THE MISOSYS QUARTERLY

In this issue:

MISOSYS Products: Virtually Bug Free!

Volume Vi.iii $10 Spring 92

PRICE LIST effective March 1, 1992

TRS-80 Software

Product Nomenclature	Mod III	Mod 4	Price	S&H
AFM: Auto File Manager data base	P-50-310	n/a	$49.95	D
BackRest for hard drives	P-12-244	P-12-244	$34.95	
BASIC/S Compiler System	P-20-010	n/a	$29.95	B
BSORT / BSORT4	L-32-200	L-32-210	$14.95	
CP/M (MM) Hard Disk Drivers		**H-MM-???**	**$29.95**	**B**
CON80Z / PRO-CON80Z.	M-30-033	M-31-033	$19.95	
diskDISK / LS-diskDISK	L-35-211	L-35-212	$29.95	
DISK NOTES from TMQ (per issue)			$10.00	
DoubleDuty		M-02-231	$49.95	
DSM51 / DSM4	L-35-204	L-35-205	$49.95	
DSMBLR / PRO-DUCE	M-30-053	M-31-053	$29.95	
EDAS / PRO-CREATE	M-20-082	M-21-082	$44.95	D
EnhComp / PRO-EnhComp	M-20-072	M-21-072	$59.95	D
Filters: Combined I & II	L-32-053	n/a	$19.95	B
GO:Maintenance	n/a	M-33-100	$39.95	B
GO:System Enhancement	n/a	M-33-200	$39.95	B
GO:Utility	n/a	M-33-300	$39.95	B
Hardware Interface Kit	n/a	M-12-110	$24.95	
HartFORTH/PRO-HartFORTH	M-20-071	M-21-071	$39.95	B
LDOS 5.1.4 User Manual	L-40-020	n/a	$15.00	D
LDOS/LSDOS Reference Manual	M-40-060	M-40-060	$30.00	B
LDOS/LSDOS BASIC Reference Manual	M-40-061	M-40-061	$25.00	A
LDOS 5.3.1 Diskette - M1	M-10-110	n/a	$15.00	
LDOS 5.3.1 Mod1 Upgrade kit	M-10-133	n/a	$39.95	B
LDOS 5.3.1 Diskette - M3	M-10-130		$15.00	
LDOS 5.3.1 Mod3 Upgrade Kit	M-10-333	same	$39.95	B
LED / LS-LED	L-30-020	L-30-021	$19.95	
LB Data Manager-M4 (Ver 2.2)	n/a	M-50-510	$99.00	D
LS-DOS 6.3.1 Upgrade Kit - M4	n/a	M-11-043	$39.95	B
LS-DOS 6.3.1 Diskette - M4	n/a	M-11-243	$15.00	
LS-DOS 6.3.1 Upgrade kit - M2/12/16		M-11-002	$39.95	B
LS-Host/Term	n/a	L-35-281	$39.95	
LS-UTILITY	n/a	L-32-150	$24.95	
MC / PRO-MC	M-20-064	M-21-064	$79.95	D
Mister ED	n/a	M-51-028	$39.95	B
MRAS / PRO-MRAS	M-20-083	M-21-083	$59.95	D
PowerDot (Epson or Tandy)	P-32-21?	n/a	$19.95	
PowerDraw	P-32-220	n/a	$19.95	
PowerDriver Plus (Epson).	P-50-200	P-50-200	$17.95	
PowerMail Plus	P-50-003	P-50-004	$39.95	D
PowerMail Plus TextMerge	P-50-100	P-50-100	$15.00	
PowerScript	P-50-142	P-50-142	$24.95	B
PRO-WAM	n/a	M-51-025	$74.95	D
PRO-WAM Toolkit	n/a	M-51-225	$29.95	B
Programmer's Guide DOS 6.	n/a	M-60-060	$20.00	B
QuizMaster	L-51-500	n/a	$19.95	
RATFOR-M4		M-21-073	$59.95	D
RSHARD - R/S HD driver	M-12-013	same	$29.95	
ST80-III	P-35-300	n/a	$39.95	B
SuperUtilityPlus	P-32-132	P-32-104	$44.95	D
SuperUtilityPlus CMD file diskette	P-32-832	P-32-804	$20.00	
Supreme HD Driver (PowerSoft-RS)	P-12-113	P-12-113	$34.95	
TBA / LS-TBA	L-21-010	L-21-011	$19.95	D
THE SOURCE 3-Volume Set	n/a	L-60-020	$40.00	D
Toolbox/Toolbelt	P-32-203	P-32-245	$24.95	B
UNREL-T80	same	M-30-054	$29.95	
UTILITY-I	L-32-070	n/a	$19.95	
XLR8er Softawre Interface Kit (M3 mode)		M-12-X10	$20.00	B

MSDOS Software

LB Data Manager 2.2	M-86-510	$99.00	D
DED-86 [Disk/Memory sector editor]	M-86-020	$29.95	D
RATFOR-86	M-86-073	$59.95	D
HartFORTH-86	M-86-071	$59.95	D
SAID-86 [Text Editor]	M-86-040	$29.95	
Super Utility PC	P-86-407	$29.95	B
TRSCROSS (transfer <> Mod III/4	P-86-212	$89.95	B
FM-86 (File Manager)	L-86-050	$29.95	
Lair of the Dragon	M-86-021	$19.95	

TRS-80 Game Programs

Bouncezoids (M3)	M-55-GCB	$14.95
Crazy Painter (M3)	M-55-GCP	$14.95
Frogger (M3)	M-55-GCF	$14.95
Kim Watt's Hits (M3)	P-55-GKW	$9.95
Lair of the Dragon (M3/M4)	M-55-021	$19.95
Lance Miklus' Hits (M3)	P-55-GLM	$19.95
Leo Cristopherson's (M3)	P-55-GLC	$14.95
Scarfman (M3)	M-55-GCS	$14.95
Space Castle (M3)	M-55-GCC	$14.95
The Gobbling Box (M3/M4)	M-55-020	$19.95

MSDOS Game Programs

Lair of the Dragon	M-86-021	$19.95

Hardware

Floppy Disk Controller M3/M4	H-MM-FDC	$45.00	F
Double Density Controller (DDC) M1	H-MM-DDC	$45.00	F
RS232 Serial Card M3/M4	H-MM-SPC	$45.00	F
RS232 Serial Card Kit M3/M4	H-MM-SPK	$50.00	F
TeleTrends TT512P modem (M4P)	H-4P-512	$74.95	E
Floppy drives (5.25" 360K 1/2 ht)	**H-FD-360**	**$75.00**	**D**
Floppy drives (3.5" 720K 1/2 ht)	**H-FD-720**	**$85.00**	**B**
Floppy Drive Case (2-1/2 ht drives)	H-FD-2SV	$60.00	F
MSCSI HD kit e/w clock, 20Meg M3/M4	**H-HD-020**	**$450.00**	**?**
MSCSI HD kit e/w clock, 40Meg M3/M4	**H-HD-040**	**$575.00**	**?**
Aerocomp HD - 5 Meg M3/M4	**H-MM-005**	**$250.00**	**?**
Aerocomp HD - 20 Meg M3/M4	**H-MM-020**	**$400.00**	**?**
Aerocomp HD - 40 Meg M3/M4	**H-MM-040**	**$500.00**	**?**
Hard drive joystick port option	H-HD-JSO	$20.00	
Hard drive: Seagate ST225 (20M)	R-HD-020	$200.00	F
Hard drive: Seagate ST251-1 (40M)	R-HD-040	$320.00	F
Hard drive: Seagate ST-157N (SCSI)	R-HD-S40	$300.00	F
Cable: dual floppy extender	H-FD-2EX	$18.00	
Cable: 4Ft floppy (1 34EDC each end)	H-FD-C04	$12.50	
Cable: 4Ft M3/M4 printer	H-RC-PM4	$20.00	
Cable: 4Ft Radio Shack hard drive	H-HD-CT4	$20.00	
Cable: 4Ft MISOSYS hard drive	H-HD-C04	$22.50	
Cable: 26-1069 internal floppy	H-FD-2NG	$20.00	
Cable: 26-1069A/26-1080 internal floppy	H-FD-2GA	$20.00	
Cable: 26-1080/A internal floppy	H-FD-24P	$20.00	
Cable: drive power Y	H-HD-CPY	$5.00	
Cable: XT hard drive set	H-HD-CXT	$5.00	
Cable: Custom IDC ribbon (M3/M4/M2)	?-??-???	varies	
Standby Power System: 200VA	**R-PS-200**	**$199.00**	**?**
Standby Power System: 450VA	**R-PS-450**	**$399.00**	**?**
HD Controller: Adaptec 4010A	H-HD-CA4	$75.00	D
HD Controller: Xebec S1421A	H-HD-CX2	$75.00	D
T80 to SCSI host adaptor	H-HD-MHA	$75.00	D
ZOFAX 96/24 **Fax/Modem** (PC XT/AT)	R-Z1-FAX	$125.00	G
Infochip Systems Expanz! (PC)	R-IC-EXP	$150.00	G
DJ10 Tape Backup (PC)	R-TD-D10	$199.00	G
DJ20 Tape Backup (PC)	R-TD-D20	$265.00	G
AB10 Tape Adaptor (PC)	R-TD-A10	$75.00	D
KE10 External tape adaptor/case (PC)	R-TD-K10	$110.00	F
Tadiran TL-5296 AT 6V lithium battery	R-PB-TL6	$19.95	B

The Fine Print

Freight codes: A = $3.50; B = $4.00; C = $4.50; D = $5.00; E = $5.50; F = $6.00; G = $7.00; H = $12.00; ? = varies; All unmarked are $3.00 each; Canada/Mexico add $1 per order; Foreign use US rates times 3 for air shipment. Virginia residents add 4.5% sales tax. We accept MasterCard and VISA; Checks must be drawn on a US bank. COD's are cash, money order, or certified check; add $4 for COD.

MISOSYS, Inc.
P.O. Box 239, Sterling, VA 20167-0239
703-450-4181; Orders only: 800-MISOSYS

The MISOSYS Quarterly is a publication of MISOSYS, Inc., PO Box 239, Sterling, VA 22170-0239, 703-450-4181.

THE MISOSYS QUARTERLY

subscription rate information

Each issue of TMQ has information on MISOSYS products, programs and utilities, patches, significant messages from our CompuServe forum, and articles on programming. Not only that, TMQ will keep you up to date with information, news, and announcements concerning our entire product line and related machine environments. Subscription cost varies by rate zone as follows:

A = $25; United States via 3rd class bulk mail
B = $30; Canada, Mexico, via 1st Class
C = $32; Colombia, Venezuela, Central America via AO Air
D = $35; South America, Europe, & North Africa via AO Air
E = $40; Asia, Australia, Africa, Middle East via AO Air

TMQ Toolbox

The MISOSYS Quarterly is published using the following facilities:

The hardware used to produce the "camera ready" copy consists of an AST Premium/386 computer (20 MHz) with 9 Megabytes of RAM, a Seagate ST4096 80M HD, ST251 40M, Expanz! card; a CMS DJ10 tape backup, a NEC MultisyncII monitor driven by a Video Seven VGA card, an AST TurboScan scanner (Microtek MS300), and a NEC LC-890 PostScript laser printer.

Text is developed, edited, spell-checked, and draft formatted using Microsoft WINWORD Version 1.1; Submissions on paper and letters are scanned and converted to text using ReadRight optical character recognition software by OCR Systems. Final page composition is developed using PageMaker 4.0 by Aldus.

Table of Contents

List of Advertisors

List of Patches in this Issue

TMQ to Continue

You may have noticed that my last issue of TMQ, VI.ii, took on a new look. I dropped the glossy "kromekoat" cover, went to a black separately affixed perfect binding, and migrated to a colored cover stock printed in black. The change was required as a way to seriously reduce production costs. That did the trick. With these costs in line with the subscription levels, there is no reason to not continue publishing. For the same price of a subscription as it has been the past six years, look forward to TMQ Volume VII. And if you want to make me feel good about this decision, please get your renewal in early!

Points to Ponder

Remember the DRAM shortage of a few years ago? I recollect the problem as being caused by a few factors: shut down of 256K production lines to bring up 1 meg lines which were slow to come on line at high yields; a volcanic eruption in Japan which closed down some fabs; and downright hoarding. But the one thing I remember most is the prediction that in the future, computers will be small potatoes when it comes to DRAM use. That was supposed to mean that the computer industry would not be in the best position to bargain for production quantities for their purposes.

We seem to be pretty smug in the thought that RAMs are used primarily in computers, right? Not necessarily so; consumer electronics has been making tremendous inroads into consumption of memory chips. About the time of the Winter Olympic games, I came across a news item which brings that *small potatoes* prediction closer to reality.

The Blurb

by Roy Soltoff

Canon has recently developed a high-speed Hi-Vision video camera/recorder unit that has the ability to pick up images up to three times faster than conventional cameras. Instead of recording at 60 frames a second, it records at 180 frames a second. The underlying design choice for the camera was to better able to deal with stop motion recording/playback of actions which take less than a sixtieth of a second, such as instant slow-motion replay of the Olympics field events in high density television (HDTV). The Canon unit uses 15,000 4-Mbit DRAMs to store digitally, 5,400 still frames, or 30 seconds worth of viewing at high definition. Take a single 64K TRS-80, and that one camera is equivalent to 480,000 computers!

The Canon recorder can perform instant random access of digital frames, real time slow motion, variable-speed reproduction of images, and can simultaneously convert real-time images picked up at three-times speed into regular-speed Hi-Vision signals.

Canon's video camera/recorder is not the only one available. NEC developed last year an HDTV solid-state recorder using 5,000 4-Mbit DRAMs. That records 17 seconds of pictures, but can be extended to 51 seconds by stacking three memory units.

Much of the special effects capabilities appearing in recent top-of-the-line VHS-type video recorders as well as Laser Disk players is attributed to the inclusion of massive amounts of memory. The sheer size of consumption of DRAMs for the video industry will simply dwarf the usage in computers - as computer memory requirements in the *modern world* begin to approach a need of 8-10 megabytes per machine.

In previous issues, this column has remarked on the increase in density of hard drive storage capacities. In the quest for ever-greater rapid-access data storage, or the converse, lighter and more power-efficient storage, drive capacity has had tremendous increases, yet some manufacturers have shrunk the size to minuscule proportions.

Maxtor, for instance, has joined the ranks of the gigabyte+ drives with it's MXT-1240 - a 1.24 Gbyte 3.5" drive which has a spindle rotational speed of 6300 rpm, and average seek rate of 8.5 msec, and a data transfer rate of 40-plus Mbps. Host transfer rate using SCSI-2 is 10 Mbytes per second.

A company called Cambex Corp. (where are these guys coming from?) offers a 2.1-Gbyte 5.25" drive with 11-ms seek; the spindle rotates at a measly 5,400 rpm. No, this drive is not designed for your TRS-80, but rather for high-end work-stations where 300x300 dpi 4-color images can consume up to 32-Mbytes of disk storage.

Meanwhile, Areal's (wasn't that a mermaid?) 2.5" drive sports 91.5 megabytes formatted, weighs 3.4 ounces, stands but 11.7mm high, and uses a glass disk. This is a junior-sized version of their dual-platter 180-Mbyte 2.5" drive announced last fall.

You know, when Winchester disk drives first appeared on the scene, the drive size was, I believe, 30 megabytes in an 8" form factor. Then came 5.25" drives - a *standard* for quite some time. These drives

will be soon totally outdated as 3.5" drive capacities are shooting through the roof. But the low-power and small weight needs of laptops, have driven the push to reduced drive sizes. I thought that the 2.5" drive was going to fill that need, but it seems that before 2.5's became cost-effective, the size shrank some more (shades of, "Honey, I shrunk the drives").

There were start-ups trying out a 1.8" form factor. Now Hewlett Packard is showing a 1.3" drive with a measly capacity of 20 Mbytes. That drive is about the distance from the tip of my thumb to the first knuckle. Where's the use targeted? Why palmtops and calculators, that's where!

Trade-in Policy

The relatively new policy for trade-ins of an equivalent non-MISOSYS software product for a MISOSYS software product is to just send in an original **Table of Contents** page with the trade-in fee which is 50% of the price of our product. So for LB 2.2, trade in any other database product and you can purchase LB or LB-86 for $49.50 plus S&H. How's that for a deal?

Corrections to last issue

As you will read in the *Letters* column, a few snags crept into the last issue. The biggest blunder was that the text of Matthew Reed's *SYSFLEX* article was prematurely truncated! That text is re-printed in this issue in its entirety.

The last sentence of Matthew Reed's PRO-WAM Help Displayer article was truncated. It should read, "Assemble HELP/APP as a core image file (use the -GC and -CI switches with MRAS) *and add it to your PRO-WAM application library with WAMLIB*." The italicized portion is what was omitted.

In both of the cases, the DISK NOTES had the complete text.

In this issue...

I was initially planning on making this a C-language focus issue. I have been saving for re-print, an update of an old six-part series originally written by Earl "C" Terwilliger for the old *LSI Journal*, along with a series of programs by Rich Deglin which introduce an environment facility to LS-DOS. But it seems that the graphics-related articles rolled on in, to the point (no pun intended) that I shifted gears. So the apparent focus in TMQ VI.iii is high resolution graphics.

I start with my LBFILL, which has nothing to do with graphics, but solves a thorn Australia's Ivan Kennedy was having after converting to LB from Profile. I then re-print Matthew Reed's SYSFLEX article which was clobbered in the last issue.

I soon jump into hi-res graphics, though, with the introduction of Jonathan Armstrong's POINT article which also illustrates the techniques of optimizing code at the assembly-language level. This is followed by *Monochrome Graphics Programming* by Hans de Wolf. Wolf's article covers many of the graphics formats in use, as well as provides a program for exporting images to other formats. Finally, hi-res graphics fans will welcome Frank Slinkman's native-mode high-resolution graphics library for PRO-MC. C-language programmers now have direct access to great low-level functions.

Now for some requests. Since TMQ has at least five more issues to run, In the past, a few product reviews have been contributed. Programs which perform a specific purpose have also been submitted, as well as techniques which could be used to perform some job. I would now like to also see some articles of another vein submitted for publication. Charles Ainsworth presented an excellent article demonstrating his use of our Disk/Sort/Merge product. I know there are many users who utilize PRO-WAM to a great degree in their overall computer use. That topic should prove useful to many others. You will read in this issue of the LB database templates provided by Herman Hardin. With all of the LB users out there, it is advantageous for each of you to share your database structures as templates. Help out a fellow user. I'll be making them available on disk form as well as via our CompuServe forum.

TMQ Schedule

I try to **target** mailing *THE MISOSYS QUARTERLY* approximately every three months. The last issue was late. This issue should be mailed less then three months since the issue VI.ii was mailed, so I'm hanging in there - time-wise.

Note that your mailing label usually has the expiration date of your subscription. For instance, **those with "92/05" complete their subscription with this issue.** The renewal fee to continue with the next issue is 25% of the subscription rate shown on page 1. If you want to re-subscribe for the next volume, then the cost is 120% of the yearly fee (I'll give you a 5% discount) at this point in time.

MISOSYS Forum

I sponsor a forum on CompuServe. You can reach some "experts" on TRS-80 and MS-DOS subjects by dialing in, then GO PCS49, or GO LDOS.

The forum contains many programs to download, as well as lively discussions which thread through the message system. It no longer contains the listings from *Programmer's Journal*. You can direct a message to me at 70140,310. Post a message in private if you don't want it "broadcast"; some folks even send me orders via a PRIVATE message.

DISK NOTES 6.3

Each issue of *THE MISOSYS QUARTERLY* contains program listings, patch listings, and other references to files we have placed onto a disk. DISK NOTES 6.3 corresponds to this issue of TMQ. If you want to obtain the patches and the listings, you may conveniently purchase a copy of DISK NOTES priced at $10 **Plus S&H.** The S&H charges are $2 for US, Canada, and Mexico, $3 elsewhere.

PD Software Librarian

Vic McClung has apparently left the TRS-80 environment and is no longer functioning as a librarian for TRS-80 public domain diskettes. Public domain and shareware programs for the TRS-80 are available from Computer News-80 and TRSTimes, among other sources such as TRS-80 Bulletin Boards, as well as our own CompuServe forum, PCS-49. p.s. this column will now be dropped!

LB Data Manager 2.2.1

In the last issue of TMQ, I announced the release of LB 2.2.0. That version included a database conversion utility to create an LB database directly from any of the following: pfsFILE4, Profile4 (or Profile III+), DIF, dBASE II, dBASE III, and fixed fielded records. It also directly generates the following types of database files or output from an LB database: DIF, dBASE II, dBASE III, tab delimited, comma delimited, and ASCII strings.

Other enhancements within 2.2.0 include a ***Dupe*** command used to find potential duplicate records; the Find command displays the second key name if an index is active which uses more than one field and the primary key is A, B, L, or U; the Print module's output is a line buffer to allow a TAB-to-a-column control in print definition screens; the Define screen sub-menu can automatically generate an Edit/Update/Delete screen; and one more editing key is added - HOME the cursor to the start of the field being edited.

Since that release, I had a request from one LB user to support parentheses in calculated fields; seems he wanted to perform a calculation of the form: -F1*F2+F1/(1-F3)/. It sounded like a reasonable request; I proceeded to implement it so I could pass him a copy. Next, Richard King asked for a few enhancements: a way to force LB to wait after each page so he could handle single sheet forms; a way to suppress the printing of blank lines; and to have PRINT DEFINITION prompt for another print file rather than immediately return to the main menu.

With the above changes, plus one more small fix in the sort module of LB86, I groomed a 2.2.1 version. The change to a line buffer in 2.2.0 wound up causing a problem with imbedded print control codes. This was partially fixed in 2.2.1 - almost fully with a workaround, and will be fully fixed in the next release; however, I would like to groom in some additional features. As a for instance, I will be adding a command to switch on/off the automatic date updating. I use that field in my customer data base to indicate the date of the last order. But then I cannot edit the record without altering that date. So I will add a means of toggling the auto-update feature so I can alter fields without the last update date changing. Perhaps existing LB users also have some suggestions for improvements; please submit them to me.

This brings up a problem. In the MS-DOS world, some software houses have introduced changes into the normal production of a product's version; the term *slipstreamming* has been applied. But the term usually applies to the introduction of the change without announcement; I'm announcing my changes. The problem here at MISOSYS is how to best distribute the changes. Until I come up with a better plan, for now, the only way is via a disk refresh. I may explore other methods; however, since LB is a large number of separately compiled modules, coming up with a set of patch changes may be too cumbersome to deal with.

If you are an existing LB 2.1 user and wish to move up to the new release 2.2.1, or an LB 2.2.0 user who needs the small changes, return your existing two disks with $10 for the update. Existing LB 1.0 users can still upgrade to version 2.2 by sending in the Table of Contents page from the *LB User Manual* with the $40 upgrade fee plus appropriate shipping.

International LS-DOS 6.3

Since the release of LS-DOS 6.3.0 by LSI, folks have requested international versions. In the last issue, I finally noted the availability of both French and German versions of LS-DOS 6.3.1 (F & G). To date, only two copies have been ordered! So I'll put out the notice once again. If you have purchased a 6.3 version in the past, you need to order only a replacement disk - but you must specify which version you want - F or G. The cost is $15 plus S&H. If you are starting with 6.2 or earlier, order the upgrade kit for $39.95 + S&H - or perhaps a new combined L(S)DOS x.3 Reference Manual plus a disk.

DOS and BASIC Reference Manuals

Here's the announcement of two new reference manuals now available from MISOSYS. First, we have the the new 349-page "LDOS™ & LS-DOS™ Reference Manual", catalog number M-40-060. This single manual fully-documents both LDOS 5.3.1 and LS-DOS 6.3.1 in a convenient 8.5" by 5.5" format. If you use one, or the other, or even both DOS versions, you may want to bring yourself up to date with a single manual. Gone are the many pages of update documentation. Price is $30 plus $5 S&H.

Also newly published is the "LDOS™ & LS-DOS™ BASIC Reference Manual". This 344-page book, catalog M-40-061, covers the interpreter BASIC which is bundled with LDOS 5.3.1 (even the ROM BASIC portion), the interpreter BASIC which is bundled with LS-DOS 6.3.1, and both Model I/III-mode and Model 4-mode EnhComp compiler BASIC. One convenient 8.5" by 5.5" manual covers all four BASIC implementations for $25 plus $3.5 S&H. Since this new manual covers our compiler BASIC, you can purchase the disk version of EnhComp for $23.98 plus $1.50 S&H when purchased along with a BASIC Reference Manual, or the disk version by itself for $29.98 plus $3 S&H if purchased separately.

MS-DOS Products

MISOSYS is a reseller of products purchased from Ingram Micro; thus, we have access to a huge array of MS-DOS products. So if you are looking for some hardware or software to go with your MS-DOS system, why not get in touch with us for a quote. Call, write, or FAX.

ZOFAX FAX/Modem

Speaking of faxes, can the cost of a FAX get any lower? Why waste time and money communicating through voice or mail when a FAX gets you down to business. A one-page letter can usually be faxed in less than a minute. And you also get a complete 2400 baud modem along with your 9600 baud Group III send/receive fax - with both BitFax and Bitcom software. You can't afford not to take advantage of this low cost to modern day communicating!

PC-AT Lithium Batteries

When was the last time you replaced the battery in your MS-DOS computer? Don't wait until you lose your configuration data. Original batteries last anywhere from a year to several years. We stock Tadiran TL-5293/W 6V universal lithium batteries which last years and years. And it fits into more machines than any other battery. Models from Acer, Amdek, AST, AT&T, Club, Dell, Epson, Everex, Goldstar, HP, IBM, IDS, ITT, Mitac, NEC, Novell, Samsung, Sharp, Tandon, Tandy, Victor, Wang, Wyse, and more. Only $19.95 + $4S&H.

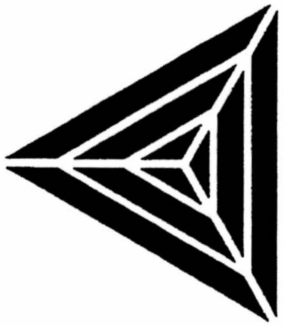
Address Change

The United States Postal Service has seen fit to change the ZIP code of our Post Office Box (mailing address) effective July 1, 1992. The new ZIP is 20167. Please update your records; I don't want to lose any of your orders.

FAX Number

If you want to reach us by fax, try 703-450-4213.

Letters to MISOSYS



Model 3/4 parts

Fm: Robert Hengstebeck: Roy, is it possible to have both CP/M and LS-DOS on the same HD, for the Model IV. From TANDY's owner's manual, it appears that you must have the two operating systems on different HDs. I have the 5 Meg HD bubble, with the tandy 2000 10Meg HD installed in place of the 5 Meg.

Fm MISOSYS, Inc: If you have a Radio Shack HD, you can have CP/M, LDOS, and LS-DOS all on the same hard drive provided that you are partitioning strictly by head. That means using TRSHD6/DCT and TRSHD3 or 5/DCT or RSHARD using head partitioning. I sell the Montezuma Micro CP/M drivers. The documentation which comes with those drivers is very specific as to the procedure to run CP/M along with LDOS or LS-DOS.

As a matter of fact, I have a 15M R/S drive sent to me by Aerocomp which uses four heads for CP/M (drives A: - D:) and two heads for LDOS.

LSDOS: Upgrade versus Replacement

Fm Bob Cunningham: Roy, I recently acquired a Model 4P and ordered LS-DOS 6.3.1 Upgrade Kit for it since it had the DOS that came with the machine back in the mid-80's. Out of curiosity, what is the difference between the 6.3.0 to 6.3.1 replacement package and the Upgrade Kit? I realize the price is different, but what else is different? I am looking forward to receiving it and seeing the changes.

Fm MISOSYS, Inc: The 6.3 upgrade kit includes a 40-page booklet which supplements the TRSDOS 6.2 User Manual. The 6.3.1 diskette is just a disk with a single sheet covering the improvements in 6.3.1 over 6.3.0. Also, I have a royalty obligation to pay on upgrade kits; however, since a 6.3.1 disk is supposed to go to a 6.3.0 owner where the royalty was previously paid, I do not have a downstream obligation. That is the essential difference.

But look for some pricing changes. Now that I have my new combined DOS reference manual covering L(S)DOS x.3.1; and a combined BASIC reference manual, you may want to obtain new manuals and a disk instead of the "Upgrade Kit". Each reference manual is approximately 350 pages.

800-MISOSYS

Fm Allan Higgins: I'm in Canada. U.S. 800 numbers don't usually cross the border. Do you have a local number?

Fm MISOSYS, Inc: Our 800-647-6797 (800-MISOSYS) is usable in the 50 United States plus CANADA - I pay extra for Canadian in-dialing support. I don't provide international 800 service which is considerably more costly, but also available from 50 or so countries into the United States. But if you need my non-800 number, its 703-450-4181. FAX is 703-450-4213 - but the FAX is not always up. I use that same number for modem com-

munications.

Note that some international users of AT&T international long distance can call US 800 numbers by an international gateway service. From time to time, I get calls from Great Britain that way. I believe the caller pays a lower international rate as it uses strictly AT&T facilities; I pick up some of the call charges on my 800 bill which covers the domestic US-portion of the 800 call. That service is different from international 800 service where the called party pays for the entire call.

TRS-80 PD Software

Fm Bob Cunningham: Roy, See page 6 of TMQ Vol VI.i. I wrote to Vic McClung a few weeks ago making inquiry regarding TRS-80 PD software. His answer: "Sorry, Bob, but I haven't had any TRS-80's in years!"

Did you get some bad information somehow? Is there an alternate person in charge of TRS-80 PD software? Thanks for the help.

Fm MISOSYS, Inc: It's not that I got bad information; apparently he never notified me that he no longer took care of that function. It was he who originally approached me. Guess I'll drop his name from *The MISOSYS Quarterly*. I have no idea what he did with any of the programs he collected from users.

At this point, you may want to check with CN80 or TRSTimes. They should have collections of public domain and shareware available.

ARC4V3 Upload

Fm David Huelsmann: Version 03.00.00 of the ARC4 utilities uploaded as ARC4V3.CMD to LIB 0 of the LDOS forum (go LDOS) and LIB 3 of the TRS80PRO (go pcs-21) forum. Enhancements include squashing and desquashing, storing only, better compression rates when crunching over ARC4ADD, enhanced verbose listing routine, and elimination of the ability to add any SYS files in a wildcard selection. Speed improvements of about 10% were seen as well. Self-extracting archive - no complicated bootstrap routines necessary. You will need about 214K of free space on target drive to fully extract. Takes about 14 minutes or so to download using CIS-B at 2400bps.

ARCTOOLS.CMD Upload

Fm David Huelsmann: I recently uploaded ARCTOO.CMD (ARCTOOLS.CMD) to LIB 0 of the LDOS forum (go LDOS). This self-extracting archive needs about 151K free on target drive to fully extract. The archive contains docs and updated versions of MARC4 (Merge archives, XARC4 (Extract all files from one or multiple archives) and SWEEP4 (clean-up loose files already contained in an archive). Primary change was to add support for self-extracting archives though there were minor bug fixes to MARC4 and XARC4.

Last TMQ

Fm Daniel L. Srebnick: I am sure someone else has already pointed this out, but what happened to the end of the SYSFLEX article. It isn't in my copy, nor a friends? It appears to be a sort of ALTRES utility, but I'm not sure.

Fm MISOSYS, Inc: Actually, I don't know how the rest of the article got to be missing. For those who are really interested, the whole article was put on the DISK NOTES, and I put the missing piece (or the whole article) into this issue. What SYSFLEX allows you to do is to use a small one-bank MemDISK to put some system modules in. The SYSFLEX module then provides an alternate means for the DOS to check some drive other than :0 for additional system modules not capable of being loaded into the one-bank RAM drive. SYSFLEX doesn't load system modules into memory, but provides essentially an alternate drive :0 strictly for system module access.

Fm Lawrence Rossiter, Canada: Dear Roy, I dread life without *The MISOSYS Quarterly* but fully understand your reasons, I don't know anyone else who owns a TRS-80 so I can't help find subscribers.

Back to my 'sharp-eye' tricks: pages 2 to 5 in Vol VI.ii were titled "Vol V.iv" and "Summer 1991"! Also Matthew Reed's article ended in mid-sentence on page 40!

Oh well, yesterday we went out for a couple of hours and I left my front door wide open and the keys in the lock! All the best to you and the family.

Fm MISOSYS, Inc: I need a better proofreader, someone other than myself. Here's what happened with *The Blurb*. Obviously from the title page, I prepare *The MISOSYS Quarterly* using Pagemaker. Recent issues are typically three sections: *Blurb*, *Letters*, and *Inside TMQ*. The title page is part of *The Blurb*, but I prepare it separately since it is a totally different construction. Thus, the title page, which I actually refer to as TOC (table of contents), is a separate file. Pagemaker maintains headers and footers on master pages. When I prepare an issue, I load the previous file, delete all old text and graphics, revise the header, then add the new text. I used to use a master template, but then I still had to revise the headers and footers on each section - so it sort of was six of one and a half dozen of the other. In last issue's *Blurb*, I forgot to revise the header master page. That's why it printed the wrong information. And I never caught it.

The good news is that you will have many more opportunities to catch my snafus as the production costs of the new format allow me to economically print TMQ at lower volumes. So for now, *The MISOSYS Quarterly* goes at least another year to Volume VII.

FORMAT1/FIX in TMQ VI.ii

Fm Dan Yertzell: Hello, Roy. I just received the latest TMQ, and was surprised to find a patch to Model 1 LDOS 5.3.1 to allow DSDD boot disks. I have made my own DSDD boot disks, but it is a rather cumbersome process and the resulting disk cannot be copied. So I applied the FORMAT1/FIX patch as described on page 9 and tried a FORMAT (SYSTEM) command. It went into an endless loop after reformatting track 0 as single density.

There is an error in the published patch. The line after "Call new code (the patch)" should read "D01,DE=CD 63 67" instead of "D01,DE=CD63 61". The error is fairly obvious, since the following line says "New code located at X'6763'".

With this correction, DSDD boot disks are finally easy to create! So now I'm running LDOS 5.3.1 on a 20 meg HD divided into 7 partitions (using RSHARD5 patched by me for Model 1 use), and a single DSDD 360K floppy drive. The Model 1 lives on!

As they used to say, "Thanks For Your Support".

Fm MISOSYS, Inc: Seems like I was all thumbs with that issue. Pages 2-5 have the wrong heading; Matt Reed's article was not continued on another page; and his PRO-WAM article on page 30 omitted a portion of the last sentence.

But the good news is that with the reduced production cost of the "new" binding style, I will be able to continue to publish after issue VI.iv.

USTAT in MC

Fm David Huelsmann: Roy, I may have stumbled on either a definition error or one of those unpublished limits that I sometimes run up against. In the latest version of MC, a function ustat() was provided. Within the structure ustat defined in ustat/h is daddr_t f_tfree which is defined as the free blocks in file system. Looking in types/h I find that daddr_t is typedef as an int. For a hard drive, I understand the limit on any one partition to be about 13.3 megs. This is about 53,200 standard blocks of 256 bytes. With f_tfree defined as an int, the function could only return up to 32,767 blocks.

Hopefully, this is just a declaration error in the header file and f_tfree should really have been an *unsigned int*. ARC4E relies on this function to determine the amount of free space on available drives and returns a negative value for free space of slightly more than 8,000,000 which would be consistent with an int that had exceeded its limits and gone negative.

Would appreciate any feedback you would care to offer.

Fm MISOSYS, Inc: Okay, I dug into that one. It took a while to explore the details, but here is the easy solution. The ustat() function is written in assembly language. Both the number of free blocks (f_tfree) and the number of allocated blocks (f_talloc) are derived from the results obtained from RAMDIR function 255. This service call returns free and allocated space in units of K (1024 bytes). The standard block size is 256 so these figures are multiplied by 4 to obtain the actual values in standard block units.

The maximum size of any logical drive under L(S)DOS is 203 cylinders times 256 sectors per cylinder which is equal to 51968 standard blocks. This maximum number easily fits into a 16-bit field; thus it is correctly calculated and stored into the ustat data structure by ustat(). Of course, this needs to be referred to as an *unsigned int*. But as you pointed out, the header file uses a declaration of *int* via the typedef of daddr_t. The description of daddr_t is the type of a disk address. This really should be an unsigned int. As the only external use of that is in referencing ustat() data, simply change the declaration in the types/h header to unsigned int. A scan of the high-level-written library functions revealed no other function using daddr_t, so it should be safe to simply change your header; I do not need to recompile any library function.

MC's floor() has a "ceiling"

Fm Frank Slinkman: Roy: compile and run this, and let me know what you think

```
/ *      limit/ccc       */
#include        <stdio.h>
#include        <math.h>
main()
{       double  f;
        f = 32767.5;
        tryit(f);
        f = 50000.5;
        tryit(f);
        f = 65535.5;
        tryit(f);
        printf( "format is 12.2;
so where's the left
padding?\n\n" );
}
tryit( f )
double  f;
{       double  a, b, c;
        a = f * f;
        b = dfix(a);
        c = floor(a);
        printf( "number is %12.2f
<- is this aligned?\n", f );
        printf( "square is
%12.2f\n", a );
        printf( "  dfix is %12.2f
<- any non-zeros to right of
decimal?\n", b
);
        printf( " floor is %12.2f
<- any non-zeros to right of
decimal?\n\n",
c );
}
```

Fm MISOSYS, Inc: Frank, One of your problem with printf formatting is a "local" one. The program uses a specifier of "12.2f" to print out the numbers. The "32767.50" is small enough to print into a 12-character field. Note that the "12" portion of the specifier is the **field width** and not the number of characters to the left of the decimal point (as it would be in FORTRAN). When the larger number was printed, 1073709056.25, that required a 13-character field. The MC manual states, "if the converted value has more characters than the width, then it (the width) is adjusted to hold the full result." MC's printf() did exactly as it was supposed to do. When your program printed out the smaller number, it printed in a 12-char field; the larger numbers were printed in a 13-character field. That's why the numbers were not right-aligned as you expected!

As far as why floor() and fix() did not perform as expected, the maximum value you can take the fix() or floor() of and get a "valid" result is 1 less than 2**31 (i.e. 2 raised to the 31st power less one); that's 2,147,483,646. The second and third numbers were larger than that. The dfix() and floor() functions use long integers to deal with the numeric handlings.Longs only support a magnitude maximum of the above-mentioned number. Short of using extended longs (which MC doesn't support), it would take quite a huge special purpose routine to deal with the dfix() and dfloor() of a larger number, as a total fix would require a specialized assembler function to truncate the fractional part and allow seven bytes of significance. We'll just have to live with that restriction.

If you really need the ability to handle larger numbers, you could write a high-level function to perform the algorithm. Simply compare the number to the maximum identified above. If equal or smaller, take the floor(). If larger, repeatedly subtract off the maximum until the result is less than the maximum, keeping track of how many times you subtracted. Then take the floor of the result and add back the value of the subtracted amount. You could also divide the number by the maximum (once initially comparing the number to the maximum), take the floor of that result as the multiple; subtract off the multiple times the maximum from the original number, then take the floor of the result; finally adding back the multiple of the maximum. In all cases, you need to store the sign of the number, use the absolute value for your calculations, then negate the result if the sign was negative.

As an aside, I did determine that the Microsoft C compiler version 5.1 does correctly handle that number size in floor(). MSC does not provide a dfix() function.

Lastly, if the number is larger than 7.20576E16, you won't have a fractional part as that would exceed the significance of storage in a 7-byte characteristic. Any fractional part would be noise.

As a final, note to readers, subsequent to my preparation of this material for *The MISOSYS Quarterly*, Frank posted the limit/ccc program to me on the LDOS forum (Cumpuserve PCS-49). It appears to follow one of my suggested workarounds quite nicely. See the next page.

Aerocomp Hard Drives

Fm Lloyd Evans: About six months before AeroComp shut down, I bought three 5 meg drives from them. Their Model III/4 drivers were the pits. I had to hand construct a DCT with only 152 tracks to get anything reliable. If you have corrected this with a patch, please send me a copy. If it took a complete rewrite, let me know what I need to do to get a copy.

Fm MISOSYS, Inc: Since acquiring the Aerocomp hardware, I haven't discovered any significant problems with the software. There's a minor change needed to the Model III-mode driver when installed on a Model 4 in III-mode using full lower RAM; but I touched on that already in a previous issue of *The MISOSYS Quarterly* in response to Jane Layman's query. She also raised another slight issue with the Model 4-mode driver when installed into low-memory; I touch on that in this issue next. I have no intentions on re-writing their software.

Most of my frustration is with the lack of documentation, a problem I am slowly dealing with. The hardware itself proved

```
/* limit/ccc
 *
 * fixit() performs accurate truncation of doubles when
 * the value is < -2147483648 or > 2147483647, the limits
 * for a long integer. It's a little slow, but it works.
 * Fm: Frank Slinkman 72411,650
 */

#option INLIB
#include        <stdio.h>
#include        <math.h>
#define MAXFILES        3
#define REDIRECT        0

main( argc, argv )
int             argc;
char    *argv[];
{       double  number;
        if ( argc == 2 )
                number = atod( argv[1] );
        else
        {       puts( "usage: limit number" );
                exit(1);
        }
        tryit( number );
}

tryit( number )
double number;
{       double  b, c, d;

        b = dfix( number );
        c = floor( number );
        d = ceil( number );

        printf( "\tnumber is %13.2f\n\n", number );

        printf( " un-fixed dfix is %13.2f\n", b );
        fixit( &b );
        printf( "    fixed dfix is %13.2f\n\n", b );

        printf( "un-fixed floor is %13.2f\n", c );
        fixit( &c );
        printf( "   fixed floor is %13.2f\n\n", c );

        printf( " un-fixed ceil is %13.2f\n", d );
        fixit( &d );
        printf( "    fixed ceil is %13.2f\n", d );
}

fixit( value )
double *value;
{       double  factor, temp, limit = 2147483647.0;
        int             sign;

        sign = dsgn( *value );
        temp = fabs( *value );
        if ( temp <= limit ) return;
        *value = temp;
        ++limit;
        factor = ceil( temp / limit );
        temp = factor * floor( temp / factor);
        temp -= floor( temp - *value );
        while ( temp > *value ) -temp;
        *value = temp * sign;
}
```

quite reliable - once I got the proper parts. I was not supplied with everything needed to assemble the drives. It does take quite a level of labor to assemble a drive. You are aware of the preponderance of screws, nuts, spacers, standoffs, and lock washers. Aerocomp was apparently out of stock on those items as well as cooling fans, rubber feet, rear panel connectors, 50-pin ribbon cable and associated connectors, and AC power switches - including AC power cables. They had a very limited supply of AC sockets and other wiring harnesses; I had to procure them through my suppliers. I initially had trouble with the standoffs I acquired as I could not track down the original part numbers. I have subsequently obtained a different line of standoffs which should prove to be top-notch.

I don't think you can beat the Aerocomp case for durability - that case is rock solid. Of course, it added about two to three pounds of weight to a completed package. I never did like those pin header connectors used on rear panels - they are very problematic and quite susceptible to pin breakage. Since I was not supplied with any of them, I am using the SCSI (50-pin Centronics-type connector) which is far more reliable. This is the same type of connector I was using with the Leadman case. But the size of this connector requires me to nibble the rectangular mounting hole in the rear panel to accommodate the larger SCSI connector. However, the end product is better.

Because I have ample hardware (read as drive cases) to last me a lifetime, I have since converted my MISOSYS hard drive package to the Aerocomp case. This also allows me to add the status light panel giving the user additional feedback (i.e. power, ready, select, read, and write).

Fm Jane A. Layman: Dear Roy, This letter is to thank you for the patch to the LDOS version of the Aerocomp driver that enables its use with the XHIK utility package [see TMQ VI.ii, page 15]. It works just fine. I would have written

sooner, but I was waylaid by the flu and the Christmas holidays.

Since MISOSYS has acquired Aerocomp's remaining drives and software, I wanted to alert you to an annoying bug in the Model 4 version of the Aerocomp 20 meg driver, HD20SA4/DCT. (This one I managed to fix for myself - at least in my configuration file.) When the driver is installed in low memory, the header is set up incorrectly. Specifically, the address immediately after the initial JR instruction references the first FREE byte of used memory instead of the last byte used by the driver. Program's relying on this information cannot function properly.

For example, when Houde's Eramdisk is loaded into low memory after the Aerocomp driver, it will operate satisfactorily as a RAM drive but it can not find itself to disable or uninstall itself. LS-DOS' MEMORY command cannot list any installed modules names when they are placed in low memory after the Aerocomp driver, e.g., XLR8SET or ERAMDISK.

Now that you have the Aerocomp drives, I am in hopes you may find it a worthwhile project to develop better drivers. I'd send in an ecstatic review to CN80 to get the word out. I suspect Aerocomp's drivers were modeled on the original TRSHD3/DCT and never saw a major upgrade after that.

Again, thank you for the patch to the Aerocomp driver. I hope you find the information for the Model 4 version useful.

Fm MISOSYS, Inc: Jane, I appreciate your quest to find me new projects; however, re-writing the Aerocomp drivers is not one of the ones I plan to tackle. But I do have a more permanent solution to HD20SA4/DCT than the twiddling you have to do to the configuration file.

The driver I currently have is version 1.30 so I will confine my remarks to that one. The source code starting from address 3173H is as follows:

```
      PUSH    HL      ;Save low-
mem pointer
      LD      HL,1E6H ;The
driver length
      ADD     HL,DE   ;Added to
current low-mem
      PUSH    HL      ;Potential
new low-mem
      LD      BC,1300H
;See if driver fits low
      XOR     A       ;Clear CF
before subtract
      SBC     HL,BC   ;Subtract
max low from potential
      JR      NC,318CH
;Go if no room low
      LD      (HL),C  ;Update
low-mem pointer
      INC     L
      LD      (HL),B
      LD      (3415H),BC
;Stuff "last used"
```

The bug occurs because what is installed into the driver pointer is not the address of the last byte used by the driver, but the first free byte following the driver. That's why the value is one byte too high as you discovered.

What is needed is to add a decrement instruction immediately prior to the "stuff". The solution is an easy one, although at first glance it does not appear that one byte can be squeezed in there. Let's capitalize again on the use of Z80 flag contents after an instruction; I have mentioned this before in previous issues of *The MISOSYS Quarterly*. Here's where we separate the "men" from the "boys".

The purpose of the XOR A is to ensure that the carry flag is reset prior to the subtract with borrow (SBC) instruction; we don't want extraneous results interfering with the issue. However, if we examine the ADD HL,DE instruction in detail, we find that it sets or resets the carry flag based on the result of the ADD. So what are we adding? We are adding the driver length of less than 512D to a pointer value which should be lower than 1300H. The result is guaranteed to not overflow a 16-bit register; thus the result of that ADD will always result in resetting the carry flag! Thus, the XOR A coded to explicitly reset the carry flag is superfluous as the carry flag will always be reset by the ADD. Just another example of fine tuning your program. So I can re-write the above code fragment by eliminating the XOR A and inserting a DEC BC. The new code fragment is as follows:

```
      PUSH    HL      ;Save low-
mem pointer
      LD      HL,1E6H ;The
driver length
      ADD     HL,DE   ;Added to
current low-mem
      PUSH    HL      ;Potential
new low-mem
      LD      BC,1300H
;See if driver fits low
      SBC     HL,BC   ;Subtract
max low from potential
      JR      NC,318CH
;Go if no room low
      LD      (HL),C  ;Update
low-mem pointer
      INC     L
      LD      (HL),B
      DEC     BC
      LD      (3415H),BC
;Stuff "last used"
```

The associated FIX file to apply the correction is:

```
. HD20SA4A/FIX - Patch to
HD20SA4/DCT V1.30
. Corrects pointer to
last byte used by driver
. Apply via, PATCH
HD20SA4/DCT HD20SA4A
D01,8C=ED 42 C1 E1 30 09
71 2C 70 0B
F01,8C=AF ED 42 C1 E1 30
09 71 2C 70
. Eop
```

I took a peek at the other Model 4 drivers seemingly available and discovered some similarities and other bugaboos. Here's a list for other Aerocomp HD users; the "patch status" references the usability of the HD20SA4/FIX:

Driver	Vers	Patch status
HD20AD4	2.00	Same
HD5SSA4	1.20	Same code but patch address is D01,C5
HD5CAD4	1.20	Doesn't load low!
HD40SA4	1.20	Same code but patch address is D01,C9
HD40AD4	1.00	Loads low but omits the LD (nnnn),BC instruction and thus does not set the pointer; a much larger patch is needed!

The Aerocomp drivers are not modeled after the TRSHD3/DCT driver. To begin with, the Aerocomp drives use the SASI-type controllers; they used a Shugart controller of a type I am not familiar with (I have no specs on it),an Adaptec 4000, and later used a Western Digital WD-1002S-SHD (small card using surface mount components). I don't believe they used the WD controller with drives for LDOS or LSDOS as I have no driver for that controller in those DOS modes. There is a CP/M driver for the WD controller.

All SASI controllers have virtually identical command protocols for drive read/write access; the primary differences lie in the commands used for formatting. The early SASI controllers provided but a single "drive-format" command. This is one of the standard SASI commands - if you want to assume that "SASI" was a standard, which it wasn't. The WD 1000 controller used by Tandy uses a totally different register command set. Interestingly enough, since Western Digital became a leader in controllers for MS-DOS 286 machines (AT class), their register-command set became the standard for the so-called AT-style hard disk drives. This became known as IDE (integrated drive electronics) when the controller first became integrated directly onto the drive electronics board.

Most controller manufacturers added additional smarts or capabilities to their controllers in an attempt to establish themselves as a leader (added value). The Adaptec controller, for instance, used inter-record gaps on cylinder 0 to write configuration information; thus the Adaptec 4000 was able to read the drive on powerup and know the parameters of the drive, which it could then pass on to the host. Where the Xebec S1410 had to always be primed with configuration data every time you booted, the Adaptec could be up and running. In fact, since the S1410 HDC could only store the configuration for one drive at a time, if you were using two drives of different configuration (cylinders, heads, etc.), you had to upload to the controller the drive characteristics prior to accessing a drive if the prior access had been to the other drive. That's one reason why the VR Data driver (remember them) using a S1410 HDC was quite large; it was designed to handle two drives of any type. It had to keep track of the drive number being accessed. If you were reading from one and writing to the other, the configuration information had to be sent to the controller each time a different drive was accessed. That operation takes a little time - very small - but it also takes up code space. I restricted my VRHARD driver to supporting only two drives of the same configuration to minimize driver length.

Another improvement in controllers was in the ability of performing track-at-a-time formatting. But each manufacturer chose to implement that a little differently. Thus, if you wanted to provide the ability of formatting individual partitions separately - like what was available in TRSFORM using a Western Digital controller, your driver would be different.

Now my solution to that kind of problem would be to totally separate the driver code needed for formatting from the driver.That's why my drivers: VRHARD, RSHARD, M80HD, and MSCSI all have separate formatters. The drivers omit all code associated with formatting other than to trap the system format-drive or format-track commands to return an error code. This makes the driver a little smaller, and also a little more safe from a crashing program inadvertently branching to a resident format drive code fragment.

Aerocomp drivers are a little larger than mine because they supposedly take advantage of a feature in the SASI controllers called *bad track formatting*. When the controller is issued the format drive command, part of the information which can be sent to the controller is a bad-track table. Drives usually ship with a label listing these defective areas, if any (remember, drive units are not guaranteed to be defect free - only cylinder 0 must be defect free). The Aerocomp formatter, HDFORM, does have a parameter which can be used to input a bad-track list. The automatic BASIC program and JCL procedure does not use this capability. I don't know if they ever documented it, as I never saw specific documentation which was shipped with their drives. Nevertheless, the driver does set aside memory space for this table - 49 bytes to be exact.

So the end result is that Aerocomp drivers are designed around the SASI interface which uses a totally different command protocol and a totally different formatting command protocol than the TRSHD drivers.

As an aside, the command protocol relies on a command/status port and a data port and an ACK/NAK handshake. SASI also defines a number of signaling leads on the 50-pin bus. However, there is no standard at the host side as to what bits the SASI signalling leads are to be assigned. That's why there is no uniformity among all the third-party vendors of TRS-80 hard drives. My VRHARD driver won't work any other drive because either the port assignments or the status/signalling leads are assigned differently from the VR Data drive. My MSCSI driver won't work the Aerocomp drive because my driver/formatter is designed around the stored configuration data and the formatting con-

ventions of the Adaptec 4010 (different from the Adaptec 4000) and Xebec S1421 (different from the S1410) controllers. Also, the port assignments and status/signalling assignments on the Aerocomp host adaptor are different from my host adaptor. There's also another feature, pioneered by Lobo Drives, in adding a hardware automatic ACK to the interface. I also have that feature in my host adaptor. It enables a driver to utilize an OTIR or INIR instruction since the ACK, which must be passed after each byte is transferred, can be provided in hardware. The alternative is to use byte I/O (IN and OUT instructions). The auto repeat OTIR and INIR instructions are much faster.

This wound up as a huge response to your query, but I think it shed light on a subject many may be interested in.

LBASIC 4; SubDISKs

Fm John P. Jones: Dear Roy, I am commenting on your plans for LBASIC4 as outlined on page 22 of TMQ VI.ii. I agree as stated in my previous letter that memory is of the utmost importance. Using Model 3 BASIC with 4 files, there are 36603 bytes available. My HIMEM routines (including HD driver) reduces this to 32064 bytes. One of my longer programs loaded reduces this to 7914 bytes for variables, and this is with all remarks and spaces removed (they are very much needed for listings, but I wrote a M/L routine to remove then). This program must be loaded and compressed in two halves. No spaces, and as little memory loss as possible, please.

Regarding your "wondering" what pokes & peeks are used for, I use them for two basic purposes. First, I have a 20-byte buffer in HIMEM available to all programs to store such things a the following (examples only): 1) Information on printer codes, which differ from printer to printer (a menu program pokes the proper values for the printer on line). 2) Information on what programs have been used which tells us which files to backup at the end of the day. These could of course be stored to HD, but peek/poke is faster and I find it useful. Also, I sometimes pass variables to HIMEM for the M/L programs, although the main method is a string array with the USR passing the VARPTR (per "Basic Faster and Better"). I see no need to pass variables to DCB or FCB, as the M/L programs take care of this themselves.

Your diskDISK worked fine for me, but I had to use something less than the default values to allow backup to single-sided disks (because of the "overhead" on the floppies). The most convenient way is to reduce the default number of cylinders by 3-5.

Fm MISOSYS, Inc: John, It's true that you can't copy a standard-sized "/DSK" file directly to a floppy of the same configuration as the "/DSK" file has an additional configuration header sector. The way in which I back up my DiskDISKs or SubDISKs is to "mount" them to a logical drive via a SD or DD command, then DISKCOPY or QFB the DiskDISK logical drive to a floppy. In that way, I can create and use standard-sized 180K (5.25" 40D1), 360K (5.25" 40D2), or 720K (3.5" 80D2) subdisks (diskdisks) depending on the amount of capacity I need for the file collection.

Another LBASIC 4 supporter

Fm Herman Hardin: Dear Roy & family, I was finally was able to locate a Model 4P at a price that I could afford, what with the loss of our military cost of living allowance at the base where I am currently stationed.

Here is my vote for a multi-machine DOS manual and a Model III BASIC interpreter for the Model 4 in Model 4 mode. Having just purchased a 4P all the BASIC programs that I own or wrote run in my Model III that I have had since May of 1983, so if you do come out with LBASIC for the Model 4 put me dawn for one.

Enclosed you will find pfs:FILE and pfs:REPORT for the Model III as trade-in for LB 2.2.

Like I said. I am currently stationed overseas at Lajes Field, Azores. To find it look on a world map just below the 40øN line and on the 30øW line in the Atlantic Ocean and you will find the Azores. Lajes Field is located an the island of Terceira. I have been here with my family since Nov 1988, and we will be leaving Nov 1992. Have no idea where I will be next but will keep you informed when we move.

Fm MISOSYS, Inc: The DOS Reference Manual is available as well as a BASIC Reference Manual. The former covers LDOS 5.3.1 and LS-DOS 6.3.1. The latter covers Model III LDOS BASIC, Model 4 LS-DOS BASIC, and our EnhComp Compiler BASIC for Model III and 4 modes.

I started on the "LBASIC4" but it is currently on the shelf.

LB86 Sort Problem

Fm MISOSYS, Inc: Chris Mangelly, of Augusta reported a problem in being unable to sort a database with a particular number of records. I did locate the prob-

lem in the LBSORT module. Actually, it is something which I thought was fixed when version 2.0 was released, but apparently, under certain conditions, a memory conflict could still prevail. Here's what was happening, and it relates strictly to the MS-DOS version of LB.

The MS-DOS version of LB-86 1.0 was compiled using Computer Innovation's C86 - a C compiler available in the early stages of MS-DOS programming. LB-86 version 2.0 was compiled using Microsoft's MSC compiler; version 2.0 using MSC 4, and version 2.1 and above using MSC 5. One of the nagging irritants associated with moving from one compiler to another - even from moving from one compiler release to another - is the side affects introduced by using a different library of compiler-provided C functions. The LBSORT module of V2.x is virtually identical to V1.0, yet, I had detected a sorting memory problem when testing out V2.0. The problem was caused by the fact that MSC does not allocate a memory block for a file buffer when a file is opened via fopen(); the buffer is allocated only when the file is first accessed. I cannot suggest why that method is more efficient. Anyone have an idea?

LBSORT opens three files in sequence, an index file, a temp file, and a data file; but the data file is not accessed right away. The temp file is also not immediately accessed under certain sort specifications. LBSORT then determines the memory available, subtracts off a fudge factor to allow for stack usage and auto allocated variables, then determines how many passes the sort will require. If but one pass, no problem will develop. If more than one, the available memory is partitioned into two chunks: one to hold the pointers associated with the records of a given pass, and another to hold the sort strings of all records in the pass. Each pass-set of records is then sorted into a work file. Once all passes have been sorted, it is necessary to merge the passes into one index. In the merge process, the two chunks are freed and the original amount of free memory (less the fudge) is re-allocated to create a sort-string buffer for each pass another set of pointers. However, in the first allocation, the sum total of the two chunks can be less than the total memory available, depending on the number of records per pass and the sort string length. Since the data file's I/O buffer is not allocated by MSC until after these two chunks are allocated, that buffer immediately follows the allocation of the second chunk. That's what I caught before releasing LB-86 version 2.0, and solved the problem by allocating a dummy file buffer, then deallocating it after the data file is accessed. What I didn't catch is that the temp file is sometimes not accessed prior to the allocation of chunks; thus, I really needed to allocate two dummy file buffers.

This is now rectified in the 2.2.1 version of LB-86. If you stumble into this problem, simply get the more recent version.

LB 2.2.0 and imbedded print control codes

Fm MISOSYS, Inc: Mark Wygent, of Christiana, PA came upon a small problem in LB version 2.2.0. He had prepared a print definition file which used imbedded control codes to select such things as character font, bold facing, etc. His sample output print looked perfect using the print definition's output command, but when he printed out records, none of the control codes had any effect. I traced the problem to my oversight when I added the capability of a control TAB in version 2.2.0. Under LB versions 1.0 through 2.1, printed output was sent to the printer character by character. In the 2.2.0 release, I shifted to using an internal line buffer where a TAB repositions the line pointer. But since control codes are not actually output but are placed into the line buffer, the pointer must be advanced just as if a standard character was being output. Thus, any control code not at the end of a line was subsequently being overwritten by the next character. I have fixed that up in release 2.2.1.

Now this really caused another problem which I have yet to fix. LB allocates a line buffer of a size equal to the physical line length (plus one for the terminating NULL). But now since control codes actually take up space in the line buffer, a line which contains actual printed characters right-justified to the physical line length and which have control codes will overflow the line buffer. Plus, the logical line position for a tab needs to be adjusted by the quantity of control codes which have been generated up to that tab position. As a temporary work around, increase the physical characters per line by an amount equal to the maximum number of control codes you have in any line. And also do not use a TAB in the same print line as you have a control code. Since I am working on release 2.2.2 (more features), I can correct this problem by using a line buffer for the actual characters and a line buffer for control codes; the actual print function can then check both for needed character output.

LB's Null-terminated fields

Fm Dr. Ivan R. Kennedy, University of Sydney: Dear Roy, I'm afraid I can't agree that the Model 4 LB 2.1 switch from Model 4 graphics characters to reverse video lower-128 value characters on exiting (see TMQ VI.1.13) is benign. This interferes with my subsequent use of PROWAM. I have a series of 'hot-keys' (from PROWAM/KSM) that do things such as invoking DIALER (<CLEAR Yarn>),

TYPER (<CLEAR Qwerty>), BRINGUP (<CLEAR Journal>), PHRASE (<CLEAR Z>), etc. These make the use of my Model 4's super-efficient and I am lost without them. Can I have a patch?

A second problem relates to the use of LBDM. I use the following BASIC program to convert Profile 4+ data files to LBDM.

```
10 OPEN"R",1,"N2FIXNOO/KEY",251
20 FIELD 1, 65 AS SO, 6 AS S1,
60 AS S2, 60 AS S3, 60 AS S4
30 OPEN"R",2, "N2FIXN00/
DAT",251
35 FIELD 2, 20 AS S5, 20 AS S6,
20 AS S7, 30 AS S8, 50 AS S9,
50 AS S10, 50 AS S11, 11 AS S12
40 OPEN"R",3, "N2FIXN00/DA2",2
50 FIELD 3,2 AS S13
60 OPEN"O",4, "AUTO/JOB:0"
70 FOR L=1 TO LOF(1)
80 GET 1,L
90 PRINT#2,S0;CHR$(13);
100 PRlNT#2,S1;CHR$(13);
110 PRINT#2,S2;CHR$(13);
120 PRINT#2,S3;CHR$(13);
130 PRlNT#2,S4;CHR$(13);
140 PRINT#2,S5;CHR$(13 );
150 PRINT#2,S6;CHR$(13);
160 PRINT#2,S7;CHR$(13);
170 PRlNT#2,S8;CHR$(13);
180 PRlNT#2,S9;CHR$(13);
190 PRINT#2,S10;CHR$(13);
200 PRINT#2,S11;CHR$(13);
210 PRINT#2,S12;CHR$(13);
220 PRINT#2,CHR$(27);
250 NEXT L
260 PRINT#2,CHR$(26);
300 CLOSE:END
```

This program, based on an example you give in the LB86 Manual, provides an AUTO/JOB file that can be automatically entered into a filename/LB file for access by LBDM. This works fine. But unfortunately, because Profile 4+ uses hexadecimal 20 characters as spaces and also for data-empty areas in its fields, LBDM is unable to use the STRIP feature for fields so that firstname and lastname, for example, are not closed up where the lastname is less than the field length. I note that you use hexadecimal 00 characters for unfilled spaces (left Arrow) to the right of data left-justified in fields. Can you suggest a way to fix this, short of adding space (00) characters to each record manually? We plan to continue using Profile 4+ for a large database but to convert these to an AUTO/JOB file for LBDM, to run on an 80386 MS-DOS computer that will be used by colleagues in China.

On an unrelated point, I would prefer LBDM's cursor to go into fields when editing records on their left hand margin rather than the far right. Alternatively, if SHIFT LEFT ARROW took the cursor directly. to the left margin I would be satisfied. I find the fact that it is necessary to single step with the LEFT ARROW key rather tedious, since editing is usually performed from the right margin). Thanks for your attention.

Fm MISOSYS, Inc: Dear Ivan, I can understand your situation regarding the screen clearing problem. I still am not going to work up a patch; however, I am now ready to ship version 2.2.0 (Note: MISOSYS is already shipping 2.2.1). I am sending you a set of LB 2.2.0 disks with this letter. This version corrects not only the screen-clearing on exit, but adds some more useful features such as duplicate record discovery and automatic generation of edit screens..

Another new addition is a data base conversion utility which should be quite useful for you. It can directly convert a Profile database (as well as pfsFILE, dBASE2, dBASE3, and fixed format) to an LB database file. LBCONV also converts from LB to a few formats.

I was just about to begin releasing 2.2 when I got your letter just in the nick of time. Because of your *unrelated point*, I added another editing function to the LB editor. On a TRS-80, SHIFT-F1 will home the cursor to the first character of a field while editing; it's the HOME key on MSDOS machines.

As far as dealing with trailing spaces, the only thing I can offer is to massage the data fields with an external program. Perhaps I may consider that as a utility. A solution internal to LB's print module would be to change the behavior of STRIP. If STRIP is enabled, LB currently prints the designated maximum length or the field string length, whichever is smaller. Changing LB to use a field string up to the last non-blank character would have STRIP operate *properly* whether spaces or NULLs buffered the field. *Properly* is a questionable word as LB works properly according to its design! How that would affect program run-time is unknown; but at this point, I'm not going to do that. Now then, let me think about such a conversion utility for TMQ.

LBCONV, etc.

Fm Ivan R. Kennedy, University of Sydney: Dear Roy, I was delighted with your rapid response to my letter of December 28. My second letter (January 17) with diskette only served to emphasize some of the points I raised, since your initial reply (January 21, 1992) answers most of my concerns. But you could find it helpful to have the text on disk. I look forward to a utility able to STRIP Hex 20. Thanks for the HOME key in Edit. This ends up the very best solution, since both ends of the fields are readily available for editing or additions.

LBDM 2.2.0 is really starting to hum with power. It seems that I will be able to have my /DIF file output, and what looks to be an easier, automatic means of PROFILE4+ to LBDM conversion. The LB to/from dBASE3 will be very useful too. Great stuff!

Unfortunately, my PROFILE4+ files won't successfully convert with LBCONV. Some sample input and output for a small database BOOKS000) is on this disk, as well as another (WRC9000) that won't convert

to LB. I get the message "Source file read error" after initialization of the /LB file when data transfer should commence. This occurred both on the hard disk and on this floppy. I tried all my databases in PROFILE with the same result. My version of PROFILE4+ is patched to 01.01.07, but I don't think that would change the format of the files. Perhaps I'm making some elementary error, but I thought you would like to have a reaction as soon as possible. Other functions in LBCONV (<=>dBASE and /LB =>/DIF) work from my keyboard, at least on first examination.

PS. Could I suggest you consider improving the binding of the LB86 Manual, to better match the quality of the LBDM program? I can understand your desire to minimize costs of production and I'm generally happy with economy versions, but I find there's too many pages for the spine provided for ease of use. I fear the manuals will soon wear out, considering the use they're likely to get. (I appreciate that the many excellent HELP screens will delay this, but I intend to rebind our manuals somehow). While I'm in a critical mood (constructively, I hope), why do the filename/DEF files of LBDM often have rubbish from the disk in them? It doesn't seem to affect operation of the program in the least but it is mildly worrying.

Fm MISOSYS, Inc: Ivan, The error you were getting was due to a problem with the LBCONV utility program. Apparently, prior to releasing it, I must have broken the Profile to LB conversion after implementing the pfsFILE to LB conversion. I found a minor problem in the convert utility which has now been corrected. Apparently I was so intent on discovering the pfsFILE data structure and supporting it in the conversion program, I made some changes to get everything to fit; one minor edit was the culprit.

I have provided a replacement to LBCONV on the disk; please remove the old copy and use this one. I am sure you will get positive results. I've just finished my new LDOS/LSDOS Reference Manual, so perhaps I can spend a little time to whip up the space <-> NULL conversion utility. I want to try it as a stand-alone, i.e. not use any of my LB library functions, so I can use it as a TMQ article.

As far as the LB binding, here's what has happened. At the time of the release, I had not gotten in the larger 3/4" binding, so the early manuals were shipped with a 5/8" binding. The 3/4" binding is ample to avoid any problems. If you get into a "bind", let me know and I can get you a replacement manual.

Lastly, the "rubbish" you refer to is probably after the end-of-file mark. LB is generated from C; thus, it doesn't perform sector buffer zeroing prior to writing. What is most likely happening is that contents of RAM are part of the buffer. Actually, you will probably find that most programs have garbage in the portion of the last sector following the EOF. Programs generated strictly by my assembler usually don't as the assembler nulls the buffer prior to using it.

PRO-WAM CAL/APP

Fm Ivan R. Kennedy, University of Sydney: Dear Roy, Thanks very much for your very prompt attention to my problem re LBDM and the LBCONV facility. With the fix, it handles Profile 4+ conversion very well. I now vote LBDM clearly superior to Profile 4+, despite my liking for the latter. I will be saying so in an article I'm just about to put in *SYDTRUG NEWS*.

Here's a minor problem that you might just note until you're fixing up PROWAM later for dates beyond 2000. I found that the "*" flag on the CALENDAR date inserted from BRINGUP actually flagged the day previous to the activity from January 2, 1992, but only for two months until March, 1992. Since then, it's been behaving properly. A bug that goes away? Okay, I can live with that but only if it stays away. Note that I can still reproduce this effect by 'Add'ing after stepping back to this 2-month period, so it wasn't a system dating fault, as I thought at first. It may be in PRO-WAM.

A second matter. In using the LDOS 5.3.1 XLR8er Interface Kit with LDOS 5.3.0, I found that XRAMDISK prepares ramdisks as advertised. These are numbered, according to the drive number nominated (SYSTEM (DRIVE =n,DRIVER ="XRAMDISK")). I then tried to swap a ramdisk (with the system files backed-up to it) to be the system drive (SYSTEM (SWAP =3,DRIVE =0)), but the computer won't accept this and reboots. SYSTEM (SYSTEM =3) has the same result. This also happened with LDOS 5.3.0 as the operating system.

Neither the manual nor David Goben's comments specifically state that an xramdisk can be swapped to be drive 0, but I'd like this to be possible, since it is normal in LS-DOS 6.3.1 with an eramdisk. Do I have a local problem, or is this as it's designed to be? Must it be so? My use of LDOS 5.3.1 is not major, but I'd like to have it performing at its best.

Fm MISOSYS, Inc: Ivan, I tried out the XRAMDISK on my machine (Model 4D with XLR8er) in LDOS 5.3.1 mode. I had no trouble either installing the RAMDISK, or of using it as the SYSTEM drive. Worked like a champ. I suspect you have some local problem. You may want to try isolating it by starting with a fresh system disk.

I am also unable to reproduce your problem with PRO-WAM and CAL. I first suspected a problem in dealing with a leap year when you suggested the problem was localized to January and February of 1992. So I scrutinized the source code for CAL and found perfectly correct coding to deal

with leap year testing. I then installed PRO-WAM, brought up BRINGUP, added a record on January 2nd, February 3rd, and March 2nd - all in 1992. I then exited BRINGUP, brought up CAL, and saw the asterisk correctly in the calendar positions for the three dates. I then even invoked BRINGUP from CAL, added another record in February, escaped back to CAL, scrolled to 1991 and back to 1992 (so CAL would re-read the BRINGUP/DAT file), and the added February record was also correct. Do you have another local problem? There were no patches to CAL, so you can't be missing one. Does anyone else have this problem?

More LB Templates

Fm Herman L. Hardin III, Dear Roy, Thank you for the continuing support for the TRS-80 Model III and Model 4P machines with such fine programs like LB 2.2, and EDAS, and outstanding DOS packages. Without your support I would be out in the cold as well as out in the middle of the ocean.

Enclosed on the disk are five of the templates that I have built in the thirty days since I received LB 2.2. They are: **DRA**, a Dragon Magazine article index; **GAMEINV**, a role playing game inventory; **STREK**, a Star Trek collection inventory; **VID**, a Video tape and Laser disk library; and **PROP**, a Valuable property record

I have eight more templates that I have built that are variations on what I have already constructed. There are at least ten more sets to put together before I am finished cataloging everything I want.

For the ease of use of LB 2.2, I only have this to say: Extraordinary, Exceptional, Phenomenal, Rare, Remarkable, Singular, Uncommon, Unique, Special, Wonderful, etc., (choose your favorite adjective).

The one template I am interested in is the one that Ken Strickler talks about in his review of LB 2.x.x for his CD collection in TMQ VI.ii.

Fm MISOSYS, Inc: Thanks for the input. And here's my request again for you LB users to forward me templates of any LB data bases you have designed. All you need do is use LBMANAGE to duplicate a database construct (i.e. file.DEF and file.LB), then bundle them together with any associated screen and print definitions, and you have a template. I'll be putting these together on a disk. Send in a blank with return postage for the latest bundle.

CLAN Genealogy System

Fm Chris Fara, MICRODEX Corporation, 1212 North Sawtelle Avenue, Tucson, Arizona 85716, (602)326-3502: Dear Roy, Stan Slater (of Computer News 80) brought to our attention the "Letters to MISOSYS" in the Winter 1991/92 issue of the "Quarterly", page 24, regarding "CLAN genealogy system".

The problem and the clear solution you described apply to the Model III Clan. Partly because of such problems we have recently published a completely new CLAN-4 program in true Model 4 mode. It runs 100% in fast machine language on floppy or hard disk systems, and does not require any patches.

It also eliminates other problems that have plagued the old Clan, such as endless loops, excessive file fragmentation, sluggish indexing, and so on. It's really a classy new program. Data files from Mod-III Clan can be used with CLAN-4.

It seems Stan brought up your article, thinking that it had to do with our new CLAN-4. To prevent similar confusion, would you be kind enough to clarify in a next "Quarterly" that the new CLAN-4 from Microdex has completely eliminated such problems?

If you'd like a copy for review, please don't hesitate to let us know. In any event, thank you for your attention.

Fm MISOSYS, Inc: Chris's input clarifies Patrick Hamels input to TMQ Issue VI.ii, page 24. Now if anyone would like to review CLAN-4 specifically as a submission to *The MISOSYS Quarterly*, please contact Cris. I am sure, based on his letter, that he will accept the first request for a TMQ review of CLAN-4.

Help needed on Anitek's memory board

Fm Pieter J. Plomp, Linnaeusparkweg 59, 1098CR Amsterdam, Holland: Roy, I have been told that you have been asked by someone if there is a Model 4 PrintMaster program. I bought one 4 years ago and found it working well. If he still needs it, he can write to me for all information.

While I am writing to you I should like to mention that I am in need of knowledge of how Lescript handles bank switching on a Memory Board. I have a homebuilt Memory board along the lines of the article in 10/1987 *80-Micro*, using Port 9Ch, but cannot get Lescript to see it. I know

that the Anitek board uses port 43H, but changing all OUT 43's into OUT 9C does not help. Unlike other well written software, such as FastTerm, Lescript does not use Bank SVC's.

Fm MISOSYS, Inc: I don't have the foggiest, since I never followed the Anitek board(s). I have always regretted Peter's insistence on going directly to hardware within his program. That totally defeats the portability of expanded memory amongst all programs needing access to additional memory.

I do recollect that David Goben did some work on an @BANK interface for the Anitek board, so perhaps he can get with you (he reads TMQ) or you can write to him in care of CN80.

Update on BOOT5

Fm MISOSYS, Inc: A few folks have asked about patches to BOOT which appeared in TMQ V.iv. BOOT 5 was written using hard-coded addresses in LDOS 5.3.0; thus, it doesn't work with LDOS 5.3.1! I made a few suggestions which have been followed up by Art McAninch and Lance Wolstrup. Lance published a set of fixes in *TRSTimes* Volume 5.1 Jan/Feb 1992. Since he republishes some of my fixes for his audience, I though he would permit me to re-publish that set in TMQ. I'll show the set of PATCH commands as a single FIX file.

```
.BOOT531/FIX - Patch to
BOOT5/CMD for LDOS 5.3.1
. Apply via, PATCH BOOT5
BOOT531
D01,92=6A;F01,92=62
D03,5F=31;F03,5F=30
D04,2B=31;F04,2B=30
D04,D3=D9;F04,D3=F6
D04,D9=50;F04,D9=6D
D04,DC=51;F04,DC=6E
D04,E1=D3;F04,E1=F0
D04,E4=12;F04,E4=2F
. Eop
```

Incidentally, if you don't yet subscribe to TRSTimes, you ought to take a serious look at it. See their ad in this issue for details.

Inside TMQ

LBFILL: An LB Database Utility

by Roy Soltoff

In this issue of *The MISOSYS Quarterly*, I have re-printed some correspondence between myself and Ivan Kennedy. Included in that dialogue is a request for support of one particular difficulty in using our LB Database Manager. Specifically, the situation develops because of the differences between Profile and LBDM, Dr. Kennedy being a recent convert to LB from Profile.

Data base file structures which store data in fixed field formats have a choice as to what to do with field fill characters. These are character positions which are beyond the scope of the field data string. For instance, if an alphabetic field is established at twenty character positions and the field is "flush left" (i.e. the character string abuts the left-most position of the field), and a particular data record has a 15-character string to be stored, what character or characters are used to occupy positions 16-20? One way could be to simultaneously store a string length field for each data field of each record; however, that does appear to waste storage space. Another way could be to add a special character which would be used to indicate the end of the string, as long as the string was shorter than the field maximum. Finally, a third way could be to use SPACE characters to fill out the unused positions, since we don't *see* SPACEs.

Profile uses the latter method; all fields have unused character positions filled with the SPACE character. LBDM uses the middle method; the actual string of data entered is NULL-terminated. So someone shifting from a SPACE-filled data structure to a NULL-terminated data structure who was making use of the SPACE-filled construct may have some level of difficulty. Let's examine LB a little more closely.

LB's print capabilities allow for columnized printing by establishing a print zone equal to the maximum field size. When printing a specific data value of size less than this maximum, SPACEs are added during printing to fill out the field. There is a caveat: LB will columnize so long as the defined print screen has provided sufficient spacing for the field maximum.

LB also provides a *strip* option which allows you to over-ride the automatic columnization; strip can be selectively optioned for each field. When strip is active for a field during printing, LB will not add any SPACEs to fill out the field size when printing a shorter-than-maximum data string. But LB will NOT strip any trailing SPACE characters specifically entered in the record's data field. These are trailing SPACEs entered by a user after the last non-blank character. This is how LB was designed many years ago.

In response to Ivan's request, I considered changing LB to automatically strip any SPACE characters explicitly entered after the last non-blank character in addition to suppressing columnization when the strip option was active. However, some folks may be utilizing the actual design of permitting trailing SPACE characters; thus, a change in behavior needs to be carefully weighed. Besides, trailing SPACE characters entered by a user are either overtly entered for a reason, or inadvertently entered through extra typing or importing data. There's no way to determine why trailing SPACE characters would be in a field.

Because of the above, I decided that the best solution was to provide a utility which could convert an existing LB data base file to either NULL-terminated fields or SPACE-filled fields. Therein was the reason for whipping up LBFILL. When using LBFILL to NULL-terminate, trailing SPACE characters are all removed and a NULL is added immediately following the last non-blank character - provided the field is not totally filled with data. When SPACE-filling with LBFILL, the NULL is removed and all character positions following the last non-blank character are filled with SPACE characters.

LBFILL will do nothing with four field types: Calculated, Float, Dollar, and Right Justified Numeric. The Calculated field actually has a zero length; data is not stored in this field but is calculated when needed for output. The Float and Dollar field types completely fill the field or are flush right. The last type is always flush right; thus, there are no trailing unused character positions in each of the four types mentioned.

LBFILL allows you to alter the field buffering characteristics for either all records or only the records associated with an index file. In either case, deleted records are automatically skipped over. Because LBFILL was designed as a simplistic approach to a situation, there was no attempt to provide bullet-proof behavior, such as permitting you to escape from the process once it started. But I did take the time to utilize some of the functions used in LB's code modules which may make it easier for folks to generate their own utility programs for whatever reason they may have.

The two functions which have been used in LBFILL but for which source code is not being provided are fext() and rdpath(). The object modules, which can be linked into your programs, are available on DISK NOTES 6.3. Fext() is used to develop a full-path string needed to access any of the LB data base files. The rdpath() function is used to read the database path file (dbname.pfl) and store the data read in the

LBFILL.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>
#include "lbfill.h"
char dbname[9];
char fldbuf[255];
char recbuf[MAXBYTES];
char *p, *q;
unsigned curinp;        /* current record being processed */
int Index = 0;          /* Index file, 0 if none requested */
int Fill = 0;  /* Fill character, NULL if none requested */
int Itotal;             /* Total number of records to process */
int ret, x, rec, len;
int nfields=0;          /* number of fields to process */
struct fields Fp[MAXFIELDS];
struct fields *pFp;
/* This structure defines some of the data from the .DEF file */
struct datares LB =
  { "", 0, 0, 0, 0, 0, "", 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, "", { 0 }, 0, 0L, 0, 0, 0, "", { 0, 0 }, 0 };
/* This structure holds info on each individual field */
struct fldres Fn[MAXFIELDS];
struct Ares A = { "", NULL, NULL, NULL };
char Abuff[FULLPATHLEN+1];
char dpath[4][MAXPATHLEN+1];

main(argc,argv)
    int argc;
    char * argv[];
{
    puts("\nLBFILL Version 1.0\n");
    if (argc < 2)
        showsyntax();                   /* show syntax and abort */
    if (strlen(*++argv) < 9)
         strcpy(dbname, *argv);         /* grab database name */
    else
        showsyntax();
    while (--argc > 1)                  /* while we have arguments */
        {
        if ((*++argv)[0] != '/')
            showsyntax();   /* if not a switch, show and abort */
        switch ( *(argv[0]+1) )
            {
            case 'F':
            case 'f':
                    Fill = 32;          /* set to SPACE */
                    break;
            case 'I':
            case 'i':                   /* get the index arg value */
                    ret = getval(argv[0]);
                    if (ret < 1 || ret > 10)
                        {
                        puts("Index must be 1-10\n");
                        exit(1);
                        }
                    Index = ret;
                    break;
            default:
                    showsyntax();       /* error and exit */
            }
        }
    if (! (rdpath(strcpy(Abuff, dbname))))
        {
        puts("Can't open path file\n");
        exit(1);
        }
    fext(Abuff, "DEF");  /* full path spec of db.DEF file */
    if (!(A.fDef = fopen(A.Fspec, "r+b")))
        {
        puts("Can't open definition file\n");
```

```
        exit(1);
        }
    (void) fread(&LB, sizeof LB, 1, A.fDef);        /* get LB
def data */
    ret = (int) fread(Fn, sizeof Fn, 1, A.fDef);    /* get field
data */
    (void) fclose(A.fDef);                          /* close DEF file */
    if (ret != 1)
        {
        puts("Error during field load\n");
        exit(1);
        }
    fext(LB.Dbname,"LB");          /* build db.LB full spec */
    if (!(A.fData=fopen(A.Fspec, "r+b")))
        {
        puts("Can't open data file\n");
        exit(1);
        }
    if (getind() == NO)
        cleanup();
    if (!Index)                    /* if no index file */
        Itotal = LB.Urec;          /* ... use User rec count */
    printf("Database = %s, Fill = %d, Index = %d, Records =
%d\n",
        dbname,        Fill,        Index,          Itotal);

    buildfields();     /* build structure of fields to process */
    for (rec = 0; rec < Itotal; rec++)
        {
        errno = 0;                 /* reset errno */
        ret = Index ? getinrec() : rec;     /* use index or
sequence */
        if (errno)     /* would be <> 0 if index error */
            cleanup();
        if (readinp(++ret) != YES)     /* get a record */
            cleanup();
        if (!(*recbuf & DELETEDREC))       /* ignore if deleted */
            {
            for (ret=0, pFp=Fp; ret < nfields; ret++)
                {
                p = fldbuf;            /* field buffer */
                q = recbuf+pFp->fpos; /* field in record buffer */
                x=0;
                while (x++ < pFp->flen)    /* shift field to work
buffer */
                    *p++ = *q++;
                *p = '\0';         /* ensure NULL terminator */
                len = strlen(fldbuf);      /* what's it's actual
length? */
                p = fldbuf+len;  /* position of NULL */
                while (*--p == ' ' && len--)
                    *p = '\0';     /* strip trailing spaces */
                x = pFp->flen - len;       /* calc fill length */
                while (x--)
                    *++p = Fill; /* fill with filler */
                p = fldbuf;        /* field buffer */
                q = recbuf+pFp->fpos;      /* field in record
buffer */
                x=0;
                while (x++ < pFp->flen)
                    *q++ = *p++; /* move to record buffer */
                ++pFp;
                }
            if (saveout())/* save to disk */
                cleanup();
            }
        }
    (void) fclose(A.fData);
    printf("\nProcessed %d records\n",Itotal);
    exit(0);
}
```

dpath data array, thus making it available to fext().

A close examination of LBFILL's source code by a C-programmer will reveal methods to access a database's file set, access and manipulate field data, and read/write records either sequentially or via an index file.

For the non-programming LB user, LBFILL is simple to use. It is invoked by a command of the form,

LBFILL dbname [/Fill] [/Index=n

In the command string, *dbname* is the data base name entered in the same format as you would use when passing the data base name on the command line when you invoke LB (i.e. no extension, no drive specification, or other path sub-string).

Both parameters *fill* and *index* are optional. If *fill* is not specified, LBFILL will NULL-terminate the fields; if *fill* is specified, the fields will be SPACE-filled. Note that the parameter is preceded by a slash character, "/", and can be abbreviated to a single character, "f" - which can be entered in either upper or lower case. You can actually misspell the parameter as only the first character is significant. If *index* is not specified, the entire file will be processed sequentially, ignoring deleted data records, of course. Specifying an index of 1 to 10 results in using the associated index file; only records selected in that index will be processed, still ignoring deleted data records (i.e. records may have been deleted after the index file was created).

LBFILL is provided as an executable file on DISK NOTES 6.3. I hope you will find it useful. And perhaps LB users who are also C programmers may find this program example instructive on techniques which may be used to access an LB data file structure.

```
void cleanup()
{
    (void) fclose(A.fData);
    if (Index)
        (void) fclose(A.fDind);
    exit(1);
}
void showsyntax()
{
    puts("Syntax: LBFILL databasename [/Fill]
[/Index=n]\n");
    exit(1);
}

int getval(p)
    char * p;
{
    int i, len;

    while (*++p && *p != '=')       /* while
chars, find the '=' */
        ;    if (!(len = strlen(++p)))        /
* error if null length */
        return -1;
    for (i=0; i<len; i++)
        if (! isdigit(*(p+i)))
            return -1;
    return atoi(p);
}

/* Get the current index to use */
int getind()
{
        int c, ret;
        static char Sfil[] = "SL ";

        if (!Index)
            return YES;
        Sfil[2] = (char) Index+'/';/* convert
to ASCII (-1) */
        fext(LB.Dbname, Sfil);
        if (! (A.fDind = fopen(A.Fspec, "rb")))
            {
            Index = 0;
            (void) puts("That index file not
found\n");
            return NO;
            }
        Itotal = getinrec();       /* Read recs
in file */
        if (errno || ! Itotal)
            {
            if (! Itotal)
                (void) puts("No records in that
index file\n");
            Index = 0;
            (void) fclose(A.fDind);
            return NO;
            }
        (void) fseek(A.fDind, 20L, SEEK_SET);
/* Position to 1st recnum */
        return YES;
}

unsigned getinrec()
{
        unsigned y;

        y = (unsigned) getw(A.fDind);
        if (ferror(A.fDind))
        {
                (void) puts("Error during index
file read\n");
                return 0;
        }
        errno = 0;
        return y;
}

/* Save the record to disk */
int saveout()
{
        seekinp();
        (void) fwrite(recbuf, LB.Lrl, 1,
A.fData);
        if (fflush(A.fData))
        {
                if (errno != EROFS)
                        (void) puts("Error
during data write\n");
                return errno;
        }
        return 0;
}

/* Read the record */
int readinp(n)
        unsigned n;
{
        curinp = n;
        seekinp();
        if (fread(recbuf, LB.Lrl, 1, A.fData)
!= 1)
        {
                (void) puts("Error during data
read\n");
                return NO;
        }
        return YES;
}

/* Seek the start of a source record */
void seekinp()
{
        (void) fseek(A.fData, (long)(curinp -
1) * (long) LB.Lrl, SEEK_SET);
}

void buildfields()
{
    int i, pos;
    struct fldres *pfld;

    for (i=pos=0, pFp = Fp, pfld = Fn; i<
LB.Nfld; i++)
        {
        switch (pfld->Ftyp)
            {
            case 'C':  /* don't do calc */
            case 'D':  /* dollar & float */
            case 'F':  /* are full right */
            case 'R':  /* right-just is full */
                       break;
            default:   /* set recbuf position */
                pFp->fpos = pos;
                (pFp++)->flen = pfld->Flen; /*
set field length */
                ++nfields;/* bump field count */
                break;
            }  /* calculate next position */
            pos += (pfld++)->Flen;
        }
}
```

```
/*
   lbfill.h
 */

#ifdef MSDOS
#define MAXPATHLEN 64
#define FULLPATHLEN 76
#endif
#ifdef LDOS
#define MAXPATHLEN 2
#define FULLPATHLEN       14
#option FIXBUFS    ON
#option REDIRECT   OFF
#option MAXFILES   2
#endif
#define MAXFIELDS  64
#define MAXBYTES   1024
#define FSLENG            8
#define YES               1
#define NO          0
#define DELETEDREC 0x80
struct fields
{
    int fpos;             /* starting position
in field */
    int flen;             /* length of field
*/
};

/* This structure defines some of the data
from the .DEF file */
struct datares
{
    char Dbname[FSLENG+1];        /* the NAME,
used to name ALL files */
    unsigned Psrec;       /* print - starting
record */
    unsigned Perec;/* print - ending record */
    int Pcopies;   /* print - copies */
    char Pdevtype; /* print - output device
type */
    char Pcronly;  /* Y if suppress LF after CR
(MSDOS) */
    char Dbpswd[9];
    unsigned Arec; /* total record space
allocated */
    unsigned Urec; /* records in use */
    unsigned Lrl;  /* total record length */
    unsigned Ldel; /* last deleted record */
    int Nfld;      /* number of defined fields
*/
    char Pcbrk;    /* print - control-break
paging on/off */
    char Upfld;    /* set YES if unprotected
fields */
    int Cind;      /* current index number */
    int Cscr;      /* current screen */
    int Cprt;      /* current print format */
    int Pfrom;     /* last overlay executed by
print setup */
    int Where;     /* overlay to which select/
sort must return */
    int Fault;     /* sort module return code
*/
    char Sheet;    /* TRUE if single sheet feed
& prompt */
    char Sparel;   /* unused */
    int Found;     /* TRUE if database opened
successfully */
    int Level;     /* set if password entered
correctly */
    int Auto;      /* auto job state: capture/
on/off */
    char Aspec[FSLENG+1];  /* auto job filespec
*/
    char Spare2[14];       /* unused */
    int Pappend;           /* print - append to
output file */
    long Pos;       /* seek position in auto job
file */
    unsigned Adindx;       /* number of records
in add index file */
    unsigned Ttldel;       /* number of deleted
records */
    int Addindexflag;      /* add index on/off/
none */
    char Pspec[FSLENG+1]; /* print - redirec-
tion filespec */
    int Pnl_tof[2];        /* print - newlines/
TOF before/after report */
    char Pdetail;          /* print - suppress
detail on/off */
};

struct fldres      /* LB field structure */
{
    int Flen;
    char Ftyp;
    char Fprt;
    char Fnam[20];
};

struct Ares
{
    char Fspec[FULLPATHLEN+1];     /* A global
buffer used to open a file */
    FILE *fDef;    /* File descriptor, defs */
    FILE *fData;   /* "" data */
    FILE *fDind;   /* "" current index */
};

#ifdef MSDOS
#define NEEDPROTOS
#endif

#ifdef NEEDPROTOS
void    buildfields(void);
void          cleanup(void);
int     getind(void);
unsigned   getinrec(void);
int     getval(char *);
int     readinp(unsigned);
int     saveout(void);
void          seekinp(void);
void    showsyntax(void);
extern void fext(char *, char *);
extern int  rdpath(char *);
#else
void          buildfields();
void          cleanup();
int     getind();
unsigned   getinrec();
int     getval();
int     readinp();
int     saveout();
void          seekinp();
void          showsyntax();
extern void fext();
extern int  rdpath();
#endif
```

SYSFLEX, The Article

by Matthew Reed
Box 368
West Lebanon, NH 03784-0368

A 128K Model 4 contains two "extra" 32K memory banks. These memory banks are generally unused by the operating system, although many programs do exist that can take advantage of the extra 64K. For example, the MEMDISK driver included with LS-DOS can use the memory as a small, super-fast disk drive. If you copy all the LS-DOS system files (SYS0-SYS13) onto this RAM disk and then establish it as drive zero, your Model 4 will experience truly spectacular performance. Library commands will execute instantly, and physical drive zero will then be available for data diskettes.

Another excellent use for the extra 64K is PRO-WAM, the window controller and applications manager from MISOSYS. Once PRO-WAM has been installed, pressing a special key combination "pops up" a menu of small applications. These applications, which can include a calculator, calendar, address book, to-do list, and more, can be used from within any program, without exiting to LS-DOS and without disturbing your program in any way. PRO-WAM applications can even transfer data to and from the interrupted program; for example, you could "import" numbers from a word processing document into the calculator and "export" the sum back to the document.

Now that I have used a RAM disk and PRO-WAM, I never want to work without them. The problem is that PRO-WAM uses 32K of the Model 4's extra memory. That leaves only 32K for the RAM disk, and that isn't enough for all the LS-DOS system files. Creating a RAM disk with only some of the system files won't work, because if LS-DOS can't find a necessary /SYS file, it displays "Error 07H" and often locks up the machine. This means I could install PRO-WAM, or I could install a RAM disk as the system disk, but not both at the same time. On my system, PRO-WAM won, but it seemed to me that it should be possible to do it all at the same time.

What I really wanted was a way to have the most frequently used overlays resident in one 32K memory bank, in a type of caching system. The ones used less often, such as those containing libraries B and C, could still reside on an actual disk. First I thought of writing something similar to SYSRES, but which would load /SYS files into a single alternate bank instead of main memory. Then I came up with a better idea. Why not tell LS-DOS to search more than one drive for system files? Then I could put the most important system files on a small RAM disk and leave the rest on physical drive zero.

SYSFLEX, the flexible /SYS file loader, is the final result of my efforts. SYSFLEX accepts three parameters. Upon installation, the "SYSTEM" parameter specifies the drive that LS-DOS will search first for /SYS files, and the "DRIVE" parameter specifies the drive that LS-DOS will search second. (Both parameters can be abbreviated to one letter, and if you don't include them on the command line, SYSFLEX will default to SYSTEM=0, DRIVE=1.) When you are done with SYSFLEX, the "REMOVE" parameter disables SYSFLEX and (if possible) removes its small high memory module. ("REMOVE" cannot be abbreviated.) For example, to install SYSFLEX using a primary system drive of 2 and a secondary system drive of zero, enter the following command from "LS-DOS Ready":

SYSFLEX (S=2,D=0)

To disable SYSFLEX, enter:

SYSFLEX (REMOVE)

How It Works

Any system disk's directory contains sixteen reserved entries. Three of these entries are used for BOOT/SYS, DIR/SYS, and SYS0/SYS; the remaining thirteen are for SYS1/SYS through SYS13/SYS, the LS-DOS overlay files. When LS-DOS needs to load in one of these overlays, it calculates the overlay's directory entry and checks to make sure that the /SYS file is in fact on the disk. (See The SOURCE, volume 1, pages 97-98, for more information.)

This is where SYSFLEX hooks itself into the system. SYSFLEX performs exactly the same operations as the LS-DOS system loader, with one exception: if necessary, it goes through the whole procedure twice, once for the primary drive, once for the secondary drive. If the requested /SYS file exists on either disk, then SYSFLEX returns control to LS-DOS so the proper file can be loaded. If the requested /SYS file does not exist on either disk, SYSFLEX returns control to the LS-DOS error handler.

Due to the complicated nature of its operation, SYSFLEX works on a much lower level than is proper for ordinary applications programs. This is because SYSFLEX, by necessity, must become part of LS-DOS. For example, to avoid some serious problems, the SYSFLEX high memory module calls the DIRRD@ routine within LS-DOS directly, rather than using the equivalent SVC.

Conclusions

SYSFLEX can be used with any two disk drives; one drive does not have to be a RAM disk. This means you could use SYSFLEX to make a more flexible hard disk configuration, or to distribute system files between two floppy disks. SYSFLEX probably even has uses that never occurred to me when I was writing it.

I have been using SYSFLEX on a 128K Model 4 for over a year without a single problem. The ability to have PRO-WAM installed and keep most system files on a RAM disk has made the effort I put into designing and programming SYSFLEX well worthwhile. I hope you find SYSFLEX as useful as I have.

POINT:

An example of highly optimized code for the Z-80.

by Jonathan Armstrong
208,2973 Pembina Hwy
Winnipeg, MB R3T 2H5
CANADA

Some time ago Roy pointed out in *The MISOSYS Quarterly* that expert assembler coders keep careful track of the ongoing status of flag bits and the contents of registers in order to make the best possible use of the hardware and instruction set. This requires a thorough knowledge of the complete effects of every instruction (or in my case, frequent reference to the programmers' manual!). In general, there are usually several ways to achieve a given objective when programming, and picking the "best" (i.e. fewest bytes and/or fewest T-states) can be quite an arduous process of comparing several possible techniques. In the discussion below, I use the "T-state" (memory cycle) as a measure of speed. It is actually a time interval — on a Z-80 running at 4 MHz, one T-state is 0.25 microseconds; on an HD64180 (the processor in an XLR8er) running at 6.144 MHz, it is 0.1628 microseconds.

I was not satisfied with the routine for plotting points on the Hi-Res graphics display that is provided in the documentation of The Grafyx Solution (TM) board (listing 1). This is quite a good routine — note at the end the use of DJNZ to accomplish what in a high-level language would be called a CASE construct — but I didn't like the repetition of relatively slow rotate instructions in the middle to accomplish the division by 8 (far faster than a general-purpose divide, to be sure). Also, this routine does not check for points out-of-range; on principle, this should be done at the lowest level, so high level routines which call the point-plotting routine as a subroutine (e.g.. line-drawing) need not bother to check that each point they are plotting is on-screen. What I eventually came up with (listing 2) is almost 25% faster, and includes boundary checking. This is accomplished in part by integrating the check with the process of scaling the values, and in part by keeping values in the registers rather than in memory. The latter point is also responsible for the net reduction in size of 3 bytes (assuming no padding is necessary in front of the bit mask table — see explanation below). I employed a number of tricks and shortcuts to get the job done as quickly as possible, which I will explain as an illustration of the way in which "side-effects" of an instruction can be utilized to good effect. Unfortunately, I doubt that this explanation will be very helpful to anyone not having a Z-80 programmers' manual.

I started with the idea that it must be possible to use the RRD instruction (intended for use with BCD — Binary Coded Decimal — arithmetic; it is essentially a 4-bit rotate) as a divide-by-16 instruction. What is needed in order to extract the X value to be sent to the hardware from the software X value is a divide by 8 (with remainder), so I needed to combine the RRD with a rotate left (multiply by 2). There is a snag, however: RRD works only partially in the Z-80 registers; it references a byte in memory pointed to by the HL register. This meant I had also to store a byte, then retrieve it for output. I could see the advantage disappearing very rapidly. There are output instructions which use the byte pointed to by the HL register, however, e.g.. OUTI . . . but this requires that the C register be loaded with the output address, so the advantage was pretty marginal until I realized that to check that X is less than 640, it is sufficient to multiply it by 2, then check that the most significant byte is less than 5. Thus I can use the same multiply by 2 for two purposes: to check that the value is in range, and as part of the scaling process. Note that to simplify checking the Y value, I am allowing a range of 0 - 255. Only 0 - 239 can appear on the screen, but the hardware will accept up to 255 as valid (these points are plotted, and can be tested, etc., but are below the bottom edge of the screen).

To proceed through the routine: Examine the entry conditions specified at the top of listing 2. Firstly, the Y value is checked by examining the most significant byte. If any bits are set, the value is out-of-bounds, and the routine exits with the Carry flag reset (i.e. NC condition, established by the OR A instruction). Once this byte is known to be empty, it is used to hold the plot mode (which cannot stay in the L register, as HL is about to be used). The most significant byte of the X value is moved to the accumulator, HL is pointed to a temporary storage location, and the least significant byte of the X value is multiplied by 2 and then placed in the storage location. Note that because the carry bit was reset earlier, RL C would have been as good as SLA C, but since it makes no difference to speed or size, it is better to use the instruction which most accurately represents the operation being performed. Now the upper byte of X (in the accumulator) is multiplied by 2 (incorporating the bit carried out of the C register) and compared with 5. If it fails this test, the routine again exits with Carry flag reset (thus if the calling routine needs to know if the point was out-of-bounds, it can do JR NC,*errlabel* after POINT returns). Having passed this test, the point coordinates are known to be valid, so the RRD is performed. This leaves X/8 in the temporary storage location, and twice X mod 8 in the accumulator (i.e.. the least significant 3 bits of X, shifted one place left). Thus the quotient and remainder have been calculated simultaneously. The C register is loaded with 128 — the hard-

ware address of the graphics board X register — and the X value is output. Now the bit pointer in the accumulator (twice X mod 8) is added to the HL register. Remember that HL was incremented by the OUTI, so now points to the first entry in the bit mask table. There are two points to be aware of here: Firstly, we are adding an 8-bit value (range 0 - 14) to a 16-bit value. To ensure that this cannot generate a carry into the most significant byte (the H register) the bit mask table is carefully positioned so that it does not straddle a 256-byte page boundary. At worst, this results in 14 wasted bytes (with a probability of only 1 in 18 of any being wasted). Secondly, the value in the accumulator is double what is needed. Rather than take the time to divide it by 2 (with RRA) I chose to use only every second byte in the table, "wasting" another 8 bytes. Both of these decisions reflect my preference for speed. All the preparation has now been done and the accumulator is now free, so the Y value is output. From this point, this routine is essentially identical to the original, except that I provided a separate return from each case to save a few T-states, and also the carry flag is set to provide a "success" (i.e. valid point address) indication to the calling routine.

I am not foolish enough to think that I have invented the fastest possible way to achieve this function — and note that it does not set the hardware control byte, which the old routine does (but which in my view should be done only once, before drawing begins) — but I bet it's pretty close. I spent a ridiculous amount of time optimizing this routine (if I was doing it for pay, I'd have been fired for low productivity!), and if anyone can do better, I'll be very impressed (for the HD64180 I found it is possible to save an additional 3 T-states. I'll leave this as an exercise for the reader). I actually spent about 4 days on it, which I suppose illustrates the concept of diminishing returns — the speed gain is quite small (about 13 microseconds per point, plus whatever it took to check the boundaries with the old system), and it took a lot of effort on my part. Moreover, there aren't many applications for which

```
LISTING 1:  Public domain point-plot routine for TRS-80 hi-
res graphics.
;
; Takes 263 T-states (for COLOR = 2) on a Z-80, i.e. 66
; microsec. @ 4 MHz or 231 on an HD64180, plus refresh and
; any extra wait states,i.e., 38.6 microsec. @ 6.144 MHz,
; plus 10 microsec for each memory wait,
; assuming 2-state refresh every 80 states.
;
; Entry Parameters:
;
;     XPOS: X-position (16-bit value) Range 0 - 639
;     YPOS: Y-position (16-bit value) Range 0 - 239
;     COLOR: Plot mode  (8-bits) 0 = Clear, 1 = Set,
;           2 = complement,3 = read and return value
;
; Exit Parameters
;
;     If COLOR=3,  then for bit=0, Z flag is set and Accumu-
lator is clear
;else for bit=1, Z flag is clear and Accumulator is not.
;
POINTLD     A,(YPOS)        ;Assume valid value
      OUT     (129),A             ;Output Y coordinate
      LD      A,243           ;Control byte (no auto-inc.)
      OUT     (131),A
      LD      HL,(XPOS)       ;Assume valid value
      LD      A,L             ;Extract LS 3 bits
      AND     7
      LD      C,A             ;Save for later
      LD      B,0             ; - as 16-bit value
      SRL     H               ;Divide X by 8
      RR      L
      SRL     H
      RR      L
      SRL     L
      LD      A,L             ;Result to Accumulator
      OUT     (128),A               ;Output X coordinate
      LD      HL,BTABLE       ;Point to bit mask table
      ADD     HL,BC           ;Add bit pointer
      LD      A,(COLOR)       ;Get plot mode
      LD      B,A
      INC     B               ;Adjust for DJNZ
      IN      A,(130)         ;Pick up existing graphics byte
;Perform required operation:
      DJNZ    NOT0            ;Skip if not color 0
;Clear point:
      CPL
      OR      (HL)            ;Set bit high
      CPL
WRITEOUT    (130),A         ;Write modified byte back
      RET                     ;EXIT
NOT0  DJNZ    NOT1            ;Skip if not color 1
;Set point:
      OR      (HL)            ;Set bit high
      JR      WRITE           ;Output new value
NOT1  DJNZ    NOT2            ;Skip if not color 2
;Complement point:
      XOR     (HL)            ;Complement point
      JR      WRITE           ;Output new value
;Read point:
NOT2  AND     (HL)            ;Set flags
      RET
;
```

```
BTABLE DEFB 128          ;Table of bit mask values
     DEFB  64
     DEFB  32
     DEFB  16
     DEFB  8
     DEFB  4
     DEFB  2
     DEFB  1
;
XPOS DEFS  2             ;Storage for X coordinate
YPOS DEFS  2             ;Storage for Y coordinate
COLORDEFS  1             ;Plot mode
     END
```

Total length 76 Bytes.

```
LISTING 2: Point-plot routine
; (C) 92-1-27  Jonathan R. Armstrong
;
; Takes 211 T-states on a Z-80 (for mode = 2), i.e.. 53
microsec. @ 4 MHz
; or 187 on an HD64180, plus refresh and any extra wait-
states,
; i.e.. 31.2 microsec @ 6.144 MHz plus 8 microsec for each
memory wait,
; assuming 2-state refresh every 80 states.
;
; Entry parameters:
;
;    BC : X-coordinate
;    DE : Y-coordinate
;    HL : plot mode:
;           0 : CLEAR point
;           1 : SET   point
;           2 : FLIP  point
;           3 : READ  point
;
; Exit parameters:
;
;    A  : Bits written, or bit read (for valid point only)
;    F  : CARRY flag:
;           NC : invalid point address
;           C  : valid
;         ZERO flag (for read bit (mode 3) only) :
;           Z  : graphics bit was CLEAR
;           NZ : graphics bit was SET
;    B  : 0
;    C  : 128  (X-reg. hardware address)
;    D  : Plot mode
;    E  : Y-coordinate (range 0 - 255)
;    HL : Pointer to bit mask for current bit
;
POINTLD    A,D           ;0 <= Y < 256 ?
     OR    A             ; (set status flags for high byte)
     RET   NZ            ;NO: ABORT
     LD    D,L           ;OK: Save output mode
     LD    A,B           ;MSB of X
     LD    HL,BTABLE-1   ;Pointer to temp. loc'n
     SLA   C             ;2*LSB of X
     LD    (HL),C        ;Save in temp. loc'n
     RLA                 ;2*MSB of X
     CP    5             ;0 <= X < 640 ?
     RET   NC            ;NO: ABORT
     RRD          ;OK: temp <== X/8 ; A <== 2*(X.MOD.8)
     LD    C,128         ;Set output address
```

the slight gain is worthwhile. I have a program which slightly resembles the Moire' program used as a Macintosh (TM) screen-saver, and which is quite fast (hundreds of millions of points per day!). Also HAT/BAS compiled via EnhComp and using fast line-drawing routines based on this version of POINT is about twice as fast as with GBASIC. If anyone wants to see these, send me $7 and I'll mail you a disk. Whether POINT was worth doing is arguable, but it was an interesting exercise. Note that the gain does not come entirely from the use of RRD — this takes 18 T-states, and the associated multiply by 2 takes another 12, for a total of 30, vs. 47 in the original program for the divide by 8 and extraction of the remainder — but from making use of other instructions which dovetail with the register use of RRD. The integration of the various effects of instructions in this way, so that advantage is taken of as many as possible of the results of each instruction, is a feature of code optimization for the Complex Instruction Set philosophy used in the design of the 8080 and its successors (the 8086 through to the 486). Reduced Instruction Set (RISC) designs should in theory be simpler to work with... but such features as branch instructions which delay taking effect for one instruction tend to make life complicated again. I suspect that some people like to make things difficult!

```
      OUTI   ;O/P X, inc HL points to BTABLE
      ADD    L              ;Add 2*BIT# to
mask address LSB
      LD     L,A            ;HL now points to
specific bit mask
      LD     A,E            ;Y coordinate
      OUT    A,(129)        ;O/P Y
      LD     B,D            ;Get plot mode
      INC    B              ;Adjust for DJNZ
      IN     A,(130)        ;Get old byte
      DJNZ   NOT0           ;Mode = 0 ?
        CPL                 ;Yes: CLEAR bit
        OR   (HL)
        CPL
        OUT  (130),A        ;     O/P new byte
        SCF          ;      Set success flag
        RET                 ;     EXIT
NOT0  DJNZ   NOT1           ;Mode = 1 ?
        OR   (HL)           ;Yes: SET bit
        OUT  (130),A
        SCF
        RET
NOT1  DJNZ   NOT2           ;Mode = 2 ?
        XOR  (HL)           ;Yes: FLIP bit
        OUT  (130),A
        SCF
        RET
```

```
NOT2  AND    (HL)           ;Get bit
      SCF           ; and assign Z flag
      RET
;.....................................................................
;                    LOCAL STORAGE AREA
;.....................................................................
; To permit the use of an 8-bit add to form
the 16-bit address, all of
; BTABLE must be on the same 256-byte page
(except the last byte).
; Note that the temporary storage location
used for the LSB of the X
; coordinate can be on the prior page; the
; pointer to this is incremented by OUTI,
;  which works with the full 16-bit value.
PCLSB DEFL   $.AND.0FFH
IF    PCLSB.GT.0F0H
      DEFS   256 - PCLSB
ELSE
      DEFS   1      ;Ensure 1 byte minimum
for temporary storage
ENDIF
BTABLE DEFW 128,64,32,16,8,4,2,1
; (MSB's are not used; thus the last byte
can be on the next page)
```

Total length 73 Bytes, assuming PCLSB triggers no filler.

Monochrome Graphics Programming

by Hans de Wolf
Hendrik Schaarstraat
1544 WH Zaandijk
The Netherlands

Internet address for e-mail:
hw23316@nlr.nl
For compuserve users that is:
>INTERNET:hw23316@nlr.nl

Summary

One of the most interesting and spectacular subjects in computer programming is high-resolution graphics. In this article I will discuss items that are related to this subject: details about popular image file formats, programming techniques in the C language, and little-known and even undocumented features of the Radio Shack hires board for the TRS-80. These ingredients are used to create the MONOSAVE program, which will allow you to export monochrome pictures to other platforms.

Introduction

A TRS-80 equipped with a high-resolution graphics card may not be the equivalent of a state-of-the-art Apple Macintosh or an IBM PS/2 with XGA display, but it can display graphical images with surprising quality. And, when you make a printout of these images, you certainly cannot tell which engine was used to process them: a 80486, 68040 or 'just' a Z-80.

A good example of what can be done with a TRS-80 equipped with a graphics board and good software is the GIF4MOD4 package created by Frank Slinkman. Using the programs in that package you can display, store and print out pictures in the Graphics Interchange Format (GIF). This GIF format offers a lot of advantages: it stores the images in very compact way and is system-independent, giving you a graphics connection to nearly any other computer type. However, the format is hardly ideal for the TRS-80: a lot of processing is required to display a picture on the TRS-80 screen, and the time required to create a high quality printout on a 300 dpi printer allows you more than enough time to get another cup of coffee.

This observation is not intended as a negative remark about the GIF4MOD4 package: I regard it as top-quality software! It is just the GIF format that requires a lot of processing power: before an image can be displayed or printed it must be read from disk and decompressed, the colors must be translated into a gray scale value, and a dithering algorithm must translate the gray scale value in a pixel pattern on the screen or printer. If you keep this in mind, GIF4MOD4's speed is impressive!

While I was shopping around for GIF pictures on MS-DOS bulletin boards I encountered also other common formats: PCX, IMG, MacPaint, TIFF. I got curious about how images were stored in these files, and, after some research, I found out that some of these formats allowed transport to and from the TRS-80 in a much easier and faster way than the GIF format - that is, for monochrome bitmap images! Only one small detail had to be solved: I could not find TRS-80 software to handle these formats, except Mel Patrick's HIRES4 programs for MacPaint files. However, this was just the kind of problem that I liked: a bit of challenge, but one that I should be able to solve. So I decided to create software that would allow me to access monochrome bitmap images in commonly used bitmapped formats, starting with the Monosave program to save monochrome images in common formats.

Existing TRS-80 formats

At the start of this project I decided that the MonoGraf software should be compatible with the standard TRS-80 formats. But what are these TRS-80 formats?

Radio Shack's /HR files
Simple but Large

The first format used to store TRS-80 hires images was defined by Radio Shack, and was supported by the software package that accompanies their hires hardware. Images stored in this format normally use files with the /HR extension, and take up 19200 bytes of disk space. The format is very simple: it is a direct copy of the information displayed on the hires screen. Each line of the display consists of 640 pixels (picture elements: the dots on the screen), stored in 80 bytes (using a single bit for each pixel). The complete screen consists of 240 lines. To create a /HR file, a program reads the 80 bytes from the hires video memory that make up the first line, and writes them to the file. This is repeated for all 240 lines in the image, making a file of 80 columns * 240 lines = 19200 bytes.

The function **hrfile** shows how a /HR file can be created:

```
x0 = 0; x1 = 79;
y0 = 0; y1 = 239;
for
(line=y0;line<=y1;++line)
{  /* load current line
into line buffer */
     loadline
(line,x0,x1,outbuf);
     fwrite
(outbuf,1,length,outfile);
}
```

Like many other functions in the monosave program, **hrfile** calls the function **loadline** to retrieve a row of pixels (from x0 to x1) from hires video memory into a buffer **outbuf**:

```
loadline
(line,left,right,bufstart)
unsigned int
line,left,right;
char *bufstart;
{     int column;
      char *bufpos,abyte;
      bufpos=bufstart;
      outport (XPORT,left);
/* set x-address */
      outport (YPORT,line);
/* set y-address */
      for
(column=left;column <
right+1;++column)
      {    abyte = inport
(DATAPORT); /* read byte */
           /* x address
increments automatically
after read */
           if (invert) abyte
= ~abyte;
           *bufpos++=abyte;
      }
}
```

The /CHR Format
Introducing RLE for Crunched Rows

Because the /HR format is simple, it is easy to write software that can handle these files. But that is about the only advantage of using the /HR format. On the negative side, each image costs a lot of disk space and due to the amount of disk access, requires much time to save or load.

A solution to this problem was presented by Mel Patrick, who created the /CHR (Crunched High Resolution) format. This format was derived from (but not identical to) the compression technique used to encode MacPaint pictures, known as Run Length Encoding (RLE).

RLE is a common technique to compress image bitmaps. It works because graphics files often contain repeated long sequences (called 'runs') of the same pattern. If a single byte is repeated 37 times on a row, the RLE technique does not store all of these identical 37 bytes, but stores the byte only once, as well as the count 37.

In order to create a /CHR file, the program scans each row of the image (80 bytes), and looks for repeating bytes. If the program finds a sequence of non-identical bytes (called a 'string run'), it must first write the count of the bytes in this sequence, followed by the actual bytes themselves. If a sequence of identical bytes is detected (a 'pattern run'), the program stores only the length of this sequence and one of the bytes in this sequence. In order to indicate the difference between a string run and a pattern run, the most significant bit of the count byte is set in a pattern run. This process is repeated for all runs in a row, and all 240 rows in an image.

An example demonstrates this technique more clearly:

Suppose that a row of an image contains the following bytes (listed as hexadecimal codes):

20 45 30 30 30 30 22 65 65 65 65 65

The first two bytes differ, and form a string run. In a /CHR file this will be encoded by a '02' byte indicating the length of the run, and the actual bytes of the run:

02 20 45

Next, we encounter a pattern run: the byte 30 is repeated four times. This is written to the /CHR file as 84 (the count 04, but with the most significant bit set to indicate a pattern run) followed by the pattern byte (30):

84 30

Then, a string run of only one byte (22), which must be preceded by a count of one:

01 22

and, finally, a pattern run of five bytes:

85 65

The complete sequence

20 45 30 30 30 30 22 65 65 65 65 65

is now encoded as:

03 20 45 84 30 01 22 85 65

requiring not 12, but only 9 bytes - a reduction of 25% in this simplified example. A better reduction ratio is reached when the repeated patterns are longer: a line consisting of 80 repeated bytes would cost only 2 bytes instead of 80!

The Monosave program uses several routines to create a /CHR file. The **chrfile** function reads each line into a buffer by means of a call to **loadline**, and uses **outline** to write the line to a file:

```
chrfile
(outfile,x0,y0,x1,y1,outbuf)
FILE *outfile;
unsigned int x0,y0,x1,y1;
char *outbuf;
{  unsigned int line,size;
   int chrrun();
   size = x1 - x0 + 1;
   for
(line=y0;line<=y1;++line) /
* loop over all lines */
   {  /* load current line
into line buffer */
      loadline
(line,x0,x1,outbuf);
      outline (outbuf,
size, outfile, SINGLELINE,
chrrun);
   }
}
```

The **outline** function is used to write a single line of the image to the output file using the RLE encoding technique. Because the other file formats also employ RLE encoding this function is constructed in such a way that it can be reused for these other formats. The **outline** function detects series of pattern runs and string runs, and calls a format-specific routine is called to write these runs to the output file.

```
outline
(buffer,size,outfile,repeatcount,outrun)
char *buffer;
FILE *outfile;
unsigned int size,repeatcount;
int (*outrun) (); /* pointer to function
that writes a run package */
{   unsigned int column,aratio;
    int runlength;
    char current,previous,state;
    char *runstart;
    if (repeatcount > 1)
       if (outformat == IMGFORMAT )
          imgrpt (repeatcount,outfile);
       else
          fatal ("internal error: replica-
tion packet for non-IMG format");
    state = EMPTY;  /* try to compress the
outbuffer */
    for (column = 0 ;column < size;++column)
    {   current = *buffer;
       switch (state)
       {   case EMPTY: /* empty run buffe */
          previous = current;
          state = ONLYONE;
          runstart = buffer;
          runlength = 1;
          break;
       case ONLYONE:/*1 byte in run buffer*/
          if (previous == current)
             state = IDENTICAL;
          else
             state = DIFFERENT;
          previous = current;
          runlength = 2;
          break;
       case IDENTICAL: /* more than one
identical byte in run */
          if (current == previous)
             ++ runlength;
          else
             {   /* output current run */
             (*outrun)
(state,runstart,runlength,outfile);
             /* reset status */
             state = ONLYONE;
             runlength = 1;
             runstart = buffer;
             previous = current;
             }
          break;
       case DIFFERENT: /* more than one
nonidentical byte in run */
          if (current != previous)
             ++runlength;
          else
             {   /* output current run */
             (*outrun)
(state,runstart,runlength-1,outfile);
             /* reset status */
             state = IDENTICAL;
             runlength = 2;
             runstart = buffer;
             }
          previous = current;
          break;
       default:
          fatal("undefined state in loop in
function outline");
       }
    ++buffer; /* end of loop, next column */
    }

/* loop completed, now flush leftovers */
    switch (state) {
       case EMPTY: /* do nothing */ ;
          break;
       case ONLYONE:
          (*outrun)
(DIFFERENT,runstart,runlength,outfile);
          break;
       case IDENTICAL:
       case DIFFERENT:
          (*outrun)
(state,runstart,runlength,outfile);
          break;
       default:
          fatal("undefined state after loop
in function outline");
    }
}
chrrun (state,runstart,runlength,outfile)
char state;
char *runstart;
int runlength;
FILE *outfile;
{   char abyte;
    switch (state) {
       case IDENTICAL/* write pattern run */
          abyte = *runstart;
          while (runlength > 128)
          {  putc (0x80,outfile);
             runlength -= 128;
             putc (abyte , outfile);
          }
          {  putc (runlength|0x80,outfile);
/* anything left? */
          putc (abyte,outfile);}
          break;
       case DIFFERENT: /* it is string run*/
          while (runlength > 128) /* max
CHR run length is 127 */
             {  putc (0x00,outfile);
             runlength -= 128;
             fwrite
(runstart,1,128,outfile);
             runstart += 128;
             }
          putc (runlength,outfile);
          fwrite
(runstart,1,runlength,outfile);
          break;
          default:
                fatal ("undefined state
in function chrrun");
       }
}
```

Here is a programming trick that may interest C-programmers: the **outrun** argument of the **outline** function contains a pointer to the function that writes the string or pattern run in the correct format to the output file. In the body of the **outline** function that format-specific output function is called by the statement **(*outrun)(state,runstart,runlength,outfile);**. For /CHR files this output function is **chrrun**.

SuperCrunched /SHR
The Best Compression Up To Now

David Goben has introduced a variation on the /CHR technique in his Supercrunched /SHR files, which may produce even smaller files (depending on the image). This is because the /CHR format works line-by-line; the repeat count never exceeds 80 because each line consists of 80 bytes. In the /SHR format the line length is ignored, the repeat count can be a large as 127 (not 255, because one bit is needed to indicate the difference between pattern runs and string runs).

The best compression results are achieved if the screen is completely filled with identical bytes. The /HR format stores this screen in 19200 bytes, /CHR requires 480 bytes (2 bytes for each line), /SHR is satisfied with just 304 bytes (19200 bytes is 151 runs of 127 bytes and one run of 23 bytes).

Because Monosave works line-by-line the /SHR format is not supported.

The /BLK Format
Cropped Images from Pro-Draw

The /BLK format that is used by the Pro-Draw program and others differs from all other formats described above in two ways: it can contain more than one image, and the images can be cropped (that is: they can be smaller than the full screen). These unique properties were the reason to include this format in the types supported by Monosave.

Each /BLK file consists of a series of image blocks, and is terminated by a pair of 00h bytes. Each block starts with a 12-byte header:

Bytes 00 and 01 contain the length of the block in bytes, including the header, but excluding these two bytes themselves. Byte 00 is the least significant, byte 01 the most significant.

Bytes 02 to 08 contain the name of the block (7 characters,padded with blanks).

Bytes 09 and 10 contain the width of the image, in pixels, minus one. Byte 09 is the low order byte, byte 10 is the high order byte.

Byte 11 contains the height of the image, in rows, minus one.

In the Monosave program this header is written by the **blkhead** function:

The rest of the block consists of actual image. The bytes as they are read line-by-line from hires memory in the same way as the /HR file format, but only for the selected part of the image.

```
blkhead (x0,y0,x1,y1,outfile,filename)
FILE *outfile;
unsigned int x0,y0,x1,y1;
char *filename;
{
    unsigned int length;
    char BLKheader[12];
/* calculate length of block, except for the two length bytes */
    length = (y1 - y0 + 1) * (x1 - x0 + 1) + 10;
    BLKheader[0] = length & 255;  /* low byte */
    BLKheader[1] = length >> 8;   /* high byte */
    /* store blockname in next seven bytes */
    strncpy (&BLKheader[2],filename,7);
    /* store number of columns in bits, minus one */
    length = (x1 - x0 +1) * 8 - 1 ;
    BLKheader[9] = length & 255;
    BLKheader[10]= length >> 8;
    /* store number of rows,minus one */
    BLKheader[11]= y1 - y0;
    /* write header */
    fwrite (BLKheader,1,12,outfile);
}

blkfile (outfile,x0,y0,x1,y1,outbuf,filename)
FILE *outfile;
unsigned int x0,y0,x1,y1;
char *outbuf;
char *filename;
{
    unsigned int line,lengthb;
    length = x1 - x0 + 1;
    /* write header */
    blkhead (x0,y0,x1,y1,outfile,filename);
    /* loop over all lines */
    for (line=y0;line<=y1;++line)
    {   loadline (line,x0,x1,outbuf);/* load current line
into buffer */
        fwrite (outbuf,1,length,outfile);/* write buffer to
the file */
    }
    /* write terminator to /BLK file */
    putc (0,outfile); putc (0,outfile);

}
```

These blocks are repeated for every image in the file. In order to indicate the end of the file a pair of two 00 bytes are appended, indicating a block with a length of 0 bytes:

Note: the format specification of /BLK file allows extended (1024 by 255 pixel) images to be stored in this format, and this feature is supported by the Monosave program if both the X and B parameters are specified on the command line. If you use this feature keep in mind that not all programs that can accept /BLK files will handle this large image size correctly.

Other TRS-80 Graphics Formats

In addition to the formats described above, there may be some other file types in use. Is anyone out there willing to describe these in a next issue of TMQ ?

Alien formats

If you think that the difference in hires file formats for the TRS-80 is confusing: the situation outside the TRS-80 world is even worse. Nearly every computer type and graphics program uses it's own format.

The following sections describe the most common of these formats.

MacPaint

One of the first formats defined for personal computers is the MacPaint format. While this format was originally designed for the Apple Macintosh, it is also used by MS-DOS computers. MacPaint images are always 576 pixels (72 bytes) wide and 720 pixels high - a full printed page when printed at 75 dots per inch. MacPaint files store images using the RLE technique. Apart from the image itself, the MacPaint files contain also 38 filling patterns used by the original MacPaint program, and a 'MacBinary' header. This 128-byte header is added to the beginning of any file when it is transferred from a Mac to any other computer type. It holds the information that is normally stored in the directory on a Macintosh disk (long file name, file type, creation date, etc.), and in the 'Resource Fork' of a Mac file. Note: some MacPaint files found on bulletin boards do not contain that MacBinary header, they are known as 'headerless MacPaint' files.

The current version of Monosave does not support MacPaint files.

GEM/IMG - for Ventura and Atari

The GEM/IMG format is mainly used by the Ventura Publisher program and Atari computers. This format combines efficient storage and flexibility. The file consists of a 16 byte header, followed by the image itself. The header holds eight 16-bit words in Intel format (least significant byte first, the same sequence as used in the Z-80) that have the following meaning:

The first three words must be 0001H, 0008H and 0001H, indicating the IMG file type (Note: there are also color IMG files, in which these bytes are different. Who can tell me about their structure ?) The fourth word contains the 'pattern length', normally a 1, indicating that string runs and pattern runs will be based on patterns of 1 byte long. The 5th and 6th word contain the size of the pixels (width and height), in microns (one millionth of a meter). A commonly used value is 0055H or 85 microns, corresponding to 1/300th of an inch - the resolution used by most laserprinters. The last two words in the header define the size of the picture: the width and the height (in pixels, not in bytes).

The image itself follows directly after the header. This image is constructed out of 'packets'. The monochrome IMG format uses 4 different packet types.

Because an IMG file uses the RLE image compression technique, it needs a **pattern run packet** to represent repeating patterns. This packet consists of a 00h byte, followed by a byte containing the length of the run (the number of times that the pattern is repeated along the line). The rest of the pattern packet consists of the bytes that make up the actual pattern (as many bytes as indicated by the 'pattern length' word in the header, normally just one byte).

The image parts without repeating byte patterns are stored in **string packets**. These string packets consist of a 80H byte, a count byte indicating the number of bytes in the string run, and finally the string of bytes themselves.

In addition to these basic packets, the IMG format uses additional packets to achieve even better compression. The 'solid run' packets are each only one byte long, and are used to indicate completely white or completely white sequences of bytes. The structure is simple: any byte that is not a 00H or 80H represents a **solid run**. If the most significant bit is set (values above 7FH) the packets indicates black bytes, otherwise it is a white solid run. The length of run is found by ignoring the most significant bit by ANDing the byte with 7FH: a 89H byte indicates a black run of 9 bytes.

Finally, there is a 'replication packet' that indicates that the next image row from the file must be repeated a number of times on the display or printer - a kind of RLE technique in the vertical direction. The structure of a replication packet is (in 4 bytes):

00h 00h FFh replication_count

Note: this replication packet is used by the MONOSAVE program to repeat each line in order to compensate for the 2-to-1 aspect ratio of the TRS-80 pixels. The user must request aspect ratio adjustment by specifying the A parameter on the command line.

GEM/IMG files are completely supported in MonoSave. In addition to this I have included a demonstration program IMGVIEW/BAS, which will read and display GEM/IMG files. You do not need GBASIC or BASICG to run IMGVIEW; the graphics hardware is accessed by IN

and OUT statements, not by special BASIC statements. Note that IMGVIEW/BAS contains only the core of a GEM/IMG display program: I have not included any error handling (for pictures that exceed the maximum size), and the user interface is very primitive. IMGVIEW contains at least one bug: the image does not start at the left hand side, but is shifted one byte (8 pixels) to the right. Fixing this bug and improving the program is left as an exercise for the reader ...

PCX - for MS-DOS hardware

The most common format for MS-DOS pictures is the PCX file created by ZSoft's PC Paintbrush programs. The structure of the PCX reflects the development of the MS-DOS graphics hardware: there are different versions for monochrome images, 16-color images and 26 color images. In order to store our TRS-80 pictures, only the monochrome version has to be considered.

A PCX file consists of a 128-byte header, followed the image data and (only for 256-color images) a color palette.

Let's take a look at the header structure:

The first byte indicates the manufacturer, and must be a A0H - if not, it is not a PCX file.

The second byte contains the version number: 0 indicates PC Paintbrush 2.5 (monochrome), a 2 or 3 comes from PC Paintbrush 2.8 (with or without palette information). A 5 indicates PC Paintbrush 3.0 or later and may contain up to 256 colors.

The third byte is assigned to indicate the encoding technique, but is always a 1.

The fourth byte contains the number of bits per pixel: 1 is monochrome, 4 is 16 colors (EGA) and 8 is 256 colors.

The next two words contain the x and y coordinates of the origin (upper left hand corner) of the picture, and are mostly set to 0. They are followed by another word

```
1 CLS:PRINT"IMGView reads monochrome GEM/IMG files used by
Ventura Publisher and Atari ST":PRINT"computers and dis-
plays the image on a TRS-80 model 4 with hires graphics."
2 PRINT"Created by Hans de Wolf - version 0.0.4 - June 16,
1991":PRINT
3 PRINT"Notice: this is an early development version -
program aborts on end-of-file":PRINT"and if image exceeds
1024 x 256 pixels. Some images may not display
correctly.":PRINT
5 DEFINT A-Z:LUNIMG=1:XPORT=&H80:YPORT=&H81:
DATAPORT=&H82:OPTIONPORT=&H83:OPTIONVALUE=&HB1
6 SIGNATURE$=
CHR$(0)+CHR$(1)+CHR$(0)+CHR$(8)+CHR$(0)+CHR$(1)
7 DEF FN WORD(ASTRING$,POSITION)=
256*ASC(MID$(ASTRING$,POSITION,1))+ASC(MID$(ASTRING$,POSITION+1,1))
8 DEF FN
GETWORD(LUN)=256*ASC(INPUT$(1,LUN))+ASC(INPUT$(1,LUN))
10 INPUT"Enter drive number or filename";FILE$:IF
LEFT$(FILE$,1)>="0" AND LEFT$(FILE$,1)<="7" THEN
SYSTEM"dir /img:"+LEFT$(FILE$,1):GOTO 10
20 OPEN "I",LUNIMG,FILE$
25 REPEAT.COUNT=1
30 HEADER$=INPUT$(16,LUNIMG)
40 IF LEFT$(HEADER$,6) <> SIGNATURE$ THEN PRINT"Not a GEM/
IMG file":CLOSE:END
50 PATTERN.LENGTH=FN WORD(HEADER$,7):  PRINT"Pattern length
= ";PATTERN.LENGTH
60 PIXEL.WIDTH   =FN WORD(HEADER$,9):  PRINT"Pixel width
= ";PIXEL.WIDTH
70 PIXEL.DEPTH   =FN WORD(HEADER$,11): PRINT"Pixel depth
= ";PIXEL.DEPTH
80 LINE.WIDTH    =FN WORD(HEADER$,13): PRINT"Line width
= ";LINE.WIDTH
90 IMAGE.DEPTH   =FN WORD(HEADER$,15): PRINT"Image depth
= ";IMAGE.DEPTH
100 'OUT XPORT,0:OUT YPORT,0:OUT OPTIONPORT,OPTIONVALUE
110 XBYTE=1:YBYTE=1:OUT XPORT,XBYTE:OUT
YPORT,YBYTE:BUFFER.SIZE=INT(LINE.WIDTH/8):IF (LINE.WIDTH
AND 7) THEN BUFFER.SIZE=BUFFER.SIZE+1
120 DIM BUFFER (1000)
122 DIM PATTERN(1000)
130 ABYTE=ASC(INPUT$(1,LUNIMG))
140 IF ABYTE<>0 THEN 300
141 'Packet type is 00. Check next byte to determine repli-
cation count or pattern run
150 ABYTE=ASC(INPUT$(1,LUNIMG))
160 IF ABYTE <> 0 GOTO 200
170 '=== REPLICATION PACKET
180 ABYTE=ASC(INPUT$(1,LUNIMG)):IF ABYTE<>255 THEN PRINT
TAB(40);"FATAL: x'FF' missing in replication packet":OUT
OPTIONPORT,0:CLOSE:END
190 ABYTE=ASC(INPUT$(1,LUNIMG)):REPEAT.COUNT=ABYTE:GOTO 130
200 '=== PATTERN RUN
210 RUN.LENGTH=ABYTE
220 FOR I=1 TO
PATTERN.LENGTH:PATTERN(I)=ASC(INPUT$(1,LUNIMG)):NEXT I
230 FOR I=1 TO RUN.LENGTH:FOR J=1 TO
PATTERN.LENGTH:ABYTE=PATTERN(J):GOSUB 700:NEXT J:NEXT I
250 GOTO 130
300 REM
310 IF ABYTE<>&H80 GOTO 400
320 '=== STRING RUN
330 RUN.LENGTH=ASC(INPUT$(1,LUNIMG))
340 FOR I=1 TO RUN.LENGTH:ABYTE=ASC(INPUT$(1,LUNIMG)):GOSUB
```

```
700:NEXT I:GOTO 130
400 'Solid run
410 BLACK.RUN=ABYTE AND 128:RUN.LENGTH=ABYTE AND &H7F:
420 FOR I=1 TO RUN.LENGTH
430 IF BLACK.RUN THEN ABYTE=255 ELSE ABYTE=0
440 GOSUB 700
450 NEXT I
460 GOTO 130
700 'add to buffer
710 XBYTE=XBYTE+1:IF XBYTE>BUFFER.SIZE THEN GOSUB
800:XBYTE=1:RETURN:ELSE BUFFER(XBYTE)=ABYTE:RETURN
800 '
810 FOR Z=1 TO REPEAT.COUNT
820    OUT OPTIONPORT,OPTIONVALUE
830    OUT XPORT,0
840    FOR ZZ=1 TO BUFFER.SIZE
850       OUT DATAPORT,(NOT BUFFER(ZZ))AND 255
860    NEXT ZZ:YBYTE=YBYTE+1:OUT YPORT,YBYTE
870 '    OUT OPTIONPORT,0:XBYTE=1
875 NEXT Z
880   RETURN
999 OUT OPTIONPORT,0
```

pair containing the coordinates of the lower right hand corner (639 and 239 for a TRS-80 image).

The two words that follow specify the horizontal and vertical resolution of the device that created the image, and are mostly ignored.

The next part of the header is intended for color images: 48 bytes specifying a the color palette for up to 16 colors, one currently reserved byte, and a byte specifying the number of color planes - not important for our purpose.

The next word eliminates some work: it specifies the number of bytes required to store one row of the picture.

Finally, there is a word specifying the palette type (if any: grey values or color).

The rest of the header is filled with dummy values to make up the required 128 bytes.

The rows in a monochrome are encoded by means of the RLE technique. A pattern run is indicated by a byte with it's most significant bits high. When such a byte is encountered, it must be ANDed with 3FH to obtain the actual repeat count that must be used for the next byte from the file.

There are no 'string run packets' in a PCX file, as there are in GEM/IMG files. Any byte without its two high bits set is simply stored into the picture. But what about image bytes that happen to have their two high bits set? Well, in order to prevent that they are interpreted as pattern runs you must put them into a pattern run with a length of one. An example: a completely white byte (FFH) will be interpreted as a run length indicator of 63 bytes. To prevent this, it must be stored as C1H (=run length 1) FFH (=the actual byte). This means that in some cases the storage algorithm will make the 'compressed' image larger than the original one !

Other formats

There are many other formats used in addition to the ones described above. More details about these and other formats (color PCX, TIFF, GIF) can be found in the book 'Bit-Mapped Graphics' by Steve Rimmer (ISBN 0-8306-58-0). The code of Monosave was developed using the information from that book, but no code was copied from it.

(Un)Documented Features of the Radio Shack Board

Some functions of the Monosave program use some special features of the Radio Shack hires board. Some of these functions are well known, others are documented but little known, and some are even undocumented in the manual. Because these special features may be interesting for other hires programmers, I will discuss them below.

Extended Images

The image displayed by the TRS-80 hires screen is 640 by 240 pixels, requiring 19200 bytes. This image is not stored in normal RAM or in the video RAM banks of the model 4, but in special memory on the hires board. Here is a surprise: the Radio Shack board contains much more memory than these 19200 bytes, it is equipped with a full 32 kbytes of memory. This feature of the Radio Shack board is documented in the Computer Graphics manual (26-1126), chapter 5: Programming the Graphics Board). What is not told in that manual, is that you can use this memory to store an image of 1024 by 256 pixels by using values between 0 and 127 for the X-address (Z-80 port 80H) and between 0 and 255 for the Y-address port (81H).

Note: this feature may not be supported by other hires boards.

Image Panning

The ability to store larger images may be interesting for programmers, but it is of little use if you cannot view these extended images. The solution to this is an undocumented feature of the R/S board: it allows image panning. This means that you can use the rectangle of 640 by 240 pixels displayed on the monitor as a window that you can shift over the 1024 by 256 image in memory. The way to do this is simple for programmers: writing a byte to two Z-80 ports with an OUT instruction shifts the image by the amount specified by the value of this byte. Port 140 controls the horizontal shift, port 141 the vertical shift. The maximum shift values are 48 bytes (384 pixels, more than a half hires page) in the horizontal direction, and 48

lines vertically.

Some graphical programs use this technique to display palettes of tools and patterns in the invisible part of the screen, and shift them into the visible part of the screen when needed. Because only a single OUT instruction accomplishes this operation it is much faster any other technique.

Note: if you plan to use these techniques in your own programs, remember that they may not work with all R/S boards. I expect that it will NOT work with the MicroLabs board because that board does not contain the full 32k RAM required to store a 1024 by 256 pixel image.

Text/graphics overlay

Another trick that you can use: it is possible to display the text screen and the graphics screen at the same time (at least, it is possible with the R/S hires board that I am using - others may not have this capability). The memory of the text screen and graphics screen are completely independent. The text screen memory is accessed as normal memory by the Z-80 after some memory bank switching, but the graphics video RAM can only be accessed via IN and OUT instructions (that is why an image in hires memory survives a reboot). However, the video circuitry in a model 4 accesses both kinds of video memory, and is capable of combining both images. This gives you three video modes: text only, graphics only and text and graphics both at the same time. As documented in the manual for the R/S board, the least significant bit of the options port 83H controls the first two modes: if this bit is a 0 the text screen is displayed, a 1 activates the graphics screen. The third mode is controlled by port 142: an OUT 142,1 instruction will combine both screens by means of a XOR-operation. If a pixel is on in either of the two screens it will light up on your display, if on or off on both screens it will be dark.

Using MonoSave

If you want to save a hires image by means of the MonoSave program, you must enter the command:

MONOSAVE filename (parameter,...)

If you do not specify an extension for the filename, MonoSave will supply one matching the format type. The parameters can be used to specify the file format and other options. Each parameter consists of a single letter, and can be specified in LDOS style "(A,X,I)", in UNIX style "-axi" or in VMS style "/A /X /I". Starting Monosave without any arguments displays the correct command syntax; specifying the H or ? parameter gives you a short help summary. More detailed information is given in the MONOSAVE/HLP file, and can be accessed with the LS-DOS help program (type HELP MONOSAVE).

All of the files used to compile monosave are provided as part of DISK NOTES 6.3 available separately. The complete list of monosave files on this disk is as follows:

BLK/C	MC source code for creating Pro-Draw /BLK files
CHR/C	MC source code for creating /CHR and /XHR files
HR/C	MC source code for creating /HR files
IMG/C	MC source code for creating GEM /IMG files
IMG/H	Header file for IMG/C
IMGVIEW/BAS	Demo program for displaying GEM /IMG files
MONOHELP/TXT	Source file for MONOSAVE/HLP (use with HELPGEN)
MONOSAVE/CCC	MC source for Monosave, #includes /H and /C files
MONOSAVE/CMD	Executable version of Monosave program
MONOSAVE/HLP	LS-DOS style HELP file for Monosave
PCX/CMC	Source code for creating /PCX files
PCX/H	Header file for PCX/C
TRS4KEYS/H	Header file for MONOSAVE/C, defines TRS-80 keys
TRSHIRES/C	MC source code accesses Hires hardware
TRSHIRES/H	Header file for TRSHIRES/C

by Frank Slinkman
1511 Old Compton Road
Richmond, VA 23233

with a forward by Roy Soltoff

PRO-MC High Resolution Graphics Library

Forward

When Tandy first introduced a high-resolution (hires) graphics interface for the TRS-80, it included a library of graphics subroutines usable from FORTRAN. Programmers had explored ways of utilizing the hires board from a different high-level language, such as C. Starting from scratch, designing and implementing low-level routines to plot points, draw lines and figures, etc., was a task more suited to an individual with dual talents: an expert programmer who also had an in-depth foundation in graphics algorithms. Not too many TRS-80 programmers fit the bill. I suspect that Ted Carter of MicroLabs developed a good deal of expertise on the hires graphics board, but nothing involving C-access was released publicly.

Harry Clayton was the first person, to my knowledge, to tinker with accessing the high-res graphics board from C. He developed, in 1986, a set of functions which provided an interface from the function calling conventions used in our MC compiler, to the subroutine linkage conventions used by Microsoft in their F80 FORTRAN compiler and associated GRLIB graphics library. This effort culminated in an article, HIRES Graphics for C which appeared in *The MISOSYS Quarterly* issue II.ii (Fall 1987). Thus, C programmers could make direct use of GRLIB from the C-level.

But adding another layer of subroutine interfacing is the last thing you want to do with graphics functions; speed is of the essence. Thus, the next approach was to have modified the graphics routines to directly interface with C using the C-function argument passing protocol. That never materialized.

Those of you who have followed the hires graphics world surrounding the TRS-80, know that in recent years, another graphics guru has materialized in the form of Frank Slinkman. Frank has always appeared to tackle the difficult and esoteric problems. His work with the XLR8er's 64180 processor has produced many highly specialized and successfully implemented programs. Not too many folks tackle that area. Frank also turned his attention to hires graphics. From graphics conversion programs, to graphics printing programs, to graphics scanner interfaces, he has produced some interesting stuff. Last I heard, he was working on a program to interface one of the new digital cameras to capture still photography directly into the TRS-80. That's a challenge! I expect him to one day tackle a software interface to a FAX board. What with low-cost pocket FAX boards which connect to a serial port, it should be a snap for Frank.

Somewhere between all of this activity, Frank found the time to develop for MC, a set of functions which can replace the block graphics functions. The graphics library functions are strictly for the Model 4-mode compiler. Obviously, the amount of code necessary to produce such a set of functions is much too much to print here. But I will provide a flavor of his work by printing a representative sample of the functions and corresponding programs which illustrate their use. All of the source appears on DISK NOTES 6.3.

Frank presents a method of revising your installation library - that's where MC's graphics functions are stored. I also want to provide another approach - one which creates a separate hires library accessible via a simple *option* statement. This method is also required for those using M80 who are without a librarian utility to manage relocatable libraries; such a librarian, MLIB, is bundled with our MRAS products.

Finally, with this new access to hires graphics from C, I expect to receive more input of graphics-rich programs as submissions for TMQ articles. Let's let Frank take a well-deserved break!

Integrated Library Installation Instructions

There are six accompanying /REL files which must be merged into the PRO-MC file, IN/REL. They are:

PAINT	contains rewritten paint() function;
PLOC6	contains rewritten ploc() function;
PMODE6	contains rewritten pmode() function;
PIXEL6	contains rewritten pixel(), point(), reset() & set() functions;
GFXCLEAR	contains new gfx_cls() and gfx_neg() functions; and
GFXMODE	contains new gfx_mode() function.

Editor's note: The six "/REL" modules are combined on DISK NOTES 6.3 as a single library module, HIRES/REL. To perform an integrated library installation, it is necessary to first extract each individual module by using MLIB. This can be performed by invoking MLIB, loading the HIRES library, then in turn, extracting each module. Note that module names are at most, six characters when stored in the library.

Installing these new and rewritten functions will remove and disable the old block character graphics functions. To install the new functions, all of the above files, plus IN/REL and MLIB/CMD must be available to the system at the same time.

NOTE: When making IN/REL available to the system, do NOT use your original Pro-MC master disk, but a backup, "working" copy instead.

A. From LS-DOS Ready, enter: MLIB

B. From the MLIB menu, select "L" (load library). Type [R] when prompted for type, and enter "IN" when asked for name. First, you must replace the existing members. Follow this procedure for each of the first four files listed above, in the SAME ORDER they are listed.

1. Select [R]eplace Module from the MLIB menu.

2. Enter the module name (without the "/REL") when prompted to do so.

3. Type [R] when prompted for type.

4. Enter the module name (sans "/REL") again when prompted for file name.

Now you must add the two new members. Follow this procedure for each of the remaining two files listed above:

1. Select [A]dd Module from the MLIB menu

2. Type [R] when prompted for type.

3. Enter the file name (sans "/REL") when prompted for file name.

Next we have to do a little "housekeeping" — namely changing the order of the members of IN/REL. Specifically, we have to change the location of the FREEME {freemem()} member. To do this, perform the following steps:

1. Select [E]xtract Module from the MLIB menu.

2. Enter the name "FREEME" when prompted for the element name.

3. Type [R] when prompted for type.

4. Enter the name "FREEMEM:d" when prompted for the file name (where ":d" is the drive number to which you want to write FREEMEM/REL).

5. Select [D]elete Module from the MLIB menu.

6. Enter the name "FREEME" when prompted for the element name.

7. Select [A]dd Module from the MLIB menu.

8. Enter "FREEMEM" when prompted for the element name.

9. Type [R] when prompted for type.

10. Enter "FREEMEM" when prompted for the file name.

Now select [S]ave from the MLIB menu, type [R] for type, and enter "IN" as the name, and [Y]es, replace. Now type [X] to return to LS-DOS Ready.

IN/REL on your working copy of Pro-MC now supports the new high resolution graphics plotting functions.

As a test, it is suggested you study, compile and run each of the accompanying test files:

GFXDEMO/CCC - utilizes all the new and modified functions except ploc() and pmode().

TPMODE/CCC - puts the pmode() function through it's paces. Since the new pmode() simply invokes the new ploc(), this is also a thorough test of ploc(), as well as circle().

TPAINT/CCC - puts the paint() function through it's paces.

SINEWAVE/CCC - doesn't test anything that hasn't been tested above, but elaborates on the example for the ploc() function.

Separate Hires Library Installation Instructions

The six accompanying /REL files mentioned above are in the HIRES/REL library on Disk Notes. This will be a separate library automatically searched when "#option HIRES" is specified. Here's the procedure if you use MRAS; adaptations to this procedure for M80 follow:

First load the MC/ASM file into SAID or your own text editor. Locate the line "IF @_INLIB". Insert three lines in front of this statement. They are:

```
IF      @_HIRES
$REQ    HIRES
ENDIF
```

Save that file, delete all, then load the MCMACS/ASM file. Locate the line near the end of the file which reads, "@_INLIB DEFL 0" and insert a line immediately prior to it which reads, "@_HIRES DEFL 0". Resave that file. You have completed your assignment; there is no need to resequence any module in IN/LIB.

Now any program in which you want to use the hires graphics functions, specify an #option HIRES in addition to any #option INLIB which may be required. Since the hires library will be searched prior to the installation library, any refer-

ences to the hires graphics functions with a name identical to the block graphics functions in the installation library will be satisfied by linking modules from the hires library. This requires you to use either hires or lores functions; because Frank has provided the functions with names identical to lores functions, you cannot intermix their use. If you really want to, you would have to carefully rename the modules in the source code, compile and/or assemble the source to relocatable modules, then build the library. But I don't personally see a reason for doing that.

If you are using M80 in lieu of MRAS, make the first insertion in the M80/H file. Make the second change in the MCMACS/MAC file.

Frank Slinkman's Pro-MC High Resolution Graphics Library

Function DOCUMENTATION

All existing Pro-MC plotting functions have been implemented, although one, ploc(), has been completely changed. The main thing to remember is that TRS-80 pixels have an aspect ratio of 1:2 (i.e., they are half as wide as they are high, or twice as high as wide).

Pixel addresses are determined by X and Y coordinates, where X runs left-to-right and Y runs top-to-bottom. Unlike the standard Cartesian system, the positive direction for Y is down, not up.

The normal screen has the upper-leftmost pixel at coordinates X=0,Y=0, and the lower-rightmost pixel at coordinates X=639,Y=239.

The following Pro-MC plotting functions are unchanged: box(), line(), pmode(), reset(), set().

The following Pro-MC plotting functions either require additional explanation, have been added, or are substantially changed:

circle(IN)

This function is unchanged. The radius of a circle is expressed in Y units, not X units. Thus a circle with a radius of 17 will be 35 (2 * r + 1) Y-pixels high and 69 (4 * r + 1) X-pixels wide.

paint(IN)

This function is unchanged. However, due to memory constraints, it is limited. If invoked to fill in an extremely complex area {e.g., every other pixel on the X-axis is set white on each even-numbered row, and the paint() "color" is set to 1 (white)}, paint() may fail, and fill in only part of the black area. A new error code, -2, will returned in this case. This limitation will rarely, if ever, be encountered in "normal" circumstances. The error code "-1" will be returned if the color is outside the range <0-1>; or if either the X coordinate or the Y coordinate is off the currently displayed screen.

pixel(IN) and point(IN)

These functions are unchanged. However, the return code of "-2", indicating "non-graphics character," is no longer applicable, and therefore is no longer implemented.

gfx_cls(IN)

This new function clears the graphics screen. See the example for ploc().

```
#option INLIB
 void gfx_cls();
```

gfx_mode(IN)

This new function sets the high resolution graphics display mode.

```
#option INLIB
int gfx_mode( funcod );
int funcod;
```

funcod:
- 0 = graphics off, text on
- 1 = graphics on, text off
- 2 = graphics on, text on (overlaid)
- 3 = return current graphics mode

Description:

High resolution graphics display is separate and distinct from normal text display. Thus, you can display either text, graphics, or both. The function codes 0, 1 and 2 determines which type, or "mode," of display is active.

Return Code:

A return code of "0", "1", or "2" indicates the current mode when a gfx_mode(3) function is invoked. A value of EOF (-1) is returned if "funcod" is not in the range <0-3>. A value of "-2" is returned if no high resolution graphics board is installed.

See the example for ploc().

gfx_neg(IN) (new function)

This function reverses (creates the photographic negative) of the current hi-res graphics display.

```
#option INLIB
 void gfx_neg();
```

See the example for ploc().

ploc(IN)

This function is substantially changed. ploc() no longer stores copies of the graphics screen image in memory, for reasons of memory size limitations. Instead, it

now changes the mapping of the pixels, allowing the plotting of pixels with either or both the X and Y coordinates normally "out of bounds" (i.e., off the screen).

The new operation of this function is as follows:

```
#option INLIB
void ploc( new_x,new_y );
int new_x, new_y;
```

new_x the X coordinate of the pixel to display at the upper left corner of the graphics display.

new_y the Y coordinate of the pixel to display at the upper left corner of the graphics display.

Description:

The ploc() function is useful for creating graphs and mathematical plots in which some coordinates will be either negative or otherwise off of the normal graphics screen. For example, the pixel at coordinates 0,0 is normally displayed in the upper left hand corner of the screen. Invoking ploc(-319,-119); will cause the pixel at coordinates X = -319,Y = -119 to be displayed in the upper left of the screen. This, in turn, causes the pixel at coordinates 0,0 to display at actual location 319,119, roughly the middle of the screen. This allows all pixels with X coordinates in the range -319 to +320 and Y coordinates in the range -119 to +120 to be displayed.

Example:

See sinewave/ccc.

```
/* sinewave/ccc */
#option     INLIB
#include    <stdio.h>
#include    <math.h>
main()
{   int     c, i;
    double  x, y, rad_to_deg = M_PI/180;

    gfx_cls();                  /* clear graphics screen */
    printf( "\x1c\x1f" );       /* clear text screen */
    gfx_neg();                  /* make background white */
    gfx_mode(2);     /* select graphics w/text overlay */
    ploc(-319,-119 ) /* put pixel 0,0 in center of screen */
    printf( "\n\x0f  Y = 100 * sin X (degrees)" );
    for (i = -320; i <= 320; i+=10 ) reset ( i, 0 );
    for (i = -120; i <= 120; i+=5 ) reset ( 0, i );
    for ( x = -270; x <= 270; x++ ) {
       y = 50 * -sin( x * rad_to_deg );
       reset( (int)x, (int)( y +.5 ) );    }
    cursor( 57,22 );
    printf( "\x0ePress any key to exit" );
    while ( ( c = inkey() ) == 0 );
       printf("\x1c\x1f" );
    gfx_mode(0);   /* turn hi-res graphics off, text on */
}
```

```
;PIXEL2/ASM - common code for SET, RESET, POINT graphics
routines - 04/19/92
;int pixel(code, X_coordinate, Y_coordinate)
;     int code, X_coordinate, Y_coordinate;
;Code can be one of the following:
;     0 means RESET the pixel
;     1 means SET the pixel
;     2 means test if pixel is SET or RESET
;Return codes:
;     0 if pixel reset
;     1 if pixel set
;     -1 if X or Y coordinate is out of bounds
;     -3 if illegal function code
;     Change log
; 09/27/83 - Modified - RS
; 11/04/84 - Added save of IX and IY - RND
; 07/20/85 - adapted to MRAS, merged DOS 5 and DOS 6 ver-
sions - RND
; 10/19/87 - changed PMODEA, PMODEB, PLOCA to @PMODA,
@PMODB, @PLOCA - RS
; 04/19/92 - adapted to hi-res boards, errcode -2 deleted -
jfrs
*GET MCMACS
MAX_X EQU   640          ;maximum x value
MAX_Y EQU   240          ;maximum y value
     PUBLIC PIXEL,PIX@,@PMODA,@PMODB
;*=*=*
*M
;*=*=*
;    Data Area
;*=*=*
     DSEG
PIX@ DC     6,0          ;<0-4>=storage, 5=CRT flag
;*=*=*
     CSEG
PIXEL PUSH  IY     ;xfer register variable to BC for now
     POP    BC
```

```
      LD    IY,PIX@       ;point IY to data
area
      POP   DE      ;get return address
      POP   HL            ;get CODE
      LD    (IY+4),L      ;save LSB
      LD    A,H    ;Save MSB for range test
      POP   HL            ;get X-coordinate
      LD    (IY),L        ;save LSB
      LD    (IY+1),H      ;save MSB
      POP   HL            ;get Y-coordinate
      LD    (IY+2),L      ;save LSB
      LD    (IY+3),H      ;save MSB
      PUSH  HL            ;restore stack
      PUSH  HL            ;to old status
      PUSH  HL
      PUSH  DE      ;put back return address
      PUSH  IX            ;now save both
register variables on stack
      PUSH  BC            ; (was IY)
      CALL  @PIXEL1     ;do rest of function
      POP   IY            ;restore register
variables
      POP   IX
      RET
;
@PIXEL1     OR    A       ;test msb of CODE
      JR    NZ,$?3        ;Illegal function
code if > 255
;
      LD    L,(IY)
      LD    H,(IY+1)      ;HL = X coord
      LD    DE,0   ;DE = offset from 0,0
@PMODA      EQU   $-2
      ADD   HL,DE         ;adjust X coord
;
      LD    DE,MAX_X
      LD    A,H           ;CP HL,DE
      CP    D
      JR    NZ,$?1
      LD    A,L
      CP    E
$?1   JR    NC,$?8        ;out of bounds if
X > 27FH
      PUSH  HL
;
      LD    L,(IY+2)      ;Y-coordinate
      LD    H,(IY+3)
      LD    DE,0
@PMODB      EQU   $-2
      ADD   HL,DE
;
      LD    DE,MAX_Y
      LD    A,H           ;CP HL,DE
      CP    D
      JR    NZ,$?2
      LD    A,L
      CP    E
      EX    DE,HL         ;DE = Y coord
$?2   POP   HL            ;X coord
      JR    NC,$?8
      LD    A,E
      OUT   (81H),A             ;Y-reg port
      LD    A,L    ;MSB of X-coordinate
      AND   7
      LD    B,A
      SRL   H
      RR    L             ;max = 13FH
      SRL   H
      RR    L             ;max = 9FH
      SRL   L             ;max = 4FH
      LD    A,L           ;p/u gfx board
byte column #
      OUT   (80H),A             ;X-reg port
      LD    C,1           ;init bit mask
      INC   B             ;range = 1 to 8
      RRC   C             ;bit mask for this
X-coordinate
      DJNZ  $-2
;
      LD    B,(IY+4)      ;get set/reset/
point code
      INC   B
      DEC   B             ;test for zero
      JR    Z,$?6         ;reset routine
      DEC   B             ;test for one
      JR    Z,$?4         ;set routine
      DEC   B             ;test for two
      JR    Z,$?7         ;point routine
;*=*=*
$?3   LD    HL,-3         ;illegal function
code
      RET                 ;return to caller
;*=*=*
;     SET pixel routine
;*=*=*
$?4   IN    A,(82H)       ;gfx data port
      OR    C
$?5   OUT   (82H),A
      LD    HL,0   ;no error return code
      RET                 ;return to caller
;*=*=*
;     RESET a point routine
;*=*=*
$?6   LD    A,C           ;p/u mask
      CPL                 ;reverse bits
      LD    C,A
      IN    A,(82H)
      AND   C             ;reset masked bit
      JR    $?5
;*=*=*
;     POINT routine
;*=*=*
$?7   IN    A,(82H)
      AND   C     ;is masked bit set?
      LD    HL,1          ;pixel is SET
      RET   NZ
      DEC   HL            ;pixel is RESET
      RET
;*=*=*
$?8   LD    HL,-1         ;set error code
      RET                 ;return
;
      END
```

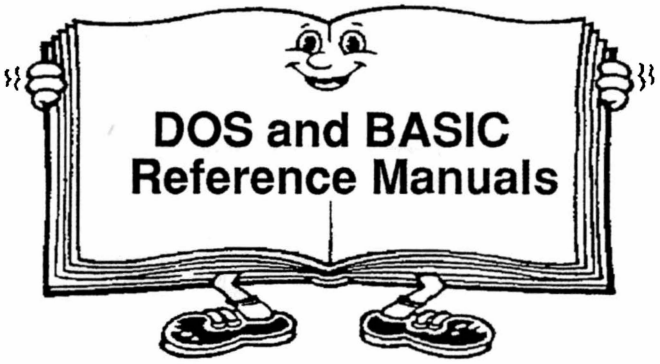
```
/* gfxdemo/ccc */
#option   INLIB
#include  <stdio.h>
#define   MAXFILES  3
#define   ARGS      0
#define   REDIRECT  0
#define   CLSputs("\x1c\x1f")
;
char *str1 = "\x0fThis graphics demo was
compiled using . . .";
char *str2 = ". . . with Frank Slinkman's
high resolution graphics library";
char *str3 ="\x0ePress any key to exit";

main()
{   int x, y;
    char  c;

    gfx_cls();   /* clear gfx screen */
    CLS;      /* clear text screen */
    gfx_mode(2);/* gfx on w/text overlay */
    dsp_string( str1, 18, 7 );
/* P */
    circle( 1,124,109,6 );
    circle( 1,124,109,10 );
    paint( 140,109,1 );
    line( 0,123,99,123,119 );
    paint( 120,117,0 );
    line( 1,85,99,85,134 );
    line( 1,85,134,94,134 );
    line( 1,94,134,94,119 );
    line( 1,94,119,123,119 );
    line( 1,94,115,123,115 );
    line( 1,94,115,94,103 );
    line( 1,94,103,123,103 );
    line( 1,85,99,123,99 );
    paint( 86,100,1 );
    gfx_neg();
/* R */
    circle( 0,204,109,6 );
    circle( 0,204,109,10 );
    paint( 218,109,0 );
    line( 1,203,99,203,119 );
    paint( 201,117,1 );
    line( 0,203,99,165,99 );
    line( 0,165,99,165,134 );
    line( 0,165,134,174,134 );
    line( 0,174,134,174,119 );
    line( 0,174,119,203,119 );
    line( 0,203,115,174,115 );
    line( 0,174,115,174,103 );
    line( 0,174,103,203,103 );
    line( 0,195,120,215,134 );
    line( 0,215,134,224,134 );
    line( 0,224,134,204,120 );
    paint( 166,100,0 );
    paint( 200,120,0 );
    gfx_neg();
/* O */
    circle( 1,275,114,11 );
    circle( 1,275,114,15 );
    circle( 1,275,119,11 );
    circle( 1,275,119,15 );
    paint( 275,101,1 ); paint( 275,133,1 );
    paint( 250,117,1 ); paint( 300,117,1 );
    reset( 254,116 );   reset( 254,117 );
    reset( 296,116 );   reset( 296,117 );
    set( 255,116 );  set( 255,117 );
    set( 295,116 );  set( 295,117 );
    paint( 275,116,1 ); paint( 275,116,0 );
    gfx_neg();
/* - */
    box( 0,330,115,379,119 );
    paint( 331,116,0 ); gfx_neg();
/* M */
    line( 1,405,99,414,99 );
    line( 1,414,99,434,113 );
    line( 1,435,113,455,99 );
    line( 1,455,99,464,99 );
    line( 1,464,99,464,134 );
    line( 1,464,134,455,134 );
    line( 1,455,134,455,106 );
    line( 1,464,99,435,120 );
    line( 1,434,120,405,99 );
    line( 1,414,106,414,134 );
    line( 1,414,134,405,134 );
    line( 1,405,134,405,99 );
    reset( 462, 100 );  reset( 406, 100 );
    paint( 434,119,1 ); gfx_neg();
/* C */
    circle( 0,519,116,13 );
    circle( 0,519,116,17 );
    circle( 0,519,117,13 );
    circle( 0,519,117,17 );
    paint( 489,117,0 );
    line( 1,543,112,552,112 );
    line( 1,543,121,552,121 );
    paint( 549,116,1 ); gfx_neg();
    dsp_string( str2, 10, 15 );
    cursor( 29, 20 );
    box( 1,74,66,561,165 );
    line( 1,75,66,75,165 );
    line( 1,562,66,562,165 );
    sleep(5); printf("%s", str3 );
    while ( ( c = inkey() ) == 0 );
       CLS;
    gfx_mode(0);
}

dsp_string ( string, column, row )
    char  *string;
    int column,row,i,j,k,x1,y1,x2,y2, y3;
    j = strlen(string);
    y1 = 10 * row;
    y2 = y1 + 9;
    y3 = y1 + 1;
    cursor( column, row );
    for ( i = 0; i < j; i++ )
    {  x1 = 8 * column++;
       x2 = x1 + 7;
       box ( 1, x1++, y1, x2, y2 );
       paint( x1, y3, 1 );
       putchar( string[i] );
       paint( x1, y3, 0 );
    }
}
```

Choose LDOS 5.3.1 or LS-DOS 6.3.1

Something for everyone

- ☆ Both Model I and Model III LDOS support similar commands; DOS commands are virtually similar to Model 4 LS-DOS 6.3.1 syntax where possible.
- ☆ The DATE command, "Date?" prompt on boot, and the @DATE SVC now support a date range of 32 years; from **January 1, 1980 through December 31, 2011.**
- ☆ **Enable or disable the printer time-out** and error generation with SYSTEM (PRTIME=ON!OFF)
- ☆ Both ASCII and hexadecimal display output from the LIST command is **paged a screen at a time.** Or run it non-stop under your control.
- ☆ MEMORY displays (or prints) the status of switchable memory banks known to the DOS, as well as a **map of modules** resident in I/O driver system memory and high memory.
- ☆ Specify SYSTEM (DRIVE=d1,SWAP=d2) to **switch drive d1 for d2.** Either may be the system drive, and a Job Control Language file may be active on either of the swapped drives.
- ☆ The TED text editor has commands to **print the entire text buffer**, or the contents of the first block encountered. Obtain directories from TED, too!
- ☆ Have extended memory **known to the DOS**? The SPOOL command now permits the BANK parameter entry to range from 0-30 instead of 0-7.
- ☆ **Alter the logical record length** of a file with "RESET filespec (LRL=n)"
- ☆ Specify "RESET filespec (DATE=OFF)" to restore a file's directory entry to the old-style dating of pre-6.3 release. Specify "RESET filespec (DATE=ON)" to establish a file's directory date as that of the **current system date and time.**
- ☆ SYSTEM command supports removable and reusable BLINK, ALIVE, and UPDATE memory modules.
- ☆ **Double-density BOOT support for Model I** with embedded SOLE and FORMAT (SYSTEM). Supports mirror-image backup, too. Reworked FDUBL driver eliminates PDUBL and RDUBL and takes less memory; enhanced resident driver eliminates TWOSIDE.
- ☆ Model III version auto-detects Model 4 for installation of KI4 keyboard driver; supports CAPS, CTRL, and function keys.
- ☆ The SPOOL command offers Pause, Resume, and Clear parameters. (OFF) attempts to reclaim memory used.
- ☆ Both Model I and Model III support similar commands: all features of Model III 5.3.0 are in Model I 5.3.1. That includes such facilities as DOS and BASIC help files, SETCOM and FORMS library commands, TED text editor, BASIC enhancements, etc. All DOS commands have been groomed for Model 4 LS-DOS 6.3.1 syntax where possible.
- ☆ Felt uncomfortable with the *alleged* protection scheme of 6.3? **LS-DOS 6.3.1 has no anti-piracy protection!** Neither does LDOS 5.3.1. MISOSYS trusts its customers to honor our copyrights.
- ☆ Best of all, **a 5.3.1 or a 6.3.1 diskette is available as a replacement for your 5.3.0 or 6.3.0 diskette for $15** (plus $3 S&H in US and Canada, $4 elsewhere). There's no need to return your current master.
- ☆ The 5.3.1 or 6.3.1 diskette(s) come(s) with a 30-day warranty; written customer support is available for 30 days from the purchase date. Versions of 5.3.1 for the Model I and Model III are available. Versions of 6.3.1 for the Model 4 and Model II/12 are available; Model 4 French and German versions are also available (specify 6.3.1 F or 6.3.1 G). **If you do not already have an LDOS 5.3.0 or LS-DOS 6.3.0, order the 5.3.1 or 6.3.1 Upgrade Kit with 90 days of customer support for $39.95** (+$4 S&H). Some Model I 5.3.1 features require lower case or DDEN adaptor.

MISOSYS, Inc.
P. O. Box 239
Sterling, VA 20167-0239
703-450-4181
[orders to 800-MISOSYS (647-6797)]



Two new reference manuals are available from MISOSYS. First, we have the the 349-page "LDOS™ & LS-DOS™ Reference Manual", catalog number M-40-060. This single manual fully-documents both LDOS 5.3.1 and LS-DOS 6.3.1 in a convenient 8.5" by 5.5" format. If you use one, or the other, or even both DOS versions, you may want to bring yourself up to date with a single manual. Gone are the many pages of update documentation. Price is $30 plus $5 S&H.

We also publish the "LDOS™ & LS-DOS™ BASIC Reference Manual". This 344-page book, catalog M-40-061, covers the interpreter BASIC which is bundled with LDOS 5.3.1 (even the ROM BASIC portion), the interpreter BASIC which is bundled with LS-DOS 6.3.1, and both Model I/III-mode and Model 4-mode EnhComp compiler BASIC. One convenient 8.5" by 5.5" manual covers all four BASIC implementations for $25 plus $3.50 S&H. Since this new manual covers our compiler BASIC, you can purchase the disk version of EnhComp for $23.98 plus $1.50 S&H when purchased along with a BASIC Reference Manual, or the disk version by itself for $29.98 plus $3 S&H if purchased separately.

MISOSYS, Inc.

MISOSYS sponsors a forum on CompuServe: PCS49

When you don't have to write in stone, don't let your editor weigh you down. You need SAID-86!

Editing was never so easy!

SAID-86 is a fast, flexible, full screen text editor for PC's. It is perfect for editing batch files, program listings, README files, CONFIG.SYS files, and anything you now do with EDLIN or the non-document mode of a word processor. Why struggle with huge editors; when all is said and done, SAID-86 will be your text editor of choice!

Check out this list of features

- ✓ WordStar-like editing commands are easy to use
- ✓ Pull-down menu system for commanding SAID-86
- ✓ Supports nine editing buffers with automatic swap to disk
- ✓ Supports up to 30 user-defined macros; 255 characters each
- ✓ Undelete the last nine deleted lines can save your bacon
- ✓ MOUSE support with automatic recognition
- ✓ HELP facility; shell to invoke DOS commands from SAID-86
- ✓ SAID-86 can expand or contract TABs

SAID-86 is reasonably priced at just $29.95 + $3S&H

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 20167-0239
800-MISOSYS
(US&Canada)
or 703-450-4181
M/C & VISA accepted.
S&H are U.S. only.

TRSCROSS

Now you can transfer TRS-80 Model III/4 files directly to your MS-DOS disks right on your PC. Convert BASIC programs; Convert SuperScripsit document files to DCA-RFT. **Only $89.95 + $4S&H**

HartFORTH-86™

HartFORTH is a Direct Threaded Code implementation full 79-STANDARD FORTH which runs under DOS; the Virtual Memory that it accesses for storage and retrieval purposes is a file created and controlled by the operating system. HartFORTH's enhancements include functions to call the DOS file handling routines so that other files may be created and accessed if required. A library of standard screens is supplied with HartFORTH to provide often used extensions to the language, such as double length and floating point math, editing of source screens, string manipulation, arrays, etc. **Priced at $59.95 + $5S&H**

- HartFORTH programs can invoke other programs via **EXEC** and **EXEC.PROG**.
- Functions create new files from within HartFORTH, and allow the current Virtual Memory file to be changed for another and manipulated at the individual block level.
- Provides the recommended 79-STANDARD DOUBLE NUMBER STANDARD EXTENSION word set that implements 32-bit operations.
- **CASE:** and **SWITCH:** functions allow multi-way branching decisions to be taken with execution continuing in-line once the word branched to completes.
- String manipulators include: `"VARIABLE`, `"CONSTANT`, `"!`, `"LEFT`, `"RIGHT`, `"MID`, `"+`, `"COMPARE`, and `">`.
- DOS software and hardware interrupt vector access support via: `GET.VECTOR`, `PUT.VECTOR`, `THIS.SEG`, `DI`, and `EI`.
- V24 program input, output, and interrupt input support.
- *Overlay management words:* `FORGET.OVLY`, `OVLYNAME`, `PUT.DATA`, `OVLY.ENTRY`, `SAVE.OVLY`, `CORRECT?`, `LOAD.OVLY`, `NEW.OVLY`, `RUN.OVLY`, and `LEAVE.OVLY`.
- Screens provide trigonometric functions: `SIN.`, `COS.`, `TAN.`, `SIN`, `COS`, and `TAN`.

Are you still fussing with floppies for BACKUP? CMS' DJ10 or DJ20 tape drive from MISOSYS is your solution!

The Colorado Memory Systems' JUMBO tape drives fit all computers. Internal mounting in AT's, XT's, and PC's, they connect to your floppy disk controller. Tape adaptor board needed when two floppies are in use. Kit converts Jumbo to external use.

- In about 5.5 minutes, a DJ10 backs up 10MB's file-by-file - the fastest in the industry! 40MB's gets backed up in about 18 to 20 minutes. Uses industry-standard DC2000/DC2120 tape cartridges.
- DJ10/DJ20 plugs into your floppy disk controller to save cost, power, and a slot. Needs 5-1/4" (or 3.5" with faceplate) mounting slot.
- Optional adapter board mounts in your computer to provide a tape port. When used with the external DJ10/DJ20, it lets you share your drive between computers. Note: external adaptor includes "Tape Adaptor"
- DJ10 has up to 120 megabytes of capacity using compression with a DC2120 tape; DJ20 has up to 250 megabytes of capacity.

DJ10 Jumbo	**$199 ($7S&H)**
DJ20 Tape drive	**$265 ($7S&H)**
A11 Adaptor	**$45 ($3S&H)**
K10 External Kit	**$110 ($5S&H)**
DC2000 tape	**$20.00**
DC2120 tape	**$25.00**

NEW LOWER PRICES

Why buy just a FAX board, when the ZOFAX 96/24 from MISOSYS includes a 2400 baud modem for a few bucks more? Turn your PC into a FAX machine!

- ✓ Send and receive FAX from any CCITT Group III Fax Machine or PC Fax
- ✓ Auto receive and print incoming Fax messages ✓ Background receiving
- ✓ Distribut Fax messages to multiple destinations ✓ Fax mail merge
- ✓ Time schedule transmission to take advantage of low nightime rates
- ✓ 2400 bps Fully Hayes Compatible Modem
- ✓ Includes powerful but easy to use BITCOM and BITFAX software

Further price reduction: Just $125 + $7 S&H

EXPANZ!™ Disk Expander Card

With the EXPANZ! data compression card, you can boost hard disk capacity up to three times. EXPANZ! plugs into any open slot and intercepts calls to and from the disk controllers. Compresses and decompresses in real time. Requires PC/XT/AT or compatible running DOS 3.x or higher. **Now Just $150 + $7S&H.**

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 20167-0239

800-MISOSYS
or 703-450-4181

TRSCROSS™

(Pronounced TRISS-CROSS)

TRSCROSS runs on your PC or compatible, yet reads your TRS-80 diskettes! Copy files in either direction!

The FASTEST and EASIEST file transfer and conversion program for moving files off the TRS-80™and over to MS-DOS (or PC-DOS) or back

TRSCROSS™
Copyright 1986, 1987 by MISOSYS, Inc.
All rights reserved

1 - Copy from TRS-80 diskette
2 - Copy to TRS-80 diskette
3 - Format TRS-80 diskette
4 - Purge TRS-80 diskette
5 - Display directory (PC or TRS-80)
6 - Exit

Shown above is the Main Menu displayed when running TRSCROSS on your PC or compatible.

TRSCROSS is as easy to use as it looks to be! The program is very straight forward, well thought out, and simple to operate. TRSCROSS has several "help" features built into the program to keep operation as easy as possible. Just pop your TRS-80 disk into your PC and copy the files right to your PC data disk or hard disk. *It couldn't be any faster or easier!* All steps are detailed in the instruction manual. Advanced features, for those that desire to use them include executing menu options right from DOS or from a batch file or macro. This can really speed up transfers when similar operations are performed frequently.

TRSCROSS allows you to "TAG" all files to be moved in ONE pass!

TRSCROSS converts TRS-80 BASIC programs and SuperSCRIPSIT files in ONE PASS while COPYing to MS-DOS!

No need to save your programs or files in ASCII or run a separate conversion program first before transferring. TRSCROSS reads your tokenized BASIC program or SuperSCRIPSIT files directly off your TRS-80 disk and performs the conversion all in ONE pass while being transferred directly to your PC or compatible computer. ***Automatically*** **converts most BASIC syntax,** and lines that need special attention can be listed to a printer. (Does not convert PEEKs, POKEs, graphics, machine language calls or subroutines.)

TRSCROSS will even FORMAT a TRS-80 disk right on your PC! (Handy for those who use both machines!) Former TRS-80 users who no longer have their TRS-80, but still have diskettes with valuable data... this is exactly what you've been waiting for!

TRSCROSS will READ FROM and COPY to the following TRS-80 double-density formats:
TRSDOS 1.2/1.3, TRSDOS 6.2*, LDOS 5.3*, DOSPLUS, NEWDOS/80*, & MultiDOS.

DOS formats listed above flagged with * signify that earlier versions of these DOS's are readable as well, but one or more sectors may be skipped due to a format problem in that version of the DOS. (Disks that were formatted with SUPER UTILITY™ or SUW4/4P™ do not have this problem.) TRSDOS 6.02.01, or higher should not have this problem. Disks formatted in any 5.25" 80 track format, or single density are not supported; 3.5" 720K disks are readable in a 720K 3.5" MSDOS disk drive.

TRSCROSS Requires: PC or compatible computer, 128K and a normal 360KB (40 track) PC or 1.2MB (80 track) AT drive. Double-sided operation is fully supported. If you have more than one disk drive, fixed drive, or RAM disk, operation will be much smoother. TANDY 1000 requires more than 128KB memory (DMA). TANDY 2000 is not supported at this time due to a difference in disk controller and floppy drives. "Special" data files (like PROFILE+) would need to be converted to ASCII on a TRS-80 first before they would be of use on a PC or compatible.

If you use both types of computers, or you plan to retire your TRS-80, this is for you. TRSCROSS will allow access to your TRS-80 diskettes for years to come. Copy your TRS-80 word processor data files as well as your Visicalc data files over to MS-DOS and continue using them with your new application.

Only $89.95

Plus $4 S&H (U.S.) or $5 Canada or $6 Foreign
Virginia Residents must add appropriate sales tax.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 20167-0239
Phone: 703-450-4181 (Orders only: 800-MISOSYS)

ZOFAX™ 96/24: Send and receive fax 9600 bps + 2400 bps Data Modem

Advanced Features:

Automatically receives and prints incoming fax messages.
Working like your regular fax machine, BitFax can print incoming fax messages directly to the printer or save them onto disk.

Flips/rotates incoming fax pictures.
If the sender sends a page upside down, BitFax lets you flip the page so you can see it on the screen. If the page is sent sideways, you can rotate the page 90° so you can view it on the screen.

Time scheduled transmission.
You can schedule to send a fax at night or any time when the phone charges are less expensive.

Background receiving.
While receiving a fax message, BitFax lets you run other programs such as copying a file, viewing a file directory or running word processing software.

Fax mail merge.
Mail Merge lets you fax customized letters to several people with their own name, address, or company information merged into a form letter. The names, companies, addresses or other information is pulled from the phone book database which is a dBase format. You can create a comprehensive database with dBase and use the data for the Fax Mail Merge. Doing so will save you time and get impressive results from your recipients.

Broadcast and distribution list management.
BitFaX lets you send a message to several people selected from the phone book. You can also assign names into groups and just send to the selected groups without having to key in or select their numbers each time you send.

Features and Benefits:

Integrated data modem and send/receive Fax capability on a single board.
Auto-answer: the modem will detect whether it's data or fax.
Supports Multiple File Formats. You can send any of these file types directly from your PC to any Group III Fax machine: ASCII (text), PCX (pc Paintbrush), IMG (GEM Artline), TIFF (scanner) and FAX (received from proFAX PC Fax card). Color PCX Files will be converted to gray scaling.
Auto redial.
Sends Multiple Files. So you can combine more than one file to be sent to a fax machine. For instance, you might choose a picture file with graphs or logos, and combine it with text message, and then send every thing with one phone call. Doing so will reduce phone charges, as well as save the time needed to make multiple calls.
Flexible phone number management. You can add, change, delete and sort phone numbers easily with BitFax. The BitFax Phone Book also allows you to store company names and addresses. You can group sets of phone numbers for distribution lists. The Phone Book Database is stored in a dBASE III type format (.DBF file name extensions), so you can access with dBASE or other utility program.
Software selective for data modem and Fax mode - software automatically switches between data modem or fax.
"AT" command set compatible for use by most communications software.
Low transmission cost - schedule to send fax at night rate.
Fax's are better quality - No degradation due to printer quality or scanner quality.
Distribute fax messages to multiple destinations.
Highest speed and lowest cost solution by combining 9600 bps fax and 2400 bps data modem on one board.

Technical Specifications:

Compatibility: Data modem
300 bps Bell 103 and CCITT V.21
1200 bps Bell 212A and CCITT V.22
2400 bps CCITT V.22

Compatibility: Facsimile
Send and receive Fax from CCITT Group III fax Machines. Fax format compatible with CCITT T.30 and T.4 Group III Fax 9600 bps CCITT V.29 ter with fall back to 7200, 4800, 2400, CCITT V.27 ter CCITT V.21 (Synchronous)

Command language.
"AT" command set compatible with extensions provided for fax operations.

Data format.
Asynchronous operation supports 7 or 8 data bits, plus start, stop and parity bits.

Operations.
Full or half duplex with pulse and tone dialing. Dial up or leased phone mode with auto answer mode.

Audio Monitor.
On-board monitoring with software speed control.

Line equalization transmit.
Adaptive equalization for PSK/QAM mode.

Test and diagnostics.
Install and Test program
Self Diagnostics

Modular phone jacks.
Two RJ11C phone connectors for phone and line.

Approval.
FCC part 15, Class B; FCC part 68

Warranty.
1 year warranty.

The ZOFAX™ 96/24 Fax/Modem gives you the best value at an affordable price. It's a 2400 bps fully Hayes compatible modem with an integrated 9600 bps send/receive fax that is compatible with any Group III fax machine. Now you can turn your PC into a fax machine with the ZOFAX™ 96/24 Fax/Modem.

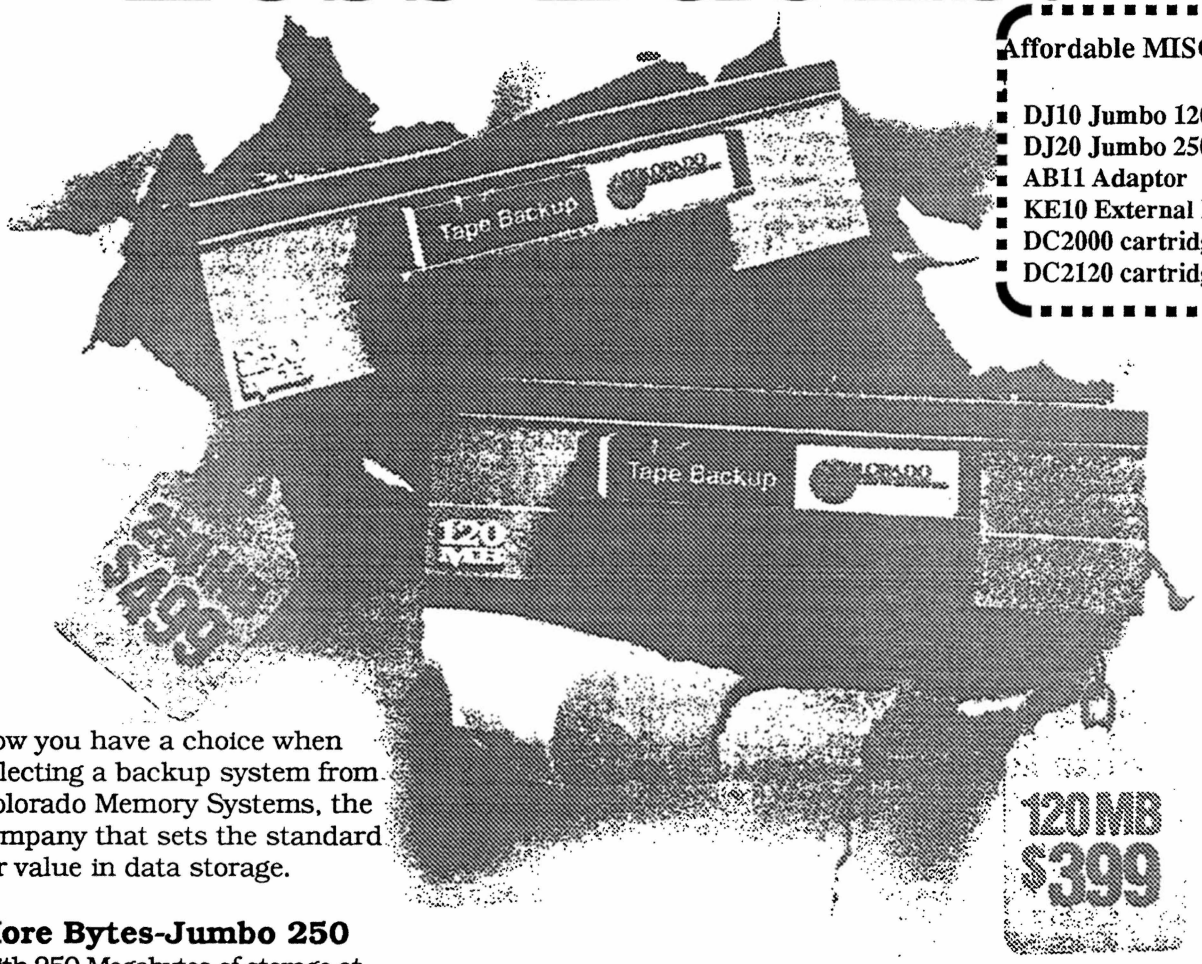
Just $125
+ $7S&H

Zoltrix
Solely For Tomorrow's Technologies

Products available from MISOSYS, Inc.

MISOSYS, Inc
PO Box 239
Sterling, VA 20167-0239
703-450-4181
800-MISOSYS (647-6797)

More Bytes, Less Bucks.



Affordable MISOSYS pricing

DJ10 Jumbo 120	$199+$7S&H
DJ20 Jumbo 250	$265+$7S&H
AB11 Adaptor	$45+3S&H
KE10 External Kit	$110+5S&H
DC2000 cartridge	$20
DC2120 cartridge	$25

Now you have a choice when selecting a backup system from Colorado Memory Systems, the company that sets the standard for value in data storage.

More Bytes-Jumbo 250

With 250 Megabytes of storage at $499 SRP ($265 at MISOSYS) you can save more while spending less. Jumbo 250 delivers superior performance while using all applicable QIC industry standards for format and data compression.

Less Bucks-Jumbo 120

At $399 SRP ($199 at MISOSYS) Jumbo 120 continues to provide *jumbo performance for peanuts.* With 120 Megabytes and QIC compatibility, Jumbo 120 is the backup of choice when less storage is needed,

Jumbo 250 and Jumbo 120

Both work with IBM® PC, XT, AT, 386™, 486, PS/2™ and compatibles and the price includes software that will let you backup nights and weekends with an unattended scheduler. Network compatibility is standard, you can also get optional Xenix®/UNIX® software offering an easy to use menu system, another first from Colorado Memory Systems. While both drives use the standard floppy controller, a variety of optional jumperless controller cards offer easy to install performance enhancements.

COLORADO MEMORY SYSTEMS INC.

Available from:
MISOSYS, Inc.
PO Box 239
Sterling, VA 20167-0239
800-MISOSYS

IBM PS/2, Xenix, Unix are trademarks of their respective companies. 386 is a trademark of Intel Corp.

Let our LB Data Manager solve your data storage problems

LB Version 2.2: A Flat File Data Manager with more powerful and easy to use features in this latest enhancement of Little Brother! Now with data conversion utility for DIF, pfs, Profile, dBASE, ...

We've added many features asked for over the past few years by LB users; yet LB is still about the easiest, most flexible data manager you can use for managing your data. Absolutely no programming is needed to create a database with numerous fields, construct input screens for adding and editing data, and create your own customized report. Quickly you define your data fields in response to LB's prompts, and then draw your data input screen using simple keystrokes - or have LB automatically create your input screen. In no time at all, you're entering data. Customize your printed reports with user-definable print screen definitions. LB is just what you need in a data manager! Now even more in version 2.2!

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

Field types supported:
LB allows ten field types for flexibility: *alphabetic* {A-Z, a-z}, *calculated* {operations on "numeric" fields using +, -, *, /, with 2-level parentheses}, *date last modified* {YYYY/MM/DD automatically maintained}, *dollar* {±ddddddd.dd}, *floating point* {±ddddddd.dddddddd, *literal* {any ASCII character}, *numeric* {0-9, -, .}, *right-justified numeric*, *upper case alphabetic* {A-Z, automatic conversion of a-z}, and *upper case literal* { literal with automatic conversion of a-z}. All field types utilize input editing verification so invalid data cannot be added to a record. Field name strings can be up to 19 characters long.

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

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

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

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

Maintenance utilities:
To make it easy for you to grow your database as your data needs grow, we provide two utility programs for managing your database. One allows you to construct a new database with an altered data structure and populate it with data from your existing database Another allows you to duplicate your database structure, copy or move records from one to another, or automatically purge un-needed records. **A third utility converts to LB from pfsFILE4, Profile4, DIF, dBASE II&III, and fixed record; also converts to DIF, dBASE, and delimited.**

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

Trade-up policy:
Send in an original Table of Contents page from any existing database program and get LB Version 2 for half price.

Specify MS-DOS or TRS-80 version. LB is priced at $99 + $5 S&H (US; $6 Canada; $7 Europe; $9 Asia, Pacific Rim, and Australia). To trade up from any other database, send Table of Contents page and $49.50 +S&H. Remit to:

MISOSYS, Inc.
PO Box 239
Sterling, VA 20167-0239
703-450-4181 or orders to 800-MISOSYS

MISOSYS, Inc.

With a 20 or 40 MB MISOSYS Hard Drive connected to your TRS-80 Model III or 4, your computer will sail through data access.

Order any hard drive kit or unit from MISOSYS and we'll pre-install either LS-DOS 6.3.1 or LDOS 5.3.1 at no extra

MISOSYS has been shipping complete drive kit packages since September 1989 which plug into Model 4/4P/4D and Model III computers; let us build one up for you! Our host adaptor, which interfaces the 50-pin expansion port of the TRS-80 (host) to the 50-pin SCSI port of the HDC, sports a hardware real time clock using a DS1287 clock module. With its internal battery lifetime in excess of 10 years, never enter date and time again. It even adjusts for daylight saving time! An available option is a joystick port and Kraft MAZEMASTER joystick with a port interface identical to the old Alpha Products joystick; thus, any software which operated from that joystick will operate from this one.

Software supporting the S1421 and 4010A controllers includes: a low level formatter; an installation utility and driver; a high level formatter; a sub-disk partitioning utility; utilities to archive/restore the hard disk files onto/from floppy diskettes; a utility to park the drive's read/write head; a utility to set or read the hardware clock; a keyboard filter which allows the optional joystick to generate five keycodes; and a utility to change the joystick filter's generated "keystroke" values after installation. Optional LDOS 5.3 software is available.

Twenty megabyte drive packages are currently built with a Seagate ST225 hard drive; Forty megabyte packages use a Seagate ST251-1 28 millisecond drive. Drive packages are offered as '*pre-assembled kits*'. Your 'kit' will be assembled to order and fully tested; all you will need to do is plug it in and install the software. Drive kits include a 50-pin host interface cable and the hardware clock. Full implement of status lights included: power, ready, select, read, and write. Add a joystick for but $20 additional (see price schedule).

Aerocomp Hard Drives now available from MISOSYS

MISOSYS is also the sole source of remaining brand new Aerocomp hard drives. A limited number of NEW 5 megabyte units are available. All Aerocomp drives include status LEDs, software driver and formatter, power and host cables, and installation Job Control Language. We are also building their 20M and 40M drives.

Prices currently in effect:

Complete MISOSYS Drive Kits:	
20 Megabyte kit:	$450
40 Megabyte kit:	$575
Joystick option	$20
LDOS software interface	$30
Aerocomp Hard Drives:	
5 Meg unit	$250
20 Meg unit	$400
40 Meg drive	$500
MISOSYS H/A with software	$75
Xebec 1421 HDC	$75
Adaptec 4010 HDC	$75
Drive power Y cable	$5
XT drive cable set	$5

Note: freight charges are additional.
Prices subject to change without notice.

MISOSYS, Inc.
PO Box 239
Sterling, VA 22170-0239
U.S.A.

Contents: Printed Matter

BULK RATE
U. S. POSTAGE
PAID
Sterling, VA
PERMIT NO. 74

Effective 7/1/92, our ZIP code will be 20167-0239

Attention Postmaster: Address Correction Requested
Forwarding and return postage guaranteed