUB 40:
V=14:H=34:L=29:W=1:CH=32:GOSUB 40:
A$="Enter your move, please: ":GOSUB 50
510 GOTO 550
520 I=VAL(I$):IF I<1 OR I>11 OR I/4=INT(I/4) THEN 420
530 M=M+1:V=7:H=49:A$="":GOSUB 50:
PRINT USING"###";M;
540 TP=B(I):B(I)=B(I+4):B(I+4)=B(I+5):B(I+5)=B(I+1):
B(I+1)=TP
550 J=I
560 IF J>4 THEN J=J-4:GOTO 560
570 V=3*(INT(I/4)+1):H=7+5*(J-1):L=2:W=2:CH=B(I):
GOSUB 40
580 H=H+5:CH=B(I+1):GOSUB 40:IF FF THEN FF=0:
GOTO 610
590 V=V+3:CH=B(I+5):GOSUB 40
600 H=H-5:CH=B(I+4):GOSUB 40
610 GOSUB 80
620 IF FL THEN 630 ELSE 420
630 V=8:H=34:L=29:W=7:CH=32:GOSUB 40
640 V=9:H=41:A$="Puzzle solved":GOSUB 50
650 V=13:H=40:A$="Try again (Y/N) ":GOSUB 50
660 I$=INKEY$:IF I$="" THEN V=3:H=2:L=4:W=11:
CH=RND(63)+128:GOSUB 40:H=26:GOSUB 40:GOTO 660
670 IF I$="Y" OR I$="y" THEN 230 ELSE IF I$="N" OR I$="n"
THEN I$="Q":GOTO 200
680 V=2:H=34:A$="The object of ROTATE is to":GOSUB 50
690 V=3:A$="put the letters A-O in alpha-":GOSUB 50
700 V=4:A$="betical order. This is done":GOSUB 50
710 V=5:A$="by rotating groups of four":GOSUB 50
720 V=6:A$="letters clockwise one posit-":GOSUB 50
730 V=7:A$="ion. The letters are number-":GOSUB 50
740 V=8:A$="ed 1-16 as shown below.":GOSUB 50
750 V=9:A$=" 1  2  3  4   However, only":GOSUB 50
760 V=10:A$=" 5  6  7  8   the numbers":GOSUB 50
770 V=11:A$=" 9 10 11 12   1,2,3,5,6,7":GOSUB 50
```

```
780 V=12:A$="13 14 15 16  9,10 & 11 are":GOSUB 50
790 V=13:H=48:A$="valid moves.":GOSUB 50
800 V=14:H=35:A$="Press any key to continue":GOSUB 50
810 I$=INKEY$:IF I$="" THEN 810
820 V=2:H=34:L=29:W=12:CH=32:GOSUB 40
830 V=2:A$="To make the puzzle easier to":GOSUB 50
840 V=3:A$="solve you can exchange any 2":GOSUB 50
850 V=4:A$="adjacent letters. This move":GOSUB 50
860 V=5:A$="is only allowed once, so be":GOSUB 50
870 V=6:A$="careful not to use it too":GOSUB 50
880 V=7:A$="early. As with other moves,":GOSUB 50
890 V=8:A$="a valid exchange can be:":GOSUB 50
900 V=9:A$="1,2,3,5,6,7,9,10 & 11.":GOSUB 50
910 V=10:A$="Press <Q> at any time to":GOSUB 50
920 V=11:A$="abort the current game and":GOSUB 50
930 V=12:A$="return to the menu.":GOSUB 50
940 I$=INKEY$:IF I$="" THEN 940 ELSE RETURN
```

## Source code for the ML routine

```
00100        LD     HL,15361   ;start of video+1
00110        DEC    L          ;now start of video - done
00120                          ;to avoid byte of 00H.
00130        LD     DE,0101H   ;bytes to be POKEd with
00140                          ;value of screen pos.+257.
00150                          ;done to avoid potential
00160                          ;bytes of 00H.
00170        DEC    D          ;now adjust MSB
00180        DEC    E          ;and adjust LSB
00190        ADD    HL,DE      ;HL now points to top
00200                          ;left corner of box
00210        LD     B,1        ;height of box - actual
00220                          ;value POKEd from BASIC
00230 LOOP1  PUSH   BC         ;save heigth counter
00240        PUSH   HL         ;save screen position
00250        LD     B,1        ;length of box - actual
00260                          ;value POKEd from BASIC
00270 LOOP2  LD     (HL),1     ;chr to screen - will be
00280                          ;POKEd from BASIC
00290        INC    HL         ;next screen position
00300        DJNZ   LOOP2      ;go do it again
00310        POP    HL         ;restore screen position
00320        LD     DE,0140H   ;DE=320 (256+64)
00330        DEC    D          ;subtract 256 - done to
00340                          ;avoid  byte of 00H
00350        ADD    HL,DE      ;HL now points to the
00360                          ;next line
00370        POP    BC         ;restore height counter
00380        DJNZ   LOOP1      ;go do it again
00390        RET               ;return to program
```

*The inspiration for this program must be credited to David Ahl, editor of "More Basic Computer Games" from Creative Computing. He wrote the first version of Rotate and, though this is a complete rewrite, had I not thumbed through his book, I would have never thought of this game.*

**MORE GOODIES FOR YOUR TRS-80**

**Get the latest issue of TRSLINK**

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

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

Believe it or not:
TRSLINK is FREE

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

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

**VTHG**

**BOX 9747**
**N. HOLLYWOOD, CA. 91609**

**SUPPORT for your TRS-80**

**THE ONLY MONTHLY PUBLICATION THAT SUPPORTS YOUR MODEL I, III, 4, 4P & 4D**

**CONCENTRATION IS ON THE USER APPLICATION OF PROGRAMS, SOURCES OF PRODUCTS, PRODUCT REVIEWS, FEED BACK LOOP AND NEWS ITEMS FOR THE TRS-80 USER**

**SUBSCRIPTION RATE: $24.00**

**Computer News 80**
**P.O. Box 680**
**Casper, Wyoming 82602-0680**
**(307) 265-6483**

# A Tale of Two File Formats
## How to convert Macintosh graphics for use with your TRS-80

**by Ben Mesander**

I like to do computer graphics, so it might seem odd that I own a TRS-80. However, I have found that if I am persistant enough, I can do almost anything on my TRS-80. This program allows me to convert graphics created on Apple Macintosh computers to a TRS-80 format that I can edit and print out.

My program, MAC2PWR/C, requires that you own the Powerdot program available from Powersoft. This is a great program that allows creation of huge high-resolution graphic files on your TRS-80. Powerdot does not need a high-resolution graphics board, because it uses the screen as a scrolling window on a large graphics page. Graphics files may be larger than memory - up to the size of your disks. Powerdot runs on model 1, 3, and 4 in 3 mode, as well as LOBO, PMC, and LNW computers

MAC2PWR is written in the small C language on a model 1 running NEWDOS/80 V2.0 with my own C compiler, but it should not be very hard to convert it to other operating systems and compilers - but it is not trivial either. I believe LDOS users have a compiler called LC that should work, Misosys sells a C compiler, and Alcor also made C compilers for the 1,3, and 4. There is a public-domain compiler from the Valley TRS-80 Hackers' Group (VTHG) for the model 3 called ZC that should work, but my program will require quite a bit of modification to work with it. MAC2PWR pretty much sticks to the letter of the CP/M and MSDOS small C standards.

Macintosh disks cannot be read on a TRS-80, so the graphic files must be downloaded from a BBS. The ones you are looking for are fairly common, and usually have the extension .MAC on MSDOS boards. Others may call them ReadMac files. This program does not seem to work on all MacPaint graphics, but out of the ten I have tried it on, only one has failed to convert properly.

MacPaint files are stored in a compressed format, because otherwise the files would be very large. A conversion program first must skip the first 512 bytes of the Macintosh file. If the 513th byte of the file is a zero, skip an additional 128 bytes. Now the file is positioned at the beginning of the dot data, which consists of a bit map 576 pixels wide by 720 pixels tall. Each line is compressed in a format that is best described by example:

First there is a byte which specifies whether or not the data is packed, and is also the count byte. It is a negative number if packed (i.e. the high bit is set). If the high bit is set, then that complete byte is a two's complement number that tells you how many bytes were packed. If it is a positive number, then it is simply a zero-based count of how many discrete data bytes there are. Consider the following example (all in hexadecimal):

Unpacked data: AA AA AA 80 00 2A AA AA AA AA 80 00 2A 22 AA AA AA AA AA AA AA AA AA AA

After being packed by the Mac program MacPaint:
FE AA ; (-(-2) + 1) = 3 bytes of pattern AA Hex
02 80 00 2A ; (2) + 1 = 3 bytes of discrete data
FD AA ; (-(-3) + 1) = 4 bytes of pattern AA
03 80 00 2A 22 ; (3) + 1 = 4 bytes of discrete data
F7 AA ; (-(-9) + 1) = 10 bytes of pattern AA

or: FE AA 02 80 00 2A FD AA 03 80 00 2A F7 AA
(note the savings from the unpacked data!)

The Powerdot graphics file format is based on the TRS-80 graphics characters. The first two bytes of a file are the width of the file in characters - remember each character is two pixels wide, so our width will be 576 / 2 = 288 (120 Hex). The bytes are in LSB, MSB order so the first two bytes of the file will be 2001 (Hex). The minumum width is 64 decimal which is the width of the screen. Next there is a comment field of 200 bytes. There may be a text string in this field terminated with a 0D hex (carriage return). Next there is a 54 byte unused area before the start of dot data, which begins on the second sector of the file (2 + 200 + 54 = 256). The dot data is stored as TRS-80 graphics characters until the end of file. A converted Macint osh file takes about 69K of disk space, so be sure you have enough!

To compile the conversion program with my small C compiler, issue the following commands:

```
cc -lop mac2pwr/c mac2pwr/mac
m80 mac2pwr, =mac2pwr
l80 mac2pwr,clib-s,mac2pwr-n,-e
```

To run the program type:

```
mac2pwr macintosh_file_name powerdot_file_name
```

**System requirements:**
**Model I, III, 4 (III mode) - NEWDOS/80 v2.**
**POWERDOT program from POWERSOFT**
**a C compiler (small C is OK.)**

The program will open the files and ask you for the message to insert in the Powerdot header. After that, it will unpack the Macintosh file line by line and convert it to Powerdot format. It will tell you what graphic line it is currently working on. When it reaches line 720, conversion is done, and the program exits to DOS.

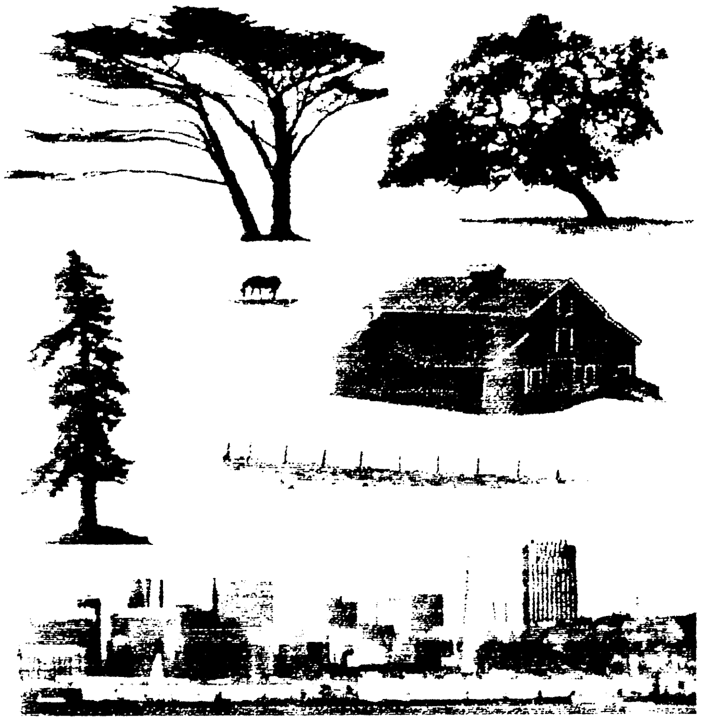
You can then edit the files with Powerdot or print them out. The two illustrations in this article show converted Macintosh graphics printed out with Powerdot.

*Powerdot is from:*
*Powersoft*
*17060 Dallas Parkway, Suite 114*
*Dallas, Texas 75248*

*Powerdot is a copyright of Powersoft.*
*MacAnything (except for MacDonald's) is a copyright of the Apple computer Company.*

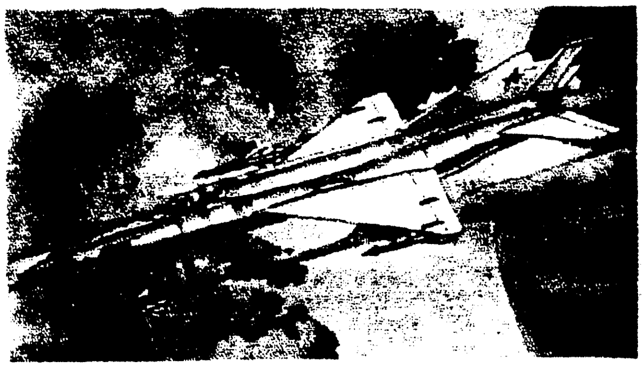
## MAC2PWR/C

```
/*------------------------------------------------------------
convert MacPaint/MacBinary format paint files to Powerdot
format - Ben Mesander
------------------------------------------------------------*/
#include      "stdio/h"
#define BEGIN  0     /* cseek from begin of file */
#define CURRENT 1     /* cseek from current point in file */
#define       MACWIDE 576   /* width of mac file */
#define MACBYTE 72    /* 72 bytes per line */
#define MACLEN  720   /* length of mac file */
#define MACHDR  512   /* length of mac header */
#define MACBIN  128   /* length of addtnl padding for Mac-
Binary */
#define ISCOUNT 0x0080 /* mask to see if Mac file counter
*/
#define TEXTLEN 200   /* length of text in pwr header */
#define PWRHDR  256   /* length of pwrdot header */
#define PWRWIDE MACWIDE/2      /* width of powerdot file
*/

main(argc,argv)
int   argc;
int   *argv;
{
    FILE   ifd,ofd;      /* input & output file descriptors */

    printf("mac2pwr/c 1.0  Ben Mesander\n");
    if (argc != 3) {
        fprintf(stderr,"usage:\nmac2dot <mac_file <power
dot_file>");
        exit(1);
    }

    ifd = efopen(*(argv+1),"r");
    ofd = efopen(*(argv+2),"w");

    printf("Creating Powerdot header.\n");
    pwr_header(ofd);            /* make powerdot header */
    printf("Reading MacPaint header.\n");
    mac_header(ifd);            /* skip Macpaint header */
    printf("Converting graphic...\n");
    convert(ifd,ofd);           /* convert bit maps */
}
/*------------------------------------------------------------
pwr_header creates the powerdot header record
------------------------------------------------------------*/
void pwr_header(fd)

FILE   fd;
{
    int    i;
    char   *buf;

    buf=memory(TEXTLEN);
    printf("Enter the message to put in Powerdot header: ");
    fgets(buf,TEXTLEN,stdin);

    /* write out 2 byte Powerdot width field */
    i = PWRWIDE;   /* width of powerdot file */
    write(fd, &i, 2);     /* write out 2 byte length */

    /* write out header string */
    write(fd, buf, strlen(buf));

    /* now pad to 256 bytes total length */
    for (i=strlen(buf)+2; i <= PWRHDR-1; i++) {
        fputc('\0', fd);
    }

    if (ferror(fd)) {
        fprintf(stderr,"error writing Powerdot header\n");
        exit(1);
    }
    free(buf);
}
/*------------------------------------------------------------
mac_header skips over the MacPaint header
------------------------------------------------------------*/

void mac_header(fd)

FILE fd;

{
    char   c;

    cseek(fd, MACHDR, BEGIN);      /* skip over mac header
*/
    read(fd, &c, 1);            /* get current byte */
    if (c==0) {
        cseek(fd, MACHDR+MACBIN, BEGIN);        /* pack-
ed with Macbinary */
    }
    else {
        cseek(fd, MACHDR, BEGIN);           /* not Mac-
Binaried */
    }

    if (ferror(fd)) {
        fprintf(stderr,"Error skipping Mac header\n");
        exit(1);
    }
}
/*------------------------------------------------------------
convert(ifd,ofd) converts the MacPaint file positioned at the
start of dot data on fd "ifd" to the Powerdot file positioned to
the start of dot data on fd "ofd"
------------------------------------------------------------*/

void convert(ifd,ofd)

FILE   ifd,ofd;

{
    char   *ibuf,*obuf;
    int    x,y,macline;

    obuf = memory(PWRWIDE); /* allocate powerdot buffer
*/
    pad(obuf,0x0080,PWRWIDE);     /* clear buffer */
    y = 0;              /* graphics line in powerdot file */
    x = 0;              /* graphics column in powerdot file */
    ibuf = memory(MACBYTE); /* allocate MacPaint buffer */
    auxbuf(ifd,4000);      /* buffer input */
    auxbuf(ofd,4000);      /* and output */

    /* convert */

    for (macline=1; macline <= MACLEN; macline++) {
      fprintf(stderr,"Now processing line: %d\n",macline);
      unpackbits(ifd,ibuf);  /* unpack a line of MacPaint file */

        /* transfer a line */

        for (x=0; x <= MACWIDE-1; x++) {
            if (mactest(x,ibuf)) {
                pwrset(x,y,obuf);
            }
        }

        /* if Powerdot buffer full, dump it */

        if (y==2) {
            write(ofd, obuf, PWRWIDE);
            if (ferror(ofd)) {
                fprintf(stderr,
                    "Error writing Powerdot dot data");
                exit(1);
            }
            pad(obuf,0x0080,PWRWIDE);
        }
        y = (y+1)%3;   /* increment y pointer */

    }
}
/*------------------------------------------------------------
unpackbits unpacks a line of the Macpaint file at a time
------------------------------------------------------------*/
void unpackbits(fd,buf)

FILE   fd;
char   *buf;

{
    int    numbytes;      /* number of unpacked bytes */
    int    n;
    char   c;          /* input char */
    char   *curpos;

    curpos = buf;
    numbytes = 0;
    while (numbytes  MACBYTE - 1) {
        read(fd, &c, 1);      /* get character */
        /* literal */
        if ( !(c & ISCOUNT) ) {
            n=c+1;        /* number of bytes to xfer */
            read(fd, curpos, n);
            curpos += n;   /* update line ptr */
            numbytes += n;  /* update number of bytes */
        }
        /* repeat count */
        else if ( c != ISCOUNT ) {
            n=(-c)+1;     /* number of bytes to xfer */
            read(fd, &c, 1);      /* get rep byte */
            pad(curpos, c, n);
            curpos += n;
            numbytes += n;
        }
        /* if c = ISCOUNT, nop */

    }

    if (ferror(fd)) {
        fprintf(stderr,"Error reading MacPaint dot data");
        exit(1);
    }
}
/*------------------------------------------------------------
mactest  tests to see if a bit is set in the MacPaint buffer
------------------------------------------------------------*/

mactest(x,buf)

int    x;
char   *buf;

{
    int    mask;

    mask = 0x0080 > (x & 7);     /* bit within byte */
    return ( *(buf + (x > 3) ) & mask);
}
/*------------------------------------------------------------
pwrset(x,y,buf) sets a point in the powerdot file
------------------------------------------------------------*/
void pwrset(x,y,buf)

int    x, y;
char   *buf;

{
    int    mask;

    mask = 0x0001 < ( (y < 1) + (x&1));

    *(buf + (x > 1) ) |= mask;
}
/*------------------------------------------------------------
memory - allocates a block of memory, returns a pointer,
 checks for errors
------------------------------------------------------------*/
memory(nbr)
int    nbr;
{
      char    *ptr;

      if ( (ptr = malloc(nbr)) = = NULL ) {
            fprintf(stderr,"out of memory.");
            exit(1);
      }
      return (ptr);
}
/*------------------------------------------------------------
open a  file and abort on an I/O error
------------------------------------------------------------*/
efopen(file,mode)
char *file;
char *mode;
{
      FILE fd;
      if ( (fd=fopen(file, mode)) = = NULL ) {
            fprintf(stderr,"\nError opening: %s in mode %s\n"
                  ,file,mode);
            exit(1);
      }
      return(fd);
}
```

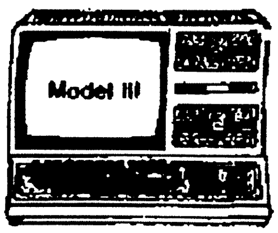


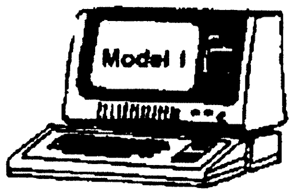
I would like to make a correction to my LPRINT article from the Mar/Apr 1989 issue (page 10). The system requirements are stated there as being:
Model I
NEWDOS/80 v2.
EDTASM
STAR NX-1000.

This is not correct. The only system requirements are:
Model I
EDTASM



*Ben Mesander can be reached at 11237 E. Brooks Street Apt 4., Norman, Oklahoma 73071*





*Editors note:*

*The original graphics which accompanied Ben's article are wonderful examples of what can be done. Unfortunately, the above, which are third generation photo-copies, do not do justice to the originals. Use your imagination and picture them sharp and clear.*

# Assembly 101
## Z-80 without tears
**by Lance Wolstrup**

*Up to now we have concentrated on the Assembly language versions of the PRINT command. Here is a quick recap what we have learned so far:*

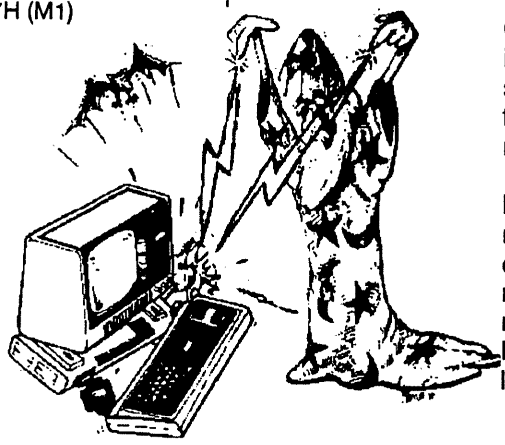
**PRINT**

Load the HL register with the address of the first character of the text to display. Then CALL 21BH (model III), or CALL 4467H (model I).

```
00100        ORG    7000H
00110 MSG1   DEFM   'HI, I AM YOUR TRS-80'
00120        DEFB   0DH
00130 START  LD     HL,MSG1
00140        CALL   21BH - CALL 4467H (M1)
00150        RET
00160        END    START
```

**PRINT@**

Load register HL with the PRINT@ location + 15360.
Load the contents of HL into (4020H).
Then load register HL with the address of the first character of the text to display.
Finally, CALL 21BH (model III), or CALL 4467H (model I).



```
00100        ORG    7000H
00110 MSG1   DEFM   'HI, I AM YOUR TRS-80'
00120        DEFB   0DH
00130 START  LD     HL, 15424
00140        LD     (4020H),HL
00150        LD     HL,MSG1
00160        CALL   21BH - CALL 4467H (M1)
00170        RET
00180        END    START
```

**CLS - clear the screen**

**CALL 1C9H**

```
00100        ORG    7000H
00110 START  CALL   1C9H
00120        RET
00130        END    START
```

OK, now let's get on with some new stuff. When you program in Basic, you might decide to POKE the text or graphics to the screen instead or PRINTing. For example, let's POKE the text "HI, I AM YOUR TRS-80" to the screen beginning at the second line (line 1). That would be POKE location 15424. The code might look something like this:

```
10 HL$="HI, I AM YOUR TRS-80"
20 HL$=HL$+CHR$(13)
30 HL=1
40 DE=15424
50 A=ASC(MID$(HL$,HL,1))
60 IF A=13 THEN 110
70 POKE DE,A
80 HL=HL+1
90 DE=DE+1
100 GOTO 50
110 END
```

Notice that line 20 adds CHR$(13) to the end of HL$. This is done because line 60 checks to see if A=13. If A does equal 13, then the program knows we have reached the end of the string.

I know that this code is sort of backwards and certainly, if this was meant to be a Basic program, we could have written it somewhat more elegantly. However, this is meant to be an example that can be translated directly to Assembly language, so let's do just that:

```
00100        ORG    7000H
00110 MSG1   DEFM   'HI, I AM YOUR TRS-80'
00120        DEFB   0DH
00130 START  LD     HL,MSG1
00140        LD     DE,15424
00150 LOOP   LD     A,(HL)
00160        CP     0DH
00170        JP     Z,EXIT
00180        LD     (DE),A
00190        INC    HL
00200        INC    DE
00210        JP     LOOP
00220 EXIT   RET
00230        END    START
```

This program introduces three new commands: CP (compare), INC (increment) and JP (jump). Also, we are using register A and register DE for the first time.

The CP instruction always compares the specified value to the contents of register A. In essence, it subtracts the specified value from the value in register A and, depending on the outcome, it manuipulates certain bits in register A's companion register: F (also known as the

FLAG register - we will look at this in just a few minutes). Do keep in mind that, even though the specified value was subtracted from the value stored in register A, *the value in register A remains unchanged... only register F is changed.* You might think of the CP instruction as Assembly language's IF - THEN statement.

The INC instruction is very simple. It INCrements the contents of the specified register by 1.
*INC HL does to register HL what HL = HL + 1 does to variable HL in BASIC.*

The last new instruction is JP. This does exactly the same as the GOTO command in Basic.

Now, let's compare the Assembly language program, line by line, to the Basic program:

00100 **ORG 7000H**, mandatory in Assembly - is not needed in Basic.

00110 **MSG1 DEFM 'HI, I AM YOUR TRS-80'** is the same as: 10 **HL$ = "HI, I AM YOUR TRS-80"**

00120 **DEFB 0DH** serves the same purpose as:
20 **HL$ = HL$ + CHR$(13)**. That is, the text now has an ending byte that will be checked for in line 60 of the program.

00130 **LD HL,MSG1** points register HL to the first byte of the text in MSG1.
Basic line 30 **HL = 1** points to the first character of HL$ when we get down to line 50.

00140 **LD DE,15424** is exactly the same as:
40 **DE = 15424**. Register DE (in Assembly) and variable DE (in Basic) now points to the screen location.

00150 LOOP **LD A,(HL)** is identical to:
50 **A = ASC(MID$(HL$,HL,1))**.
Here we better stop and analyze exactly what is going on, both in the Assembly and Basic line. First, in the Basic line let's figure out just what MID$(HL$,HL,1) means. Since HL$ is "HI, I AM YOUR TRS-80" and HL at this point in the program equals 1, then we are looking at the first character of HL$, "H". We now take the ASC value of that character (72) and put that into variable A. This variable now holds the value 72.
The Assembly line is much less complicated, but needs explanation because we are using a new concept. First, register A is very special. It is the only one of the 8-bit registers that is capable of performing math functions. Since we will need to see if we have reached the ending byte, we must put each character into the A register one at a time. At this time register HL is pointing to the first character of the text, therefore we copy the character pointed to by HL into register A. Note that when we surround a register pair with a parenthesis, the value stored in the register is treated as a memory location. Thus LD A,(HL) means: get the value stored in the memory location pointed to by register HL and copy it to register A. (whew!!)

00160 **CP 0DH**. Since the A register is the only one capable of math, CP 0DH means: compare register A to 0DH.
This is the first half of Basic line 60: **IF A = 13.....**
The second half is found in line 00170

00170 will be discussed in more detail below. For now, **JP Z,EXIT** means:
*if the A register holds a value identical to the one we specified in the CP instruction (0DH), then we will jump to the label called EXIT.*
This is the second half of Basic line 60: **.....THEN 110.**
(note that 0DH is 13 decimal and the EXIT label has a RET instruction which brings us back to DOS; line 110 in the Basic program simply ends the program).

00180 **LD (DE), A**. Since we did not jump to EXIT, obviously the value in register A was *not* the ending byte 0DH. Therefore it is part of the text, and we put it into the memory location pointed to by register DE, which is a screen location. Thus, the character is displayed on the screen. Basic line 70 **POKE DE,A** does just that.

00190 **INC HL** increments register HL to point to the next character in the string. Line 80 in the Basic program, **HL = HL + 1**, provide the same function.

00200 **INC DE** increments register DE to point to the next screen location, same as line 90 in Basic:
**DE = DE + 1**

00210 **JP LOOP** provides the same function as Basic line 100 **GOTO 50**. We go back and check the next character if it is CHR$(13). This loop continues until CHR$(13) is stored in the A register. Then the loop is broken and program flow is directed to the EXIT label or, in the case of the Basic program, to line 110.

00220 **EXIT RET** is where we end up when the end byte, CHR$(13), is found. Since we CALLed this program from DOS, the RET instruction returns us to the caller, thus ending the program. Line 110 **END** in the Basic program is only needed because we chose to GOTO 110 when CHR$(13) was detected.

00230 **END START** - the END statement is mandatory in Assembly language - not needed in Basic.

## F - THE FLAG REGISTER

The flag register F is never used to hold data. It contains several bits, logically called 'flags', that are set according to the *RESULTS* of other instructions. It is an 8-bit register, even though there are only six flags, and only four of these are are really important for most programming purposes.

```
Bit  7  6  5  4   3  2  1  0
Flag S  Z     H      P/V N  C
```

The four important flags are:
Z (zero flag)
S (sign flag)
C (carry flag)
P/V (parity/overflow flag)

The other two flags are:
H (half-carry flag)
N (add/subtract flag)

This tutorial will not discuss the H & N flags. Frankly, I have never used them, nor do I remember ever seeing an application that did. We will also ignore the S & P/V flags. They have no relevance to what we are trying to accomplish, at least for the moment.

This leaves us with the Z and C flags.

- The Z flag is set ONLY if the result of an operation is zero.

- The C flag (*don't confuse this with register C*) is set whenever an add instruction produces a result that is too large to store in a single register. It is also set when a subtract operation produces a borrow. Other instructions will also affect this flag.

Relating this to our program (lines 00160 & 00170), we compared the contents of register A to the value 0DH and the outcome of this operation was stored in the pertinent flags.

There are basically six different ways you can compare one thing to another:

1. EQUAL TO
2. NOT EQUAL TO
3. GREATER THAN
4. LESS THAN
5. EQUAL TO OR GREATER THAN
6. EQUAL TO OR SMALLER THAN

If the value of register A is EQUAL to the specified value, the Z flag is set (bit 6 of register F is 1), the C flag is reset (bit 0 of register F is 0).

If the value in register A is NOT EQUAL to the specified value, the Z flag is reset (0). Should we wish to find out *how* they are not equal, we consult the C flag.

If the value in register A is LESS THAN the specified value, the C flag is set (1). If, on the other hand, register A has a value LARGER THAN (or equal to) the specified value, the C flag is reset (0).

Thus, since we merely wanted to find out if register A is EQUAL to the specified value (0DH), we JP to label EXIT only when the Z flag is set.

Both Z and C can be used for their alternate state; that is, NZ (non-zero) and NC (non-carry). This enables us to make comparisons using four of the six methods mentioned above:

- JP Z,EXIT - jump if the comparion is EQUAL

- JP NZ,EXIT - jump if the comparison is NOT EQUAL

- JP C,EXIT - jump if value in register A is LESS THAN the specified value

- JP NC,EXIT - jump if value in register A is EQUAL TO OR GREATER THAN the specified value

'GREATER THAN' and 'EQUAL TO OR SMALLER THAN' are really not needed, as they can be accomplished using the above four methods.

Boy, I am running out of room, so before quitting, let me quickly tell you that in the next installment we will get to the Assembly language versions of INKEY$ and INPUT. We will also begin writing our mailing label program.

Until next issue........keep practicing.

---

*** NEW ***

**Recreational & Educational Computing**

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

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

the Magic of Schram's 123 String - the probability of an N game at Bingo - time to complete a collection - 6174 - Next Number in Sequence - Locate the Bomb - perfect numbers - Fibonacci numbers - prime number generation and contest - self-reference and paradoxes - self-listing program challenge and solution - pi - mystery programs explained - probability - Monte Carlo simulations
Also:
Fractal art - the world's best card trick (based on algebra) reviews of best software and books - editorial - humor - cartoons - art - reader solutions and more!

Programs supported for: TRS-80, Tandy, MS-DOS and others.
REC is available for $24.00 per calendar year of 8 issues

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

# A REVIEW OF: MAGIC MATH PLUS

*Dr. Michael W. Ecker*
*Recreational Mathematical Software*
*129 Carol Drive Clarks Summit, PA 18411*
*(717) 586-2784*

## Reviewed by Dr. Allen Jacobs

Occasionally, a software package comes along to remind us of our mathematical roots. Such a package is "Magic Math Plus". It is a menu driven compilation of demonstrations and games embodying several basic numerological and mathematical principles. In essence, it is a "mathematical museum" on a disk.

The entire presentation is comprised of 35 "exhibits" on 5 menus. It was written by Dr. Michael W. Ecker, Mr. David B. Lewis, Mr. Jim Kyle, and Mr. Edward M. Roberts and is available at a cost of $37.50 + $2.50 shipping & handling.

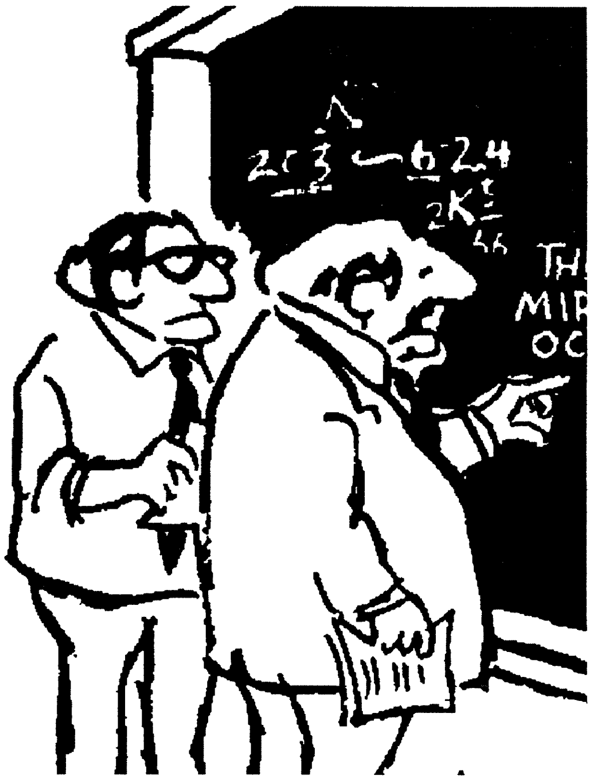
The disk uses a proprietary Model III disk operating system named XDOS, which is compatible with TRSDOS 1.3 format. The system automatically loads Basic, fillowed by an on-screen introduction, a user involved upper case test, and then the first menu named Volume 1 arrives. Thereafter, the entire package runs under a menu driven shell, in Basic.

Selecting a choice from the menu is as easy as using the UP and DOWN ARROW keys to point to your choice and pressing <ENTER>. The selected program loads and runs.

After a title screen, a self documenting introduction to each program usually appears. It tells the user what the program will do and how to interact with it. Usually, some structured input or action is required from the user. At the end of the demonstration, at the user's option, the program gives an explanation of the mathematical principle involved.

Other times the user is challenged to provide the answer himself. The <BREAK> key is always operable so that the commented basic code can be examined. After a demonstration is over, it can be restarted, or the user can return to the menu and select another program with one or two keystrokes.

The menu for each volume provides opportunities to select the menu for the previous or the next volume. One representative program from each volume follows:



**"Fastloan!"**

"Fastloan!" is a truly fast loan amortization program. Its simple screen-only I/O is adequate enough to be practical and it is approximately as fast as my *dedicated financial calculator*. Remember that this is Basic running at 2 MHz. If the program were to be run at 4 MHz on a Model 4 (even in Model III mode with the clock speed doubled), I would use it more than the calculator. This is because my calculator has no possibility of hard copy output whereas a few well placed LPRINT statements in the program can get the numbers down on paper.

"*Not important*", you say, because the monthly payment is all you need? Well, typing "A" at the appropriate prompt gets you a complete amortization table scrolling on the screen, one screen at a time, showing the Payment Number, the Interest portion of the payment, the Principal pay-down, and the Balance Owed. Your financial calculator can do this, with the proper volume of keystrokes, but you can only see one value at a time. That is unless we are talking about an expensive printing calculator.

If you don't want to bother adapting "Fastloan!" for hard copy, Dr. Ecker offers a more full featured version of the program named "Fastloan II". I would guess that it can LPRINT but you would have to write or call to find out. I have never seen "Fastloan II" but I would like to.

**"Super-Blackjack"**

"Super-Blackjack" (The game of 110) is analogous to the standard game of "Blackjack" and is as much of a challenge. The goal of the game is to accumulate a hand totalling 110, as one would normally try to hit 21 without going over (going bust), in regular Blackjack. The major difference between the games is the way in which the total for a hand is calculated. Cards are initially dealt just as in regular "Blackjack", however, each pair of cards in a hand is multiplied. The pair products are then added together. Unpaired cards are simply added, to achieve the total. Since the computer does all the math, it's easy to play and as much fun as any regular Blackjack game in Basic. Once you get used to thinking in a nonlinear manner, which takes about two hands, you are on your way to becoming a "product-sum" riverboat gambler.

**"Collatz/Ulam Conjecture"**

The "Collatz/Ulam Conjecture" will become the favorite of all you "*mathematical degenerates*" out there. The demonstration shows that any number input to it is repeatedly processed according to the following rule. The number is tripled plus 1 if it is odd, and it is halved if it is even. The pattern thus produced eventually degenerates into the numerical sequence: 4..2..1..4..2..1.

Although this conjecture apparently works every time (try it), the author did not call the demonstration a proof. The demonstration apparently does not formally prove that this pattern will arise every time it is attempted, even though it may. To me, formulating an actual proof for this "conjecture" seems as though it would be more interesting than the phenomenon itself. I can not even conceive of how such a formal proof would be "gone about". It would be interesting to research it. It appears as though it would make an excellent and inexpensive but difficult science fair project to attempt.

**"Number Guesser"**

The "Number Guesser" is a game which will pick any number you can think of within a user selected range. The computer will then make a series of guesses which will soon converge upon the number you picked.

For assembly language hackers, the principle of this trick should be second nature. Essentially, you are requested to become the conditional branch algorithm of a binary search. If this explanation does not make any sense to you, and you want to gain an insight as to how computers can look something up so rapidly, muse about this simple game. The demonstration explains itself, at your request.

**"The Fibonacci Numbers"**

"The Fibonacci Numbers" is a demonstration of the unique properties of this number series. The series is generated by each term being the sum of the two previous terms in the series. The first term is defined as 1. The second term is 1 plus nothing, so it too is 1. The third term is 1 + 1 = 2. Continuing, we get: 1, 1, 2, 3, 5, 8, 13, 21, etc.

The advantage to having the program operating in Basic over reading about it in print is that the computer is doing all the calculating for you. As long as you can verify and then "trust" the computer, you can watch the relationships between the members of the series without being distracted by having to do the math. This is true to the spirit of any "museum exhibit". You can watch the mechanism on display and learn something from it without having to run it or build it yourself.

Overall, this package seems to be most suited to three general types of computer users. One type is the user who is not as interested in programming as in being entertained by more intellectually stimulating games than "shoot-em-ups" requiring fast reflexes.

The second type of user who would be most interested in this kind of software would be the computer novice. He (she) would be both entertained by the demonstrations and can also learn some programming techniques from them.

The third category of user is the youngster who is not yet consumed with interest in computers or mathematics. Being "turned loose" on an educational package like this might just spark one of those revelations we have all had, such as when we suddenly understand or discover a new concept.

Barring these artificial categories, I think that all but the most jaded mathematics or programming professional would find this package at least somewhat interesting. Personally, I would like to have seen a greater variety of phenomena demonstrated. Namely, I wanted to see more and different examples. I guess, that this is a possible criticism of any "museum". We always want to see just one more exhibit before closing time.

Here is an additional mathematical phenomenon I just found in the Sunday comic section of the "Daily News", a San Fernando Valley newspaper. The issue of March 19, 1989 contained a strip entitled "The Family Circus" by Bill Keane, in which the following was presented: If you select any whole number from 1 to 100 and multiply it by 99, the sum of the resulting digits is always equal to 18. I tried it and it works. The question is: Why? No explanation was given in the comic strip.

It appears to me as though the answer could be worked into yet another interesting basic program. Maybe you could add your own additional menu of demonstrations to this package, using the conventions you can learn from studying its examples. Buying the package may encourage the authors to expand their efforts and create a second edition.

It is obvious that just reviewing this software has already sensitized me to the fact that many more "Magic" mathematical phenomena exist out there in the world. Some are known and some are not. The opportunity to explore this aspect of our universe is the same for all of us. It just takes the desire, the insight, the effort, and a little bit of time.

# BASIC FULL SCREEN EDITOR

by Gary Edwin Campbell

## Requirements: Model 3,4,4P,4D TRSDOS 1.3 with BASIC Rev 1.3

All model 3 basic programmers most likely edit their basic programs using the built in basic line editor. Whether you have a cassette or disk based system, use LDOS, DOSPLUS, NEWDOS or TrsDos 1.3, the ROM line editor provides adequate methods to edit basic program text. You can add/delete/change a line, and immediately run the program to test out the changes you made.

Basic programs can also be written on a word processor. As most word processors are "screen oriented", the basic program can be written much faster, due to the more advanced editing features. When you think the program is ready to test, the file must be converted to ascii format, which is then processed by Basic.

Most screen editors are separate stand alone programs. Although they offer more powerful editing capabilities, the basic code cannot be run, tested or debugged by the word processor.

The program presented here offers you the best of both worlds. All basic programs will run as usual with this utility enabled. It can be turned ON or OFF in immediate mode at any time. Line editing may still be used. When in screen edit mode, all data displayed on the video screen ( program lines or displays ) can be moved, split, inserted, deleted or typed over. Line numbers can be added or erased. Then, ANY PART of the display can be "sent" to basic as a NEW/OLD program line, or as an immediate mode command line. The program is self relocating, uses only 532 bytes, requires no high memory setting, and is 99.99% compatible with all basic programs. Once installed, it becomes part of Disk Basic itself, reserving its own space in low memory usually allocated for disk file buffer Input/Output.

PLEASE NOTE:
*The version published here is for use with either TRSDOS 1.3 Basic Rev 1.3 or LIBDVR/CMD (System 1.5). If you desire to add these capabilities to LDOS 5.1.3, LDOS 5.3 or cassette basic, please see the end of article for ordering information.*

Type in the basic program titled EDITOR/BAS.
Save it as EDITOR/BAS.
Run it to create EDITOR/CMD assembler program.

Now you are ready to "test" it out. Exit to DOS, and enter Basic as required by your basic program. (ie: set files/memory etc.)

At the Basic "Ready prompt" type:

```
CMD"L","EDITOR/CMD"
DEFUSR=&H4E00H
X=USR(0)
```

If you make a syntax error before typing the last line, you must start over.

These commands load and initialize the screen editor. This is required only once each time Basic is entered. As the initialization code calls the NEW command ROM routine, any basic program in memory will be deleted.
**BASIC *** should not be used to re-enter Basic once the editor has been installed.

Now, load or run your basic program as usual. Hitting the BREAK key at the basic ready prompt '>' (command mode) toggles the editor on or off. The <BREAK> key does not affect the ON & OFF status when a basic program is running. When the editor is ON, you will see an asterisk flash at the top right hand corner of the video. This reminds you that you are in screen edit mode. To exit the screen editor, hit the <BREAK> key at any time. The flashing will stop, and you are returned to normal input mode.
The screen editor uses the following keys:

| | |
|---|---|
| Up Arrow | Moves non destructive cursor up. |
| Down Arrow | Moves non destructive cursor down. |
| Left Arrow | Moves non destructive cursor backward. |
| Right Arrow | Moves non destructive cursor forward. |
| BREAK key | Enters or Exits Screen Edit Mode. |
| CLEAR key | Marks the start of video text. |
| ENTER key | Marks end of text, sends text to BASIC. |
| Left Shft CLEAR | Clears the screen, marks start (CLEAR). |
| Left Shft ENTER | Clears the screen, LISTS your program. |
| Left Shft < | Deletes the character after the cursor. |
| Left Shft > | Inserts a space at cursor position. |
| Left Shft Up Arrow | Moves text below cursor up 1 line. |
| Left Shft Down Arrow Z | Moves text after cursor down 1 line. |
| Cntl Z | Moves text after cursor down. (Mod 4) |
| Cn1,n2 | Copies existing Line n1, inserts it as Line n2 (This immediate mode command supported only when editor is OFF!) |

All other keys react as usual, except other control codes are ignored. To print a < or > character, use the right shift key.

Ok, lets see what we can do! Load and initialize the screen editor as directed above. At Basic ready, enter the following:

```
10 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
20 K=2:P=101:PRINTSTRING$(30,191)
```

To enter the screen edit mode, hit the <BREAK> key.

You should see the asterisk flashing at the top of your screen.

Tap your arrow keys. See how the cursor moves?

Hold down the Left Shift key, and press <ENTER>. Your program will be listed.

Position the cursor to the top of the screen. The cursor should now be right on top of the first character of the line number of line 10.

When the cursor is on top of the 1 press the <CLEAR> key. This notifies the screen editor to mark the starting position.

Now, type over the '10' by pressing the 3 and 1 keys. Your display should look like this:

```
31 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
20 K=2:P=101:PRINTSTRING$(30,191)
```

Move the cursor to the end of line 31, just after the PRINTJ program text.

Now tap the <ENTER> key. You will notice that a > character appears in about 1/4 second.

As you have probably guessed, this procedure COPIES or REPLICATES a line. Line 31 should now be added to your program.

List your program by holding down the left shift key, and hit <ENTER>. You should see the following:

```
10 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
20 K=2:P=101:PRINTSTRING$(30,191)
31 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
```

Remember, we marked the beginning of the line with <CLEAR>. We marked the ending of the line with <ENTER>. The screen editor tricked the Basic Interpreter into thinking that the text between the two markers was typed in from the keyboard!

Another way to copy a line is by using the Cn1,n2 command. The Cn1,Cn2 command cannot be entered when the editor is on.

Exit the editor by hitting <BREAK>.
Now type: C10,11 <ENTER>.

Now list your program. It should look like this:

```
10 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
11 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
20 K=2:P=101:PRINTSTRING$(30,191)
31 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
```

The C10,11 command works by locating the existing line 10 in memory. If found, the line text is moved to an internal buffer. Then the line is re-numbered, and sent to basic for inserting.

Ok, lets restore our original program. Delete line 11 and line 31.

D11 <ENTER>
D31 <ENTER>
*Note: the L, D, E and A (List,Delete,Edit,Auto) commands cannot be used when editor is on!*

Turn the screen editor back on by hitting the <BREAK> key.

The editor can also send commands to basic without line numbers. The <CLEAR> key marks the start of text. The <ENTER> key marks the end of text. The text between is treated as if it was typed in directly from the keyboard. If the text between the <CLEAR> and <ENTER> markers was:

```
CMD"D:0":?"HELLO"
```

the computer would display the catalog of drive 0, and then print "HELLO". You would then be returned to screen edit mode.

So far, just basic program text has been edited. But a screen editor has another use. Video displays, including graphics, can also be edited. For example, if the CMD"D:0" command displayed:

```
Drive:0  MYPRG/BAS   DATA/TXT     BASIC/CMD   TEST
         TYPE/CMD    SPOOL/CMD    RECIPE/BAS
READY
```

By moving the video display over a few characters using Left Shft > keys, line numbers can easily be added. The Catalog data above can then be "stored within a program, as shown below:

```
1 ?" Drive:0  MYPRG/BAS  DATA/TXT    BASIC/CMD  TEST
              TYPE/CMD   SPOOL/CMD   RECIPE/BAS"
Ready
```

The Left Shift < and > key combinations will insert a blank space, or delete a character at the cursor position.

Restore your video by hitting the Left Shift Enter key.

If the cursor was positioned on top of the first 'N' in line 10, and the Left Shift key and the > was pressed, this would happen:

Before:
```
10 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
```
After:
```
10 FORI=1TO4: NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
```

If the < was pressed, the character *following* the cursor is deleted. If the cursor was on the N, the E gets deleted. ei:

Before:
```
10 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
```
After:
```
10 FORI=1TO4:NXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
```

The Left Shift Down Arrow Z will move the program text located after the cursor character down 1 line, opening a 64 space line buffer. For example, if the cursor was positioned on the first colon on line 10, and the Left Shift Down Arrow Z was pressed:

Before:
```
10 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
20 K=2:P=101:PRINTSTRING$(30,191)
```
After:
```
 10 FORI=1TO4:
                NEXTI:FORJ=1TO12:NEXTI:PRINTI:PRINTJ
 20 K=2:P=101:PRINTSTRING$(30,191)
```

The Left Shift Up Arrow will move the program text directly below the cursor up. By using the Left Shift key, with the Left Shift Up Arrow keys, you can do a "cut and paste".

Before:
```
10 FORI=1TO4:NEXTI:FORJ=1TO12:NEXTJ:PRINTI:PRINTJ
20 K=2:P=101:PRINTSTRING$(30,191)
```
After:
```
10 FORI=1TO4:PRINTSTRING$(30,191)
20 K=2:P=101:
```

Another nice advantage of a screen editor is that graphics can also be stored within program text. By sending a immediate mode command to print a variable that contains graphics, you can eliminate the CHR$(X)+CHR$(X) text. For example, type:

```
<CLEAR>A$=CHR$(244)+CHR$(245)+CHR$(246):
?"A$="CHR$(34)A$<ENTER>
```

Basic will respond , storing the graphics into variable A$. If the graphics switch is on (not space compression codes) the results that are printed on your screen can be stored into your basic program. (This example prints a little pointing hand). To turn on or off the graphics/compression toggle, use the command PRINTCHR$(21) <ENTER>. This type of editing saves memory space, uses no string storage (as in CLEAR 1000), and improves speed.

Program lines can also be moved from one program to another. Just list the line(s), load your new program, and store it!

## About the program:

Disk Basic calls address 41AFh to process immediate mode input. Extended Disk Basic commands, such as L(ist), D(elete), E(dit) A(uto) & the ., keys are enabled by DOS calling address 41AFh. This call address jumps to 57DAh. The code required to process these "added" commands starts at address 57DAh. The Cn1,n2 Copy command is "patched" into this area. Before DOS checks for the L D E or A bytes, it now checks for a C command. This is how the copy command "links" itself to the basic interpreter.

During immediate mode keyboard scanning, a call is made to 49h. This call address waits for a keyboard character. Instead of calling 49h at 588Ch, processing jumps to BREAK key testing. The 49h call is made, and the keyboard input byte is tested. If the character is not a BREAK, control returns to the original driver. If A=01h, a flag is checked, and is then set if off. The screen editor is then enabled by loading 41AFh with the screen editors address. Instead of jumping to 57DAh when DOS calls 41AFh, it calls the screen editor instead. Unfortunately only one jump can be processed by DOS at 41AFh. This explains why abbreviated commands L, A, E, D, and Cn1,n2 are not supported when the screen driver is active.

*LDOS 5.1.3, LDOS 5.3 and cassette versions are also available from GRL Software (att: Screen Editor), for $12.95 each.*

*Other versions for TRSDOS 6.x, LS-DOS, and DOSPLUS may be written if enough pre-orders are received. If not, your cheque will be returned.*

*Please state your DOS type, DOS Version number, DOS Version date, Disk Basic Revision number, and Disk Basic Rev. date for EACH request. (The 1.3 version can also be ordered). Each order is accompanied by the documentation in this article.*

## EDITOR/BAS

```
0 'Screen Editor for TrsDos 1.3 BASIC Rev 1.3
1 'Save program before installing screen editor!
2 'To install, at Basic Ready, type:
3 'CMD"L","EDITOR/CMD":DEFUSR=&H4E00:X=USR(0)
4 '
5 'By Gary Edwin Campbell, Suite 209, 1051 KLO Road,
6 'Kelowna, British Columbia, Canada  V1Y 4X6
7 'Released to the public domain 03/27/89
8 'Send me a post card stating your interests!
9 '
10 CLS:PRINT"Checking data entry..."CHR$(14);:
CLEAR1000:DEFINTA- Z:RESTORE:D$="123456789ABCDEF"
```

```
20 READA$,LN,CK:T=0:FORI=1TOLEN(A$)STEP2:
J=ASC(MID$(A$,I,1)):K=ASC(MID$(A$,I+1,1)):
IFJ=42ANDK=42THEN50ELSET=T+J+K:NEXTI

30 IFT=CKTHEN20ELSEPRINT:
PRINT"Data entry error in line #";LN:END

50 IFTCKTHEN30ELSERESTORE

55 OPEN"O",1,"EDITOR/CMD:0":PRINT:
PRINT"Creating EDITOR/CMD file ...";

60 READA$,LN,CK:FORI=1TOLEN(A$)STEP2:
B$=MID$(A$,I,1):C$=MID$(A$, I+1,1):
IFB$="*"ANDC$="*"THENCLOSE:PRINT:
PRINT"EDITOR/CMD created !":END

70 J=INSTR(D$,B$):K=INSTR(D$,C$):
PRINT#1,CHR$(J*16+K);:NEXTI:GOTO60

100 DATA 0182004E2AA440013402C5E50936002322A440
E1E5,100,2256

101 DATA 11954EB7ED52E5C1DD21594EDD6E00DD66017
CB528,101,2437

102 DATA 16E5FDE1FD6E00FD660109FD7500FD7401DD23
DD23,102,2444

103 DATA 18E0215950223E593EC3323D5921954ED1ED53
8C58,103,2350

104 DATA C1EDB0CD4D1BC3191A3E4EA54EAB4EAE4EB74E
BD4E,104,2559

105 DATA C04ECA4ECD4EF04EFB4E164F1B502A50395045
5084,105,2421

106 DATA 501E4F204F220182804E4F244F264F284F2A4F
2C4F,106,2365

107 DATA 2E4F304F324FF94F0000CD4900FE01C03E00B7
2803,107,2359

108 DATA 3E01C93D329C4E2AB04122D34E21F24E22B041
2100,108,2326

109 DATA 002201502A134022415021345022134018 2BF3
AF32,109,2192

110 DATA 9C4E2A41502213402100002 2B0413E20323F3C
F137,110,2251

111 DATA FBC93E1A18253A8038B7C4C9012A2040220150
3E0E,111,2338

112 DATA CD3300CD490021344FE5010B0182004F00EDB1
280A,112,2318
113 DATA E1FE2038E8CD330018E32BD1B7ED5229111E4F
195E,113,2412

114 DATA 2356EBE9C74E3F4FE14EE54E944FBC4FEF4F43
4F6E,114,2532

115 DATA 4F984FC04F015B0A1F08090D1B1A3C3E3E1B18
C72A,115,2411

116 DATA 20401140001911BF3FDF30BDED5B2040E5E5C1
21FF,116,2377

117 DATA 3FB7ED42E5C1E1EDB01100403E207723DF28A1
18F9,117,2401

118 DATA 2A204011C03FDF38021894EB01FF3F21BF3F01
8280,118,2360

119 DATA 4F7E020B2BDF20F906402A20403E20772310FC
18E3,119,2340

120 DATA 3E1818A93A8038FE0128043E3C18F321FF3FED
5B20,120,2399

121 DATA 40B7ED52E5C12A204023EDB03E2032FF3F18D6
3E19,121,2397

122 DATA 18D63A8038FE0128043E3E18CB3E0FCD330021
FF3F,122,2405

123 DATA ED5B2040DF28B8B7ED52E5C121FE3F11FF3FED
B83E,123,2516

124 DATA 202A20407718A33A8038B7280BCDC90121C550
0104,124,2283

125 DATA 00181D018200501100007AB3288B2A2040B7ED
5211,125,2245

126 DATA F000DF30A77DB728A34D06002A0150ED5BA740
C5ED,126,2401

127 DATA B0AF122100002201502AA7402BC1F1AFC9F53E
003C,127,2339

128 DATA 323650FE142804F1C30000AF3236503A3F3CFE
2A28,128,2339

129 DATA 073E2A323F3C18EA3E2018F7FE4328077E21A8
59C3,129,2388

130 DATA 4059D5C5E5112542010D00EDB0212542D73805
E1C1,130,2305

131 DATA D118E3CD5A1E7EFE2C014B805020F32322B350
CD2C,131,2403

132 DATA 1B380218E8F1F1F1F1B7ED42E5110500B7ED52
E5C5,132,2406

133 DATA E101040009ED5BA7401B1BC1EDB0AF06031213
10FC,133,2358

134 DATA 210000CD5A1EC12AA7402B2BAF22E640C3A71A
4C49,134,2383

135 DATA 535402022D40**,135,623

136 ' Just another reminder! SEND ME A POSTCARD!
```

# WATER SCHEDULING

**for Model 4 - Basic**
**Model I & III with changes**
**by Elton L. Wood**

I live in a small rural community which is not blessed with an abundant supply of irrigation water. During the summer drought of 1988 the tempers of the local residents became as hot as the temperatures. In fact, during the month of July the boiling point was reached. It reminded me of the old western movies in which the upstream land owners would dam up the stream and necessitate a heroic effort by some brave individual to blow up the dam and save those who lived downstream. While not resulting in the use of dynamite, many threats and insults were exchanged and some friendly relationships were destroyed.

Because of my involvement in administering the local culinary water system, I was asked to update the irrigation watering schedules, which had not been updated for several years. Since the last preparation of the watering schedules there had been numerous divisions and sales of property, so it was not simply a matter of updating the schedules to reflect new dates. Consequently, I decided that there must be an easier way of reconstructing the schedules than laboriously tracking names, dates and times with a pencil and paper, and then plugging the results for each name into my word processor.

WATRTURN/BAS is the result of my crash effort to computerize the watering schedules and, hopefully, quench some of the heat. The program is self documenting. I have "*REMarked*" each variable and routine, thus eliminating the need for a line by line discussion of the code.

```
10 ' *** WATRTURN/BAS ***
20 ' Written for TRS-80 Model 4
30 ' (c) 1988 - Elton L. Wood
40 ' Please do not remove credit lines
50 '
60 ' Calculates & Prints Water Turn Schedules
70 ' Can be used for other cyclic schedules
80 '
90 ' Enter Data lines of Names & Hours
100 DATA NAME 1,15,NAME 2,15,NAME 3,80.5
110 DATA NAME 4,22.5,NAME 5,15,NAME 6,1,NAME 7,1
120 DATA NAME 8,2.5,NAME 9,3.5,NAME 10,1,NAME 11,2.25
130 DATA NAME 12,2.75,NAME 13,2.75,NAME 14,2.25
140 DATA NAME 15,2.75,NAME 16,2.75,NAME 17,5,NAME
18,4.75
150 DATA NAME 19,4.75
160 '
170 ' Initialize
180 TH=187 ' Set TH to Total of Hours in Data lines
190 Z=19 ' Set Z equal to number of Names in Data lines
200 Y=20 ' Set Y equal to number of Turns desired for each
name
210 DIM NA$(Z) ' Names
220 DIM HR(Z) ' Hours for each name
230 DIM DE(Y) ' Day turn Ends
240 DIM TE(Y) ' Time turn Ends
250 DIM EM(Y) ' Minute turn Ends
260 BM=744 ' Sets Beginning Month which is then incre-
mented for each succeeding month (744 sets BM to May) -
determined by calculating hours from month in which 1st turn
starts
270 DS=1 ' Set DS to Day of month 1st turn is to Start
280 TS=14.5 ' Set TS to Time 1st turn is to Start (24 hour
decimal time)
290 HI=TH MOD 24 ' Hours remaining after dividing Total
Hours by 24
300 '
310 CLS
320 '
330 ' Main routine
340 '
350 FOR N=1 TO Z
360 READ NA$(N),HR(N)
370 '
380 ' Print heading
390 LPRINT TAB(20)"SPRING HOLLOW CREEK IRRIGATION
WATER SCHEDULE"
400 LPRINT:LPRINT TAB(38)"FOR 1989":LPRINT
410 PRINT TAB((80-LEN(NA$(N)))/2);NA$(N)
420 LPRINT TAB((84-LEN(NA$(N)))/2);NA$(N)
430 PRINT TAB(34);"(";HR(N);"HOURS )"
440 LPRINT TAB(36);"(";HR(N);"HOURS )":LPRINT
450 PRINT TAB(19)"START";TAB(59)"END"
```

```
460 LPRINT TAB(22)"START";TAB(59)"END"
470 LPRINT TAB(15)STRING$(20,95);TAB(50)
STRING$(20,95):LPRINT
480 '
490 ' Calculate Month, Day & Time
500 HY=(DS-1)*24+TS ' Convert Day turn Starts to Hours
510 FOR X=1 TO Y
520 '
530 ' Determine respective Starting Months for turns
540 IF HY=<BM THEN MS$="MAY"
550 IF HY>BM AND HY=<1464 THEN MS$="JUN"
560 IF HY>1464 AND HY=<2208 THEN MS$="JUL"
570 IF HY>2208 AND HY=<2952 THEN MS$="AUG"
580 IF HY>2952 AND HY=<3672 THEN MS$="SEP"
590 IF HY>3672 THEN MS$="OCT"
600 '
610 ' Determine respective Ending Months for turns
620 IF HY+HR(N)=<BM THEN ME$="MAY"
630 IF HY+HR(N)=>BM AND HY+HR(N)=<1464 THEN
ME$="JUN"
640 IF HY+HR(N)>1464 AND HY+HR(N)=<2208 THEN
ME$="JUL"
650 IF HY+HR(N)>2208 AND HY+HR(N)=<2952 THEN
ME$="AUG"
660 IF HY+HR(N)>2952 AND HY+HR(N)=<3672 THEN
ME$="SEP"
670 IF HY+HR(N)>3672 THEN ME$="OCT"
680 '
690 ' Determine respective Day turns End
700 TE(X)=TS+HR(N)
710 IF TE(X)=<24 THEN DE(X)=DS
720 IF TE(X)>24 AND TE(X)=<48 THEN DE(X)=DS+1
730 IF TE(X)>48 AND TE(X)=<72 THEN DE(X)=DS+2
740 IF TE(X)>72 AND TE(X)=<96 THEN DE(X)=DS+3
750 IF TE(X)>96 THEN DE(X)=DS+4
760 IF MS$="MAY" AND DE(X)>31 THEN DE(X)=DE(X)-31
770 IF MS$="JUN" AND DE(X)>30 THEN DE(X)=DE(X)-30
780 IF MS$="JUL" AND DE(X)>31 THEN DE(X)=DE(X)-31
790 IF MS$="AUG" AND DE(X)>31 THEN DE(X)=DE(X)-31
800 IF MS$="SEP" AND DE(X)>30 THEN DE(X)=DE(X)-30
810 '
820 ' Determine respective Time turns End
830 IF TE(X)>24 AND TE(X)=<48 THEN TE(X)=TE(X)-24
840 IF TE(X)>48 AND TE(X)=<72 THEN TE(X)=TE(X)-48
850 IF TE(X)>72 AND TE(X)=<96 THEN TE(X)=TE(X)-72
860 IF TE(X)>96 THEN TE(X)=TE(X)-96
870 '
880 ' Strip hour from Time & convert decimal part to 60 minute
equivalent
890 SM$=STR$(TS) ' Minutes part of the hour that turn Starts
900 PP=INSTR(SM$,".")
910 IF PP=0 THEN SM$=" 00" ELSE SM$=
MID$(SM$,PP+1,2)
920 SM=VAL(SM$)*.6
930 SM$=STR$(SM)
940 IF LEN(SM$)=2 THEN SM$=SM$+"0"
950 EM$=STR$(TE(X)) ' Minutes part of the hour that turn
Ends
960 PE=INSTR(EM$,".")
970 IF PE=0 THEN EM$=" 00" ELSE EM$=
MID$(EM$,PE+1,2)
980 EM(X)=VAL(EM$)*.6
990 EM$=STR$(EM(X))
1000 IF LEN(EM$)=2 THEN EM$=EM$+"0"
1010 '
1020 ' Print Month, Date & Time
1030 PRINT TAB(10)MS$;:PRINT TAB(15)USING"##";DS;
1040 LPRINT TAB(15)MS$;:LPRINT TAB(20)USING"##";DS;
1050 PRINT TAB(25)USING"##";FIX(TS);:PRINT" :";SM$;
1060 LPRINT TAB(28)USING"##";FIX(TS);:LPRINT" :";SM$;
1070 PRINT TAB(50)ME$;:PRINT TAB(55)USING"##";DE(X);
1080 LPRINT TAB(50)ME$;:
LPRINT TAB(55)USING"##";DE(X);
1090 PRINT TAB(65)USING"##";FIX(TE(X));:PRINT" :";EM$
1100 LPRINT TAB(63)USING"##";FIX(TE(X));:LPRINT" :";EM$
1110 LPRINT
1120 '
1130 ' Determine starting Day & Time for next turn
1140 IF TS+19>24 THEN DS=DS+8 ELSE DS=DS+7
1150 IF MS$="MAY" AND DS>31 THEN DS=DS-31
1160 IF MS$="JUN" AND DS>30 THEN DS=DS-30
1170 IF MS$="JUL" AND DS>31 THEN DS=DS-31
1180 IF MS$="AUG" AND DS>31 THEN DS=DS-31
1190 IF MS$="SEP" AND DS>30 THEN DS=DS-30
1200 IF MS$="OCT" AND DS>31 THEN DS=DS-31
1210 TS=TS+HI
1220 IF TS>24 AND TS=<48 THEN TS=TS-24
1230 IF TS>48 AND TS=<72 THEN TS=TS-48
1240 IF TS>72 AND TS=<96 THEN TS=TS-72
1250 HY=HY+TH
1260 NEXT X
1270 '
1280 ' Set Day and Time turn starts for next Name
1290 DS=DE(1):TS=TE(1)
1300 LPRINT CHR$(12) ' Send Page Eject to Printer
1310 CLS
1320 NEXT N
1330 END
```

*The following changes are needed for Model I & III.*

```
290 HL=INT(TH/24)*24:HI=TH-HL

410 PRINT TAB((64-LEN(NA$(N)))/2);NA$(N)

430 PRINT TAB(26);"(";HR(N);"HOURS )"

450 PRINT TAB(11)"START";TAB(51)"END"

1030 PRINT TAB(2)MS$;:PRINT TAB(7)USING"##";DS

1050 PRINT TAB(17)USING"##";FIX(TS);:PRINT" :";SM$;

1070 PRINT TAB(42)ME$;:PRINT TAB(47)USING"##";DE(X);

1090 PRINT TAB(56)USING"##";FIX(TE(X));:PRINT" :";EM$
```

*Elton L. Wood can be reached at:*
*2536 W. Old Hwy Rd.*
*Morgan, UT. 84050*

CP/M - The Alternate DOS for Model 4

# More on The CP/M Directory, and a little on Super Utility

by Roy T. Beck

*In the last (March 1989) issue, I discoursed at length on the file structure of CP/M, and I have some more information to add along those lines. Since this is a continuation, I recommend reading the previous article before reading this one. Also note the Super Utility sector printout which was inside the back cover of the March issue.*

I promised some more information on directory bytes 9, 10, and 11. These are the three bytes holding the extension of a filespec. Digital Research identifies these as t1', t2' and t3', respectively. These bytes have some additional attributes. Since all eleven bytes in the filespec of the directory entry are 7 bit ASCII bytes, there is inherently the possibility of doing something with bit 7 of these bytes. Digital Research took advantage of this to put some "hidden" information in there.

Bit 7 of the first byte (t1') of the extension is assigned to mean Read Only (R/O) Status. This can be set or unset by the STAT command. If set, the file is read only, and cannot be inadvertently written to. (A similar thing happens automatically when you swap disks in a drive and forget to log the new one in with ^C, but that does not involve the t1' byte).

The middle byte t2' of the extension can have its bit 7 turned on by the STAT command to make the entry invisible in the DIRectory display. This avoids cluttering up the screen with files you do not wish to see, but do wish to keep available. Since TRSDOS SYSTEM files are usually also INVisible, you can think of this CP/M flag as the INVisibility flag. Actually, Digital Research calls this the SYSTEM flag, but you and I can think of it as whatever we please. I believe DR's reasoning was that a user may have some system-type files (FORMAT, for instance) which are usually desired to be present, but which tend to clutter the directory display, and they opted to term this property SYSTEM. Being an old TRS hand, I tend to think of this property as being INVisible rather than SYStem, but who cares so long as we know how it works and how to use it.

The third extension byte t3' can be used as part of an archiving system. The definition by DR is that this bit 7 should be set to a one whenever the Backup function is performed. The BDOS will clear the bit to zero whenever the file is altered in such a way that the data map is altered. This scheme would therefore limit the backup effort to files which have been altered since the last Backup. I don't know if this scheme is implemented in the real world, but at least the BDOS will perform its portion of the act by clearing bit 7 of t3' if a file is altered and bit 7 of t3' was previously set.

DR also allows use of bit 7 of the first 4 bytes of the filespec for som tagging features unique to MM's CP/M. When I become more adept in use of the hard disk, I will discourse on these and other aspects of life with a hard disk under CP/M.

The following is not really part of the CP/M directory structure, but I will include it here anyway.

If you consult the CP/M manuals and look for the AUTO command, you will find it isn't there. But CP/M insiders have long known that the equivalent of the TRS AUTO command has been there all along, hidden away from the light of day. In the days before Monte, the AUTO command was only accessible by Zappers. The command had to be ZAPped into the disk, and was never mentioned in the official CP/M documentation.

But now, praises be to Monte, he has made the AUTO command accessible to us mere mortals! But did you really recognize it when Monte showed it to you?

When you return from the CONFIG operations, you are given the opportunity of executing one command at BOOT time. That's the AUTO command. When you receive your system disk from Monte, he has the MDIR command set up in the AUTO function, but you can have any command you want in that space. You can call one filespec, or issue one command. But don't forget, that one command can be a SUBMIT command, which gives you an entire JCL script.

This just goes to show there really is a lot of capability in CP/M if you know where to look for it and how to use it. I found the location of the command by using Super U, as usual. By installing an unique command as the AUTO command, and then using SU's Find String command, I immediately located the command at byte 8h of sector 3.

When I examined the sector, I saw 16 bytes of space evidently set aside for the command. But what immediately followed was a Copyright notice. I deliberately overwrote the copyright notice with a long string of X's, and rebooted. Up came the long string of X's, followed by **?**, which is CP/M's way of saying it could not find the file.

As an old hacker, I have on occasion used the space occupied by a copyright as patch space when modifying a program, but this is the first time in my experience I have ever seen an author (Monte) deliberately plan for overwriting of his own copyright notice! I proved this by going to CONFIG and filling the auto command space with as much as it would hold. Sure enough, it would take 127 bytes, overwriting the copyright notice in the process.

*Why 127 bytes?*

I believe the answer is that this is the length of the command buffer in CP/M. If you use the CONFIG command to install an AUTO command, all is automatic. If you ZAP an AUTO command into sector 3, be sure and terminate the command with 00h, else the command parser won't know when to quit!

I have now revealed all my knowledge of CP/M directories, so I will quit while I am ahead. If anyone has other info to contribute in this area, I am all ears and will write it up in future columns.

BUT! How many of you gave thought to how Super Utility, a TRS-type Zap program, could show the CP/M directory sectors? Did Kim Watt hide some features away from us? Maybe so. Read on!

In the course of preparing this article, I needed to refresh my own memory on details of the directory structure, and looked for an easy means of accessing the sectors of the CP/M disk, as we do with Super Utility (SU) on TRS disks.

There are CP/M utilities for this purpose which work in a fashion similar to SU, but I don't have any of them.

Casting about for an alternative method, and knowing SU can copy entire tracks into memory, I immediately booted up SU and began exploring the MM CP/M disk.

I tried ZAP, and SU told me it could not find sector 0.

Next, I tried one of SU's clever features. If you don't know the configuration of a disk, SU4+ (but not V3.2) contains a command which causes it to attempt to identify the format of an unknown disk. To use this, you reply to SU's "Drive, Sector, Track?" command with **!0**, (assuming your unknown disk is in drive 0). The "**!**" causes SU to analyze the disk and select a TRS format for its Configuration Table which matches the unknown disk.

I knew, of course, that SU doesn't know a CP/M disk from a pancake, but I thought it would be interesting to see what happened. Therefore I tried the ZAP command with **!0** on my SS DD CP/M system disk, and what do you know, the sector 1 image came up on the screen!

I then went to the Config Table to see what SU thought it was looking at. The table entry was now **T3D'**, which translates to Model III TRSDOS V1.3! WOW!

Now the ZAP function worked correctly, and showed the original MM System CP/M disk is structured with 40 tracks of 18 DD sectors each, numbered 1 to 18.

Would anything else work? I tried the directory read command, but got only rubbish. That's not surprising, as the directory structure is very different.

The ZAP trick really only worked because Monte opted for 256 byte, DD sectors on his SS system disk. Sectors of any other size would not have been correctly read by SU, but praises be to Monte for choosing 256 byte sectors for his system disk! (I suspect he may have been forced into this by the BOOT ROM coding in the machine, but let's give him the benefit of the doubt). In any event, we now have a powerful tool with which to read (and write) CP/M sectors on single sided system disks. But be aware this won't work on our data disks which have larger sectors.

But, I have just discovered another quirk of Super Utility. If you try to install **T3D"** in the config table, SU will give you **T3D'** instead, because it knows a double sided TRSDOS V1.3 is illegal. BUT...... If the table entry shows **T4D"** (for example), and if you give it the command **T3D**, then it will change the T4D to T3D, but leave the "!

Now we can read the backside of a DSDD disk whose sectors are numbered from 1 instead of 0. This is the quirk I alluded to above.

Using the (illegal) **T3D"** configuration, I now attempted to read a DS DD system disk. The Zap command immediately showed me sectors 1 through 18, as expected, but when I hit the arrow key for the next sector, SU told me it could not find sector 19.

I then told it to "skip" sector 19. Just to see what would happen, I then attempted to step one more sector, and up popped sector 20! Continuing to step, I was able to view sectors 20 though 37, followed by sector 1 of the next track. What the heck happened to sector 19?

I then went on to other tracks, and the same pattern showed. It is no fluke, there is no sector 19! So Monte's sequence is sectors 1 to 18 on the front, followed by sector 20 to 37 on the back for a total of 18 sectors of 256 bytes each.

Why? I dunno.

Super Utility also contains a Read Track Command. This command will read an entire track from a disk in one swoop, laying it into memory in a continuous swath, and giving you access to it. For reasons which are too complex to explain in this limited space, you cannot simply write that entire block of memory back to a disk track and expect it to work. Maybe I will write an essay on that another time.

Anyhow, you can explore the track image to your heart's content, including all the funny little things hidden away between the actual data sectors. The screen print function of SU will allow you to print out the memory contents in 256 byte blocks, which is great for analyzing how tracks are configured.

Can we copy an entire second side of a track into memory in the same fashion? I think so, but must confess I have never done it. There probably is a command somewhere in SU to tell it to take the back side of a track, but I have never looked for it. Anyone know the answer? If I solve this one, I will tell you about it.

Meanwhile, that's all for this time, Have fun!

Roy

BOOK REVIEW:

David R. Cecil

# Debugging BASIC Programs.

TAB 1984 171 pp.

Reviewed by Robert M. Doerr

Mr. Cecil's book presents nine chapters: 1. Debugging fundamentals. 2. Prevalent errors. 3. Errors in logic. 4. Arithmetic errors. (Going from single precision to double without employing VAL() 5. String errors. 6. I/O errors. 7. Disk BASIC errors. (Using disk BASIC commands while in Level II) 8. When all else fails. (Use TRON) 9. Error messages.

The de-bugging principles presented still apply today, however, published in 1984, the book should have been up-dated to cover Mod 4 BASIC (1983) and BASICA (1982). He wisely urges programmers to work in an organized manner, to document changes, to make changes one at a time, etc. It is also explained that DEBUG can be used for BASIC programs. (That prompted me to consider using Roxton Baker's STOPPER for editing Basic programs. This program is, or was, available from The Alternate Source, although it may require NEWDOS and may not run in the Mod 4 mode.) ON ERR as a de-bugging tool is also shown.

Mr. Cecil writes to watch closely for the common case of a needed item bypassed in an IF construct. The specifics that he presents include many that are inapplicable to Mod 4 BASIC. For example, he writes categorically that all FOR - NEXT loops in Microsoft BASIC execute at least once. The beauty of Mod 4 BASIC is that no execution occurs if the limit is satisfied before the loop is entered. Much of his text on reserved words does not apply to Mod 4 BASIC. Also, not being updated, this book fails to cover the difference between Mod III and Mod 4 BASIC by which the double NEXT is not allowed in Mod 4 mode:

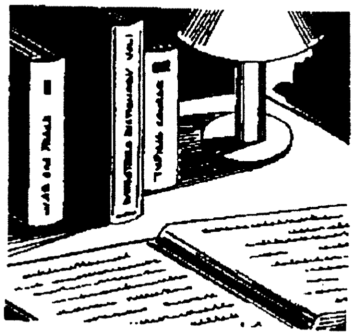
MOD III:
```
10 FOR I = 1 TO 40
20 IF U(I) = V THEN GOSUB 55: NEXT I: GOTO 50
30 NEXT I
50 V = V + 9.5: REM Program continues
```

MOD 4:
```
10 FOR I = 1 TO 40
20 IF U(I) = V THEN GOSUB 55: I = I + 1: GOTO 50
30 NEXT I
50 V = V + 9.5
```

Also not covered is the problem of overloading the stack by caused by jumping from loops, or subroutines that GOTO instead of RETURNing.

WRONG:
```
10 FOR I = 0 TO 33
20 IF U(I) = V THEN 50
30 NEXT I
50 V = V + 3.14
```

CORRECT:
```
10 FOR I = 0 TO 33
20 IF U(I) = V THEN
IH = I: I = 33
30 NEXT I
50 I = IH: V = V + 3.14
```



An interesting display presented is the list of reserved words for Mod III BASIC:

```
10 FOR I = 5712 TO 6175: A = PEEK(I): IF A>127 THEN
 A = A - 128
20 B$ = CHR$(A): IF PEEK(I + 1)<127 THEN PRINT B$;
ELSE PRINT B$ " ";
30 NEXT
```

The reader is reminded that most built-in numeric functions return only single-precision values, and that, in the world of rounding errors, IF A = B is a poor test. He presents a chi-square test of randomness. Readers are warned of rounding errors and a round-up user function is given, acceptable to bankers, but not consistent with good practice, as specified in the American Society for Testing Materials (ASTM) rounding rules.

Also shown, in Basic, is the align-tab function, but, unfortunately, it is done with an abnormal exit from a loop shown in a manner that risks stack overflow. Also covered is sorting by using VARPTR and altering string pointers instead of moving the strings in memory.

The 'If all else fails' chapter includes checking whether high-memory machine language programs overlap, if a MERGE is done incorrectly, if all POKES are OK, and whether the program is data sensitive.

To enable saving by a simple GOTO, I have long started each BASIC program:

```
1 GOSUB 9000: GOTO 100
2 SAVE "PROGNAME/BAS:2": STOP
3 SAVE "PROGNAME/ASC:2", A: STOP
99 REM Main *****
100 Program starts here
```

Cecil recommends just such a procedure to avoid lost versions. If a BASIC programmer finds this book in a library, it may be worthwhile to check it out.

---

*ROBERT M. DOERR can be reached at*
*39 McFarland Drive*
*Rolla, MO 65401*

# ATTENTION TRSDOS 1.3 USERS!

**GRL SOFTWARE PROUDLY ANNOUNCES "SYSTEM 1.5", THE MOST COMPREHENSIVE 1.3 UPGRADE EVER OFFERED!**

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

While maintaining 100% compatibility to TrsDos 1.3, this DOS upgrade advances TrsDos 1.3 to 1989! SYSTEM 1.5 supports 16k-32k bank data storage and 4 MGHZ clock speed (4/4P/4D). DOUBLE SIDED DRIVES ARE NOW 100% UTILIZED! (all models).

| | | | |
|---|---|---|---|
| CONFIG=Y/N | CREATES CONFIG BOOT UP FILE | DATE=Y/N | DATE BOOT UP PROMPT ON or OFF |
| TIME=Y,N | TIME BOOT UP PROMPT ON or OFF | CURSOR='XX' | DEFINE BOOT UP CURSOR CHAR |
| BLINK=Y,N | SET CURSOR BOOT UP DEFAULT | CAPS=Y/N | SET KEY CAPS BOOT UP DEFAULT |
| LINES='XX' PRLEN='XX' | SET *PR LINES BOOT UP DEFAULT | WP=d,Y/N (WP) | WRITE PROTECT ANY or ALL DRIVES |
| ALIVE=Y/N | GRAPHIC MONITOR ON or OFF | TRACE=Y/N | TURN (SP) MONITOR ON or OFF |
| TRON=Y/N | ADDS an IMPROVED BASIC "TRON" | MEMORY=Y/N | BASIC FREE MEMORY DISPLAY MONITOR |
| TYPE=B/H/Y/N | HIGH/BANK TYPE AHEAD ON or OFF | FAST | 4 MGHZ SPEED (MODEL 4's) |
| SLOW | 2 MGHZ SPEED (MODEL III's) | BASIC2 | ENTER ROM BASIC (NON-DISK) |
| CPY (parm,parm) | COPY/LIST/CAT LDOS TYPE DISKS | SYSRES=H/B, 'XX' | MOVE /SYS OVERLAY(s) to HI/BANK MEMORY |
| SYSRES=Y/N | DISABLE/ENABLE SYSRES OPTION | MACRO='XX',TEXT STRING + (??) | DEFINE ANY KEY TO MACRO |
| SPOOL=H/B,SIZE='XX' | SPOOL to HIGH or BANK MEMORY | SPOOL=D,SIZE='XX' | LINK MEM SPOOLING to DISK FILE |
| SPOOL=N | TEMPORARILY DISABLE SPOOLER | SPOOL=Y | REACTIVATE DISABLED SPOOLER |
| SPOOL=RESET | RESET (NIL) SPOOL BUFFER | SPOOL=OPEN | OPENS, REACTIVATES DISK SPOOLING |
| SPOOL=CLOSE | CLOSES SPOOL DISK FILE | FILTER *PR,ADLF=Y/N | ADD LINE FEEDS BEFORE PRINTING 0Dh |
| FILTER *PR,IGLF=Y/N | IGNORES "EXTRA" LINE FEEDS | FILTER *PR,HARD=Y/N | SEND 0Ch to PRINTER (FASTEST TOF) |
| FILTER *PR,FILTER | ADDS 256 BYTE PRINTER FILTER | FILTER *PR,ORIG='XX',CHNG='XX' | TRANSLATE PRINTER BYTE to CHNG |
| FILTER *PR,FIND='XX',CHNG='XX' | TRANSLATE PRINTER BYTE to CHNG | FILTER *PR,RESET | RESET PRINTER FILTER TABLE |
| FILTER *PR,LINES='XX' | DEFINE NUMBER LINES PER PAGE | FILTER *PR,WIDTH='XX' | DEFINE PRINTER LINE WIDTH |
| FILTER *PR,TMARG='XX' | ADDS TOP MARGIN to PRINTOUTS | FILTER *PR,BMARG='XX' | ADDS BOTTOM MARGIN to PRINTOUTS |
| FILTER *PR,PAGE=Y/N or 'XX' | NUMBER PAGES, SET PAGE NUMBER | FILTER *PR,ROUTE=Y/N | SETS PRINTER ROUTING ON or OFF |
| FILTER *PR,ROUTE=*DO | ROUTE PRINTER to VIDEO if on | FILTER *PR,ROUTE=*RO | ROUTE PRINTER to RS-232 if on |
| FILTER *PR,TOF | MOVES PAPER to TOP OF FORM | FILTER *PR,NEWPG | SET DCB LINE COUNT to 1 |
| FILTER *PR,SPEC=Y/N | ADDS LPRINT CHRS(1-7) CONTROLS | FILTER *KI,EXTKBD=Y/N | ENTER GRAPHICS FROM KEYBOARD |
| FILTER *KI,CLICK=Y/N | "CLICK" KEYBOARD SOUND on/off | FILTER *KI,TONE='XX' | SETS KEYBOARD "CLICK" TONE |
| FILTER *KI,LENGTH='XX' | SETS KEYBOARD "CLICK" LENGTH | FILTER *KI,PORT='XX' | SEND "CLICK" SOUND TO PORT XX |
| FILTER *KI,ECHO=Y/N | ECHO KEYS to the PRINTER | FILTER *KI,MACRO=Y/N | TURN MACRO KEYS ON or OFF |
| FILTER *KI,FILTER | ADDS 256 BYTE KEYBOARD FILTER | FILTER *KI,ORIG='XX',CHNG='XX' | TRANSLATE KEYBOARD BYTE to CHNG |
| FILTER *KI,FIND='XX',CHNG='XX' | TRANSLATE KEYBOARD BYTE to CHNG | FILTER *KI,RESET | RESET *KI FILTER TABLE |
| FILTER *KI,DVORAK | GOODBYE QWERTY, HELLO DVORAK! | FILTER *KI,SPEC=Y/N | ADDS SPECIAL CODES Cntl 1-7 |
| ATTRIB:d,NAME="DISKNAME" | RENAME DRIVE:d DISKETTE | ATTRIB:d,DATE="00/00/00" | REDATE DRIVE:d DISKETTE |
| ATTRIB:d,PASSWORD | CHANGE DRIVE:d MASTER PASSWORD | DEVICE | DISPLAYS CURRENT CONFIG INFO |
| SCAN=Y/N/H/IG/GR | HOST OR TERMINAL MODE | MMU | USER BANK I/O |
| HELP FILE KEYWORD | DOS COMPAT HELP | HLPGEN | CREATE /HLP FILES |

All parms above are installed using a new LIBRARY command **SYSTEM (parm,parm)**. Other new LIB options include **DBSIDE** (enables double sided drive use by treating the "other side" as new independent drive, drives 0 - 7 supported) and **SWAP** (swap drive code table #'s). Previous PATCHER/CMD (DBSIDE) customers may upgrade to SYSTEM 1.5 for only $9.95 US funds, original PATCHER disk must be returned. Dump (CONFIG) all current high and/or bank memory data/routines and other current config data, to a disk data file. If your type ahead is active, you can (optional) store text in the type buffer, which is saved. During a boot, the config file is loaded back into high/bank memory, and interrupts are recognized. After executing any active auto command, any stored type ahead data will be output FANTASTIC! Convert your QWERTY keyboard to a DVORAK! Route printer output to the screen or your RS-232. Macro any key, even F1, F2 or F3. Load *01 - *15 overlay(s) into high/bank memory for a memory only DOS! Enter data faster with the 256 byte type ahead option. Run 4 MGHZ error free as clock disk I/O routines are properly corrected! Spool printing to high/bank memory. Link spooling to disk. (Spooling updates DCB upon entering storage.) Install up to 4 different debugging monitors. Print MS-DOS text files ignoring those unwanted line feeds. Copy, Lprint, List, or CATalog DOSPLUS, LS-DOS, LDOS or TRSDOS 6.xx files & disks. Add top/bottom margins and/or page numbers to your hard copy. Rename/Redate disks. Use special printers codes eg: LPRINT CHR$(3); toggles printer output to the ROUTE device. Special keyboard codes add even more versatility. This upgrade improves date file stamping MM/DD/YY instead of just MM/YY. Adds optional verify on/off formatting, enables users to examine *01-*15, DIR, and BOOT sectors using debug, and corrects all known TrsDos 1.3 DOS errors. Upgrade includes LIB/DVR, a /CMD driver that enables LIBRARY commands, such as DIR, COPY, DEBUG, FREE, PURGE, or even small /CMD programs to be used within a running basic program, without variable or data loss!

**ORDER TODAY!**

32k 48k Model III's, 48k 64k 128k Model 4 4P 4D's. Send $39.95 US funds, plus $4.00 postage/handling to:
**GRL Software, Suite 209, 1051 KLO Road, Kelowna, British Columbia, Canada V1Y 4X6**
**Attention SYSTEM 1.5.**

## STORAGE POWER

Models I, III, IV, IVD, IVP

**Hard Disk**

5 Meg Hard Disk.......$295.00
10 Meg......$425.00 15 Meg.......$495.00
20 Meg......$545.00 30 Meg & up $Call
Hard floppy combinations available.
Hard disk drivers........................................................$49.95
supports most DOS's

**III/IV Disk Drive Installation Kits**

Complete with controller, drive stands, power supply, cables, Add drives & Dos.
2 FH Drives $149.95 4 HH Drives $159.95
Controller only...............................................................$89.95

**Double Density**

Increase Mod I storage capacity up to 80% w/ this easy to install board. Works w/ most Dos's (except TRSDOS) **$84.95**

**Mod IV Memory kits**

64K Non Gate Array...$39.95 64K Gate Array...$27.95

**Mod IV Speed up kits**

Non Gate Array 5Mhz..$39.95 Gate Array 6Mhz..$39.95

**5 1/4 Diskettes**

w/ sleeves & labels
Pkg of 10.......$4.25 Pkg of 25.......$9.95

**Green/Amber CRT Tubes..................$74.95**

**External Disk Drives**

Complete w/case, power supply and cables.
2 - 40 track DS DD..$259.95 2 - 80 track DS DD..$269.95
1 - 40/1 - 80 track....$264.95 2 - 3.5" 80 track.......$299.95

**Bare Drives**

40 track DS DD HH..$89.95 80 track DS DD HH..$99.95
3.5" 80 track....................$114.95

**Specials**

| | |
|---|---|
| 40 Track DS DD FH. refurb. | $69.95 |
| Replacement for SS Mod III & IV | |
| 80 track DS DD FH | $49.95 |
| 80 track w/case & power supply | $119.95 |
| Case & power supply | $59.95 |

65w Power supply $34.95

We can supply most of the parts (new & used) that you will need in repairing & upgrading your Mod I, III. IV's.
Call or write for availability & price.
Look for our BBS Coming soon.

**STORAGE POWER**
**10391 Oakhaven Dr. Stanton, Calif. 90680**
**(714) 952-2700** (9:30 am - 8:00 pm PST)
All C.O.D. orders are cash only. Calif. residents require 6% sales tax.
All prices subject to change.

## SOFTWARE

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

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

## TRSTimes on DISK #3

Issue #3 of TRSTimes on DISK is now available.
It features the following programs from the Janauary, March and May 1989 issues:

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

TRSTimes on DISK is reasonably priced:

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

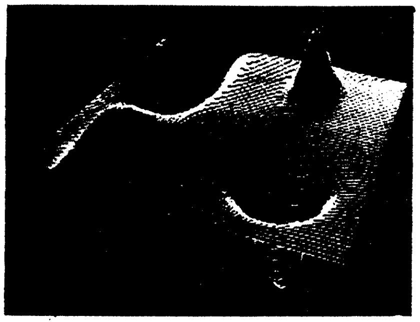
Send check or money order to:

**TRSTimes on DISK**
**20311 Sherman Way, Suite 211**
**Canoga Park, CA. 91306**
**U.S.A.**

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

## Surface Plot

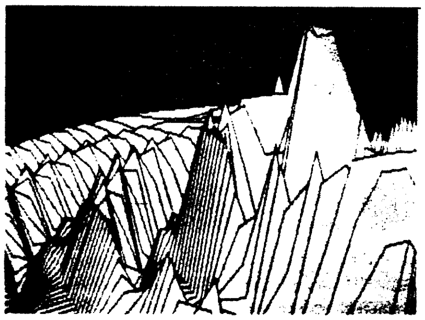
### The Grafyx Solution™ with Depth

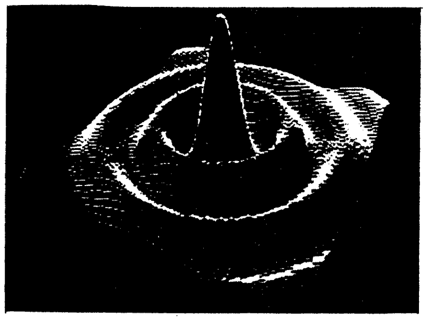


**Outstanding Graphics.** SURFACE PLOT lets you broaden your imagination and impress your friends by creating amazing three-dimensional views on any Model 4/4D/4P/III with Radio Shack's hi-res board and GBASIC 3.0 ($39.95) or Micro-Labs' Grafyx Solution. The Grafyx Solution is a plug-in, clip-on board which offers superior hardware and software.

**Flexible Graphics.** SURFACE PLOT allows you to enter an equation of the form Z=F(X,Y) where Z is the height above the surface for a given X, Y coordinate. For example, entering the equation $Z=10-X^2$ draws a hill. The final picture can be viewed from any position in space so you can see an image from underneath, above, or even inside a hill or valley on the contour plot surface. You can also specify the size and perspective of the resulting image. Learn about three-dimensional equations, viewpoints and space the fun way!

**Complete Graphics.** The program automatically removes hidden lines for





best results. The documentation contains complete instructions and sample equations so that you will have your computer hard at work without delay. The finished plot can be saved on disk or printed on any of 30 popular printers.

The SURFACE PLOT program, sample hi-res pictures, and manual is $39.95. The GRAFYX SOLUTION package includes the hi-res graphics board, 40 programs, and a 56 page manual all for $129.95. Payment is by check, Visa, Mastercard, or COD. Domestic shipping is free on pre-paid orders. Texas residents add 7% tax.

**MICRO-LABS, INC.** 214-235-0915
902 Pinecrest, Richardson, Texas 75080

---

Are you alarmed over the high cost of long distance rates while downloading Public Domain Software?

THE FILE CABINET has the answer...

## The All New 1989 TRS-80 Model 4 Disk Catalog

High Resolution/MacPaint and Orchestra-90 have been upgraded too!

MORE PROGRAMS!
EASIER TO READ!
EASIER TO PRINT!
BETTER ORGANIZED!

Send in your current catalogs for your free catalog upgrade!
Don't forget to include return postage!

TRS-80 Model 4 - $5.00
TRS-80 High Resolution/MacPaint - $4.00
Orchestra-90 Music Files - $2.00

CATALOG PRICES ARE REFUNDABLE WITH YOUR FIRST ORDER!

**The File Cabinet**

P.O. Box 322, Van Nuys, Ca. 91408
DOWNLOAD THROUGH THE MAIL

TRS-80 is a trademark of TANDY Corporation

## HINTS & TIPS

### PRODRAW PATCHES

by Arthur N. McAninch

Here are some patches for PRODRAW to allow reading directories of drives other than 0-3, as well as patches for SAVKLOAD/CMD to accomplish a similar function.

FIXPDRAW is as follows:

```
. This FIX will patch DRAW/LOD of the PRODRAW program by
. GRAFYX SOLUTIONS so that you may obtain a Directory of
. drives 0-7 instead of only 0-3
. (Note: the last patch is only cosmetic so that on Exit,
. LS-DOS Ready is displayed instead of TRSDOS Ready)
. Apply this patch by issuing the command
.                    DO FIXPDRAW
.            from TRSDOS/LS-DOS Ready
PATCH DRAW/LOD (X'94D2'=37)
PATCH DRAW/LOD (X'9760'=37)
PATCH DRAW/LOD (X'9788'=37)
PATCH DRAW/LOD (X'AEFF'=4C 53 2D)
//EXIT
```

SAVLOAD/FIX is as follows:

```
. This patch will allow you to obtain Disk Directories of
. Drives 0-7 from within SAVLOAD/CMD when utilizing
. either the (L)oad or the (S)ave command
. Apply by issuing the command:
.            PATCH SAVLOAD/CMD SAVLOAD
.              from TRSDOS/LS-DOS Ready
D02,61=37
F02,61=33
D04,23=37
F04,23=33
. Eop
```

### LITTLE KNOWN MODEL 4 FEATURE

by Lance Wolstrup

Those of us who also own an MS-DOS machine know that pressing the <F3> key will repeat the last issued DOS command. This is a very handy feature, and many times I have wished for something similar for the Model 4. Talking to Tim Sewell a few weeks ago, the subject came up. I told him that, if I had the time, I would write a small utility to perform this function.

Tim laughed and said: *"Why bother, you already got it!"*

He then proceeded to tell me that both TRSDOS 6.2. and LS-DOS 6.3. repeats the last DOS command when you press **<CTRL> <R>**.

Hey, we had it before MS-DOS.

### IT HELPS TO READ THE MANUAL

by Dennis Burkholz

One evening I was playing around, looking at LESCRIPT with a monitor program. I noticed that the program prompts and error messages were written in both English and French. This was very intriguing so I spent the rest of the night trying to reconfigure a backup copy to the French version. I didn't succeed...

The more I failed, the more determined I became. I was going to have a copy of LESCRIPT in French. To make a long story short, after failing many more times, I became so determined that I finally resorted to unfair methods: *I read the manual.*

There it was, plain as day. It is done as a parameter to the filename. Type:

**LESCRIPT %** <ENTER>

and 'voila' you will now see the prompts, headings and error messages in French.

I did encounter one problem with the French version of LESCRIPT 1.81. Wanting to see if the spelling checker also came up in French, I pressed **<CTRL> <H>**. The spelling checker came up, as usual, in English and immediately hung up the program. I must have pressed all combinations of control keys, but none worked. I had to reboot. Thought my experiences might interest your readers.

● Model I, III & 4

### The World's second smallest word-processor

by W. Barry Knight

I read with some interest that "lean and mean" wordprocessing program submitted by Ed Martin in the Jan/Feb 89 issue of TRSTimes. By including the PRINT I$; the operator can now see the text on the screen as he/she is typing, thus keeping the program almost as lean, and maybe just a tad less mean.

```
10 I$=INKEY$:IF I$="" THEN 10 ELSE PRINT I$;:LPRINT I$;:
GOTO 10
```

Ed Martin replies: *Sure, go ahead and fatten up my program with fancy bells and whistles. See if I care!!*

## CLOSE #3

I can't believe it. This issue, our ninth, marks a year and a half of publishing TRSTimes. Sure doesn't seem that long.

Our very first issue was a humble 22 pages; the next issue grew to 30 pages and we stayed right around there until issue #6, which sported 36 pages. This count carried over into the January and March 1989 issues. As you can tell, you are now reading this column on page 38. We grew another 2 pages. *And they said information for the TRS-80 was dwindling!*

As you might imagine, the flow of mail to TRSTimes is heavy. We get letters asking questions about a variety of software, hardware, patches, fixes and numerous other subjects. The three most frequently asked questions so far this year are:

1. Will we continue in 1990?
2. Will we go monthly?
3. Why don't we set up a TRSTimes BBS?

The answer to question number 1 is not finalized as of this time. TRSTimes, for reasons documented in previous issues, is a year-by-year project. Now, like last year, I will take some time off, probably catch some ball games down in San Diego, relax in the sun, while reflecting on my status with my family. I will also seriously consider whether or not TRSTimes will have anything of interest to say in 1990 and if you, the readers, are at all interested. The early indicators (sounds like an election, doesn't it?) show that TRSTimes, more than likely, will do at least one more year. How about if we change the name to '90 MICRO'? *Nah, we'll keep TRSTimes.*

Should we continue in 1990, the bi-monthly format will be kept. Publishing TRSTimes on a monthly basis is not possible. I simply cannot double the time spent producing the magazine. Also, the extra expense would mean doubling the subscription price, making it just too expensive.

A TRSTimes BBS? The answer is a definite maybe! We are investigating the possibillities, talking to a couple of people who have had past experience running TRS-80 boards. Nothing is firm yet, but hopefully we will have an announcement about this in the next issue.

Let me change the subject for a minute. As all of our readers should know, there is another TRS-80 publication available called Computer News 80. Now, while our two magazines are to a certain extent competing, we do so on very friendly terms. As a matter of fact, Stan Slater (CN80 publisher) and I communicate frequently with each other. Though I haven't discussed it with Stan, I imagine that he shares my views on the following subject:

Both CN80 and TRSTimes continue to exist because you, the readers, are kind enough to send your articles to us for publication. We do appreciate it, believe me. The problem occurs when an author sends the same article to both magazines. We may both like the material and, not knowing the other also has it, publish it simultaneously. This just is not fair. It is not fair to CN80; it is not fair to TRSTimes and, most importantly, it is not fair to the readers. Many subscribe to both magazines, and they deserve to get new and fresh information from both publication, each and every issue. So please, if you submit an article to CN80, do not send it to TRSTimes; if you send it to us, please don't send it to CN80. Enough said!

Finally, TRSTimes wishes to salute the people who helped make this issue possible by sharing their knowledge and talents with us:

to Donald Shelton for his reverse video trickery....
to Bob Rose for telling us about SYSTEM 1.5....
to Ben Mesander for helping us get to those MAC pictures....
to Dr. Allen Jacobs for the review of Magic Math PLUS...
to Gary Campbell for another installment of 'miracle' programming...
to Elton Wood for a most interesting scheduling program....
to Roy Beck for more CP/M wizardry....
to Robert Doerr for reviewing DEBUGGING BASIC PROGRAMS....
to Arthur McAninch, Dennis Burkholz & W. Barry Knight for sharing some fine tips...

**Thanks guys** - we couldn't have done it without you.

*Lance W.*

---

**The RAM software Company presents:**

**SMALL-C compiler version 3.0. on the TRS-80**

A large subset of Kernigan & Ritchie C, with a UNIX compatible I/O library. Many other library functions are inluded. This is a true compiler, not a pseudo-code generator like some others.

REQUIRES that the purchaser own Microsoft's M89 assembler & L80 linker, or compatible assembler & linker, on a 48K Model I with Newdos/80 v2.0. More than 1 disk drive is recommended.

$20.00 for executable C compiler, library object code, demo programs source code and C manual.
$20.00 additional for source code to compiler library and library building/management utilities + documentation.

Make check/money order payable to:
**Ben Mesander**
**1137 E. Brooks St. Apt. 4**
**Norman, Oklahoma 73071**

Sorry, no COD or credit cards.