TRS TIMES

Volume 2. No.2 - Mar/Apr 1989 - $4.00

Another Marathon Issue

for Mod I, III & 4

DOS commands from Basic in TRSDOS 1.3. - Shell Basic

Printer utilities - Assembly language tutorial - and much more....

LITTLE ORPHAN EIGHTY

In the old Model I days, just when you thought you had everything under control, you realized you absolutely *needed* another piece of equipment in order to do some really serious computing. First, there was the memory expansions from 4K to 48K. Next was the retirement of the cassette recorder when the disk drive was brought home. This lonely creature soon multiplied and became four.

Needless to say, when the Model III first appeared in Radio Shack stores, I went to see what it was all about. I had no sooner walked in the store than it looked at me with its big brown eyes and cried out: "*Take me home!*" Being somewhat of a 'softie', I just couldn't leave without doing the right thing!

Being completely emerged in the Mod III, I didn't realize that Model 4 was on the market until someone offered me a used one. Did I want it? "*Sure*". I didn't even ask how much. It had 128K of memory, such luxury! The 2 single sided drives were soon replaced with double sided drives. An additional 80 track double sided drive was added shortly after.

Most reasonable people would assume that this ought to be enough. *Wrong!* These machines were all too bulky to bring along on a trip, so naturally I *needed* a portable so, rather than sleeping, I could continue computing at the motel. Lo and behold, someone had a 4P for sale along with a 5 meg hard drive. Now, in all honesty, I didn't really need the hard drive; but the price was right (*hmmm*), so I bought both items.

In between the Mod I, III & 4's were excursions to Radio Shack stores for purchases of Color Computers. Yes, I have a CoCo I, CoCo 2 and a CoCo 3, along with most of the needed extras. Also, somewhere along the line I ended up with a portable Commodore 64. My older son, Alan, inherited that one rather quickly.

At this point we were using Apple IIE's at school. Being a devoted TRS-80 man, I was never tempted to own one. I just didn't like the machine. Now, however, the school was eliminating the Apples and replacing them with IBM PC's. I had been familiar enough with the intricacies (idiocies?) of the Apple to teach reasonably interesting classes; but the PC demanded a whole new learning process, so rather than spending many spare hours at school, I bought one, a bare XT clone with 640K, 2 drives and a monochrome monitor, to learn in the comfort of my home. It didn't take long to realize that survival in the MS-DOS world required a hard disk. I bought 30 megs worth. A few weeks later color became a necessity, so an EGA monitor was added.

I did pick up enough information to make the switch and teach MS-DOS, so I quietly went back to enjoying my Mod III and 4 in my spare hours. As I was not interested in Hi-rez (which was about the only thing I didn't have), my wallet stayed nice and closed for quite a while. During that time I did recoup some of my investment by selling some articles to various computer magazines. Boy, did that feel good!

Then came TRSTimes!!

In the beginning everything was done on my existing equipment, the text files created on Mod III or 4 word processors (usually LeScript), then transferred to the PC and into Ventura Publishing. Using a dot matrix printer (an Epson compatible Citizen MSP-10) produced only semi-adequate masters. To make matters worse, the Ventura Epson printer driver had a bug that, at the most inconvenient times, squashed a line. This necessitated many, many print-outs of the same page before a reasonably good page was done. Frustrating, and very time consuming.

I fought the urge to buy more equipment several times. As you know, I finally broke down right before Christmas and upgraded to a Laser printer. While I am extremely happy with it, I am also well aware that the cycle is starting all over. Now I *need* again!

The printer comes with 512K of internal memory and, believe it or not, this is not enough. I am going to have to spring for an additional 2 megabytes of RAM. Not only that, but Ventura is straining the PC's 640K of RAM. Looks like I might have to expand to 1 full megabyte there. Never ends, does it?

To update you on my adventures with the missing page numbers. They were there all the time, just too far down the page for the Laser Jet to print them. So I moved them up half an inch and, *voila*, magically they began to appear on the printed page. This issue features page numbers *without* scotch tape. I hope you enjoy them as much as I do.

TRSTimes would like to congratulate our good friend Eric Bagai from the Valley TRS-80 Hackers' Group. Eric managed to get an editorial published in the Los Angeles Times. No mean trick, as this snooty newspaper only publishes editorials from the best of writers. But then again, why not? As we all know from the pages of this magazine, Eric *is* one of the best.

TRS-80 fever!! - catch it

and now....welcome to TRSTimes 2.2.

Lance W.

TRSTimes - Volume 2. No. 2. - Mar/Apr 1989

Contents:

TRSTimes 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

Are there any typing tutor programs for the Model 4? I don't recall ever seeing one. If anyone knows of a good public domain, shareware or commercial program of this type, I'd like to find out how to get a copy. I'm most interested in a program suitable for teaching children.

Also, if anyone knows where I can find a copy of 'TROUBLESHOOTING AND REPAIRING PERSONAL COMPUTERS' by Art Margolis, please let me know. It's a TAB book which is now out of print. (The new version, 'Troubleshooting and Repairing the NEW Personal Computers' is not the one I need.)

I am looking forward to the next issue.

Don Singer
3726 Skyline Dr.
Scottsbluff, NE. 69361

I would like to ask if it is possible to remove the backup limited password out of TRSDOS 6 programs. There are 2 of them that I have, and they are: PFS: File and PFS: Report. Both are written for Model 4 TRSDOS 6. The copies I have are on their "Last Leg", so to speak, and I have some very important documents recorded in them, and I would hate to re-do them using a different data base management program.

Thank you for any help you can give me, and also thank you for your wonderful magazine. I enjoy every page of it!

Charles T. Neighbors
Longmont, CO.

Check out Roy Beck's article in the Jan/Feb 1989 issue. It gives the complete details on how to strip the copy limited protection from Model 4 TRSDOS 6 disks.

Ed.

Please sign me up for another year. I like TRSTimes, although 3/4 of every issue is useless to me since I don't have a graphics board, CP/M, Scripsit, or a Model I. I suggest that with limited space available, you would be best advised to prefer general-use articles over those requiring particular software or hardware. Naturally, you have to strike a balance, since a lot of your subscribers DO have a graphics board, CP/M, etc...!

I also do not own an assembler. I may buy one at some point, in order to access its greater speed, but I wonder if someone has a convenient way to maximize the speed of BASIC using a standardized machine language call. For example, it would be neat if a BASIC programmer could store an entire array at some specified memory location, then call a subroutine in machine language that would rapidly print that array to the screen. I have in mind a universal subroutine that could just be copied from one BASIC program to another. Or something similar...!

Ed Gracely
Sicklerville, NJ.

TRSTimes will always try to deliver program- and DOS specific articles, rather than the ones falling in the general-use category. Personally, I own neither a graphics board nor Scripsit. I do not use CP/M, at least not on a regular basis, and I have just recently re-acquired a Model I. However, since many readers are interested, TRSTimes will certainly provide as much information on these topics as possible. We do try to cover a variety of TRS-80 subjects, some which will be of interest to one group and not another. Hopefully, it will all balance out.

Your programming suggestion is quite interesting. While we don't have a specific answer, do check out Gary Campbell's info in the HINTS & TIPS section. Also, in your honor, we dusted off and revived an old idea - see the SBASIC article in this issue.

Ed.

Two weeks ago I powered up my 4P only to discover the screen filled with garbage and no reset. I've been known to be patient (once in a GREAT while), so I began the process of tracing each and every connection. Tandy saw fit to design a very delicate circuit board. I have board #8709524 REV.-. This board was the latest in production.

With the board in front of me, I powered up, (no reset), hit the period (.) and got tons of memory problems (Ram Test). The Z80A cpu chip was heating up (not REALLY hot, but much warmer), so I traced the top and bottom of the board. The saddest part is that I finally realized I could not fix this (since I was an electronics expert in security installations, I doubt ANYONE could fix it). I broke down and called Tandy National Parts' 800 number. I was told that board #8709427 REV. A. (the one prior to mine) and board #8709524 REV.-. were NO LONGER available. Needless to say, I dashed off a letter to Computer Customer Service - stating why I refuse to take the 4P to a Tandy dealer and WHY don't they have ANY boards for the 4P. I'm waiting to hear.

Since I have the Micro-Labs Hi-Res board, I'm especially looking for a 4P. I like the screen size and cassette port on the 4/4D, so hopefully, someday I will have both.

I got a pair of Teac double sided drives for Christmas. There was a big sale my best friend found. I read an article in my only copy of Northern Bytes about adding 2 external drives to the 4P (with the REV.-. board). I REALLY wanted to do that - to still be able to use the Tandons.

Radio Shack could have done a much better board, and I'm rather ticked. A single replacement of one chip could ruin a board like that. The circuitry is SO delicate that someone could accidentally pull off an entire section of the connection. Despite not having a running computer, I'm rather glad the 4P was SO obvious when it got sick. I'd rather see the problems than to install the 2 new Teac drives, only to have the cpu board die.

Thought TRSTimes readers may want (or NEED) to know about the problems with trying to get a cpu board from Radio Shack.

Carol Welcomb
Rockford, MI.

I am wondering if there is still enough readers out there that still use NEWDOS 2.0 or 2.5. For the last 8 years, I've using this DOS exclusively and hoping there are other users left out there.

Presently I'm using NEWDOS/80 to run a system called Chicago Syslink Network (312 622-4442 data) with the following configuration: 4 TRS-80 Model 4's; a megaplexer; 20 megabyte hard drive and a few modems to access the system. The system works fine letting callers access the same files. The problem is that I can't configure the DOS beyond a 40 megabyte limit. It there are any readers out there that have used NEWDOS/80 and a hard drive, I would be more than willing to share information.

Also, in response to an article by Tim in the Nov. '88 issue of TRSTimes about reviewing BBS programs, let me inform you that SYSLINK software for the TRS-80 Model 3 & 4 (Model 3 mode NEWDOS/80 only) is now in public domain. For those wanting more info. on this great software (more powerful than even TBBS in the MS-DOS world) can either call Chicago Syslink Network at 312/622-4442 (300/1200), call voice at 312/622-5969, or write directly to me at the address listed below. I'm not a programmer by nature, but will gladly support those wishing to start their own SYSLINK system and/or become part of the SYSLINK network. The only limitations in running SYSLINK is that the user must have a TRS-80 Model 3 or 4, NEWDOS/80 2.0 or 2.5, and at least 5 megabytes of disk storage (or a few double sided 80 track drives).

George Matyaszek
1718 N. Long Ave.
Chicago, IL. 60639-4321

As a relatively new TRS-80 user (1 year ago), I want to say that your mag has been very encouraging and helpful. Particularly, I have appreciated the help with Scripsit and TRSDOS 1.3. (I'd be doing this letter on Scripsit, but the computer is busy downloading).

I use my TRS-80 in connection with my work as a pastor, and I would be interested in corresponding with other people who are using TRS-80's in church works. Would you publish my address with a plea for correspondence? I would appreciate it very much. One thing I would like to hear about in TRSTimes: There has been so much talk of VIRUSES in the computer world - I've heard of viruses in IBM, Mac, C64, Amiga and even CoCo software, but I haven't heard talk of any TRS-80 viruses around. Have I just not heard of it, or since our micro is so "obscure" now, do we just not rate the attention? (now there's an advantage of running a "dinosaur"!) Also, I have heard from a local Model 2 user asking for support. What support remains for 2/12/16 users? I'll pass that info back to her.

One last question: Is TRSTimes on Disk copyrighted as a collection? I know that VTHG doesn't want their disks distributed as collections, so I ask. The reason is because I would like to ARChive them and send them up to my local BBS (8/N/1 #1). Can he post them as collections? Again, thanks for your support.

Michael E. Webb
P.O. Box 96
Bronson, FL. 32621

So far we have neither encountered, nor heard about, TRS-80 viruses. I suspect that we will remain unaffected, especially since the common target of these 'sickies', the hard drive, is still an infrequent commodity in the TRS-80 world.

The DALTRUG - Tandy users group of DALLAS, TX. has a special interest group catering to Models 2/12/16. Contact them through their BBS. (214) 234-4952 (300/1200/2400 baud-24 hrs), they may possibly know of support in your friend's area

We do ask that TRSTimes on Disk not be posted on bulletin boards, as this would violate the copyright of each of the individual authors. Sorry!

Ed.

I look forward to the assembly language tutorial, so I walzed over to Radio Shack Computer Center, where they told me that EDTASM most likely will be unattainable, but they would check. If they bomb out, can you tell me where I might be able to find it?

D.A. Crossley
Brownsville, TX

We have had several letters indicating that EDTASM is NOT readily available from Radio Shack. The answer to this unfortunate situation may be to get a hold of the NEWDOS/80 version of EDTASM. We believe this to be public domain, as Apparat modified the RS version AND

distributed it free with purchase of NEWDOS. Apparat, to the best of our knowledge, never paid royalties to RS, nor did RS ever take legal action. I have seen numerous public domain disks containing this program, so check out your PD disks, you may already have it. This version, except for a couple of minor things, is identical to RS EDTASM. The Model I version works on all Model I DOS'es. The Model III version works on all Model III DOS'es.

Ed.

I just received and reviewed my Jan/Feb 1989 TRSTimes. You did a good job. There are many small companies out there who have been using the TRS-80 I, III & 4 who should have been thinking about disk duplication and conversion long ago. All their files/records are on the old formats. This is their business records!!

The article CPY/CMD in "TRSDOS 1.3. CORNER", how about that?

I thought the hard drive business died for the TRS-80. I inquired about a HD some time back, but couldn't get a response from anyone. Rob Stewart is a new name to me.

The article 'Copy Limited Files Under TRSDOS LDOS 6.X" brings back a memory or two. One day someone came running in panic. No more backups and all my business files (on PFS files) are in jeopardy!!! HELP!!!!!! My subsequent investigation ended with SU and TRSDOS 6.3. I missed a few steps that Roy covered, but I got there. Thanks for the added info.

I noticed an ad by Hinrichs Software for statistics programs. I've never heard of them. I think I will buy the regression analysis and fourier analysis programs from them. So far, the old TRS-80 has done much better in the statistical programs than what I have seen in the IBM PCs. The stuff for IBMs cost too much and you don't know what you're getting.

In grad school, I was forced into statistical problems. These mentors like to keep doing things over and over. I tried to do the computations on their machines, well good luck. I would rather work in a hog troth. The time and inconvenience and running back and forth trying to get things right, I could go on and on..... These professors like to get you into a position where you have to use a mainframe (their mainframe). Although I work for a company that is supposed to be into computers, there wasn't any help. My Model III came to my rescue. One thing you find out right away is that these stat. programs may say they do the same thing, but many do not. The limits on what you can or cannot do can get you into trouble. I ended up with 3 stat. programs. The Tandy Advanced Statistical Analysis was excellent, fast enough, but limited. The Statistician by Quant Systems, P.O. Box 628, Charleston, SC 29402 (803) 571-2825) was fairly easy to run and less limited. STATPac by Walonick Associates, 5624 Girand Ave. South, Minneapolis, MN 55419 (619) 866-9022) actually has a version of the\mainframe SSPS program that runs on the Model III (it's not cheap). I thought the STATPac program was confusing, but you can plug through it. The mainframe version of SSPS really looks like a turkey. The schools that I was attending actually requested an SSPS printout. They couldn't believe the little Tandy was doing it, when doing it on their mainframe would have taken much longer. The mainframe set-up time was a killer. The trick with the Model III is in using virtual (disk) memory. The last stat. program I picked up which looks real good is S.P.S. by Advanced Operating Systems, (div. of Howard E. Sams) 450 St. John Road -- suite 792, Michigan City, IN 46360. I bought a copy of Patcher. And it appears to run OK. I notice there's a TRSDOS 1.5! You may know that when using the Electric Webster with one-sided drives and putting the EW files on the main Super Scripsit program disk in drive 0, you have only 50-54KB left. If you try to edit a file larger than about 50KB, S/S and EW will lock up!!! Well, you have to put the dictionary in drive 1 while the S/S and EW are in drive 0. See the problem? If you truly know what DOSPLUS is doing and you have double-sided drives, you could use DOSPLUS. But with Patcher and double-sided drives, you can put the EW dictionary on the bottom of the drive 0 disk. So far everything works OK.

A few new items -- People would probably upgrade to double-sided drives in their Model 3 & 4's if they knew how or where it could be done. The more that gets out on this the better. I have both a Model 4D and a Model III with two double- sided Tandon drives (TM100-2). People should know that the TM100-2 is the standard disk drive used in the IBM PC-XT. It's not an oddball drive or anything like that. You can get them for about $120 each. On the subject of copying disks, I have a DOSPLUS V3.4 disk that I would like to copy. It's a boot disk for one of those Micro-Merlin upgrades -- made the Model III become an IBM-PC. I'm told SU will copy it sector by sector, but.. Something strange is going on and I can't pin it down. I believe the problem is in my lack of knowledge between the capabilities of the various versions of DOSPLUS. Maybe someday someone will put to print a few words about this DOS?

It would be nice if you could wake up some TRS-80 repair shops to do some advertising.

There are rumblings in the peanut gallery, looking for a wildcard copy for TRSDOS 6.3. Anything out there?

Manley Fox
Manhattan Beach, CA.

In a future issue we will do something specific with DOS PLUS, both the Model III and 4 editions; heck, we may even play around with MULTIDOS.

As far as wildcard copy for TRSDOS 6.3, believe it or not, the TRS-80 has had this capability since the Model I, that is, if you used LDOS. More than likely, we will present a short article on this topic in the next issue.

Ed.

ITEMS OF INTEREST

ENGLISH PROGRAM for TRS-80/mod 4. $6.00 plus $1.00 shipping and handling U.S.A / $3.00 foreign to:
Basic Skills Software, 3942 Daphne Ave.
Palm Beach Gardens, Florida 33410

Casino 21 Blackjack for TRS-80s

(Note: There are so few TRS-80 products and publishers left producing software for TRS-80s that we are especially pleased to offer this at this late date: late 1988!)

Premise: You enjoy the game of Blackjack...
Conclusion: You'll love CASINO 21,
a superb implementation

The game allows all the things you expect:

- multiple players, or just one, against the computer
- random deal of cards
- hitting, betting, doubling, splitting, insurance
- betting with predetermined stakes

Offered EXCLUSIVELY by Recreational Mathemathical Software, CASINO 21 was written expressly for the TRS-80 and solely for it. It is available for TRS-80 Model I (cassette tape), Model III (tape or disk), Model 4, 4P, 4D (disk format, Model III mode).
As a special extra, at no extra charge, all disk versions will come complete on a self-booting disk with a licensed DOS that is TRSDOS 1.3-compatible. So, all you need do is make a backup and then just insert the disk. The rest is completely automatic!
There are no commands to memorize, nothing to know, exept the basics of the game of 21 itself (blackjack). A small manual is provided to make you feel more comfortable, but if you're like me, you won't need it at all!
CASINO 21 was written by New Directions Consulting in the heyday of TRS-80 computing (around 1983-4 or so), and we're thrilled to promote it now for the benefit of fellow TRS-80ers who have been left stranded by 80 Micro and the other publishers who have made the understandable migration to MSDOS.
Personally, I've had the game for several years, and it is still one of the 10 of the hundred games or so I own that I still use. CASINO 21 never becomes boring. And, it's not copy-protected so you can have peace of mind (unlike the practice of copy-protecting of most game publishers that leaves you helpless)! Heck, you can even study the listings to see how the graphical effects are achieved very simply.

~~The price is just $16.95 (15.95 + 1.00 shipping)~~
Special to TRSTimes subscribers only:
$13.95 (12.95 + 1.00 shipping)

Order now!
Please include name, full address, phone #, computer model and format preferred (tape, disk).

Michael W. Ecker
Recreational Mathematical Software
129 Carol Drive - Clarks Summit, PA. 18411

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

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 included. This is a true compiler, not a pseudo-code generator like some others.

REQUIRES that the purchaser own Microsoft's M80 assembler & L80 linker, or compatible assembler & linker, on a 48K Model I with Newdos/80 v.2.0. 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 orders payable to:
Ben Mesander
1137 E. Brooks St. Apt.4
Norman. Oklahoma 73071

Sorry, no COD or credit cards

WANTED: Model III or 4

Permantly disabled woman looking for a Model III, 4, 4P, 4D, any condition, but priced reasonably.
Also Tandon single or double sided drives.
Would give your old computer a VERY good home.
Contact: Carol L. Welcomb
11161 Edgerton N.E.
Rockford, Michigan 49341-9150

Recreational & Educational Computing

A New Column by Dr. Michael W. Ecker

Hi, fellow TRSTimes readers and TRS-80 users! This is the first of an irregular sequence of articles, programs, tutorials, and challenges to appear in this publication.

To the extent that your feedback warrants continuation, I'll keep this going. I don't get paid for this, but if a few of you subscribe to my own publication as a result, or a bunch of you write to say how much you love my stuff, then those strokes will be adequate reward for my meager bank account and fragile ego.

This Issue's Teaser:
A Mini-Contest and Program/Tutorial

Million-Dollar Word

Here's a fun challenge for you. All you need is a little number sense and a sense of word play. If you want to try to write a program to help you, you can be my guest, but I will provide one here as well, plus a tutorial.

Assign to each letter of the alphabet a monetary value based on alphabetical order: A = $1, B = $2, C = $3, ..., Y = $25, and Z = $26. (Just count out the letters.)

Let the value of a word be the product of the values of the constituent letters, counting multiplicity (frequency). For instance, the value of the word "DOLLAR" is 4 x 15 x 12 x 12 x 1 x 18 = 155,520 dollars.

First challenge: Write a program that will allow you to input any word and that will output the value of the word. (Don't peek at the one I'm providing here if you accept this challenge.)

Second challenge: Find a word whose value is precisely one million dollars - or comes as close as possible.

Help for First Challenge

To enhance enjoyment, rather than let novices have nothing to do with their computers here, I am including one program in BASIC that will let users convert words of up to 30 letters (in theory) to dollar amounts.

What follows is a program written on Dec. 2, 1988. Please be sure to lock on the capitals prior to running the program.

First, load BASIC and then type in the program.

Since we probably have several self-proclaimed novices and neophytes, we can use this program as a tutorial on some intermediate concepts in Basic programming. Please refer to the program below as we go through it line-by-line over the next page or so.

```
5 CLEAR 500 :REM Model I & III only - Model 4 can skip this line
10 CLS: DIM LT$(30), MN(30): DEFDBL A
20 PRINT "Program to convert words into dollars... per REC Newsletter, (c) 1989
30 PRINT "Dr. Mike Ecker / 129 Carol Drive / Clarks Summit, PA 18411
40 PRINT: INPUT "Using capitals, what's the string whose value you seek"; WD$
50 LET AM = 1: REM INITIALIZE PRODUCT
60 LET L = LEN(WD$): IF L > 30 THEN PRINT "Smaller word, please!": GOTO 40
70 FOR DG = 1 TO L
80 LT$(DG) = MID$(WD$,DG,1): PRINT LT$(DG);: REM CONFIRM IT
90 MN(DG) = ASC(LT$(DG))-64: 'SUBTRACT 64 TO GET CORRECT DOLLAR VALUE
100 IF MN(DG) > 26 THEN PRINT: PRINT "USE CAPITALS ONLY! TRY AGAIN!":GOTO 40
110 LET AM = AM*MN(DG)
120 NEXT DG
130 PRINT: PRINT "The amount for your word - "; WD$; " - is $"; AM
140 PRINT: INPUT "Try again (Y or N)"; MO$
150 IF MO$ = "Y" OR MO$ = "y" THEN 40
160 IF MO$ = "N" OR MO$ = "n" THEN END
170 PRINT "TYPE Y or N only, please!": GOTO 140
```

Line 10 clears the screen and also dimensions arrays designed to hold the letters of words and their dollar equivalents. DEFDBL declares that variables beginning with A are double-precision. It would take too long to explain arrays here, though you could look up my FOR...NEXT columns in PC Clones in early 1988.

Lines 20 and 30 are just print statements/ commands.

Line 40's Input command prompts the user for input, in this case of a string variable - a word - called WD$ (for WORD). Note that it is not necessary to supply a question mark within the optional text (the part asking for your string) to make a "?" appear. If you supply one in the text, two will appear.

Since we will be multiplying and also probably using the program repeatedly, line 50 initializes the variable AM to 1. (If we were working with sums, we would use 0 instead of 1.) This way, when we iterate later, we will be able to calculate the value of a word by taking the product so far and multiplying by the next letter used, as you see happen if you jump ahead to line 110. Note, by the way, that the assignment keyword LET is optional, but many feel that it should be used.

Line 60 computes the length of the word you input, checking that it's not more than the arbitrary figure of 30 characters long. The purpose of this is to make it easy to know how many times we have to pick out a letter to be converted to a value. The program, however, doesn't check to make sure you have a bonafide word, but it will check in line 100 to make sure you've used capitals, as I'll explain in a moment.

Line 70 begins the loop needed to go through the word one letter at a time. The variable DG is the name for the counter; when DG = 1, line 80 picks out the first letter with the MID$ command. (Consult a BASIC manual for more on this.)

Line 90 converts to the equivalent value (1 to 26). The trick here, rather than use DATA statements or the like, is to realize that the letters A through Z (but not lower-case a through z!) have ASCII values, accessed by ASC( ), starting with 65 and going up consecutively to 90. Thus, I subtract 64 to get the numbers to go from 1 to 26 instead.

Note that line 100 error-traps for lower-case input, since lower-case letters have higher ASCII values. If one wished, one could easily program in a subroutine to allow checking for lower case and then convert to upper case automatically. To do so would be too distracting from our purpose.

As mentioned before, line 110 keeps track of the product of the values so far, keeping a running product just as you would. Line 120 completes the loop, and the following lines report the output and allow you to repeat with a new word. You must reply Y or N when asked about doing more.

Okay, there you have it. How close can you come to a million-dollar word? Send your best entries directly to me at the address provided in this column.

Another Challenge to Readers

Here's a challenge that might prove tougher: If you consider 10 cents, there are exactly four ways you can make this amount of money: 1) 1 dime; 2) 2 nickels; 3) 1 nickel and 5 pennies; and 4) 10 pennies.

Using dimes, nickels, and pennies, there is exactly one amount of money - N cents - that can be broken down in as many ways (N) as there are cents, excluding the obvious answer of N = 1. (One cent can be given in just one way; get it?)

What is that amount?

Those who are interested in computer recreations may obtain a sample issue of REC by sending just $2 plus a self-addressed, stamped envelope (SASE) with 45 cents of US postage affixed. Full calendar-year subscriptions of eight issues cost $24; look elsewhere in this issue for the ad for additional information.

Write to me directly about this column or REC, but enclose a SASE if you expect or would like a reply:

Dr. Michael W. Ecker
TRSTimes / Recreational Computing
129 Carol Drive
Clarks Summit, PA 18411
(717) 586-2784

I have lots more, but I need to hear from you. Would you like to see more such challenges? Type-in programs in Basic? Tutorials? Recreations? Math? "Mathemagical" tricks? What??

Until next time, happy recreational computing!

Dr. Ecker is also editor and publisher of the Recreational & Educational Computing Newsletter and the president of Recreational Mathemagical Software. The owner of 10 computers - half from Radio Shack/ Tandy - and $90,000 worth of software, Mike lives in Clarks Summit, Pennsylvania, a suburb of Scranton. He welcomes your questions, insights, and general correspondence.

Note from the editor:

Mike informs me that REC is now entering its fourth year of publication. Anyone who mentions TRSTimes may get the 1989 subscription, as well as ALL the back issues, for $69.00.

• LPRINT

by Ben Mesander

NEWDOS/80 v.2 - EDTASM - STAR NX-1000

Model I

Recently, I bought a Star Micronics NX-1000 printer for my TRS-80. I had quite a bit of trouble, because it had a bug in the ROM that prevented it from talking to computers that ran slower than about 4 Megahertz clock speed.

While I was debugging this problem, I wrote a handy little printer utility called LPRINT, which now is in constant use. It allows you not only to send any sequence of characters to the printer while you are in DOS, but also to monitor the status of the printer.

This routine does not use the standard printer driver, so it will not work with printers that require special drivers or are not mapped to the standard 37E8H location. It's output cannot be changed with commands like NEWDOS's ROUTE, but on the other hand, any character can be sent without being changed directly to the printer-something many drivers won't let you do.

To assemble the program, you can use most any Z-80 assembler. If you are not using Microsoft's M80 assembler, you can use uppercase and delete the ".z80" directive from the program. Also, make sure you delete the colons at the end of all labels.

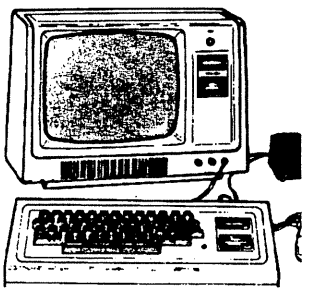
To use the program, type LPRINT followed by the hexadecimal codes you want sent to the printer, optionally seperated by spaces and/or commas for readability. For example, **LPRINT 07 0D** sends the ASCII code for a bell (07) to the printer followed by a carraige return (0D).

While I have found the program useful to see if the printer is on or off line from NEWDOS/80, it is especially great for sending control codes to the printer from inside a JCL file. For instance, when I turn on my computer, a JCL program runs that sets my printer to print zeroes with slashes (the printer defaults to no slashes).

Some printers will let you choose different text fonts or justification via control codes, and you could for example store commonly used printer settings in JCL files, and run them when you wished to choose a particular setting.

```
;LPRINT/MAC 2.0 PRINTER UTILITY
;by Ben Mesander
;1137 E. Brooks Street, Apt. 4
;Norman, Oklahoma, 73071
;
;
;usage:
;
;LPRINT xxyyzz... prints hex bytes xx yy zz on printer.
;(xx yy zz are any valid hexadecimal 1 byte numbers)
 ;
;lprint xx yy zz... you can also use spaces or
;lprint xx,yy,zz... commas for a more readable look
;lprint xx , yy , zz.. or both for that WILD look! whee!
;
;lprint with no arguments displays detailed status of
;printer port.
;
;if it locks up because printer is busy,
;press break to exit
;
;equates
;
EXIT    EQU     402DH     ;dos ready exit
PRINT   EQU     37E8H     ;address of printer port
DOSERR  EQU     4409H     ;dos error exit
BPARAM  EQU     2FH       ;bad param(s) error code
DPRINT  EQU     4467H     ;dos print routine (video)
;
;
.Z80                      ;delete this if not using M80
                          ;assembler from Microsoft
;
;
ENTER:  LD      DE,BUFFER;place to store hex bytes
        LD      A,(HL)    ;hl = command line
                          ;argument pointer
        CP      0DH       ;check for no arguments
        JP      Z,STAT    ;if so, display status
START:  LD      A,(HL)    ;else parse command line
        CP      0DH       ;check for end of line
        JR      Z,PRINIT  ;if so, print out stuff
        CP      ' '       ;sip whitespace
        JR      Z,NEXT
        CP      ','       ;chk for comma delimeter
        JR      NZ,GETBYT
LOOP2:  INC     HL        ;eliminate funny things
        LD      A,(HL)    ;like 2 commas in a row
        CP      ' '
        JR      Z,LOOP2
        JR      GETBYT
NEXT:   INC     HL        ;try again on next char
        JP      START
GETBYT: CALL    LEGAL     ;convert 1 nybble to hex
```

```
            ADD     A,A     ;store in upper 4 bits
            ADD     A,A
            ADD     A,A
            ADD     A,A
            LD      C,A
            INC     HL
            LD      A,(HL)  ;get other nybble
            CALL    LEGAL   ;and store in lower
            OR      C       ;order
            LD      (DE),A
            INC     DE
            JP      NEXT
;
;convert 1 ascii hex digit to binary
;& check on its legality
;
LEGAL:      SUB     '0'
            jR      C,ERR
            CP      'F'-'0'+1
            JR      NC,ERR
            CP      10
            RET     C
            SUB     'A'-'9'-1
            CP      10
            RET     NC
ERR:        POP     AF
            POP     AF
            LD      A,BPARAM
            JP      DOSERR
;
;print out the buffer
;
PRINIT:     DEC     DE
            LD      HL,BUFFER
LOOP:       LD      A,(PRINT) ;check print status
            CP      00111111B
            JR      Z,NEXTC
            LD      A,(3840H) ;check for brk key
            CP      04H       ;and exit if pressed
            JP      Z ,EXIT
            JR      LOOP
NEXTC:      LD      A,(HL)
            LD      (PRINT),A
            RST     18H       ;compare de:hl
            JP      Z,EXIT
            INC     HL
            JR      LOOP
;
;stat is the routine that prints
;out the status of the printer port.
;
STAT:       LD      HL,MESS0
            CALL    PRINTS
            LD      A,(PRINT)         ;get status info
            LD      HL,MESS1A
            RLCA
            JR      C,OVER1
            LD      HL,MESS1B
OVER1:      CALL    PRINTS
            LD      HL,MESS2A
            RLCA
            JR      C,OVER2
            LD      HL,MESS2B
OVER2:      CALL    PRINTS
            LD      HL,MESS3A
            RLCA
            JR      C,OVER3
            LD      HL,MESS3B
OVER3:      CALL    PRINTS
            LD      HL,MESS4A
            RLCA
            JR      C,OVER4
            LD      HL,MESS4B
OVER4:      CALL    PRINTS
            JP      EXIT
;
;
MESS0:      DEFM    'LPRINT/MAC 2.0'
            DEFB    0DH
;
MESS1B:     DEFM    'Not '
MESS1A:     DEFM    'Busy'
            DEFB    0DH
;
MESS2B:     DEFM    'Not '
MESS2A:     DEFM    'Out of Paper'
            DEFB    0DH
;
MESS3A:     DEFM    'Device Not '
MESS3B:     DEFM    'Selected'
            DEFB    0DH
;
MESS4A:     DEFM    'No '
MESS4B:     DEFM    'Fault'
            DEFB    0DH
;
; print a display on screen
;
PRINTS:     PUSH    AF
            CALL    DPRINT
            POP     AF
            RET
;
BUFFER      EQU     $
;
            END     ENTER
```



A small essay on pullup resistors

Hardware - by Roy T. Beck

Recently I decided to replace the single sided Tandon TM50-1 drives on my Radio Shack Model 4P computer with double sided Fujitsu M2551A08 drives. "Simple "I sez to myself, just pull out the old drives and put the new ones in the same places". Well, yes and no!!

The first task was to dismantle the Mod 4P, which is pretty compactly packaged. Sardines have about as much freedom of movement in their shipping containers as do the internals of a 4P. After a while, I had the old drives out, and discovered the new drives had a different bezel, which required them to be mounted about 3/8" deeper in the computer. Further, the new drives are slightly longer, overall, than the old ones. This required me to locate and drill some new mounting holes for the new drives. Fortunately the Fujitsus and the Tandons both used the same 6-32 mounting screws; I did not have to chase after metric screws, which sometimes is necessary in this kind of surgery.

Next problem was cable length, since the power connectors were in a different location on the Fujitsus. By judicious tugging and wiggling, I managed to find enough slack to connect the old cables to the new drives. Fortunately, the ribbon cable connectors were in the same relative locations on both drives.

I then set the drive select jumpers for "0" and "1". Time for the smoke test. I attempted to boot, and got a diagnostic out of the ROM, saying something to the effect that the drives were inaccessible, or unselectable, or wouldn't respond; I don't remember exactly. The red LEDs on the drives were not lighting up, I noticed. I then unplugged the "1" drive and tried again. Success! DOS READY came up. But why not with both drives? I then swapped the drives, and now the other one would boot as "0". But plug both in and the diagnostic told me a sad tale.

I thought about it overnight and talked to a friend at work. He suggested removing the pullup resistor packs on the drives. I then perused the schematics of the 4P, and noted that 150 ohm pullup resistors were installed internally in the computer on all the drive control lines EXCEPT the drive select lines.

Dug out the schematics for the Tandon drives, and lo and behold, they have an internal pullup resistor for the drive select line, separate and apart from the pullup resistor pack socket, which is empty on my Tandons. What about the Fujitsus? Resistor packs installed on both. No schematics available. GRR....

I then dismantled the board on one of the new drives, and yessiree, there was the tiniest 150 ohm resistor I have ever seen, tucked away under the board, and yes, it was on the drive select trace. So, pull the resistor packs on both Fujitsu drives, carefully reassemble every piece of sheet metal as it was, and test as I go. Yes, the thing will boot with one Fujitsu connected. Yes, it will now boot with both Fujitsus connected! Hooray!

Finished reassembling, got out the Memory Minder test disk, and ran a test on the new drives. All works OK. Speed is 300 RPM on one, 301 on the other. Head alignment is good on both. Drink a Pepsi Cola in toast to my success.

Comments and after thoughts.

The pullup resistors are necessary because the industry standard for floppy drives is to use negative logic for floppy drive control. That means a control line standing at +5V is inactive, and the same line pulled down to 0V (ground) is active.

To insure the +5V for inactive line, each of the several control lines must have its own 150 ohm resistor connected to +5V to provide the inactive high condition. The computer then uses an open collector chip, typically a 7416, to pull the line down when it is selected. What happens is that a transistor in the 7416 chip turns on, thus connecting the control line to ground. We now have a 150 ohm resistor connected between +5V and the grounded control line, 33 MA flows to ground, and the control line is essentially at 0 Volts relative to the ground bus because the 33 MA flows through the transistor with almost no voltage drop, which is the active low condition.

Apparently the original industry practice was to install the pullup resistor pack on only one floppy drive in a system, which served to pullup all the control lines. But to simplify drive installations by non-electronic types, the practice has been changed to install most of the pullups in the computer, thus avoiding the need for removable packs on the drives. This is fine as far as it goes, except the drive select resistor is permanently installed on each drive, and not in the computer. This is probably so that the computer can sense whether or not a drive is actually installed on any given select line.

My problem was due to too much of a good thing. I had a set of pullup resistors in the 4P plus another set on each drive, making a total of 3 resistors in parallel to all the lines (except the drive select lines). This meant 3x33 MA or 100 MA was being fed into each of the other control lines, and this was apparently too much for the 7416 gates to handle. Fortunately I did not burn them out, but they could not swallow so much current and still hold the line voltage near enough to 0 Volts. Removing the pullup packs from

the drives reduced the current to 33 MA and all is now well.

Those of you familiar with circuit theory may well say that the resistors are also line terminating resistors, and really SHOULD be at the nethermost drive to suppress ringing on the line. This would certainly be true if we were dealing with data on the control lines, but we are not. The control voltages persist for long (relatively) periods of time, and a little ringing at the beginning of the enable or disable condition is apparently acceptable, at least at low CPU speeds. (The 4P runs at 4MC) .

By placing the pullups at the computer end of the lines, the designer can simply say to the drive installer "remove the pullup packs from all drives" and know that the system will work. My problem was in not knowing of the apparent change in design philosophy which has occurred since the Model 1 was the "state of the art".

I have done a little research on the 1, 3 and 4's, and will pass along what I know.

The Model 1 expansion interface came in two versions, and both versions had no pullup resistors. Consequently on the Model 1, it was necessary to install a pullup resistor pack on one of the drives. I will discuss the Model 3 later on.

There are four versions of the Model 4. These are the cassette version, the early single sided disk version with PAL chips, the later single sided disk version with gate array construction, and finally the 4D version with double sided drives. I believe the latter was only built in a gate array version. There are only two versions of the 4P, both with two single sided drives These were the PAL version and the gate array version.

In reviewing the early Model 4 manual, I noted a couple of peculiarities. The disk version of the Model 4 was built with two internal disk drives and a ribbon connector for two external drives. One peculiarity of the Mod 4 is that there is an internal pullup resistor assembly for the internal drives, but NO resistor pack on the lines to the external drives. The other peculiarity is that the external connector only uses drive select lines 10 and 12 in the external cable, corresponding to drives 0 and 1, which is the same setup as for the internal drives! However, the drive select latch correctly responds to software commands for drives 0, 1, 2, & 3. Consequently, the user must be sure the dip switches on the external drives are set for drives 0 and 1, not 2 and 3 as one might casually expect.

I believe this curious arrangement was to permit R/S to continue to sell the same drives sold for the Model 1 for use as external drives on both the Models 3 & 4. All this relates to the original R/S practice of using a ribbon cable with "pulled teeth" in the female connectors. I believe the Model 3 drive circuitry was identical to the Model 4, but I lack the technical manual to prove it. Anyone with a Mod 3 Tech manual is invited to review it and send in a summary of what is found . Since the Model 4P's were not originally designed to have external drives, only the internal portion of the Model 4 circuitry was included, and this has the pullup resistor pack on the motherboard, with no pullup packs on the drives.

Many of us have added external drives to both versions of the 4P Due to the way R/S laid out the circuitry for the two internal drives, the usual "field modifications" to provide for external drives means that the added external drives share the existing pullup resistor pack with the internal drives. Consequently, all drives on Model 4P's should have their resistor packs removed.

As an aside, in the Nov. 88 issue of CN-80, there is an article similar to this one which implies that 4P's require a pullup pack on ONE drive. Since this ignores the presence of the built-in resistor pack, I believe the advice is wrong. While it may not hinder operation to have one extra resistor pack in the system, it definitely increases the current and therefore the heat dissipation in the 7416 chip, and increased heat definitely shortens the life of components.

In conclusion, what started out a as a casual "mechanical" conversion forced me to do a little research and thinking, and was the genesis of this article.

SHELL BASIC
for Model I, III & 4

by Lance Wolstrup

There are as many ways to write a program in Basic as there are programmers to write it. I thoroughly enjoy this flexible language, easy enough for a beginner to learn, and yet always challenging, even for us old-timers. Whenever my time affords (not too often these days), I like nothing better than to sit in front of my Mod III or 4 and program something or other. The final outcome of the program itself is secondary; it's the process of writing code, trying new ideas that will either fail or flourish, that is my relaxation and fun.

It was at one of these sessions that the idea struck me: *Why not create a shell of subroutines that can be loaded in from disk before I begin writing a program.*

While this was not a new and revolutionary idea, it was something I had never done, so off the top of my head I wrote a series of common subroutines. Looking over the routines, I realized that they were just too scattered; that is, they were not organized so that related functions were kept together. For example, the routine to center text might be in line 40, followed by an INKEY$ routine in line 50; then the routine to print text flush right might be at line 75, etc. I knew that without a print-out, I'd never remember which line to *GOSUB*.

In the process of organizing the routines, I decided to write them the way I would an Assembly language subroutine. I would group related routines together into one subroutine, and then have a different entry point for each function. The more I worked at this, the more it felt like a BASIC emulation of Model 4's SuperVisor Calls (SVC).

Here, let me digress and briefly explain what SuperVisor Calls are.

People that use Assembly language to program the Model I & III quite often use routines from DOS or ROM to make life easier for themselves. After all, why write a complete routine from scratch if a routine that will do the job already exists in DOS or ROM? To access such a routine, the programmer would issue the command *CALL* followed by the *memory location* where the particular subroutine begins. In other words, the programmer has to know the *exact* location of any DOS or ROM routine he/she wishes to use. For example, to clear the screen in Mod I & III Assembly language you use this command:

```
CALL    1C9H
```

Though most of the major routines are documented in the various DOS manuals in the section called Technical Information (you know, the one you took one look at, said UGH, and went on to something else), there are so many that it is not possible to remember them all. When Mod I & III Assembly language programmers wishes to access a ROM or DOS routine, they usually have to consult the manual to get the *CALL address* of the routine. In my original subroutine venture I needed a print-out to remember which line to *GOSUB* for a particular routine. Two different languages, same problem!

The Model 4 presented a new concept in Assembly language programming: the SuperVisor Call. No longer would the programmer need to know the exact address of a DOS routine (remember, Mod 4 has no ROM, so only DOS routines are available), instead the major routines were given a number (called the SVC number) and each related routine was treated as a subfunction within the major routine. These routines, as in Mod I & III, are scattered all over the DOS area, but we don't have to worry about where. The SVC takes care of that for us by giving us the command to access any of the particular routines: RST 28H.

For example, to clear the screen in Mod 4 Assembly language, you do the following:

```
LD      A,105
RST     28H
```

First, the SVC number of the routine to clear the screen (105) is LoaDed into register A. Then, the command RST 28H sends the program into a routine that, depending on the value in register A, sends the program to the correct routine.

Now, the CLS routine only has one function. Other routines, as mentioned, might have several functions. If that is the case, then register B must contain the function number. Also, if a routine needs particular parameters, they must be passed in specified registers.

For example, the @VDCTL subroutine (SVC 15) has nine various functions relating to the video display. To position the cursor, function 3 must be used, register H must contain the vertical position of the cursor (0-23), and register L must contain the horizontal position(0-79). The code to place the cursor at 2,10 would look like this:

```
LD      A,15
LD      B,3
LD      H,2
LD      L,10
RST     28H
```

First, the SVC number was loaded into register A, then the function number was loaded into register B. Next, H was loaded with the vertical position and L was loaded with the horizontal position. Finally, the command to execute the SVC, RST 28H, was issued.

Enough about Assembly language, let's get back to the Basic language at tie in the above information to the shell of subroutines.

For the sake of this article, I put together five common routines which, in order to maintain complete compatability between Mod I, III & 4, are all written in straight Basic; no tricky, imbedded ML routines, just pure and simple Basic.

Since the program already felt like a Basic emulation of the SVC concept, why not go further and give the subroutines numbers, create sub-subroutines within the routines and allow the passing of parameters. Also, I just couldn't resist the idea of accessing any and all of the subroutines with the single command: **GOSUB 40** (28H is 40 in decimal).

The program, which I call SBASIC/BAS, is listed on the next page. Type it in and save it to disk. Whenever you wish to do some Basic programming, load it back into memory, and then start writing *your* program according to the following rules:

1. your program must always start at line 1000. Your program should not use *any* A variables, nor should it use variable B. These variables are reserved for accessing the subroutines.

2.. access the subroutine of your choice by:
a. set variable A to the SBC (SBasic Call) number.
b. set variable B to the function number (if needed).
c. pass the appropriate parameters in the specified variables (if needed).

3. issue the command: GOSUB 40.

Now that you know the rules, let's go on and look at the individual routines:

SBC 0 (A=0) - screen functions

B=0 clear screen (CLS).
paramaters: none.
B=1 erase from cursor to end of screen.
parameters: AV=vertical position of cursor.
AH=horizontal position of cursor.
B=2 erase from cursor to end of line.
parameters: AV=vertical position of cursor.
AH=horizontal position of cursor.
B=3 cursor on.
parameters: none.
B=4 cursor off.
parameters: none.

EXIT conditions:

B=1 A$=CHR$(31).
all other variables preserved.
B=2 to 4 all variables preserved.

SBC 1 (A=1) - print to screen

B=0 print text anywhere on screen.
parameters: AV=vertical line.
AH=horizontal position of first character.
A$=text.
B=1 print text flush left.
parameters: AV=vertical line.
A$=text.
B=2 print text at center of line.
parameters: AV=vertical line.
A$=text.
B=3 print text flush right.
parameters: AV=vertical line.
A$=text.

EXIT conditions:

B=0 all variables preserved.
B=1 to 3 AH=horizontal position of first character of the text.
all other variables preserved.

SBC 2 (A=2) - INKEY$ routine

parameters: none

EXIT conditions:

A$=key pressed.
A7=ASC(A$).
A8=VAL(A$).
all other variables preserved.

SBC 3 (A=3) - draw graphic box

B=0 draw box anywhere on screen.
parameters: AV=vertical position of top line of box.
AH=horizontal position of left side of box.
AL=length of box (horizontal).
AW=width of box (vertical).
B=1 draw box flush left.
parameters: AV=vertical position of top line of box.
AL=length of box (horizontal).
AW=width of box (vertical).
B=2 draw box at center.
parameters: same as B=1.
B=3 draw box flush right.
parameters: same as B=1.

EXIT conditions:

A$=graphic of bottom line of box.
A=1.
B=0.
all other variable preserved.

SBC 4 (A=4) - print text inside box

B=0 print text anywhere in box.
parameters: AV=vertical line.
AH=horizontal position of first character.
A$=text.
B=1 print text flush left in box.
parameters: AV=vertical line.
A$=text.
B=2 print text centered in box.
parameters: AV=vertical line.
A$=text.
B=3 print text flush right in box.
parameters: AV=vertical line.
A$=text.

EXIT conditions:

A=1.
B=0.
AV=logical line beneath top of box.
AH=horizontal position of first character of text inside of box.
all other variables preserved.

These capabilities are only the beginning. I hereby challenge all TRSTimes readers to put together your favorite subroutines in the above manner and send them to us for possible inclusion into SBASIC. Who knows, maybe we can create our own language within Basic!

I must point out the reasons behind lines 10, 11 & 12 of the SBASIC/BAS listing:

10 CLEAR 1000
You should change the value 1000 to the actual amount of string space *your* program will need. Of course, Mod 4 does not need to clear string space, as it is allocated dynamically. But even if your program is written on a Mod 4, do clear enough so it will also run on Mod I & III.

11 A=PEEK(&H7D)
12 IF A=4 OR A=5 THEN A9=80 ELSE A9=64
To make the program compatible between Mod I, III & 4, it is necessary to know the screen width of the computer running it. Mod I & III both have a 64 column screen; Mod 4 has 80 columns. Memory location &H7D will contain 4 if the computer is a Mod 4, and 5 if it is a Mod 4P. To the best of my knowledge, all Mod I's and III's will have a different value at that location.
Unfortunately, the detection of a Mod 4 is only valid if you run TRSDOS 6.2 or LS-DOS 6.3.
TRSDOS 6.0, 6.1, DOSPLUS 4 & MULTIDOS 2.0,2.1. will not have 4 or 5 at &H7D. If you are using one of these DOS'es, change line 12 to: **12 A9=80**.

And now, here's SBASIC...

SBASIC/BAS

```
1 'SBASIC (c) 1989 Lance Wolstrup for Mod I, III & 4
2 '
10 CLEAR 1000 'change this to fit application
11 A=PEEK(&H7D) 'is it Mod 4
12 IF A=4 OR A=5 THEN A9=80:A=0:B=4:
GOSUB 40 ELSE A9=64 'find screen width
13 GOTO 1000 'on to users program
14 '
40 ON A+1 GOTO 100,110,120,130,140 'find sub-
routine - also plenty of room for more routines
45 '
100 ON B+1 GOTO 101,102,103,104,105 ' A=0
101 AV=0:AH=0
102 A$=CHR$(31):GOTO 106
103 A$=CHR$(30):GOTO 106
104 A$=CHR$(14):GOTO 106
105 A$=CHR$(15)
106 PRINT@AV*A9+AH,A$;:RETURN
107 '
110 ON B+1 GOTO 114,111,112,113 ' A=1
111 AH=0:GOTO 114
112 AH=INT((A9-LEN(A$))/2):GOTO 114
113 AH=A9-LEN(A$)
114 PRINT@AV*A9+AH,A$;:RETURN
115 '
120 A$=INKEY$:IF A$="" THEN 120 ELSE
A7=ASC(A$):A8=VAL(A$):RETURN ' A=2
125 '
130 ON B+1 GOTO 134,131,132,133 ' A=3
131 AH=0:GOTO 134
132 AH=INT((A9-AL)/2):GOTO 134
133 AH=A9-AL
134 A1=AV:A2=AH:
A$=CHR$(151)+STRING$(AL-2,131)+CHR$(171):
A=1:B=0:GOSUB 40:
IF AW THEN FOR A0=1 TO AW-2:AV=AV+1:
A$=CHR$(149)+STRING$(AL-2,32)+CHR$(170):
GOSUB 40:NEXT
135 AV=AV+1:
A$=CHR$(141)+STRING$(AL-2,140)+CHR$(142):
GOSUB 40:AV=A1:AH=A2:RETURN
136 '
140 AV=A1+1+AV:ON B+1 GOTO 144,141,142,143 '
A=4 - find function
141 AH=A2+1:GOTO 144
142 AH=INT((A2+(AL-LEN(A$))/2)):GOTO 144
143 AH=A2+AL-1-LEN(A$)
144 A=1:B=0:GOSUB 40:RETURN
145 '
1000 ' user program starts here
```

Type in the three following demonstration programs individually, save them to disk in ASCII format as DEMO1/ASC, DEMO2/ASC & DEMO3/ASC. Load SBASIC/BAS, MERGE a demo and RUN.
Next issue will feature a Mod 4 SBASIC program, as well as one for Mod I & III.

DEMO 1

```
1000 A=0:B=0:GOSUB 40 'cls
1010 A=3:B=2:AV=0:AL=A9:AW=4:GOSUB 40
1020 A=4:B=1:AV=0:A$="TRSTimes Presents:":
GOSUB 40
1030 A=4:B=2:AV=0:A$="SHELL BASIC":GOSUB 40
1040 A=4:B=3:AV=0:A$="(c) Lance Wolstrup":
GOSUB 40
1050 A=4:B=2:AV=1:A$="A demonstration of Basic
SuperVisor Calls":GOSUB 40
1060 A=3:B=2:AV=3:AL=2:AW=9:GOSUB 40
1070 A=3:B=3:AV=12:AL=10:AW=3:GOSUB 40
1080 A=4:B=2:AV=0:A$="DEMO 1":GOSUB 40
1090 A=2:GOSUB 40
```

DEMO 2

```
1000 A=0:B=0:GOSUB 40 'cls
1010 A=3:B=1:AV=4:AL=26:AW=8:GOSUB 40
1020 A=4:B=2:AV=0:A$="TRSTimes
Presents:":GOSUB 40
1030 A=4:B=2:AV=1:A$="SHELL BASIC":GOSUB 40
1040 A=4:B=2:AV=2:A$="(c) Lance Wolstrup":
GOSUB 40
1050 A=4:B=2:AV=4:A$="A demonstration of":
GOSUB 40
1060 A=4:B=2:AV=5:A$="Basic SuperVisor Calls":
GOSUB 40
1070 A=3:B=1:AV=12:AL=10:AW=3:GOSUB 40
1080 A=4:B=2:AV=0:A$="DEMO 2":GOSUB 40
1090 A=3:B=0:AV=0:AH=A9/2:AL=16: AW=13:
GOSUB 40
1100 A=4:B=2:AV=0:A$="MAIN MENU":GOSUB 40
1110 FOR X=2 TO 10:A=4:B=1:AV=X:A$=STR$(X-
1)+". Option "+STR$(X-1):GOSUB 40:NEXT
1120 A=1:B=0:AV=X+3:AH=A9/2-5:
A$="Make your selection please ":GOSUB 40
1130 A=2:GOSUB 40:DELETE 1000-1130
```

DEMO 3

```
1000 A=0:B=0:GOSUB 40 'cls
1010 FOR X=0 TO 3:A=3:B=2:AV=X:AL=A9-X*2:
AW=14-X*2:GOSUB 40:NEXT
1020 A=4:B=2:AV=0:A$="TRSTimes Presents:":
GOSUB 40
1030 A=4:B=2:AV=1:A$="SHELL BASIC":GOSUB 40
1040 A=4:B=2:AV=2:A$="(c) Lance Wolstrup":
GOSUB 40
1050 A=4:B=2:AV=4:A$="A demonstration of":
GOSUB 40
1060 A=4:B=2:AV=5:A$="Basic SuperVisor
Calls":GOSUB 40
1070 A=3:B=2:AV=12:AL=10:AW=3:GOSUB 40
1080 A=4:B=2:AV=0:A$="DEMO 3":GOSUB 40
1090 A=2:GOSUB 40
```

Model 4 - Editor/Assembler

NX-10 printer utility

by Danny C. Mullen

Quite often I have the need to pass font codes as well as other information to my printer. In the past this always involved loading BASIC, followed by the typing of a bunch of 'LPRINT CHR$(X) commands. Much work and very slow!

I have seen several BASIC programs which, by the way of a menu, allows the user to set the desired printer control codes. A nice idea, but still slow, since you have to load BASIC and then run the program. I wanted to do the job fast, so I wrote NX. As it is written in assembly language, it can be executed directly from DOS. Simply type NX <ENTER> at the DOS READY prompt and the program is up and running.

You will see the first of two menus, giving you the following options:

1) Print slashed zero
2) Print normal zero
3) Compressed Print
4) Cancel Compressed
5) Line Feed (1/8")
6) Line Feed (7/72")
7) Line Feed (1/6")
8) Italics
9) Cancel Italics
10) Reinitialize Printer
11) Set to 10 CPI

Press --> key to view next screen

Pressing the RIGHT ARROW key displays the second menu. Here these options are available

:

12) Set to 12 CPI
13) Double Wide Print
14) Set form to 60 lines
15) Set form to 11 inches
16) Emphasized print
17) Cancel emphasized
18) Double-strike mode
19) Cancel double-strike
20) Skip-over 4 lines
21) Letter quality mode
22) Cancel letter quality mode

Press <-- to view first screen

The RIGHT & LEFT ARROWS toggles the menus, while using the UP & DOWN ARROWS will move the reverse video selection bar. When the bar is at your desired selection, press the <ENTER> key and the appropriate control codes will be sent.

If the printer isn't on, a message will be displayed and a warning tone will sound. You will then be given a chance to turn it on.

When you are satified with your selections, press the <BREAK> key to exit to DOS.

Keep in mind that some of the codes may only work for one line, others will work until reset or changed. Also, some may work in combination with others, while others may not. Your printer manual will give the details.

I only used 4-byte control codes. The NX-10 has some functions that require five (or more) byte sequences and this program could be easily adapted to them if desired.

Look at the LD B,4 just prior to the label OWT1 - this is one counter to change and the others are at DWN and UP. You would also have to change all the BITS and BITS1 to the same length.

It would be easy to change the codes for other printers, just match the changes line for line (ie. the FF 'MENU' line and the first 'BITS' line, etc).

I originally tried to make this a pop up program callable by one of those great <CLEAR><??> sequences, but have been frustrated time and again even after copying several filter installation programs from 80 MICRO, etc. Lately, haven't had the time to keep trying.

Should anyone accomplish this, please let me know. I would like to see the result.

The source code was written using Radio Shack ALDS. Therefore, if you are using EDTASM, be sure to change all occurances of DB and DS to DEFB and DEFS respectively. Also, the lines that have DB followed by several pieces of data must be changed to individual lines with DEFB followed by just one piece of data. Lastly, since EDTASM is not capable of executing macros, all occurances of the SVC macro must be changed to:

LD A, equate value of label
RST 40

Type in NX, assemble it, and have fun with your newly gained control of your printer.

```
   ; NX/SRC
   ; 11/02/88 VERSION
   ; Designed to send control codes to printer
   ; (c) Danny C. Mullen
   ; PSC BOX 821
   ; APO MIAMI, FL 34004
   ; TESTED ON TRS80 MODEL 4 WITH
   ; TRSDOS 6.x.x and LS-DOS 6.3 ONLY.
   ; DESIGNED FOR STAR NX-10 PRINTER
   ;
KEY        EQU     1
PRT        EQU     6
DSP        EQU     2
DSPLY      EQU     10
VDCTL      EQU     15
EXIT       EQU     22
SOUND      EQU     104
CLS        EQU     105
   ;
   ; Menus
           PSECT   2600H
MENU       DEFM    '  1) Print slashed zero'
           DB      0AH
           DEFM    '  2) Print normal zero'
           DB      0AH
           DEFM    '  3) Compressed Print'
           DB      0AH
           DEFM    '  4) Cancel Compressed'
           DB      0AH
           DEFM    '  5) Line Feed (1/8")'
           DB      0AH
           DEFM    '  6) Line Feed (7/72")'
           DB      0AH
           DEFM    '  7) Line Feed (1/6")'
           DB      0AH
           DEFM    '  8) Italics'
           DB      0AH
           DEFM    '  9) Cancel Italics'
           DB      0AH
           DEFM    '  10) Reinitialize Printer'
           DB      0AH
           DEFM    '  11) Set to 10 CPI'
           DB      0AH,0AH
           DB      16
           DEFM    ' Press --> key to view next
                   screen '
           DB      17
           DB      0AH
           DEFM    '              NX10'
           DB      0AH
           DB      16
           DEFM    '     Use arrow keys to
                   select    '
           DB      17
           DB      0DH
MENU2      DEFM    '  12) Set to 12 CPI'
           DB      0AH
           DEFM    '  13) Double Wide Print'
           DB      0AH
           DEFM    '  14) Set form to 60 lines'
           DB      0AH
           DEFM    '  15) Set form to 11 inches'
           DB      0AH
           DEFM    '  16) Emphasized print'
           DB      0AH
           DEFM    '  17) Cancel emphasized'
           DB      0AH
           DEFM    '  18) Double-strike mode'
           DB      0AH
           DEFM    '  19) Cancel double-strike'
           DB      0AH
           DEFM    '  20) Skip-over 4 lines'
           DB      0AH
           DEFM    '  21) Letter quality mode'
           DB      0AH
           DEFM    '  22) Cancel letter quality mode'
           DB      0AH,0AH
           DB      16
           DEFM    ' Press <-- key to view first
                   screen '
           DB      17
           DB      0AH
           DEFM    '              NX10'
           DB      0AH
           DB      16
           DEFM    '     Use arrow keys to
                   select    '
           DB      17
           DB      0DH
   ;
   ; These are the codes to be sent to the printer
   ;
BITS       DB      00,27,126,1         ;print slashed zero
           DB      00,27,126,0         ;print normal 0
           DB      00,00,00,15         ;compressed
           DB      00,00,00,18         ;cancel compressed
           DB      00,00,27,48         ;line feed = 1/8"
           DB      00,00,27,49         ;line feed = 7/72"
           DB      00,00,27,50         ;line feed = 1/6"
           DB      00,00,27,52         ;italics
           DB      00,00,27,53         ;cancel italics
           DB      00,00,27,64         ;reinitialize
           DB      00,00,27,80         ;10 cpi
BITS1      DB      00,00,27,77         ;12 cpi
           DB      00,27,87,1          ;double wide
           DB      00,27,67,60         ;form length = 60
           DB      27,67,0,11          ;form length = 11
           DB      00,00,27,69         ;emphasized
           DB      00,00,27,70         ;cancel emphasized
           DB      00,00,27,71         ;double strike
           DB      00,00,27,72         ;cancel double strike
           DB      00,27,78,4          ;skip-over 4 lines
           DB      00,27,120,1         ;letter quality mode
           DB      00,27,20,0          ;cancel lq mode
   ;
   ; Messages
```

```
MESG1   DEFM    ' Select... then (ENTER) or
                (BREAK) to quit '
        DB      03H
;
;
MESG2   DB      16      ;Enable rev video
        DEFM    ' Send another code? (ENTER)
                =YES (BRK)=QUIT '
        DB      17      ;Disable rev video
        DB      03H
;
;
PRMSG   DEFM    '*** PRINTER NOT READY ***'
        DB      0AH
        DEFM    '...Hit any key when ready'
        DB      0DH
;
;   Buffers
;
BUFF    DS      4       ;Code buffer
BUFF1   DS      2       ;Address buffer
;
;   Subroutines
;
MOVE    LD      B,3     ;Move msg line to
        LD      H,17    ; Row 17
        LD      L,0     ; Column 0
        SVC     VDCTL
        RET
;
BUFLD   LD      BC,4    ;Buffer load routine
        LD      HL,(BUFF1) ;Point to origin
        LD      DE,BUFF ;Point to destination
        LDIR            ;   and load it in
        RET
;
HOME    LD      B,3     ;Home cursor
        LD      H,0     ;H = row
        LD      L,1     ;L=column
        SVC     VDCTL
        CALL    INVRT
        RET
;
;
DWN     LD      B,4     ;Crsor dwn routine
        SVC     VDCTL   ;Get crsor location
        LD      A,10    ;Is it at line 11?
        CP      H       ;H=row number
        RET     Z       ;yes-don't move it
        INC     H       ;no- move it down
        LD      B,3
        SVC     VDCTL
        CALL    INVRT
        CALL    INVRT1
        LD      HL,(BUFF1) ;Get old address
        INC     HL      ;Increase
        INC     HL      ; address
        INC     HL      ;   of pointer
        INC     HL      ;     by four
        LD      (BUFF1),HL      ;Put addrs in buffer
        CALL    BUFLD   ;Put codes in buffer
        RET
;
;
UP      LD      B,4     ;Cursor up routine
        SVC     VDCTL   ;Get cursor location
        LD      A,0     ;Is it at line 0?
        CP      H       ;H = row number
        RET     Z       ;yes-don't move it
        DEC     H       ;no- move it up
        LD      B,3
        SVC     VDCTL
        CALL    INVRT
        CALL    INVRT2
        LD      HL,(BUFF1)      ;Get old address
        DEC     HL      ;Decrease
        DEC     HL      ; address
        DEC     HL      ;   of pointer
        DEC     HL      ;     by four
        LD      (BUFF1),HL      ;Put addrs in buffer
        CALL    BUFLD   ;Put codes in buffer
        RET
;
;   Loop
;
OWT     CALL    PRTEST          ;Test if printer on line
        LD      HL,BUFF         ;Get address of
                                ; string of bytes
        LD      B,4     ;Four bytes to send
OWT1    LD      C,(HL)  ;Send it out
        SVC     PRT     ; Might try OTIR here
        INC     HL
        DJNZ    OWT1    ;Loop until done
AGAIN   CALL    MOVE
        LD      HL,MESG2        ;Need menu again?
        SVC     DSPLY
        SVC     KEY
        CP      'Y'     ;If yes, start over
        JR      Z,START
        CP      'y'
        JR      Z,START
        CP      0DH     ;(ENTER)=yes
        JR      Z,START
        CP      'N'     ;If no, end
        JP      Z,FINI
        CP      'n'
        JP      Z,FINI
        CP      80H     ;(BREAK)=no
        JP      Z,FINI
        JR      AGAIN   ;Must have a response
;
;
PRTEST  IN      A,(0F8H)        ;Get image
        AND     0F0H    ;Strip lo bits
        CP      30H
        RET     Z       ;Yes, on line
        SVC     CLS     ;Clear screen
        LD      HL,PRMSG        ;Turn on printer msg
```

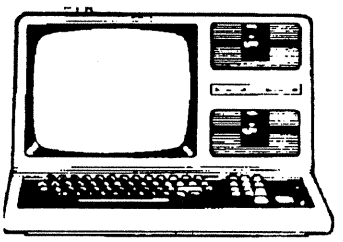
```
            SVC     DSPLY
            LD      B,5AH   ;Long high tone
            SVC     SOUND;to alert user
            SVC     KEY     ;Must get a key
            POP     HL
            JR      START
    ;
    ;
    ; Program starts here
    ;
START       SVC     CLS     ;Clean the slate
            LD      C,15    ;cursor off
            SVC     DSP
            LD      HL,MENU ;Point to 1st screen
            SVC     DSPLY   ;and display it
            CALL    MOVE    ;Put msg line on
            LD      HL,MESG1 ;bottom-out of the
            SVC     DSPLY   ;way and print it
            CALL    BOX
            CALL    HOME
            LD      HL,BITS ;Point to first table
            JR      PUTIT   ;Skip over
STRT1       SVC     CLS     ;Clean the slate
            LD      HL,MENU2 ;Point to second
            SVC     DSPLY   ;screen and dsply it
            CALL    MOVE    ;Put msg line on
            LD      HL,MESG1 ;bottom-out of the
            SVC     DSPLY   ;way and print it
            CALL    BOX
            CALL    HOME
            LD      HL,BITS1 ;Point to 2nd table
PUTIT       LD      (BUFF1),HL ;Put addrs in buffer
            CALL    BUFLD   ;Put codes in buffer
GETIT       SVC     KEY     ;Get a selection
            CP      0AH     ;Down arrow?
            CALL    Z,DWN   ;Yes, inc pointers
            CP      0BH     ;Up arrow?
            CALL    Z,UP    ;Yes, dec pointers
            CP      09H     ; + key?
            JR      Z,STRT1 ;If yes, 2nd screen
            CP      08H     ; - key?
            JR      Z,START ;If yes, first screen
            CP      80H     ;Break?
            JR      Z,FINI  ;If yes, exit to DOS
            CP      0DH     ;Enter?
            JP      Z,OWT   ;Yes-out to printer
            JR      GETIT   ;Must have one of
                            ;these responses
    ;
    ;
    ;   Exit to DOS
    ;
FINI        SVC     CLS     ;Clean the slate
            LD      C,14
            SVC     DSP     ;cursor on
            LD      HL,0    ;and exit to DOS
            SVC     EXIT
    ;
    ;
```

```
WHO         DEFM    '(c) 1988 DANNY C. MULLEN'
            DEFM    'PSC BOX 821 APO MIAMI,FL
                    34004'
    ;
    ;
BOX         LD      B,0FH   ;do 15 lines
            LD      H,0     ;row 0
            LD      L,35    ;col 35
LP1         PUSH    BC
            LD      B,2
            LD      C,0A0H ;inverted space
            SVC     VDCTL
            INC     H
            POP     BC
            DJNZ    LP1
            LD      B,0FH
            LD      HL,0000             ;row & col 0
LP3         PUSH    BC
            LD      B,2
            LD      C,0A0H
            SVC     VDCTL
            INC     H
            POP     BC
            DJNZ    LP3
            RET
    ;
    ;
INVRT       PUSH    HL      ;Routines to do invrse
            PUSH    DE
            LD      B,4
            SVC     VDCTL   ;Get curs location
            LD      A,2
            LD      L,A
            LD      B,32    ;Do 26 chars
LP4         PUSH    BC
            LD      B,1
            SVC     VDCTL   ;get char at cursor
            OR      80H     ;set high bit
            LD      C,A
            LD      B,2
            SVC     VDCTL   ;and put it back
            INC     L
            POP     BC
            DJNZ    LP4     ;loop back
            POP     DE
            POP     HL
            RET
    ;
    ;
INVRT1      PUSH    HL
            PUSH    DE
            LD      B,4
            SVC     VDCTL
            LD      A,2     ;do upper row
            DEC     H
            LD      L,A
            LD      B,32
LP5         PUSH    BC
```

```
                LD      B,1
                SVC     VDCTL
                AND     7FH     ;reset hi bit
                LD      C,A
                LD      B,2
                SVC     VDCTL
                INC     L
                POP     BC
                DJNZ    LP5     ;loop back
                POP     DE
                POP     HL
                RET
;
;
INVRT2          PUSH    HL
                PUSH    DE
                LD      B,4
                SVC     VDCTL
                INC     H       ;next row
                LD      A,2
                LD      L,A
                LD      B,32
LP6             PUSH    BC
                LD      B,1
                SVC     VDCTL
                AND     7FH     ;reset hi bit
                LD      C,A
                LD      B,2
                LD      A,0FH
                RST     28H
                INC     L
                POP     BC
                DJNZ    LP6
                POP     DE
                POP     HL
                RET
                END     START
```

Note from Ye Faithful Editor:

Some very minor modifications were made to the EQUATE labels. Danny originally preceded each by the @ sign, as is legal in ALDS. However, to make the listing more compatible with the 'patched Model 4 version of EDTASM', this was removed; also, in order to fit the listing in two column format, some of the comments were abbreviated slightly.



● **Model I & III**

Debugging Aid for Basic Programs

by John H. Shephard

Some time ago I was debugging a Basic program intended to generate a display on the screen, I inserted a STOP command at various points in the program to see how the display was developing. This proved very frustrating, as each STOP command would display several lines and scroll the display. I needed some way of pausing the program at selected lines, displaying the line number without much change on the screen, and then allowing the program to continue.

By using the "ERROR" command to simulate an error with "ON ERROR GOTO" to execute a subroutine to display the line number in a corner of the screen and then wait for a keypresss before proceeding, I was able to get the effect I needed, using "RESUME NEXT" to continue the program.

The basic code which follows is intended to be incorporated in the main program (with LBASIC use "MERGE"). To pause the program at a given line add **:ERROR 23** at the end of the line.

Error 23 will never occur with a disk-based program, it occurs only when attempting to execute a disk type command in a cassette-based machine.

```
1 ON ERROR GOTO 65500
        REM this must precede the main program

**      Insert the main program here
**      To pause at a given line - add :ERROR 23
**      at the end of the line.

65500 IF ERR/2+1 + 23 THEN GOTO 65504
65502 ON ERROR GOTO 0: RESUME
        REM Treat all but ERROR 23 normally

65504 CL% = PEEK(16416): CH% = PEEK(16417)
        REM Save cursor position

65506 PRINT@58,ERL;:
        REM Display basic line number in top
        right corner

65508 KD$=INKEY$: IF KD$="" THEN 65508
        REM Wait for any key

65510 PRINT@57,"*******"  ;:
        REM Blot out any previously displayed line

65512 POKE 16416,CL%: POKE 16417,CH%
        REM Restore cursor position

65514 RESUME NEXT
        REM Continue with the program
```

NOTES:

ERR/2+1 returns the error code.

ERL returns the line number at which the error occurred.

If there is a CLEAR statement in the main program, this will negate the effect of line 1. To avoid this, move line 1 to a position in the main program after the the line containing the CLEAR statement. Be sure that it still precedes any of the ERROR 23 statements.

● **Model I, III & 4**

The World's smallest word-processor

by Edgar Martin

Using one of the big professional word-processors to type a short note is like spraying for flies with a flamethrower; it is overkill. Here is the 'world's smallest word-processor', a one line basic program:

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

Notice that editing commands, printer drivers and other useless stuff have been cut out. When you run the program your computer will immediately turn into a lean, mean text-machine. It will respond to your slightest keypress by printing that character on your printer.

● **Model 4 in Model III mode**

● **TRSDOS 1.3. - BASIC**

Goodies from Gary Campbell

TRSDOS 1.3. has a screen size of 1024 characters. Using any Model 4, under TRSDOS 6., a 2048 character screen is used. So where does this extra 1024 byte screen go when using a Model 4, under TRSDOS 1.3.?

It is there all right, just waiting to be discovered! Under 1.3., (possibly other Model III DOS'es too), and when

using any Model 4, there are actually 2 complete video screens available for use! And they can be selected at will!

When a screen is selected, the old screen is saved completely, and the new screen will be displayed. Swap back, and PRESTO! The old screen is exactly as you left it!

I like to use this 'extra' screen to store a mini "help" screen, that can be popped up whenever required. The screen swapping is lightning fast, and the 2 screens do not interfere with each other, with the exception that both screens share the same cursor position! If you move the cursor in one screen, it will be at the same location on the other. Therefore, you may wish to save the cursor position using:

CP=PEEK(16416)+PEEK(16417)*256-15360

CP will now contain the PRINT@ position of the current cursor position.

The following program will demonstrate how to create a mini "help" screen, that may be recalled at any time! Type in and save TEST1/BAS, and TEST2/BAS. Then RUN TEST1/BAS.

Program listing "A" (save as TEST1/BAS)

```
10 GOSUB 100 'Switch in new screen
15 CLS
20 'lines 21 & 22 display what is to be recalled.
21 PRINT"MINI HELP SCREEN":PRINT:PRINT"TO EXIT
BACK TO"
22 PRINT"THE MAIN SCREEN, TAP THE 'X' KEY!"
30 PRINT:PRINT"NOT REALLY A LOT OF HELP, EH?"
40 GOSUB 200 'Switch back original screen
50 RUN"TEST2/BAS"
100 OUT 136,12:OUT 132,128:OUT 137,4:RETURN
200 OUT 136,12:OUT 132,0:OUT 137,4:RETURN
```

Program listing "B" (save as TEST2/BAS)

```
10 CLEAR 100
15 'lines 20-60 are just some display stuff to
demonstrate how the screens are swapped
20 CLS:PRINT"MAIN MENU":PRINT
30 PRINT"A) DISPLAY DRIVE 0 FILES"
40 PRINT"B) DISPLAY HELP MENU"
50 PRINT"C) END"
60 PRINT:PRINT"ENTER YOUR CHOICE:";
70 A$=INKEY$:IF A$="" THEN 70
80  IF A$="A" THEN CMD"D:0":RUN
90 IF A$="C" THEN END
91 IF A$<>"B" THEN 20
92 GOSUB 100
94 A$=INKEY$:IF A$="" THEN 94 ELSE IF A$="X"
THEN 96 ELSE 94
96 GOSUB 200:GOTO 70
100 OUT 136,12:OUT 132,128:OUT 137,4:RETURN
200 OUT 136,12:OUT 132,0:OUT 137,0:RETURN
```

Note: Line 80 of TEST2/BAS is for TRSDOS 1.3. only. To work on other DOS'es, change this line to:
80 IF A$="A" THEN **CMD"DIR :0"**:RUN

Ed.

Assembler subroutines would load register A with the OUT parameter, and then send the byte to the appropriate port.

Subroutine 100:

```
LD      A,12
OUT     (136),A
LD      A,128
OUT     (132),A
LD      A,4
OUT     (137),A
RET
```

Subroutine 200:

```
LD      A,12
OUT     (136),A
LD      A,0
OUT     (132),A
OUT     (137),A
RET
```

● **Model I & III**

ATTENTION LDOS

by Lance Wolstrup

TRS-80 Model I & III password protection has always struck me as being annoying, at best. At worst, it is ridiculous and a complete waste of DOS programming space.

The more adventurous of us have overcome this 'feature', either by using ZAP programs, or by writing our own programs to strip individual files of its dreaded passwords. In the case of Model I & III LDOS, these grand efforts were all 'doing it the hard way'.

THE EASY LDOS WAY:

The people at LDOS were always generous, giving us the passwords to all system and related DOS files. They neglected to tell us, however, that a 'backdoor' password existed; that is, a universal password that unlocks ALL files.

Yes, Roy Soltoff played a little joke on us. He allows complete access to any and all files when this password is used: **RS0LT0FF**

(that is RSOLTOFF with the O's replaced by zeroes.)

Works on both Model I & III - but LDOS only

Assembly 101
Z80 without tears.
by Lance Wolstrup

Last issue we got our feet wet, learning how to use EDTASM and, to a certain degree, how to display text on the screen. Let's do a recap:

```
00100          ORG    7000H
00110 MSG1     DEFM   'HI, I AM YOUR TRS-80
                      COMPUTER'
00120          DEFB   0DH
00130 BEGIN    LD     HL,MSG1
00140          CALL   021BH    ;Model 1 use 4467H
00150          RET
00160          END    BEGIN
```

First, we must tell the assembler (EDTASM) where to assemble our program in memory. We do this by issuing the assembler command (pseudo opcode) ORG, followed by the desired address (7000H). This assembler command is mandatory.

Then, by using another assembler command, DEFM (DEFine Message), we set up our text inside of single quotes 'HI, I AM YOUR TRS-80 COMPUTER'. We give this text the label MSG1.

Using the assembler command DEFB (DEFine Byte), we add 0DH to the end of our text. 0DH is 13 decimal and, hopefully, we all know that CHR$(13) is the (ENTER) key. More specifically, CHR$(13) is a CR (carriage return).

Note: ORG, DEFM & DEFB are commands to the assembler, they are NOT assembly language instructions.

We then point register HL to out text (by LoaDing HL with the address of MSG1). Since this is the actual start of our program, we label it BEGIN.

The next instruction is CALL 21BH. This is the Model III ROM routine that displays the text pointed to by register HL. It keeps on displaying until it encounters the byte 13. It then quits and returns to the calling program. (Model I users should substitute this with a call to the DOS routine at 4467H: CALL 4467H).

At this point we have displayed our text, so we issue the RET instruction to RETurn to DOS.

Lastly, also mandatory, we issue the assembler command END and point it to the beginning of our program (BEGIN).

If you do not understand this completely, please go back and read the previous installment (TRSTimes 2.1.) until you do.

More on the PRINT statement

Assuming that this is now perfectly clear, let's switch over to BASIC and look at the lowly (but very important) PRINT statement.

There are primarily two ways to display text with the PRINT statement:

1.) PRINT"HI, I AM YOUR TRS-80 COMPUTER"
2.) PRINT"HI, I AM YOUR TRS-80 COMPUTER";

The PRINT statement displays text beginning at the *CURRENT CURSOR LOCATION*. Example 1 prints the first character of the text at the current cursor location, the cursor is then moved one pixel to the right and the second character is displayed there, the cursor is again moved one pixel to the right, etc. When the last character has been displayed, the cursor is automatically moved down to the beginning of the next line.

Example 2 works identically to example 1, except when the end of the text has been displayed. Here the PRINT statement sees a semicolon. This means: Do not move the cursor to the beginning of the next line - *LEAVE THE CURSOR WHERE IT IS*.

Let's relate this information to our assembly language program above. We have primarily three ways to tell the Model III ROM subroutine at 21BH (or Model I DOS routine at 4467H) to handle the cursor at the end of out text:

1.) DEFB 0DH - end of text, move cursor to beginning of next line.
2.) DEFB 03H - end of text, leave cursor where it is.
3.) DEFB 0AH - NOT end of text, but move cursor to beginning of next line.

First, let's expand our program to display several lines of text:

```
00100          ORG    7000H
00110 MSG1     DEFM   'HI, I AM YOUR TRS-80 COMPUTER'
00120          DEFB   0DH
00130 MSG2     DEFM   'I HOPE I CAN HELP YOU'
00140          DEFB   0DH
00150 MSG3     DEFM   'TO LEARN ASSEMBLY LANGUAGE'
00160          DEFB   0DH
00170BEGIN     LD     HL,MSG1
00180          CALL   21BH     ;Model I use 4467H
00190          LD     HL,MSG2
00200          CALL   21BH     ;Model I use 4467H
00210          LD     HL,MSG3
00220          CALL   21BH     ;Model I use 4467H
00230          RET
00240          END    BEGIN
```

The above could have been written somewhat shorter:

```
00100          ORG    7000H
00110 MSG1     DEFM   'HI, I AM YOUR TRS-80 COMPUTER'
```

```
00120           DEFB    0AH
00130           DEFM    'I HOPE I CAN HELP YOU'
00140           DEFB    0AH
00150           DEFM    'TO LEARN ASSEMBLY
                        LANGUAGE'
00160           DEFB    ODH
00170 BEGIN     LD      HL,MSG1
00180           CALL    21BH      ;Model I use 4467H
00190           RET
00200           END     BEGIN
```

As you can see, we point HL to our text and then call the display subroutine. This routine displays the first line and then encounters the 0AH byte. 0AH is 10 in decimal. CHR$(10) is LF (linefeed), so the cursor moves to the beginning of the next line where it displays the next text. Again, at the end of the second text, the routine sees a 0AH, so we get another linefeed and the third line of text is displayed. Here, however, the routine finds a 0DH at the end of the text, so it quits and returns to our program.

Go ahead and type in both of these programs. (**W**)rite the source files to disk; then (**A**)ssemble them into CMD files. Of course, be sure to give them different names. You will notice that they execute identically.

Now, let's move on to the 03H ending byte. Type in, (**W**)rite and (**A**)ssemble the following program:

```
00100           ORG     7000H
00110 MSG1      DEFM    'HI, I AM YOUR TRS-80
                        COMPUTER'
00120           DEFB    03H
00130 MSG2      DEFM    'I HOPE I CAN HELP YOU'
00140           DEFB    03H
00150 MSG3      DEFM    'TO LEARN ASSEMBLY
                        LANGUAGE'
00160           DEFB    0DH
00170 BEGIN     LD      HL,MSG1
00180           CALL    21BH      ;Model I use 4467H
00190           LD      HL,MSG2
00200           CALL    21BH      ;Model I use 4467H
00210           LD      HL,MSG3
00220           CALL    21BH      ;Model I use 4467H
00230           RET
00240           END     BEGIN
```

Because of the 03H end byte, the cursor did not move to the next line; instead it stayed immediately next to the text it just displayed, thus displaying the next text on the same line. This works exactly like this Basic program:

```
100 PRINT"HI, I AM YOUR TRS-80 COMPUTER";
110 PRINT"I HOPE I CAN HELP YOU";
120 PRINT"TO LEARN ASSEMBLY LANGUAGE"
```

If you have typed in the examples, have gotten them to work and understand why, we will then move on to bigger and better things. If not, go back and keep plugging at it until you do (there is no other way).

PRINT@

I am sure that, when you first learned Basic, you proceeded to write programs that consisted of nothing but PRINT statements. Fun, wasn't it? However, pretty soon you realized that all the *good* programs did not display the text flush left on the screen. These programs displayed the text on the screen wherever the programmer wanted it to be. *Pretty fancy!* You wanted to do that, so you learned the PRINT@ command.

Today, you may use this command extensively, but do you *really* know what it does? "Sure", you say, "it prints the text at the specified screen location."

That is certainly the result of the command, but it is not the whole story. For example:

```
PRINT@160,"HI, I AM YOUR TRS-80 COMPUTER"
```

The first thing this command does is *move the cursor* to the specified screen location (160); then, and only then, does it display the text inside the quotes *at the cursor location*. The cursor is very important. Basic handles this for us, almost to the point where we don't have to think about it; Assembly language does not. If we want to display text at a particular screen location, we must place the cursor there first.

Both Model I and III have memory-mapped screens; that is, a series of 1024 (64x16) memory locations have been set aside, each corresponding to a screen location. The screen memory begins at 15360 decimal (3C00H) and ends at 16383 decimal (3CFFH). Some people refer to these locations by their hexadecimal addresses; I prefer to address them in decimal. It just seems easier to do it in decimal, as PRINT@ location 0 is now 15360, PRINT@ location 384 is now 15360+384, PRINT@ location 962 becomes 15360+962. EDTASM doesn't care if you use hex or decimal, or even octal (don't worry, no octal for us)!

So, now that we know where the screen is, let's learn how to position the cursor. As an example, we will position the cursor at the center of the third screen line. That would be PRINT@160 in Basic; in Assembly language it is memory location 15360+160 or, to be more specific, memory location 15520.

Since both Model I & III handle the cursor position through memory locations 4020H and 4021H, we must store the desired cursor location there.

As we know, a memory location can only hold numbers in the range 0 to 255. Our number is 15520, so we must put the LSB (Least Significant Byte) into 4020H and the MSB (most Significant Byte) into 4021H.

The code can be written this way:

```
LD      HL,15520
LD      (4020H),HL
```

We LoaD register HL with the value 15520. The value, in this particular case, is our desired screen memory location. Then we LoaD the value of HL into the cursor-handling locations. As explained above, the value 15520 is a two-byte number in hexadecimal (3CA0H), thus we must enclose the cursor-handling address inside a parenthesis so it will know to place the LSB (A0H) at 4020H and the MSB (3CH) at 4021H.

We could do the same thing from Basic with the following program:

```
10 HL=15520
20 M=INT(HL/256) :  ' M=MSB
30 L=HL-M*256 :     ' L=LSB
40 POKE &H4020,L:  ' put LSB into 4020H
50 POKE &H4021,M:  ' put MSB into 4021H
```

The cursor is placed at memory location 15520 but, since no PRINT statement was found, naturally it moves to the beginning of the next line.

Add this line:

```
60 PRINT"HI, I AM YOUR TRS-80 COMPUTER"
```

When you RUN the program, you'll see that you have managed to position your text *without* using the PRINT@ statement.

Getting back to Assembly language, we'll perform the above task as follows:

```
00100          ORG    7000H
00110 MSG1     DEFM   'HI, I AM YOUR TRS-80
                      COMPUTER'
00120          DEFB   0DH
00130 BEGIN    LD     HL,15520
00140          LD     (4020H),HL
00150          LD     HL,MSG1
00160          CALL   21BH     ;Model I use 4467H
00170          RET
00180          END    BEGIN
```

Now that you can place text anywhere on the screen, you might say something like: "If I could only erase the screen, I'd be in good shape!"

Amazingly enough, this is just what we will do next.

CLS

In Basic we would write a program like this:

```
10 CLS
20 PRINT@160,"HI, I AM YOUR TRS-80 COMPUTER"
```

This program actually performs 4 individual step:

1. Erase the screen (CLS)
2. Position the cursor (PRINT@160,)
3. Display text at cursor position ("HI, I AM....etc.")
4. Return to Basic (a Basic program will return to immediate mode when: **1.** it encounters an END statement, or **2.** it finishes executing the last line of the program.)

To write this program in Assembly language, we would need only to change step 4. Being an Assembly language program, we naturally want to return to DOS, rather than Basic. Thus, our concept would look like this:

1. Erase the screen
2. Position the cursor
3. Display text at cursor location
4. Return to DOS

Both Model I & III have a ROM routine that will erase the screen for us. It is located at adrees 1C9H. A simple CALL to that address will do the trick.

Type in, (**W**)rite and (**A**)ssemble this code:

```
00100          ORG    7000H
00110 MSG1     DEFM   'HI, I AM YOUR TRS-80
                      COMPUTER'
00120          DEFB   0DH
00130 BEGIN    CALL   1C9H
00140          LD     HL,15520
00150          LD     (4020H),HL
00160          LD     HL,MSG1
00170          CALL   21BH     ;Model I use 4467H
00180          RET
00190          END    BEGIN
```

Line 00130 makes the CALL to the clear screen ROM routine at 1C9H.

Line 00140 loads HL with our desired cursor position.

Line 00150 puts the cursor position into the cursor handling location (at this point the cursor is placed).

Line 00160 points HL to the text in MSG1

Line 00170 makes the CALL to the Model III ROM print routine at 21BH (or the Model I DOS routine at 4467H).

Line 00180 returns to DOS.

I am rapidly reaching the end of my alloted space for this issue, so I will use the remaining few lines to tell you that in our next installment we will discuss how to POKE the screen from Assembly language; we will also learn how to accept information from the keyboard (INKEY$ and INPUT).

If space permits, we will begin writing a simple, but useful program that will let you type a name & address and then print a mailing label on the printer. Who knows, we may even expand the program, adding some bells & whistles, such as a mini-editor and other stuff. In any event, we will first write the program in Basic, and then translate it into Assembly language, *line by line*.

Until next issue, keep practicing!

TRSDOS 1.3 Corner

HUNTING FOR DOS WITHIN BASIC

by Gary Edwin Campbell

When it comes to using DOS commands from within 1.3 Basic, the season is closed. Back in 1980, this feature was just not heard of.

The following program enables 1.3 Basic to support DOS command processing, while maintaining all variable storage.

****** I M P O R T A N T ******

Follow instructions A) - H) carefully!

The utility published herein is similar to the SYSTEM 1.5 (tm) utility, with some subtle changes.

****** I M P O R T A N T ******

About the program:

- The Basic program copies BASIC/CMD to LIBDVR/CMD.
- LIBDVR's end of file byte (EOF) is altered for append purposes.
- The data statements are processed, creating a file called LIB/APD.
- This file is appended to the end of the LIBDVR/CMD program.

LIBDVR is now *enhanced* Basic. To enter enhanced Basic, use **LIBDVR** instead of BASIC.

LIBDVR works with 99% of all basic programs. A limited number of programs alter their own program statement table (PST) text. If you have a program of this nature, it will undoubtedly cause syntax errors at the "alter" address.

DISK Basic is just an enhanced version of ROM Basic. The disk enhancements are enabled by placing jump instructions in memory locations 4152H - 41E2H.

When Basic sees a disk token, it calls an appropriate 4152H - 41E2H address, and control then jumps to one of the enhancement routines. The CMD command is referenced at 4174H. Any time Basic sees a CMD token, it calls 4174H.

The driver program, being loaded with DISK Basic itself, routes two disk exit jumps to its own initialization routine. When DISK Basic thinks its all done with its stuff, it ends up executing a new install routine. This routine relocates the new CMD processing code to the beginning of the PST and places an appropriate hook into 4174H. It resets the PST pointer and sets up housekeeping. Return is then passed back to the normal DISK Basic exits. Basic is now enhanced! It's open season on DOS hunting!

Let us examine BASIC's new hunting skills!

```
10 CMD"D:1"
20 CMD"!FREE"
```

Remember that 4174H hook? Well, its already baited, waiting patiently for a bite! When the Basic interpreter spots a CMD token, it calls 4174H, which jumps to the new driver. Bytes following the CMD token are examined for a "! sequence.

Line 10 fails this test, so processing jumps to the original CMD hook.

In line 20, the "! test passes, so the next bytes in the line are moved to 4225H, until the closing parentheses are found. A free memory check is then done, and a OM error is generated if less than 5220 bytes are free.

Next on the list is to examine 4225H+. If the 2nd or 3rd bytes of 4225h contain a $, the text is presumed to be a string variable. eg: A$, FS$, A$(9), etc. After a VARPTR search and some error checking, the text of the string itself is moved to 4225H. The string length must be greater than 0 or an error will be returned.

Unfortunately, most DOS commands will over write BASIC. So, those troublesome memory locations are copied to free space available in higher memories.

A few DOS vectors, such as the ABORT and ERROR vectors are patched with return "traps". OUCH!

The traps ensure that if an abort (BREAK) or error happens while the DOS command is being processed, control will return to the "recover" routine. The high memory and clear pointers are saved, and reset to 6500H.

This is done for a number of reasons. The stack is now moved to it's usual DOS location at 409FH. 429CH is then called with the HL register pointing to the text. This call executes the DOS command as if you had typed it in from

DOS ready. After 429CH has finished its job, it returns, ready to start the recovery routine.

The recovery routine moves the stack to its previous location, and all of those "troublesome" memory areas are moved back to their original locations. The "traps" are cleaned, and the high and clear pointers reset to previous conditions.

The recovery is now complete, and control is passed back to the 4174H caller.

It may be interesting to note that even DEBUG can be hunted. To exit back to Basic, type G. Debug responds with address?? Just hit <ENTER> , and presto! Back in Basic! Exiting to DOS is achieved by typing the usual **Q**. DEBUG baits its own traps, so if you do exit to DOS, remember the recovery routine has not been activated, so 4411H & 4415H will contain 6500h.

This utility cannot be used to execute any routine or program that alters the high memory pointer. Such programs that install type ahead utilities, filters, etc. would be installed lower than 6500h, and will be wiped out by the recovery process!

As a final note, the hunting license granted here is not limited to one species only! Any small /CMD critters that use memory between 5200H - 6500H are mighty tasty too!

A) Backup a system disk with BASIC/CMD on it.
B) Enter the following patch to bypass password protection, so that BASIC/CMD may be copied:

PATCH *02 (ADD=4ED4,FIND=20,CHG=18)

C) At Dos ready, enter COPY BASIC/CMD TO LIBDVR/CMD
D) Enter the above patch reversing the FIND and CHG bytes.
E) Enter BASIC, pressing <ENTER> at the "How many files prompt"
F) Enter the following program, saving it intermittently.
G) RUN line 100 to check for data entry errors.
H) SAVE the basic program before running it!
I) Type LIBDVR <ENTER>

```
1 ' Lines 10 - 16 adjust some DCB (device control block)
2 ' information about LIDVR/CMD so it may be ap-
pended.
3 ' Remember! Just hit <ENTER> at the files prompt!
4 '
10 CLEAR 1000:DEFINT A-Z:RESTORE
11 C$=CHR$(2)+CHR$(2)+CHR$(&H4D)+
CHR$(&H61)
12 OPEN"R",1,"LIBDVR/CMD":
FIELD 1,255 AS A$,1 AS B$
13 J=PEEK(VARPTR(A$)+1)+
PEEK(VARPTR(A$)+2)*256
14 J=J-50:GET1,20:K=INSTR(A$,C$):K=K-1
15 POKE J+1,40:POKE J+8,K:
POKE J+12,PEEK(J+12)-1:CLOSE
16 '
17 'Lines 19 - 24 create the append data file.
18 '
19 OPEN"R",1,"LIB/APD":
FIELD 1,128 AS A$(0),128 AS A$(1)
20 B$(0)=STRING$(128,0):B$(1)=B$(0)
21 FORK=1TO3:FORJ=0TO1:
LSETB$(J)=STRING$(128,0)
22 FOR I=1 TO 128:READ C:IF C<>-1 THEN
MID$(B$(J),I,1) = CHR$(C):NEXT I,J
23 FOR J=0 TO 1:LSET A$(J)=B$(J):NEXT J:PUT 1,K
24 IF C<>-1THEN NEXT K
25 '
26 ' Now we append the LIB/APD file to the end of
LIBDVR/CMD
27 '
28 CLOSE:CMD"I","APPEND LIB/APD TO
LIBDVR/CMD"
29 '
30 '

50 DATA 1,5,148,98,195,0,128,1,5,217,100,195,8,
128,1,130,0

51DATA 128,245,62,1,50,67,128,24,6,245,62,
2,50,67,128,241

52 DATA 245,197,213,229,221,229,253,229,42,
164,64,34,180,128

53 DATA 229,35,1,73,1,9,34,164,64,42,116,
65,34,190,128,225

54 DATA 34,116,65,205,83,128,58,67,128,254,
1,204,77,27,253

55 DATA 225,221,225,225,209,193,62,0,254,1,
40,7,241,33,240

56 DATA 99,195,179,91,241,195,25,26,33,182,
128,237,75,180,128

57 DATA 175,237,66,229,193,221,33,146,128,
221,110,0,221,102

58 DATA 1,124,183,40,24,229,253,225,253,110,
0,253,102,1,175

59 DATA 237,66,253,117,0,253,116,1,221,1,
130,128,128,35,221

60 DATA 35,24,222,33,182,128,237,91,180,128,1,
73,1,237,176

61 DATA 201,242,128,58,129,64,129,68,129,
71,129,80,129,111
```

```
62 DATA 129,134,129,140,129,166,129,170,
129,176,129,182,129

63 DATA 210,129,227,129,232,129,0,0,0,0,245,
126,254,34,40,4

64 DATA 241,195,0,0,35,126,254,33,40,3,43,
24,243,241,229,42

65 DATA 232,64,237,91,253,64,175,237,82,17,
100,20,223,225,218

66 DATA 122,25,17,37,66,35,6,60,126,254,34,40,
8,18,35,19,16

67 DATA 246,195,151,25,35,34,242,129,175,18,6,
3,33,37,66,126
 68 DATA 254,36,40,8,1,130,0,129,35,16,248,62,
13,18,24,46,33

69 DATA 37,66,205,13,38,213,221,225,26,254,0,
40,9,254,60,56

70 DATA 9,30,28,195,162,25,30,8,24,249,79,
221,35,221,110,0

71 DATA 221,102,1,17,37,66,6,0,237,176,62,13,
18,42,17,68,34

72 DATA 236,129,42,21,68,34,238,129,237,
115,240,129,42,240

73 DATA 129,17,100,20,175,237,82,34,244,129,
235,33,128,64,1

 74 DATA 32,0,237,176,33,0,65,1,0,1,237,176,33,0,
82,1,1,19,237

75 DATA 176,33,48,64,17,246,129,14,3,237,176,33,
9,68,14,4,237

76 DATA 176,62,195,243,50,1,129,128,129,48,64,
50,9,68,33,234

77 DATA 129,34,49,64,33,231,129,34,10,68,33,
0,101,34,17,68

78 DATA 34,21,68,33,159,64,249,33,37,66,251,
205,156,66,42,240

79 DATA 129,249,42,236,129,34,17,68,42,238,
129,34,21,68,42

80 DATA 244,129,243,17,128,64,1,32,0,237,176,17,
0,65,1,0,1

81 DATA 237,176,17,0,82,1,1,19,237,176,33,
246,129,17,48,64

82 DATA 14,3,237,176,17,9,68,14,4,237,176,42,
242,129,251,201

83 DATA 205,249,129,24,185,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

84 DATA 0,0,0,2,2,77,97,-1

85 '
86 '
87 ' check your data file entry, run line 100
88 '
100 PRINT"Your checksums:":RESTORE:T!=0: FOR
K=1TO 100:J=0
101 FOR I=1 TO 30:READ C:IF C=-1 THEN 103 ELSE
J=J+C:NEXT I
102 PRINT USING" ######";J;:T!=T!+J:NEXT K
103 PRINT"Total:"T!:PRINT:
PRINT"Correct checksums:"
104 PRINTUSING" ######";2430;3731;3905;4287;
3947;3561;3741;
105 PRINTUSING" ######";3769;2395;2399;2549;
3245;2490;2684;
106 PRINTUSING" ######";2644;3114;2092;2070:
PRINT"Total:55053"
107 STOP
```

The usual BASIC parameters are supported by LIBDRV. eg:

LIBDVR -F:3V -M:60000 PROGRAM/BAS

LIBDVR PROGRAM/BAS

LIBDVR

The command **LIBDVR *** should not be used, as the new appendage will overlay any basic program in memory. Besides, if you need to enter a DOS command, use the new **CMD"!** enhancement!

Please note that **BASIC *** is ok, but the enhancement will not work.

Examples below show a few ways to skin a DOS.

```
CMD"!FREE:2"                          - text method
A$="DIR :0 (INV,SYS)":CMD"!A$"  - string method
LINE INPUT A$:CMD"!A$"                - user input
READ C$:J4$(I)=C$:NEXT:INPUT K:CMD"!J4$(K)"
                                      - string select
```

Until next issue, have fun bagging your limit!

Your comments, ideas or questions regarding TRSDOS 1.3 may be sent to Gary Campbell, Suite 209, 1051 KLO Road, Kelowna, British Columbia, Canada. V1Y 4X6

Editors note: The Basic program should be used only on BASIC/CMD revision 1.3. dated May 1980 or later. Revision 1.2. of BASIC/CMD is full of bugs and will not work properly.

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

All parms above are installed using a new LIBRARY command **SYSTEM (parm,parm).** Other new LIB options include **DBSIDE** (enables double sided drive 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!

FREE WITH YOUR ORDER!

The first 50 customers will receive 2 Disks Full of Choice Model III Public Domain Programs.

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

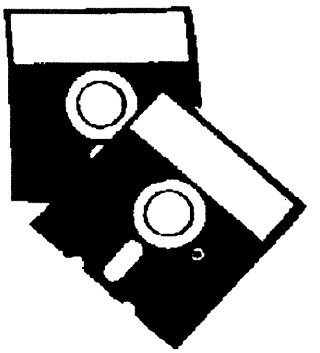
Attention SYSTEM 1.5.

FREE WITH YOUR ORDER!

CP/M - The Alternate DOS for Model 4

DISK AND DIRECTORY FORMATS

by Roy T. Beck



I remember way back in the days when NEWDOS2.1 was new, an occasion when Harv Pennington came to the OCTUG and explained the directory structure of TRSDOS 2.1. Not having disk drives then (I was still using cassette tapes), his lecture sailed over my head at about 10,000 feet above MSL (My Sensibility Level?). He did a very good job of presenting the ins and outs of the mysterious disk sectors by means of flip charts, but I really got very little out of it. Fortunately, fragments of his lecture did adhere to the lobes of my cerebellum, and when I later equipped my Model I with 2 drives (sigh, so long ago), I soon became fascinated with the GAT, the HIT, the PFDE, etc. With Harv's book "TRS-80 Disk & Other Mysteries", plus the fragments of his lecture rattling around in my brain, plus SuperZap, courtesy of Apparat in Denver, Colorado, I soon became a full fledged byte zapper, only embarrassing myself occasionally!

Having thus explored TRSDOS V2.1, 2.3, and now 6.X (see my article on copy limited disks in issue V1N6), it is high time to delve into the corresponding areas of CP/M. *YES*, CP/M also has a directory structure, but it is simpler than TRSDOS by far. That's not surprising, given the age of CP/M; it predates almost everything! GAT sector? *Nope*. HIT sector? *Nope*, more's the pity, as the HIT sector is truly an inspired feature of TRSDOS. Too bad IBMDOS doesn't have it. But of course, IBMDOS was derived from CP/M, so I guess that's one area where they followed a previous bad decision. I (or someone else) will explain the philosophy and beauty of the HIT table at some later date. Suffice it for now to say that it is a powerful tool, and neither CP/ M nor IBMDOS uses it.

CP/M has essentially no passwords, either, another area where TRSDOS shines. There is a feature in CP/M called the *User number* which functions to a slight degree as the beginnings of a subdirectory system. In this respect, CP/M has a slight edge over TRSDOS, which has no subdirectories in its basic arrangement. MISOSYS' DISKdisk does overcome that lack to a great degree, but is essentially an add-on to TRSDOS. IBMDOS really shines in the matter of subdirectories, and deserves full credit for them, even though first time users (*strugglers?*) may curse them.

One of the weak areas of CP/M is the fact that Digital Research never made an attempt to define any standard disk formats other than the 8" SS SD. The result is that every implementer of CP/M went off on his own, and we now have dozens of disk formats for different machines and different disk sizes. It would take too much of my limited time to attempt to gather and collate all of the different formats, and a duplication of effort to boot, as Monte has already done most of this. He had to in order to create the disk format interchange program with its more than 100 different formats.

Instead, the intent of this article is to explain the configuration of our "standard" MM formats as implemented in our current version 2.32 of MM CP/M. I will begin with the SS DD System disk as received from MM. There are three areas of interest on this disk, namely the system (including the Boot), the directory, and the data storage area.

By the way, when CP/M was new, the sector was almost invariably 128 bytes in length. CP/M also uses 128 bytes as the size of a logical record. Oldtimers in CP/M used the term "sector" and "record" interchangeably, and this confusion of terms still exists, to some extent. It is important to understand that CP/M always handles data in 128 byte logical records internally, but has a mechanism for blocking and deblocking logical records into sector size blocks, 256 bytes in our case. Remember that the disk drive must read or write a full sector each time; the disk controller cannot read or write more or less than one full sector at a time. Therefore, every sector on our system disk is 256 bytes long and contains two logical records. This is an important point.

The system area begins on track 0, sector 1, and extends through track 1, sector 18, on a SS disk. The first sector is, of necessity, the Boot sector, because this is the sector the ROM of either a Mod 4 or a Mod 4P will read into memory. On the SS DD MM system disk, the sectors are 256 bytes in length. The Boot sector is, of course, the actual loader for the CP/M system. The remainder of the system tracks and sectors contain the rest of the BIOS, the CCP, and the BDOS. This is all limited to 9K in length,

because there are two SS tracks of 18 sectors, each containing 256 bytes.

The next significant area of the disk is the directory which allows DOS to find the files stored in the data area of the disk. The rest of this article will be devoted to the structure of the directory.

The final portion of the disk is the data storage area where all files except the actual CP/M system files are stored. The DOS disk has 170K of storage space for files, but since the system disk, as delivered, contains numerous transient programs, there is essentially no space for additional user files on this disk. Of course, if you have double sided drives, you can prepare a double sided DOS. This will increase your data storage space by about 180K, making the machine more flexible. Of course the system disk can also be tailored to your specific needs by deleting unnecessary files, and this will free up some data space on the system (DOS) disk. Since it is possible to replace the Mod 4 and 4P drives with double sided drives, and to install 4 half height drives in a Mod 4 or 4D, there is plenty of opportunity to increase the available data storage space in our machines.

Now, on to the MM SS DD directory structure! The SS DD MM system disk has 3 directory sectors in use, shown in a *Super Utility* printout below. Before proceeding further, I had better point out the CP/M practice on interleaving sectors. Most DOSes interleave sectors in order to provide sufficient time for the CPU to handle the data from one sector before the next one comes around. In most other DOSes the interleaving is accomplished at format time, by placing the sectors around the disk in a pattern which insures sufficient rotational time between reading or writing successsive sectors of a given file. CP/M is different in this respect. The sectors are formatted around the disk in numerical order from 1 to whatever. In order to provide the necessary time between successive sectors' appearance at the read/write head, a sector interleave is established and controlled by a table within the BIOS. If you notice the sector numbers at the left edge of the three sectors shown, you will see they are in the order 1,3,5. These are the physical sector numbers, as recorded in the data address marks (DAM). This indicates the logical order of reading the physical sectors is:

1,3,5,7,9,11,13,15,17,2,4,6,8,10,12,14,16,18

While this interleaving can be ignored when reading the directory, it is crucial to reading the sectors of a file in the correct order.

The CP/M directory consists of a group of 16 sectors (32 logical records of 128 bytes each) starting at track 2, sector 1. Each directory entry consists of 32 bytes, for a total of 8 entries per sector. The first byte of each entry serves a dual purpose. If it is 00h through 0Fh, it is the User number of the file. If it is E5H, then the directory entry is either blank or killed. (Because CP/M only resets this one byte when it kills a file, it is easy to recover a killed file as long as the actual file sectors have not been written over by some other file. All TRSDOS' except V1.3 for the Mod 3 follow this same practice). The next 11 bytes of the directory entry hold the filespec, with 8 bytes for the filename and 3 for the extension. Byte #13 is supposed to hold the "extent" number, but the entries for MDM730.COM and MDM730.DOC in the second sector image both contain 01 in this position, and there appears not to be a 00 extent. I don't understand this. Explanation of bytes #14 and #15 will be deferred to a later article. Byte #16 indicates the number of logical (128 byte) records actually used in this extent.

Each of the last 16 bytes points to an "allocation block" on the disk. These bytes correspond to the similar bytes of a TRSDOS file entry, except that each entry in the CP/M directory identifies one allocation block. The TRS entry points to a track and sector location on the disk and tells how many contiguous sectors contain portions of the file. CP/M only identifies the block, as all blocks are of equal length. By having the capability of pointing to as many as 16 different portions of a disk, CP/M can reuse allocation blocks which become available anywhere on a disk as the result of killing off of old files. Further, CP/M normally can have additional extents in the case of fragmented large files scattered all over a disk. When a second or higher extent is needed, byte 13 of the next extent is set one digit higher than the previous extent in order to identify all the extents of a file and to show the order of usage. The anomalies in the MDM730 files make me wonder if Monte followed this convention.

When examining the directory of a TRSDOS disk (except V1.3), we are accustomed to seeing a multiplicity of ..../SYS files, which we know are the TRSDOS system files. But there are never any SYS files in a CP/M directory. This is because CP/M has only one system file, and the loader knows where to find it by track and sector number. Also, the BIOS knows where to look for the directory, so there is no BOOT/SYS or DIR/SYS file entry, either.

I previously mentioned the allocation block, but it deserves more attention. In CP/M, the allocation block corresponds to the gran as used in TRS DOSes. In most CP/M's an allocation block is identified by only a single byte, and therefore the highest allocation block number is FFH (255D). (There can be exceptions to this, see "Inside CP/M" referenced at the end of this article.)

We will consider now only the SS DD MM system disk. Since two tracks are devoted to the system file, the remaining available space is 38 tracks of 18 sectors each, which works out to 684 sectors or 1368 logical records. Since the largest allocation byte is 255, and there are two logical records per sector, the ratio of 1368/256 is about 5.33, and the next larger whole number is 6. This means each allocation byte must hold 6 logical records, at a minimum. But CP/M can only have allocation blocks ranging in size from 1K to 16K in powers of 2, so Monte chose 2K for us.

But now we need to examine the implications of byte #16, which shows how many logical records are actually used in the extent of which it is a part. I worked through all the logical record counts and all the corresponding allocation block assignments , and confirmed that Monte has assigned 16 logical records to each allocation block on the system disk. 16 records of 128 bytes each thus gives 2K bytes per allocation block, which agrees with the 2K choice. Of course, 2K is the minimum space assignment for a file, even if the file contains only one byte. Note also that allocation blocks 00 and 01 are not shown in any of the various extents. This is because the directory itself occupies allocation block #00 and 01.

I began this essay with the idea that I could easily discuss the whole directory in one article. But here I am, out of space and with much more to say. I wish to acknowledge the valuable information I acquired by study of the excellent manual Monte supplied with our CP/M. That is "Inside CP/M, A Guide for Users and Programmers" by David E. Cortesi. In addition, I perused an article by Gene Cotton which appeared in Interface Age for December of 1981, which was also very useful.

To be continued.....

CP/M directory sector dump as printed by Super Utility

```
#     00#0041 534D 2020 2020 2043 4F4D 0000 0040#.ASM      COM...@
HEX   10#0203 0405 0000 0000 0000 0000 0000 0000#................
DRV   20#0043 4F4E 4649 4720 2043 4F4D 0000 0065#.CONFIG   COM...E
 0    30#0607 0809 0A0B 0C00 0000 0000 0000 0000#................
TRK   40#0044 4454 2020 2020 2043 4F4D 0000 0026#.DDT      COM...&
 02   50#0D0E 0F00 0000 0000 0000 0000 0000 0000#................
TRU   60#0044 4953 4B20 2020 2046 4446 0000 0051#.DISK     FDF...Q
 02   70#1011 1213 144E 0000 0000 0000 0000 0000#.....N..........
SEC   80#0044 554D 5020 2020 2041 534D 0000 0022#.DUMP     ASM..."
 01   90#1516 1700 0000 0000 0000 0000 0000 0000#................
STD   A0#0044 554D 5020 2020 2043 4F4D 0000 0004#.DUMP     COM....
ODD   B0#1800 0000 0000 0000 0000 0000 0000 0000#................
      C0#0044 5550 2020 2020 2043 4F4D 0000 003F#.DUP      COM...?
      D0#191A 1B1C 0000 0000 0000 0000 0000 0000#................
      E0#0045 4420 2020 2020 2043 4F4D 0000 0034#.ED       COM...4
+00   F0#1D1E 1F20 0000 0000 0000 0000 0000 0000#... ............

#     00#0045 5842 494F 5320 2043 4F4D 0000 0002#.EXBIOS   COM....
HEX   10#2100 0000 0000 0000 0000 0000 0000 0000#!................
DRV   20#004B 4559 4445 4620 2043 4F4D 0000 0006#.KEYDEF   COM....
 0    30#2200 0000 0000 0000 0000 0000 0000 0000#"................
TRK   40#004C 4F41 4420 2020 2043 4F4D 0000 000E#.LOAD     COM....
 02   50#2300 0000 0000 0000 0000 0000 0000 0000##................
TRU   60#004D 4449 5220 2020 2043 4F4D 0000 0018#.MDIR     COM....
 02   70#2425 0000 0000 0000 0000 0000 0000 0000#$%................
SEC   80#004D 4449 5220 2020 2044 4F43 0000 000F#.MDIR     DOC....
 03   90#2600 0000 0000 0000 0000 0000 0000 0000#&................
STD   A0#004D 444D 3733 3020 2043 4F4D 0100 0012#.MDM730   COM....
ODD   B0#2728 292A 2B2C 2D2E 2F30 0000 0000 0000#'()*+,-./0.......
      C0#004D 444D 3733 3020 2044 4F43 0100 001E#.MDM730   DOC....
      D0#3132 3334 3536 3738 393A 0000 0000 0000#123456789:.......
      E0#004D 4F56 4350 4D20 2043 4F4D 0000 0062#.MOVCPM   COM...B
+00   F0#3B3C 3D3E 3F40 4100 0000 0000 0000 0000#;<=>?@A..........

      00#0050 4950 2020 2020 2043 4F4D 0000 003A#.PIP      COM...:
HEX   10#4243 4445 0000 0000 0000 0000 0000 0000#BCDE............
DRV   20#0052 4550 4F52 5420 2042 5547 0000 0011#.REPORT   BUG....
 0    30#4647 0000 0000 0000 0000 0000 0000 0000#FG................
TRK   40#0053 5441 5420 2020 2043 4F4D 0000 0029#.STAT     COM...)
 02   50#4849 4A00 0000 0000 0000 0000 0000 0000#HIJ...............
TRU   60#0053 5542 4D49 5420 2043 4F4D 0000 000A#.SUBMIT   COM....
 02   70#4B00 0000 0000 0000 0000 0000 0000 0000#K................
SEC   80#0053 5953 4745 4E20 2043 4F4D 0000 000C#.SYSGEN   COM....
 05   90#4C00 0000 0000 0000 0000 0000 0000 0000#L................
STD   A0#0058 5355 4220 2020 2043 4F4D 0000 0006#.XSUB     COM....
ODD   B0#4D00 0000 0000 0000 0000 0000 0000 0000#M................
      C0#E5E5 E5E5 E5E5 E5E5 E5E5 E5E5 E5E5 E5E5####################
      D0#E5E5 E5E5 E5E5 E5E5 E5E5 E5E5 E5E5 E5E5####################
      E0#E5E5 E5E5 E5E5 E5E5 E5E5 E5E5 E5E5 E5E5####################
+00   F0#E5E5 E5E5 E5E5 E5E5 E5E5 E5E5 E5E5 E5E5####################
```

CLOSE #2

Another issue under our belts. Where does the time go? I am constantly amazed by how little is done by the time we are supposed to be finished. It certainly is a good thing there are quiet hours after midnight or this issue would be rather thin. Once again, however, TRSTimes is 36 pages, full of information and fun for your Model I, III and 4.

Dr. Michael Ecker debuts a semi-regular column featuring 'fun' math programs. Even if you are of the group who never enjoyed math, give this a chance. You might even learn something.

Ben Mesander gives us another installment for Model I - NEWDOS/80. This one is a nice printer utility for a STAR NX-80. If I am not mistaken, the STAR printers are Epson compatible, so this one will be very useful.

Roy Beck shares his experience with changing his single sided drive to double sided drives. You can get a few pointers from this one.

Shell Basic for Mod I, III & 4, by yours truly, is the first of, hopefully, many sets of subroutines to come in future issues. It does slow execution time down a tiny bit, but it certainly is a convenient way to program.

Danny Mullen makes sure that Model 4 also has a printer utility. This one is for NX-10 (also Epson compatible). Check it out. It is nice to be able to set the printer directly from DOS.

The Hints & Tips installment is a collection of items that were just too short to qualify as individual articles, but good enough to share with the TRS-80 community.

Assembly 101, also by yours truly, is part 2 of our Assembly language tutorial for beginners. If you have always been mystified by this language, come on aboard and get familiarized.

Gary Campbell has done it again. This time he has modified TRSDOS 1.3. Basic into *enhanced* Basic. Now you'll be able to do DOS commands in Basic without losing your variables. Incidentally, Gary has tentatively promised a *full-screen Basic editor* for next issue. More miracles, indeed.

Finally, Roy Beck is in full swing exploring the CP/M disks and their directory formats. He presents a very interesting part 1 in this issue. By the way, he explored using Super Utility. Think about it!

A big TRSTimes thanks to all the contributors for making this issue possible.

Until May.....STAY TUNED!!