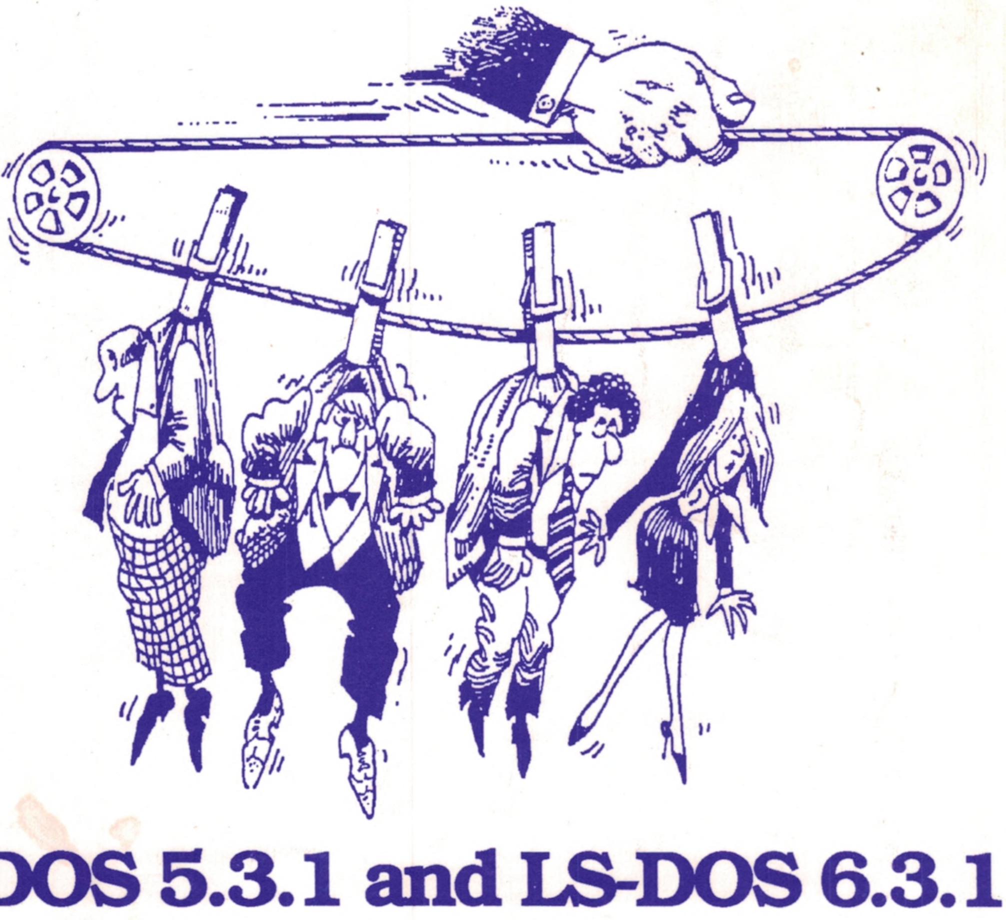
Received 30/12/91

THE MISOSYS QUARTERLY

Don't be hung out to dry



LDOS 5.3.1 and LS-DOS 6.3.1 are good through 2011

Volume Vi.i **$10** **Autumn 1991**

PRICE LIST effective September 1, 1991

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
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
GO:Maintenance	n/a	M-33-100	$49.95 B
GO:System Enhancement	n/a	M-33-200	$49.95 B
GO:Utility	n/a	M-33-300	$49.95 B
Hardware Interface Kit	n/a	M-12-110	$24.95
HartFORTH/PRO-HartFORTH	M-20-071	M-21-071	$49.95 B
LDOS 5.1.4 User Manual	L-40-020	n/a	$15.00 D
LDOS 5.3.1 Mod1 Upgrade kit	M-10-133	n/a	$39.95
LDOS 5.3.1 Diskette - M3	M-10-130		$15.00
LDOS 5.3.1 Mod3 Upgrade Kit	M-10-333	same	$39.95
LED / LS-LED	L-30-020	L-30-021	$19.95
LB Data Manager-M4 (Ver 2.1)	n/a	M-50-510	$99.00 D
LS-DOS 6.3.1 Upgrade Kit - M4	n/a	M-11-043	$39.95
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
PRO-WAM	n/a	M-51-025	$74.95 D
PRO-WAM Toolkit	n/a	M-51-225	$29.95
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
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
UNREL-T80	same	M-30-054	$29.95
UTILITY-I	L-32-070	n/a	$19.95

MSDOS Software

LB Data Manager 2.1	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
FM-86 (File Manager)	L-86-050	$29.95

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

TeleTrends TT512P modem (M4P)	H-4P-512	$74.95	E
XLR8er e/w 256K RAM (M4): board only	R-MB-004	$100.00	F
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
Hard drive kit e/w clock, 20Meg M3/M4	H-HD-020	$450.00	?
Hard drive kit e/w clock, 40Meg M3/M4	H-HD-040	$575.00	?
Hard drive joystick port option	H-HD-JSO	$20.00	
Hard drive: Kalok KL320	**R-HD-020**	**$200.00**	**F**
Hard drive: Seagate ST251-1	**R-HD-040**	**$320.00**	**F**
Hard drive: Seagate ST157A (16B IDE)	**R-HD-A40**	**$320.00**	**F**
Hard drive: Seagate ST-157N (SCSI)	**R-HD-S40**	**$350.00**	**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	$175.00	F
Infochip Systems Expanz! (PC)	R-IC-EXP	$179.10	F
DJ10 Tape Backup (PC)	R-TD-D10	$275.00	D
AB10 Tape Adaptor (PC)	R-TD-A10	$75.00	
KE10 External tape adaptor/case (PC)	R-TD-K10	$110.00	D

The Fine Print

Freight codes: A = $3.50; B = $4.00; C = $4.50; D = $5.00; E = $5.50; F = $6.00; G = $8.50; 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 22170-0239
703-450-4181; Orders only: 800-MISOSYS (800-647-6797)

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 for development of the "camera ready" copy consists of an AST Premium/386 computer (20 MHz) equipped with 9 Megabytes of RAM, a Seagate ST4096 80-Meg HD, ST251 40M, Expanz! card; a CMS DJ10 tape backup device, a NEC Multisync II color 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 1.0; Submissions on paper and letters are scanned and converted to text using Read-Right optical character recognition software by OCR Systems. Final page composition is developed using PageMaker 4.0 by Aldus. Cover art and clip art comes from CLIPPER, a product of Dynamic Graphics.

Table of Contents

List of Advertisors

J.F.R. Slinkman	48
MISOSYS, Inc.	22,44,48,IFC,IRC,RC
Pacific Computer Exchange	32
Richard R. King	44
TRSTimes magazine	47

List of Patches in this Issue

LBOV51/FIX	8
LBREDEF.EXE	9
LB/OV5 and LB/OV8	10
GOFED2A/FIX	18
OOPS/APP	19
WLINK1/FIX	19
DISKCOPY	21

Points to Ponder

The Blurb by Roy Soltoff

Motorola and Fujitsu are engaged in a war of which company manufactures the smallest and lightest cellular telephone. The size-weight war is also on for floppy drives, too. NEC introduced a 15-mm high 4.5 ounces 1.44 megabyte floppy disk drive. Mitsubishi, on the other hand, has introduced a 14.8-mm drive, but it weighs in at 6.88 ounces. NEC's drive is only 4" deep and uses only 1.3 watts when active.

I still have the first hard drive ever known to be associated with a TRS-80 Model I; it's an International Memories, Inc (IMI) 10 megabyte drive which sounded like a jet engine revving up when it started. It was used with the Lobo Drive's LX-80 expansion interface. The drive was about 18" long, 18" wide and 7" high. The thing weighed more than ten pounds and required a power supply almost the size of the drive. On the other hand, a few companies are packing up to 1.2 gigabytes onto a 3.5" hard drive, the Seagate ST11200N is a good example.

As drive capacities increase and the physical parameters shrink, what of the computers they are resident in? Well, they're shrinking too. And they will be getting smaller with increased capacity before you can blink. For instance, Chips & Technologies has been sampling a single chip containing an 8086-compatible processor, a PC/XT core-logic chip set, and peripheral controllers (a keyboard controller and LCD-oriented CGA video controller). Intel is also working on its own integrated PC chip. These types of chips target the new class of palmtop computers. See what happens when a hardware standard develops?

Memory capacities, RAM - not my own brain, usually quadruple every few years. The factor of four comes about because most memories are arranged in a rectangular array. When you double in each direction, you get a four-fold increase. The current state of the art in RAM was the 4-megabit chip. But that didn't seem to last too long as at least one supplier, NEC, has begun sampling 16-megabit chips made with a 0.55 micron process. Samsung began sampling a 16 megabit chip about a year ago. Fujitsu, Hitachi, IBM partnered with Siemens, Mitsubishi, Texas Instruments, and Toshiba have all announced 16 mbit chips or will soon do so. I also noticed that a few companies are close to sampling 64 and 256 megabit chips.

The venerable Zilog just pulled a coup with the announcement of an 8-bit processor which includes a 16-bit digital signal processor integrated on a single chip. The chip is positioned for use with sub-3.5" hard drives (the 2.5" and 1.8" form factors). The DSP handles 16-bit by 16-bit multiplications and accumulation in one clock cycle; that's 100 nanoseconds with a 24-megahertz clock. The Zilog chip also includes and 8-channel 8-bit A/D and D/A converter.

According to Technology Futures, more than 33 million PCs will carry FAX cards by the year 2000. I am one of those 33 million. With a ZOFAX 96/24 fax/modem, you can be too. Mention this paragraph in TMQ and I'll sell you one for $175 + S&H.

As you migrate to newer machines, what are you doing with your older software and files stored on that old media? If you are like me, you have all of those old diskettes and cassette tapes. Think they're still readable? If you have given up on a TRS-80 and are now using a PC or Mac, can you still read those files?

I recently caught an editorial in *Electronic Buyers' News* (EBN) which touched on some of these issues. Consider that the Census Bureau has 4000 computer tapes (remember mainframe reel-to-reel) recorded at 200 bpi. I doubt they have a machine capable of reading those tapes. There's 1.2 million reels of tape in NASA's basements. Readable? Questionable! Technology is changing too fast; we don't have time to keep up with it.

It seems that the way in which the computer industry biggies are planning their future growth is on the acquisition of similar-sized to somewhat smaller companies. Witness the acquisitions of NCR by AT&T, DRI by Novell, Ashton Tate by Borland, Norton, Zortech, and now Dynamic Microprocessor Associates by Symantec, ad infinitum.

From the *another garage operation makes good department* comes word that Dell Computer Corporation - which started as PC's Limited in an apartment - had a profit increase of 94% for their second quarter. Net income increased to $12.4 million on $200 million in sales. Meanwhile shifting from Austin to Fort Worth, we find Tandy's net income dropped from $60 million to $18.9 million for the fourth quarter on sales of approximately $1 BILLION. I wonder if the $16 million settlement, for pay and accrued and unused vacation under California law, reached in April between Tandy and a group of former employees had anything to do with that performance?

Turning to a *brighter* front, the Senate Judiciary Committee has passed a measure that would impose tough criminal penalties on software pirates. The Senate bill is aimed at large-scale and frequent copying of software. For instance, if more than 50 copies over a 180-day period of any one piece of software are copied and distributed with the copyright-holder's consent, the offense could be punishable by up to five years in prison and up to a $250,000 fine. For piracy of between 10 and 49 copies, the penalties are appropriately reduced.

Just how rampant is the crime of software *theft*? According to the Business Software Alliance, Italy has been declared as the worst violator of software copyrights. BSA estimates that eighty one percent of all new packaged software sold in Italy has been illegally copied. Other estimates are 79 percent in Spain, 63 percent in France, 52 percent in Britain, and 30 percent in the United States. Microsoft has gone so far as to put holograms on their legal packages in an attempt to thwart thieves. But when you are talking about millions and millions of packaged copies of a single software product, there's lots of opportunity for the criminal element to infiltrate the process when hungry retailers are unconcerned about their source for product. Illegal duplication mills pump out thousands upon thousands of illegally copied packages of Microsoft's MS-DOS 5.0, for example.

Now that I got your attention, here's other *good* news for mail-order buyers. The U.S. Supreme Court will soon have to resolve a dispute concerning the rights of a state to collect Sales Tax from out-of-state purchasers. With shrinking revenues, many states are trying to go after direct marketing firms. Case in point: the North Dakota Supreme Court upheld the state's right to force Quill Corporation of Illinois to collect and remit taxes on purchases made by North Dakota residents. Quill has petitioned for review by the Supreme Court. Keep your eye on that one. Already California and Tennessee have started to issue tax assessments to companies.

While I'm on the subject of news, here's some sad news we passed along to our Compuserve forum readers back in July. John Long, an employee of John Lancione and one of the folks who worked at Aerocomp, Clone Computers, Montezuma Micro, et. al. for probably well over ten years, passed away unexpectedly on July 31st while dining at a restaurant.

Incidentally in case you haven't heard, all the above businesses have ceased operation as of about July 1st. Clone Computers has re-opened under new management (i.e. Mrs Lancione) and can be reached at 214-934-2200. John Lancione, the person behind all of these companies - and a person whom I have had the pleasure of meeting on a number of occasions (who remembers the Milwaukee massacre?) - had this to say about leaving the TRS-80 market: "I had eleven wonderful years, met tens of thousands of fine people; the time came to close the business. Thanks for the memories." And to those who sneered at the venerable TRS-80, referring to it as *TraSh-80*, John had no nice words for you! John, we'll miss your presence.

On a personal note, this issue marks the beginning of the sixth year of publication of *The MISOSYS Quarterly*. That's a pretty good record. I hope that as this last year of publication proceeds, we all will have time to reflect on the past while MISOSYS looks to some future somewhere. I have introduced versions of LDOS and LS-DOS which will carry you all through to the year 2011. And if you move on to an MS-DOS environment, consider our product line.

Finally, let me close this *Points to Ponder* with a request for you LB users out there. Now that release 2 has provided the LBMANAGE utility which easily - and effortlessly - can duplicate a database structure, it would be highly desirable to help out your fellow LB users by submitting your ***database templates*** for publication. All you need to do is to duplicate your database using LBMANAGE - specify but one record - then copy the peripheral screen and print files. Bundle these together on a disk and send it to me, or combine into an .ARC or .ZIP file for uploading to our Compuserve forum. I'll make them available to other LB users. Besides, I'm interested in learning what functions you have organized with LB.

TMQ Schedule

I target mailing *THE MISOSYS QUARTERLY* the last week of the respective month as follows: Spring issue in February, Summer issue in May, Fall issue in August, and Winter issue in November. I had to drag my feet with this issue because I had too few subscribers to warrant publishing in August.

Those with **"91/11"** on their mailing label **complete their subscription with this issue.** The renewal fee to continue with the next three issues is 75% of the subscription rate shown on page 1.

TMQ advertising

If you are interested in reaching a **dedicated** TRS-80 audience, consider *THE MISOSYS QUARTERLY*. If you have a TRS-80 Model III or 4 related product to sell, you can reach these buyers by placing your advertisement in our publication. Current space rates are as follows (reduced from previous rates):

Full page	$75
Half page	$40
Quarter page	$25
Ninth page	$15

I compose the ninth-page ad layout so you have no artwork charge. Just submit your text. Ads for our inside covers are printed in the same color as the cover (TMQ alternates between PMS colors: green 354, purple 266, blue 293, and red 199). If you would like to place your ad in *THE MISOSYS QUARTERLY*, send it in. You have three issues left.

LDOS 5.3.1 Released

During late spring and early summer, I devoted time to bringing the Model III LDOS 5.3 release up to coordinate with the 6.3.1 version of the Model 4 LS-DOS release. Guess what folks - those who have not already heard - I also developed a true, official Model I release of LDOS 5.3.1. So for the few dozen or so folks who wanted a Model I LDOS with 5.3 features to carry them into the next century, you now have one.

What we wanted to do was groom the LDOS release so that every command, wherever possible, was consistent with the command syntax of the Model 4 DOS. In doing so, the command structure operation of one DOS would be no different than the other; the result should be more useful to users of all three machines running an LDOS derivative: Model I, Model III, and Model 4. As of this writing, I have not generated the MAX80 Model III version, since there may be an extremely small population of those folks left; however, if they let me know, there's no physical reason to not generate it.

In the grooming operation, there was never any intention to coordinate the two BASICs; that's why I always refer to the DOS. BASIC is a separate facility, and would be near-totally useless for the Model III to sport a Model 4-type BASIC. On the other hand, there may be justification for designing a Model III BASIC interpreter to run on the Model 4 in Model 4 mode! Would you find such a language product useful? I can envision creating an LBASIC that runs on a Model 4, but runs Model III BASIC source. The big question is how many folks would find that a useful thing to PURCHASE? Let me know.

Also, now that the Model I, III, and 4 versions are near-equivalent from a command standpoint, I expect to generate a new DOS manual covering all three versions. It appears on one hand that the Model 4 DOS manual may no longer be available from National Parts. So the resort is to purchase a used DOS manual from an outfit like Pacific Computer Exchange, one of the manuals written by Christopher Fara and sold by CN80, or a new multi-machine manual written by the author of the DOS. Note that this would not cover BASIC, as BASIC is not part of the DOS.

Okay, so what have I done to the 5.3 release to bring it up to level 5.3.1? Fair question. Other than introduce a few bugs to the Model I version which have already been corrected, all files have been groomed and enhanced so that the syntax and features are consistent with 6.3.1.

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.** For the Model I version, this obviously includes time-stamping with hour:minute. One internal difference exists between the Model I/III versions and the Model 4 version concerning dating. Without the ROM getting in the way, and a little more low-memory system space in the Model 4 version, I was able to implement the 32-year range of supported years in the system date storage data structure with a value from 80-111 - indicating 1980 through 2011. The Model 4 @DATE service call, as well as all system modules making use of the date, converted the number where necessary to 80-99 and 00-11 for display purposes. However, in the Model III version, the @DATE service function is in ROM and inhibited this technique. The Model I @DATE version is in SYS0, but insufficient space existed to handle the conversions. Therefore, the Model I/III LDOS 5.3.1 implementation stores 80-99 in DATE$ for 1980-1999 and 00-11 in DATE$ for 2000-2011.

One of the reasons I was resistant to implementing a Model I LDOS 5.3.0 - aside from the small audience - was the requirement I had for including automatic support of double density in the Model I release. I knew that it would have been extremely difficult to find space to implement the handling. Working on the 5.3.1 release for the Model I proved that to be a reality. I did not have enough free low memory to include a double density driver as part of SYS0. But I did make an improvement.

First, I enhanced the resident floppy disk driver to support two-sided disk drives. This eliminated the TWOSIDE utility. Since both PDUBL - the Percom-type double density driver - and RDUBL - the Radio Shack double density driver - have some code identical to each other and to TWOSIDE, I reworked both PDUBL and RDUBL to eliminate the code available in the resident floppy driver. I then combined RDUBL and PDUBL into a new FDUBL driver which has a parameter to choose either Radio Shack (default) or Percom-type doubler support. The new FDUBL takes up about the same space on disk as one of the earlier drivers; thus, you save some disk space. FDUBL also takes up less high memory than either of its predecessors by eliminating the code it can address in the resident driver.

Next, I altered the Model I FORMAT command so that the "SYSTEM" parameter forces cylinder 0 of a double density formatting to be re-formatted in single density. This flags the disk as being a dual-density configuration with a bit turned on in the Granule Allocation Table. Then I borrowed and modified the separate SOLE2 utility and added it as a DOS utility to add a booting double density driver to a dual-density disk. Finally, I modified BACKUP in both the Model I and III versions to be able to perform a mirror image duplication of a SOLE'd dual-density disk. Incidentally, the application of SOLE automatically recognizes whether the FDUBL driver is *tuned* for Percom or Radio Shack doublers. I did not change QFB to handle the dual-density diskette structure; BACKUP will be the tool to duplicate your booting double density system disks.

```
;*=*=*
;       Evaluate whether machine is a model III or 4
;*=*=*
        LD      C,OPREG$
        LD      A,(OFLAG$)      ;Get current port image
        LD      B,A
        LD      HL,CRT$
        LD      A,H
        LD      (HL),A          ;Put char into video
        CPL
        LD      E,A             ;Save char complement
        SET     7,B             ;Select page 1
        OUT     (C),B           ;If mod4, switch to page 1
        LD      (HL),A          ;Char comp to page 1 if Mod4
        RES     7,B             ;Select page 0
        OUT     (C),B           ;Switch to page 0
        LD      A,(HL)          ;Get char from page 0
        LD      (HL),' '        ;Make sure we insert a space
        XOR     E               ;S/b Z for Mod3, NZ for Mod4
        LD      (MODEL),A
```

Model III users without the Hardware Interface Kit add-on, get a little extra support with the keyboard driver. When you install KI/DVR, the Model III version automatically detects whether LDOS 5.3.1 is running on a Model III or a Model 4 computer. If it determines that a Model 4 is indeed the host system, then KI/DVR installs a keyboard driver which supports the CAPS, CTRL, and function keys. A Model 4 has two 1K video RAM pages which when operating in the Model III mode can be switched via a port operation. I make use of this capability by switching to page 1 and determining if I can place a character value into the RAM page and not have it appear in page 0. If its a Model 4, I'll see two distinct video RAM pages; if a Model III, I'll see the value in page 0. See the code listed above for what I used to make the machine determination:

The SYSTEM command was altered to support removable and reusable BLINK, ALIVE, and UPDATE memory modules. If you turn off the function when it was the most recent one added to high memory, it will remove itself. If it is not the most recent module added to high memory, it will remain trapped in high memory; however, if you turn the function back on, it will reuse its image in memory.

The TED text editor now has commands to **print the entire text buffer**, or the contents of the first block encountered just like its Model 4 counterpart. You can also obtain directories from TED by using the new ^Q command. TED also differentiates its prompt for the Load and File commands. For full use of TED, Model I users will need to have a lower case hardware modification.

The SPOOL command offers additional capabilities to pause despooling temporarily, then resume despooling under your command. You can also purge any text in the spooling buffer. These features are added with the Pause, Resume, and Clear parameters. The (OFF) parameter now reclaims high memory used by the spooler, providing it was the most recent module installed in high memory. If not, it will remain trapped in high memory; however, if you turn the spooler back on, it will reuse its image in memory. In order to be able to accomplish this, if the spooler finds itself in memory, it will restrict the D & M parameter values to those in effect when the spooler was installed. Actually, this version will totally ignore your D&M entry, whereas the Model 4 version gives you an error if you enter values for D or M different from that already in effect. So yes, there is a fine line difference in the response of the two versions. But I feel the Model I/III version is superior to the Model 4 version in this regard!

The RESET command had two features added to operate in concert with the Model 4 version. You can now **alter the logical record length** of a file with "RESET filespec (LRL=n)". No longer do you have to fumble with copying one file to another using C=N just to change the LRL. You can also specify "RESET filespec (DATE=OFF)" to restore a file's directory entry to the old-style dating of pre-5.3 release. Specify "RESET filespec (DATE=ON)" to establish a file's directory date as that of the **current system date and time**. This feature is very useful to be able to construct an old style dating for a file that you wish to CONVERT using Model III TRSDOS 1.3 (why?). Just format the single-density 35-track LDOS disk, copy the file to it, invoke RESET filespec (DATE=OFF), then re-boot with TRSDOS 1.3 and CONVERT the file.

And speaking of converting, the LDOS 5.3.1 CONV utility now has the DIR parameter to view a directory of the TRSDOS 1.3 disk, just like LS-DOS's CONV utility. It also supports notspecs (i.e. CONV -/BAS:2 :1 to convert all files except files with the /BAS extension). The SHRINK parameter and function was added to the CREATE command. The DIR parameter was added to the FORMAT command to allow you to specify a designated cylinder for the directory when a diskette is formatted. And ALIEN was dropped as a required parameter of REPAIR.

I checked all commands and utilities to ensure that the parameter abbreviations were consistent between Model III and Model 4 modes. I did not add the query function to FORMS or SETCOM due to what I considered to be excessive overhead in the library file to implement the code. The Model 4 DOS can make use of character to number conversion routines (and number to character) which are resident DOS services, but not so in the Model I/III DOS version. That was a judgement decision.

There was also no room in the directory display to support an am|pm versus 24-hour clock time display, so I didn't add the AMPM parameter to SYSTEM. But I did add the PRTIME parameter to the SYSTEM command to enable or disable printer timeout. The catch is that PRTIME is valid only if you have the printer filter installed.

Both the Model I and Model III versions 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. SETCOM does not support the Model I LX-80 RS232L serial driver. All DOS commands have been groomed for Model 4 LS-DOS 6.3.1 syntax where possible.

Just as I did for the Model 4 DOS 6.3.1 version, if you have a Model III LDOS 5.3.0 disk, **a 5.3.1 diskette is available as a replacement for your 5.3.0 diskette for $15** (plus $3 S&H in US and Canada, $4 elsewhere). There's no need to return your current master. Model I users, since you have never purchased a 5.3.0 upgrade, need to purchase the 5.3.1 LDOS 5.3.1 upgrade for the $39.95 price plus S&H. Some features require lower case or DDEN adaptor. See our ad for further details.

XLR8er Interface Kit

David Goben developed, for us, a software support package for the XLR8er board operating under LDOS 5.3. The package is based on our own Hardware Interface Kit; hence, the name of this new package is the **LDOS 5.3 Model 4 with XLR8er Interface Kit.** I doubt I'll put all that on one label. The package includes the following software: **XLR2RAM** Memory management handler; **XMEMDISKT** RAM disk driver; **BANKER** Memory bank utility; **VIDX** Video management extender; and **SETX** XLR8er support utility

The XLR2RAM memory management handler performs the same kind of operation as our SET2RAM utility: it switches the Model 4 operating in Model III mode to a Model III RAM mode, then populates the low RAM with an image of the ROM. It then adds the memory management bank handler for the extra 64K of memory and the extended 256K of memory on the XLR8er board. This adds banking capability for XLR8er-equipped Model 4's operating in Model III mode to the ten additional memory banks. Like SET2RAM, the XLR2RAM memory manager overwrites the cassette routines in low memory. Note that this may not be suitable for those machines equipped with M.A.D.'s XROM.

The RAM disk driver can create a RAM disk of from two to twenty seven banks; however, only ten are supported through XLR2RAM.

The Video Management Extender expands scroll protect from zero to fifteen lines rather than just three; supports reverse video characters as in the Model 4 mode, and supports the @DSP of the video character generator ROM's character codes from 0-31 and 192-255.

The price of the XLR8er Interface Kit is $20 plus $4 S&H (US, Canada), $5 S&H for TMQ zone D, and $6 S&H for TMQ zone E.

Winter vacation reminder

MISOSYS closes up between Christmas and New Year's. That means this year, we will be closed from Saturday December 19th (Saturday) until Thursday January 2nd, 1992.

PD Software Librarian

Vic McClung is librarian for a collection of TRS-80 public domain diskettes. All requests and contributions be directed directly to him at:

Vic McClung
914 Crescent
Sikeston, MO 63801
USA

Note that if you upload a "public domain" file to our CompuServe forum [PCS-49], and want it to receive general distribution, please also mail a copy on disk to Vic. There is no legal provision for downloading files from Compuserve and re-distributing them, unless you were the uploader. Some of our readers who do not have access to our forum have an interest in those submissions. So if you want to help out the most numbers of fellow users, don't limit your submissions to just one source.

MISOSYS Forum

MISOSYS sponsors a forum on CompuServe. You can reach some "experts" on TRS-80 and MS-DOS subjects by dialing in. The forum is reached via GO PCS49, or GO LDOS. If you have any questions concerning access, get on and leave a message to SYSOP. Joe Kyle-DiPietropaolo will get to you. Please don't call me here at MISOSYS because I cannot answer any questions as to its operation.

The forum contains a great deal of programs which you can download, as well as enter into the lively discussions which thread through the message system. If you do programming on a PC, the forum also contains the listings from *Programmer's Journal*. If you want to direct a message to me, my user ID is 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 6.1

Each issue of *THE MISOSYS QUARTERLY* contains program listings, patch listings, and other references to files we have placed onto a disk. DISK NOTES 6.1 corresponds to this issue of TMQ. If you want to obtain all of the patches and all of the listings, you may conveniently purchase a copy DISK NOTES priced at $10 **Plus S&H**. The S&H charges are $2 for US, Canada, and Mexico, $3 elsewhere.

Out of print TMQ's available

Volume I and II are out of print issues; they are no longer available. The price for back issues is $4 + S&H (minimum order of $15). S&H for a single issue is $2.75 in the U.S. and CANADA; $5.50 zone D; zone E is $6.50. S&H for four issues is $5 (US), $6 (CAN), $14 (ZoneD), $20 (ZoneE). Here's a synopsis of past issues:

III.i Reading NEWDOS/80 disks; An LB archival utility; Popup Application Window; XMODEM in C; Getting into computer math, part I; TMQ Volume I index.

III.ii Getting into computer math Part 2; Writing interactive RATFOR/FORTRAN programs; PRO-EnhComp: a review; Desktop publishing and the Model 4; A better TERM/APP; adding floppy drives; and a new XLR8er interface.

III.iii The CRC program; PG: a page display program; Locating high memory routines; FIXMA3; Jumbo tape backup for PC clones; New style for TMQ using Pagemaker; and an Index to Volume II.

III.iv Checking for a file from Model 4 BASIC; Surviving the Hard Disk crash; An "interview" with Niklaus Wirth; Keep your printer clean and oiled; On-line HELP with PRO-WAM; MISOSYS announces availability of Hard Drives; Logic in the C language.

IV.i Cataloging files with a word processor; Page display PRO-WAM application; File undating with FUNDATE; Array load routine for BASIC; XLR8er and the GT-180 graphics board.

IV.ii Printing from BASIC without cutting words; LOAD100 for Model 100; Generating date/time stamp; Favorite recipes; Some BASIC routines.

IV.iii Fast in-memory sort using XLR8er RAM; Using XLR8er RAM as graphics video RAM; Upgrade your 4P with external floppy drives; Doubling of files solved; SuperScripsit document file format' FELSWOOP PRO-WAM export utility.

IV.iv Five Twelve K: A better way; Multi-Command; Touch/ccc; Fixes for LS-DOS 6.3.1; DoubleDuty Version 2.6.0 released.

V.i 300 Dots on the TRS-80; Tandy 16/6000 Hard Disk Drives; NXWAM PRO-WAM application; A review of M.A.D.'s XROM; a MIDI interface for your TRS-80.

V.ii Image processing on the TRS-80 Model 4; A MAKE utility for MC; New XLR8er patches for LS-DOS 6.3.1; FORTH: A language for every application.

V.iii It's rude not to interrupt; A Model 4 mouse driver; Profile 4+ to filepro 16/dBASE III; and a complete map to *Lair of the Dragon*.

V.iv Fill low memory; Internal HD for 4P; Roll your own on the XLR8er, Profile 4" printer codes, Boot LDOS 5.3 from HD Model 4; Memory: How much and why; 300 Dots: An update; and Lair of the Dragon hint sheets.

Onesies and Twosies Closeout

I dredged up the following items from my cellar. All items are new and unused except where indicated. Numbers in parentheses indicate the quantities available; numbers in square brackets indicate UPS shipping charge. If you don't already know what an item is or can be used for, then you don't want it.

(3) Expanz! board. $125 plus $5 S&H

(1) Microsoft BASIC Decoded and Other Mysteries, by James Farvour $5 [$3];

(2) TRS-80 Disk and Other Mysteries, by H. C. Pennington $5 [$3];

(1) BFBDEM Disk Software for the TRS-80 $5 [$3];

(1) Getting Started on the Sharp 1500 & Radio Shack PC-2, by H. C. Pennington $4 [$3];

(1) The Custom TRS-80 & Other Mysteries, by Dennis Kitsz $5 [$3];

(2) A-B switchbox - DB-25 serial ports $10 [$5];

(10) bare XLR8er board with 256K $100 [$5];

(1) American Power Conversion AP200 UPS $150 [$10];

(39) Used Xebec S1410 controllers $30 [$5]

Letters to MISOSYS

LB86 and Paths

Fm Donald Simpson, Queensland, Australia: Dear Roy, Re: LB86 Data Base Manager. Thank you for your letter in April [see TMQ V.iv, page 16] advising me how to gain access to the LB Maintenance Utility from the Data Base Manager. I now have 100% success, it was as you suggested that both programs had to be on the same path.

I do have one small problem that you may be able to assist — am I correct in assuming that the "Control Codes" can only be used in the Define Printer Format, Option #9. If this assumption is correct, could you advise how it would be possible to enter 'non-keyboard' characters or say 'underlining' in certain parts of the data when using Options #2 & 3.

Fm MISOSYS, Inc: Donald, you can imbed printer control codes only in the print definition. These codes are used to emit controlling sequences of codes to the printer during the printing of a report. It is not possible to enter non-ASCII characters during data input. In fact, LB uses the 8th character bit to designate the character as appearing in reverse video. If you want to use something akin to underlining in commands #2 (Add) or #3 (edit/update), you may want to consider adding a series of dashes in the screen line immediately below the character string you wish to underline.

LB/LB86 and '\'

Fm MISOSYS, Inc: The following letter was sent to all known recipients of LBDM version 2.1.0 on May 23, 1991. Just in case I may have missed someone (unlikely), the content is published here in *The MISOSYS Quarterly*.

To LB 2.1.0 Users: Since the release of LB Data Manager version 2.1.0 on April 30th, 1991, two minor problems have been detected and corrected. These fixes are so minor, they are field installable by you.

The first problem affected the TRS-80 version (M-50-510) only. The duplicate field function in the ADD command operated incorrectly. This was corrected in the LB/OV5 module dated May 13, 1991. If your module is dated earlier than that date, please apply the following fix:

```
. LBOV51/FIX - Patch to
LB 2.1.0 LB/OV5
D08,8B=E5 2A 30 56 E5 CD
F9 33 F1 22 2A 56 D1 19
E5 21 B9 4A ED 5B 2A 56
F08,8B=ED 5B 2A 56 19 E5
21 B9 4A E5 2A 30 56 E5
CD F9 33 F1 22 2A 56 D1
. Eop
```

Type the patch into a file using TED (the text editor included with your DOS) and save the file using the name, LBOV51/FIX. This fix can then be applied via the command:

```
PATCH LB/OV5 LBOV51
```

The second problem concerns the LBREDEF utility program; the "\" field character is not acceptable. This problem was corrected on May 23, 1991. For TRS-80 users (M-50-510), apply the fix using the following command line:

```
        PATCH LBREDEF
    (D16,2F=14:F16,2F=11)
```

MSDOS users (M-86-510) can make the correction using DEBUG. The following script sequence will make the change:

```
RENAME LBREDEF.EXE XXX
DEBUG XXX
E 1641 14
W
Q
RENAME XXX LBREDEF.EXE
```

DEBUG does not permit editing of ".EXE" files; thus the need for the RENAME command. If you have your own *zapping* utility, use an offset address of 1541 as DEBUG references its offsets from a base address of 100.

LB and JCL

Fm David Huelsmann: Roy, Such prompt service! Just received your letter today concerning the fix to LBREDEF to accept the '\' char. Thanks. Now on to a couple of other minor issues:

1. If LB is called from a menu file, i.e., @xxxx, any use of the dos shell, causes the 'job done' message from the JCL with a subsequent abort of the shell. Any solution for this?

2. This one could be considered a 'feature', but I consider it a bug. If a drivespec is specified for a file specified on the command line for LB, then LB cannot find the path file (lb nmssmt:6). If, however, the drivespec is left off on the command line (lb nmssmt), then LB has no trouble finding the path file.

Fm MISOSYS, Inc: The documentation states that a filename is what is placed on the command line; you never use anything other than a filename. Drivespecs, extensions, and the like are all handled internally by LB. Trying to squeeze maximum power into the Model 4 version and still allow you flexibility with high memory availability requires some compromise in how much you want LB to guard against users not following the documentation on mundane matters.

There is no solution to the JCL abort if used to invoke LB and you subsequently invoke the SHELL from LB. LB's shell facility and original entry into LB from a JCL file are mutually exclusive. If you need to shell out to DOS to perform minimal maintenance or other DOS commands, you may want to explore installing PRO-WAM and using its library executive to perform DOS library commands. That limits you to only library commands whereas the LB shell would have let you do anything short of affecting high memory. That's a small price to pay for the capability.

Fm David Huelsmann: Roy, I was afraid you would say something like that. Oh, well, just have to make sure that I don't create another LB database with the same name anywhere on my drives.

Fm MISOSYS, Inc: It should be somewhat confusing to do that on a TRS-80 - possibly dangerous, as well. On the other hand, you could achieve duplicately named databases installed on different drives if the path files for all such-named databases were not simultaneously available. You could, for instance, use a DiskDISK (or subdisk) for each path file. Then, attach the subdisk for the database currently desired. You should be able to swap subdisks using the shell function. You would have to have a universally available database associated with LB during the shell operation, as LB would need to access the path file currently associated after returning from the shell. Under MSDOS, you would put each similarly named path file in a different subdirectory not identified in the PATH environment variable.

LB and Date of Update

Fm William J. Newman, Sanford, NC: Thank you very much for your quick response to the problem with the "Last Update" and the "Length" field insertion of the reverse slash. I started to use the Last Update, and found that editing the new data base established through LBREDEF. The field would update properly on my first EDIT giving me 1991/05/26, Subsequent EDITs, as long as I remained in the UPDATE/DELETE mode, only gave me the year of the Last Update i.e. 1991, and not 1991/05/26 as it should. If I left the UPDATE/DELETE mode and then re-entered that mode, the first record I edited would have the proper Last Update but not the subsequent records and again would only produce the year.

I tried this on both my model 4's. One with the hard drive and the other with just floppies and both systems behaved the same.

I don't feel like I am doing any thing wrong as the the entire procedure seems very straight forward. (Use the new data base and establish a new field for the screen display of the Last Update field). I have re-read the section very carefully covering the LBREDEF and the section about the Last Update, and the subject is covered very well. However, if I have goofed or overlooked something, please let me know, The manual for the LBDM is very well written and so far I have been extremely well pleased with the update from Little Brother 1.0 to 2.1. And like the man says "It has lots of bells and whistles!"

Fm MISOSYS, Inc: Bill's letter resulted in the following response from me which I sent directly to all known LB 2.1 Model 4 users on May 30, 1991. As you can imagine, I'm spending most of my time these days in the MS-DOS world. Just in case I may have missed someone, the content is published here in *The MISOSYS Quarterly*.

To LB 2.1.0 Model 4 Users. Another small problem has surfaced with LB 2.1.0 affecting only the TRS-80 Model 4 version. The problem relates to record data corruption when the second sequential update or add is performed and the data definition includes the "date-last-updated" field type. The following two command line patches will correct the problem in both the EDIT and ADD modules.

```
PATCH LB/OV5 (D1B,CB=21
38:F1B,CB=2A 3F)

PATCH LB/OV8 (D35,88=21
6C:F35,88=2A 73)
```

I apologize for any inconvenience this may have caused you. Incidentally, the name of our Version 2 data base product is "LB Data Manager"; it is not "Little Brother". Unless I slipped up somehow, there is no place in the User Manual where the term "Little Brother" is used. I will henceforth correct any future correspondence which appears in TMQ.

LB: *Read the manual*

Fm MISOSYS, Inc: In spite of the minor troubles Bill experienced (I'm going to use him as a beta site for the next release), I received the following letter dated June 25.

Fm William J. Newman, Sanford, NC: Today I received my TMQ V.iv and was duly impressed by the article you wrote concerning the development of LB 2.1. You can tell everyone out there that it works beautifully and that I have been able to enhance my reports even more than with LB 1.0 Also the ease of using the database, sometimes throws me off as an old time user of LB 1.0. I must re-read most sections of the manual to make sure I am not missing some of the NEW FEATURES and not take for granted I know how to do something! Richard Deglin and you have lots to be proud of in this program. Let 's hope those Model 4 users will recognize such an excellent database when it's available. And your generous trade-in offer is amazing! I have not used MS-DOS, but I am sure those users will also find it as powerful, if not more powerful, as any data base they have available.

For your enjoyment, as I know you must get bugged by people not "reading" their manual so, I am enclosing a copy of "SHOE", a cartoon we get in this area. I am not sure if you get Shoe in your area. Well any way, if you could get copy permission, you should use it in all your manuals!

Fm MISOSYS, Inc: I should pursue getting reprint privileges for that column of Shoe. Yes, Shoe does appear in *The Washington Post*, and yes, I did see and was amused with that column. For those who haven't come across that particular strip, Shoe is fumbling through all sorts of machinations in order to attempt proper operation of his desktop computer. The final three frames tell the tale:

"There's only one thing left to do"...

"The last resort"...

LB: Right Justified Fields

Fm Bill Newman, Sanford, NC: I have a question about a field I am entering in LB 2.1 I have a data base set up called "SLANDING" of which I am sending a copy of the field definition file and a disk concerning the database in question

In the "Update" mode, menu #3, some of the fields, blank fields, no matter what editing screen I use, as I step through the fields to that field, the cursor starts at the far right rather than starting at the left and showing the dots representing the field length. There is not just one field that behaves in this manner. In the screen definition menu, I have repositioned the field by redefining it to a different location, with the same result. However, I have noticed as I was paging through many records to update the records, it only happened occasionally that the cursor started to the far left in a "problem" field with all the dots showing on the fields in question. As it is now, I must use the left arrow and step to the far left before entering the data in some fields.

If you would start with screen #1 or #3, and "EDIT" through the fields on the first name "'ATTLESON", you will notice that field 'SL ASSESSMENT PD Y=YES", the cursor starts to far right. Also with the next field "DATE PAID mm/dd/yy", the same thing. There are some records that do not have this problem and some that do.

Also try using index #5, a sort by LAST NAME. The first name field is "ALPHAMETALS" Then I found to "FIND" my name, 'NEWMAN", requires a double "FIND". It only finds it on the second try. Now step through NEWMAN in the EDIT mode and I found the fields above behaving as I mentioned with the cursor to the far right. Appreciate any light you can shed on these "problems".

Fm MISOSYS, Inc: Bill, I have researched your reported problems and can shed light on some of your questions. When you are in the edit mode and navigate through the fields, the particular fields which you mentioned where the cursor appears to the extreme right actually contain data. Specifically, within the two records you pointed out, the "SL Assessment:" and "Date paid" fields have SPACE characters in them. I can answer specifically that the fields which are of type "R", will always be space filled once a data entry is made or they are edited. Page 24 of the manual states for R fields, *"If the data entered for the field does not completely fill the field length, 'pad' spaces will be added to the left of the data entry"*. This implies that the field will be totally filled with spaces if the entry is null. For instance, if during the ADD or EDIT mode you step through an 'R' field with *f* or _, you will not edit the field; if you step through by repeated use of ENTER, you are editing each field, possibly with a null entry. An edited 'R' field will always be space padded, which is performed at the conclusion of editing a field.

There is a difference between a null field and a space-filled field as far as the cursor position during edit. Since the cursor always is positioned to the right of the last character (not the last non-blank character but the last character), trailing spaces will move the cursor right. So your check number, lot #, ZIP, and So. Landing fields will be space filled if they are "blank".

If you enter trailing spaces to other field types, they will stay as spaces and the cursor will be positioned to the right of the last space or other character when the cursor is moved to that field during edit.

I couldn't reproduce your problem in finding "NEWMAN" with index 5. Every time I tried to find your name after attaching the index, it took but one find.

LB2: Installation

Fm Guy H. Pelletier, Dollard Des Ormeaux, CANADA: I recently ordered a Model 4 LBDM Upgrade to version 2. The following package was received:

-LB Data Manager M-50-510 Ver 2.1 installation disk
-LB Data Manager M-50-510 Ver 2.1 executable disk
-Cat No. M50-510 Installation Manual for the TRS-80 Model 4 computer
-Cat No. M86-510 User Manual for the for the IBM PC or compatible computer

Is this the right User Manual?

Furthermore, when I proceeded to make an "LB CREATION" disk as instructed on page 4 of the *Installation Manual*, the operation stopped short with the error message, *"Error 07H"*. Likewise with the "LB RUNTIME" disk, the operation aborted with an *"Error 07H"*.

I am awaiting your instructions concerning (1) the *User Manual*, and (2) the corrective action for the production of the disks by the installation method 2. Should I expect further difficulties when the above problems have been corrected?

Fm MISOSYS, Inc: To begin with, as was the case for the LB Version 1 package, **the User Manual is identical for both the TRS-80 and MS-DOS versions.** As I feel that the MS-DOS market has a greater potential for LBDM, I decided to use the mnemonic LB86 - the MS-DOS version name - on the *User Manual* cover. The installation manuals are unique to the machine environment. If I were to use a different cover for the *User Manual*, it would have created inventory problems.

As far as your problem in installing LB, I suspect that you jumped in to page 4 without working up from page 1. Getting Error 07H when entering the DO command implies that the system drive does not contain SYS6/SYS. This could be due to one of two reasons. (1) You may have failed to follow the steps on page 2 and 3 which are used to create an LB startup disk. At the conclusion of the startup disk generation, the system drive is a MemDISK, the bottom floppy is drive :1 and the top floppy is drive :2. Thus, Method 2 on page 4 would be the proper procedure and should not generate the error. (2) Assuming you performed this LBINSTAL step, that would imply the SYS6/SYS file was on the disk (otherwise, how could the DO LBINSTAL function?); thus, the only possible result would have been for the SYS6/SYS file to be defective on the MemDISK creation - a virtual impossibility.

Finally, you should not expect further difficulties - at least not from program malfunctions (knock on wood).

LB2: Cursors?

Fm Karl Krelove, Levittown, PA: I got the upgrade to LB version 2.1 that I ordered from you. The improvements are terrific, especially the changes in the report generator. I used to have to route the printer to a disk file ahead of time to generate merge files for SuperScripsit or Dot-writer, and it's handy to be able to get quick reports on the display. Several other improvements make the program much more useful and flexible for the kind of filing I do.

I have found one very small problem - completely cosmetic, concerning a sort of misplaced cursor that sits and blinks in odd places when it is not marking the spot for actual input. Most of the time it shows up near the bottom right corner of the screen at the end of whatever command bar is displayed in each mode. It appears in the middle of the display, I think at the end of the last data field, in the Update/ Edit (Menu Option 3) mode. Once I begin moving the highlighted bar with the cursor keys, the cursor jumps down to just outside the bottom right corner of the high-lighted bar. It always seems to appear directly to the right of whatever was last written to the screen. There must have been some code to turn the cursor on and off in v.i that got left out of the new version. When input is actually requested, the cursor appears at the proper place in the input mask. Has anyone else noticed this? Can it be fixed easily or should I just continue to ignore the extraneous cursor? It has no effect on the program's functioning at all that I have discovered. It's just a distraction, especially in Edit mode. Thanks for a really useful upgrade.

Fm MISOSYS, Inc: Karl, I cannot duplicate your problem of the *extraneous* cursor. Could you provide more details as to the exact sequence of events leading up to the extra cursor. For instance, starting from invoking LB, what do you do? Have you used the SHELL facility to escape to DOS? If so, what commands have you invoked at the DOS level before returning to LB. In any event, I'll need an exact sequence of steps.

LB maintains the cursor in an off state until keyboard input is requested. Cursor positioning is handled by LB; it uses a C library function which, in turn, uses the DOS @VDCTL service call to position the cursor. The bottom line is that there should not be an extraneous cursor on the screen, and that the single cursor should be properly positioned. Give me more details to investigate.

Fm Karl Krelove, Levittown, PA: Roy: Thanks for your response this week to my letter about the cursors in LB's work screens. I'll give you as much detail as I can, but I'm really not doing anything unkosher or even clever. I invoke LB with the command

```
<LB RECORDS>
```

where RECORDS is an LB file containing my collection of LP's, cassettes, and CD's. It is a file I created under version 1, in case it makes any difference. At the main menu screen I select a function, for example adding a record, with <2>, and find myself at the add screen with a command bar at the bottom of the screen. Help is highlighted and a cursor is flashing in the extreme lower right hand corner. In this mode, as in all the others on the runtime side of the disk except #3 ("Update..."), the cursor remains blinking in the corner as I move the highlight with the arrow keys to the command I want to select (normally "Add"). I return to the main menu with two presses of <BREAK>.

The cursor display I just described may, I have come to realize, be intended. Since my last letter I have gone back and reread the manual more closely. (I can already hear you from here - this has always been one of your pet peeves with software users.) I have never used the hot-key selections from the command bar, having always moved the highlight with the arrow keys to the option I wanted. I guess I forgot they were available. Anyway, after rediscovering this method of selection, the thought has occurred to me that the cursor I'm seeing is supposed to be LB's way of requesting a key if I wish to invoke a command with one. It seems unneeded, but if a cursor's going to appear at all I guess it's where it would have to be for that purpose (end of the command bar). I have also gone back and discovered that the cursor appeared in the same place in version 1. I just never noticed it.

The one mode in which the cursor behaves differently from version 1, however, is the one that first attracted my attention, #3 - Update, etc... (Edit mode). When I invoke the Edit mode from the main menu, the screen is drawn, then the data is added to the display from the first record in the data file. When all the screen drawing is finished, the cursor is blinking at a position that appears to be the next one after the end of my record data. (See the screen dump I've included; I've drawn the cursor where it appears, since it doesn't show up in the dump). Seems Like an odd place for a cursor requesting a key from one of the command bar options. In my copy of version 1, it appears at the end of the command bar, as it does in all the other modes.

Further key presses have the following effects: <KEY> (for any of the available options) - cursor jumps to right hand end of highlight, option executes. <R or L ARROW> - cursor remains attached to right-hand end of highlight. <U or D ARROW> - new record data appears on the screen, cursor is in the data area again.

I forgot in all of this to say that when the cursor is needed to indicate where input is to be entered (e.g. edit option of the Update mode or anywhere a prompt appears asking for input), it is right where it needs to be (in the data field or at the end of the prompt).

If I've stirred up any degree of fuss about a screen feature that really was intended, I am genuinely *sorry* to have wasted your time. The appearance of a cursor in the command bar seems unnecessary but it certainly is not a problem and is consistent with version 1. The one at the end of the data fields in #3 - Update..., did cause me some initial confusion the first time I used the new version. I was probably only half awake, but for an instant I thought I was expected to enter data where the cursor was. And I'm not sure why, even given the purpose of requesting a hot-key response, the cursor follows (or, depending on the direction, leads) the highlight around the command bar after an option has been selected. This is all the more puzzling since it is not carried over from version 1. If the cursor's appearance in these situations was intended, then of course no problem means no fix. As I indicated in my first letter, the problem I thought I saw is purely cosmetic anyway (once I got used to ignoring the one in Update...) and doesn't in any way diminish the program's flexibility or its usefulness.

I apologize for not having read carefully first. I read the sections concerning the changed features and tended to skip over the stuff that hadn't changed, so I really didn't review the whole manual carefully when I got the new version. If you are not seeing what I'm describing, please let me know what more information I can give you. If I've essentially answered my own question, you can just let it ride if you don't have the time to tell me so. In any case, thanks for your attention and for your devotion to TRS-80 users.

Fm MISOSYS, Inc: Karl, Apparently you were mistaken about the cursor observation. In all of the cases you noted, that's exactly where the cursor was supposed to be. I don't know if other LB users feel as you do, that some of the positions are extraneous, but if I get more feedback to that effect, I'll look into changing the design in a future release.

So how about it, LB 2 users, do you feel the positioning of the cursor has a positive, negative, or neutral effect?

LB2: CLS Exit

Fm MISOSYS, Inc: Roy T. Beck reported a very minor difficulty I wanted to pass on to Model 4 LB 2.1 users. Roy uses ALLWRITE and found that invoking it immediately after exiting LB 2.1, the "ALLWRITE" logo written directly to the screen - probably through VDCTL, was in reverse video lower-128 value characters in lieu of Model 4 graphics characters. This did not happen in LB version 1. I dug into the LB code and found the reason. LB does issue a clear-screen just prior to exiting; however, it uses its own internal clear-screen function. The DOS clears the video screen with a HOME-CLREOF sequence (that's a cursor home code followed by a clear to end of frame code: (28D, 31D). The screen clear function in LB uses a re-positioning of the cursor to location 0,0 then a clear to end of frame. Unfortunately, that doesn't reset or clear out the reverse video engagement - only HOME does. As this problem is exceedingly benign, I'll get it touched up in the next release - not as a patch to this one. Just be aware of the problem if it occurs and recognize that there is no need to send me a report.

LB86: Installation

Fm MISOSYS, Inc: Lastly, here's a minor one which could affect only MS-DOS LB2 users trying to install the product onto floppy disks - until I cured the problem.

Fm John Keane, Charlotte, NC: Roy; As we discussed, I am enclosing my two original diskettes for LB86. I am also enclosing a copy of my IBM PC Dos 3.0 and my Run Time and Creation Diskettes. I have also made a back up copy of LB86 in case you are able to fix my problem and notify me by calling me at home, I would appreciate your swift attention to this matter so that I might start using the program as soon as possible.

Just as a reminder, I am encountering an error on line 299 of the install program "@Ansisys not expected." I am running the program on a Tandy 1000 with dual floppy drives under IBM Dos 3. 0. Thank you for your help in this matter. Sincerely yours,

Fm MISOSYS, Inc: John: This is in response to your letter of July 5th concerning the inability to install LB86 onto your Tandy 1000 using dual 360K drives. This letter documents the solution as discussed with you via telephone a few days ago.

The problem occurred due to an error in the INSTALL.DAT file associated with the automated INSTALL utility. At line 299 in the file, the statement @ANSIsys = "ANSI.SYS" should have read, **@Device = "ANSI.SYS"**. The statement would have been reached only if the ANSI.SYS driver were not already installed at the time the INSTALL program was invoked. Unfortunately, all the machines utilized here have the ANSI driver in their configuration; thus, the error was overlooked. I have made the correction to your master LBDM diskette.

I also installed LB86 onto the CREATION and RUNTIME disks you provided. As a full PCDOS 3.0 system disk with the ANSI driver, config.sys file, and COMMAND.COM do not provide sufficient space for the LB RUNTIME files, I deleted the config.sys and ANSI.SYS files. It will be necessary in your configuration to use a separate boot disk using a disk with only COMMAND.COM to install the LB RUNTIME files. I apprised you of this during our telephone conversation. An appropriate note has been added to the README file.

As an aside to other Tandy 1000 users, the latest version of MS-DOS available from Tandy for the 1000, 1000EX, or 1000HX is Version 3.20.22. It can be ordered from Radio Shack Mail Order (817-624-1196) using catalog number 25-1170; the cost is $29.95 plus approximately $3 for shipping

LDOS Fix format

Fm MISOSYS, Inc: Now let's move away from LB for the time being to address other topics of importance.

Fm L. Dean Dorsey, Little Neck, NY: This concerns the MSCI patches on pages 15 & 16 of TMQ V.iii. I had no trouble applying the Model 4 LS-DOS 6.3.1 patches MSCRES62/FIX and MSCARC61/FIX to my 20 meg hard disk even though I expanded the comments for a more complete description. I did have trouble with the Model III LDOS 5.3 patches. MSCARC51/FIX required removal of my expanded comments before it would take. MSCRES52/FIX failed because of a mismatch to the all 0's second find line. Please advise. Do I need to use Mr. Ed to look at this section of the program file?

In reviewing the Logical Systems, Inc. LDOS manual I did not find mention of the 'REMOVE' function for patches in the Dxx, Fxx form but 'YANK' was mentioned for the direct address patches. Can I assume that 'REMOVE' is implied? I didn't see anything on patches in your blue pages addendum for LDOS 5.3.

Fm MISOSYS, Inc: Dean, You are correct about the MCRES52/FIX in TMQ. When I create a patch, I keep each "D" and "F" statement on a separate line. Sometimes for space considerations in TMQ, I manually edit the fix. In editing the TMQ text, I inadvertently used a semicolon for both the Model 4 and Model III versions of the patch. It is correct as it appears on DISK NOTES. But LDOS 5 uses a colon logical line ending character whereas LS-DOS uses a semi-colon. LDOS's use is consistent with its use of a colon logical line separator on the PATCH command line.

The "REMOVE" parameter is not in the LDOS 5.1.4 User Manual as that parameter was added with the 5.3.0 release; actually, the "D" and "F" patch facility was also in that release. This is documented in the blue 5.3 upgrade documentation on page 11.

pfs:FILE and SYSRES

Fm Henry A. Blumenthal, Jacksonville, FL: This letter answers the problem that Paul D. Moore of Birmingham, England, was having with SYSRES and pfs.

The pfs manual — my version, at least — contains an error. In one place it advises that a speed-up in performance can be had by SYSRES'ing 1, 2, 3, 10, and 12. But in another place, it omits the 1. If just 2, 3, 10, and 12 are SYSRES'ed, there is no problem, and pfs will run.

Mr. Moore may have to make a copy of an unconfigured disk and then add the four valid SYSGEN numbers, unless he has a utility that can remove SYSRES; I've heard that such exists, but I don't have one. (Of course, since I've "graduated" to a MISOSYS hard drive, SYSRES has been unneeded.) If Mr. Moore is using SYSRES to conserve disk space rather than for a speed-up in performance, then I hope he will consider installing floppy drives of larger capacity. In 1991 there is no reason for a TRS-80 user not to have drives of at least 360K, if his budget will allow better. Having drives of larger capacity would make his TRS-80 a better value than ever.

Fm MISOSYS, Inc: Thanks for the tip, Henry. On the other hand, some folks SYSRES the system modules when using pfs-FILE to be able to run without a system disk in drive :0. Doing that lets them use both drives as data drives. Then it is essential that SYS1/SYS be resident as that contains the command interpreter - as well as a few other DOS services needed from within running programs. I think your suggestion about larger capacity drives has merit. Also, one should make use of the memory capacity of their system. Model 4 computers should be populated to at least 128K thereby gaining the advantage of operating such useful software as PRO-WAM, DoubleDuty, or a MemDISK. With a MemDISK as the system drive, one needn't bog their high-memory region down with SYSRES of five system modules.

Date-Not-Current!

Fm Gary W. Shanafelt Ph.D., Abilene, TX: Dear Roy, The main reason I'm writing is to bring to your attention a rather strange situation which I'm at a loss to explain. It's a file which fails to be recognized by the LS-DOS 6.3.1 BACKUP command. I don't know if there is something defective in the directory information of the file or a fix to BACKUP/CMD which I have missed. Anyhow, the file on the enclosed disk is HPCR/CMD, a one-sector file. It is neither protected nor invisible. But if you issue a normal backup by class command, i.e., BACKUP :1 :2 (Q=N), everything else will be backed up except this file. Yet it can be transferred from disk to disk with no problem if you use the COPY command. I'm including my copy of BACKUP/CMD on the disk with the problem file. Any ideas? I notice that LDOS 5.3 BACKUP works fine; the problem seems to he only with the Model 4 DOS.

The other files on the disk are for your personal interest. They comprise MDRAW II, a hi-res drawing program using a serial mouse and Matthew Reed's mouse driver. The program should work equally well with David Goben's driver - David even helped with the BASIC interfacing. However, so far it does not, though I've sent David a copy and hope he can figure out what the problem is. Anyhow, since I asked you about Reed's driver on the phone, and include it with the program, I thought you might want to see what I was doing. Of course, it's the SVC interface added to BASIC with LSDOS 6.3 that makes using the driver possible to begin with. I'm uploading the program to various BBS's; it should be appearing in the LDOS forum on CompuServe whenever my brother gets some time on his phone (he subscribes to CompuServe while I do not).

MDRAW ll is, I think, a good example of how your work with the *Quarterly* helps other people with theirs, to the benefit of the whole TRS-80 community (or what's left of it). So, thanks again!

Fm MISOSYS, Inc: Gary: This is in response to your letter of July 4th - I thought that was a day for celebrating, not writing letters. In any event, the mystery on why HPCR/CMD won't back up via 6.3's BACKUP but can be copied by COPY as well as 5.3's BACKUP is no mystery at all. The Model 4 DOS has limited backup protection applied to certain files. This has been discussed some time ago in an old issue of TMQ, but I don't recollect which one - see how a universal index would be handy? Under limited backup protection, a file can be copied to permit its movement to a hard drive; however, since BACKUP moves files en masse, BACKUP inhibits files which have been designated as limited.

A limited file has a bit in its directory record - bit 4 of DIR+1 to be exact - set to a one. This bit is identified as "used internally by the DOS" in my *Programmer's Manual to LDOS/TRSDOS Version 6.* Okay, it seems like that file is not part of a commercially prepared package, is it? If not, your next question is how did that bit get set? Good question. Here's the typical answer to such a question. Under Model I and Model III LDOS prior to LDOS 5.3.0 release, that bit was used to designate that the directory date for the file was not current. The TECH INFO in any *LDOS User Manual* will confirm that use. Under earlier LDOS releases, if no system date was established when a file was updated, the file date would be left unchanged, but that directory bit would be set. It was used in the display of directory information to reflect a plus sign between month, day, and year in lieu of the normal minus sign. For example, a date of "01+May+86" would point out that the file was updated sometime after May 1st, 1986, whereas "01-May-86" indicated the exact date of update. During the development of TRSDOS 6.0, one bit was needed for the limited flag. Since the "date-not-current" bit was the least important, it went the way of the wind. It was also dropped from use without mention, I believe, when I upgraded LDOS to version 5.3.0. Typically, a Model 4 disk could get that bit set inadvertently if someone used an older version of LDOS to copy the file to the disk when the date was not current. The bit was then propagated to all subsequent copies. Alternatively, some other DOS, such as NEWDOS, may have been used to copy the file. If you wish to "unlimit" the file, reset that bit in the directory using a tool such as FED2 - part of GO:CMD.

Concerning MDRAW II, yes, it does demonstrate the utility of publishing examples and techniques to be extracted for other works. I took the liberty of excerpting a little from the documentation of MDRAW II so that readers will understand what it is about (see the sidebar on the next page). Perhaps if they find it applicable, they can search out a copy from my CompuServe forum or other source.

Fm Gary Shanafelt, Abilene, TX: Roy, Thank you for your detailed response to my question about why certain files failed to backup, You were entirely correct, of course: I pulled out my trusty copy of LSFED-II, changed the directory bit, and presto — no more problem. You were also correct in your surmise as to the cause of the problem. Though the example file I sent you was a Model 4 program, I wrote it by taking an old 1-record Model 3 file, then used LSFED-II to clear the old code and type in the new (perhaps I should learn how to use an assembler). And the file originally was undated, so I guess LSFED-II's "update" feature stamped a date on it without changing that one bit. Thanks again!

P.S., You're right about an index, too: I vaguely remembered something about this backup anomaly in a past TMQ but, though I have all the issues since the first, I couldn't track it down. Too bad I don't have a photographic memory!

Fm MISOSYS, Inc: LS-FED-II can't bother with that bit because LS-FED-II is a Model 4 program, and the Model 4 directory structure uses that bit for protection, as noted. There's no reason for FED-II to do anything with that bit when you *touch* the file's date. The snag in that bit's utilization will eventually go away as more folks shift to 5.3; I eliminated the Model III use of that bit for indicating "date not current".

Miscellaneous Queries

Fm Keith Nightenhelser, Greencastle IN: Here's a couple of suggestions about the PROWAM CARD(X) utility: (1) It would be useful if CARDX displayed the number of the card file the utility had on the screen. I think there's room for the number. (2) It would be useful to me if, as an option, I could cause any editing to be filed upon pressing the BREAK key; sometimes I get distracted and lose updates because I neglected to press <CTL-F>.

MDRAW II: Scott McBurney's MDRAW with various revisions and enhancements

Documentation by Gary Shanafelt
Dept. of History
McMurry University
Abilene, TX 79697

INTRODUCTION

Scott McBurney first wrote this program as a basic drawing utility for his Model 4 hi-resolution board. He called it DRAW/BAS. Though he said it was simply a stopgap for MicroLabs' Pro-Draw, which he couldn't afford, it became part of everyone's hi-res library because it included a routine to print Dotwriter letters on the screen. It had something no other program had. George Madison liked it so much, in fact, that he rewrote the Dotwriter routines and ported them over to TRSDRAW/BAS, another hi-res drawing program. Then, a few years later, Scott wrote a revised version called MDRAW/BAS. It was basically DRAW but with the addition of a driver to support a Microsoft serial mouse: hence the "M" in the name, for "mouse." This was again a first — no other TRS-80 program would work with a serial mouse. And when the MicroLabs mouse hardware interface was sold out, MDRAW/BAS became the ONLY way to use a mouse with the hi-res board.

In the last few months, a number of TRS-80 mouse drivers have become available, all using the same interface for the Model 4. They were inspired by (you guessed it) the original mouse driver that Scott wrote for MDRAW. It thus seems like the time to upgrade MDRAW to work with them. David Goben rewrote the interfacing to work with the new mouse standard; yours truly worked over much of the rest of the code, adding or changing various features. Finally, to enable the program to use both the normal /HR format hi-res files and the compressed /CHR format, the program now utilizes two utilities written by Mel Patrick (with minor modification by me), BCHRLOAD/CMD and BCHRSAVE/CMD. This really has been something of a cooperative effort, but it all happened because Scott McBurney wrote such an original series of programs to start with. Despite all the changes, MDRAW II is still Scott's program.

WHAT YOU NEED

MDRAW/BAS is a very exclusive program. Not just anyone can run it. First, you need a Model 4 TRS-80 computer with a hi-res board. Second, you need LS-DOS, for the program uses the USR11 call introduced with LS-DOS BASIC to access the DOS SVC's (and thus the mouse driver). It won't work with TRSDOS 6.2. Nor has it been tested with any other Model 4 DOS's BASIC. Third, you need MicroLabs' GBASIC, for it is programmed in that language. Fourth, you need one of the new mouse drivers, either that by David Goben or that by Matthew Reed. Ironically, though the point of all this is to use a mouse, the program will run without one if you have all the other things — you can use the arrow keys instead. (You MUST have a driver even if you don't have a mouse, by the way, because of the way the program interfaces with the DOS SVC's). The program should work with either a two-button or three-button Microsoft-compatible serial mouse. The serial mouse is so called because it plugs into the serial (RS232) port of your computer with no need for any additional hardware. You should make sure you don't get a bus mouse, which requires a separate interface board. And even when looking at serial mice, you need to be careful, for not all serial mice on the market conform to the Microsoft standard. I'm using MDRAW with a Tandy serial mouse, catalog number 25-1040, and several adaptor plugs to make it connect to my RS232 cable.

In the package, you should find the following files. All but the DOC files should be online when you are ready to run the program:

MDRAW/BAS	The main program
MDRAW/HLP	Command menu for MDRAW/BAS
MDRAW/DOC	Documentation for MDRAW/BAS
MMOUSE/CMD	Matthew Reed's Microsoft mouse driver
MMOUSE/DOC	Documentation for MMOUSE/CMD
BCHRLOAD/CMD	Mel Patrick's utility to load CHR files
BCHRSAVE/CMD	Mel Patrick's utility to save CHR files
DEMO/CHR	Hi-res picture in CHR format
BONOST1/PR	A public domain Dotwriter font

1) A silly question to which I think I may know the answer: why is the output of the DIR command not alphabetized if the command is executed from within COMM?

2) Why can't I use FORMS to set a margin (e.g., with FORMS (MARGIN=5)) unless I also designate chars=something as one of my forms parameters? The documentation doesn't indicate this is necessary. and I don't see the reason for it.

Two things I'd like to find (haven't had time to write them for myself) — any tips where I could get Kermit for the Model 4 running LSDOS 6.3? How about a *CL filter, or maybe a wammie that could be toggled on and off, that would beep when data came over the *CL device. I often am hooked up to a mainframe while doing wordprocessing, and it would be nice to have a notification that a piece of email or a system message had been received. I assume the com/dvr keeps running even when I'm not in comm, so the data sent by the mainframe is there, if only something else in the interrupt chain was on the lookout for it. (By the way, I appreciate how data coming into COMM is saved up while I'm using PROWAM, and appears after I break out of the WAM application.

I have not been able to use the MEMORY command to read a system flag (with a command like memory (add=a)), Has this feature been dropped from LSDOS 6.3? I didn't find any statement to that effect in the new doc's (or any explicit example of the command's syntax in the old docs; I may be getting something wrong with the parameter (add=A); an obvious difficulty would be (add=x), which would be ambiguous between reading flag x and reading some hex address in memory).

The reason I wanted to read a flag: I have had a funny glitch intermittently; I don't know how to reproduce it. Sometimes I come back to my running machine to find the debug display on the screen and the machine locked up. When this occurred I have not had debug active, so far as I knew, so I was puzzled. Therefore I thought

I'd check the flags to see if (unknown to me) debug had somehow been activated and the proper flag set, Have you heard of this glitch from anyone else?

Finally, let me praise your 40 meg hd kit; I've been using it now for a couple of years with nary a hitch.

Fm MISOSYS, Inc: Keith: Your suggestions for PRO-WAM's CARDX application appear to be good ones. Sometime in the future, I have to re-groom date-sensitive applications in PRO-WAM to operate beyond 1999; I'll look into CARDX changes at that time.

The only silly question is the question you don't ask. Here's the answer on DIR and sorting. File names are not kept in alphabetic order in a directory. In order to produce the listing in alphabetized order, the data must be sorted. That requires an in-memory sort. Each file record requires 32 bytes of memory. Since there could be a maximum of 256 file entries, that requires a scratch memory area of 8K. When DIR is run from the DOS command line, it has memory available up to where HIGH$ is pointing to - which could be in the range of 50K. That leaves plenty of room for a sort buffer. When DIR is invoked from a program using the @CMNDR service call, the program first of all is not using the library overlay region extending from 2400H through 2FFFH, and second prohibits library commands from using memory above 2FFFH. That leaves no memory for a sort buffer. Thus, when DIR detects that it has been invoked from the @CMNDR service call, it turns off the sort parameter as it knows that no memory is available for the sort buffer.

I'll have to guess as to why you have to have a characters-per-line value in order to enter a left margin parameter in FORMS. The way that the FORMS filter is written, it uses the zero state of the character column counter to designate a margin control. The counter is contained in a byte; thus, it's value can range from 0 to 255. The counter is reset to zero on a RETURN, or when the character count exceeds the characters per line at which a forced line feed, indent as required, and margin as required, then the character is generated. If a MARGIN value was permitted without a character-per-line count, you could output a string of more than 256 characters without an ENTER at which point the character counter would overflow, restart at 0, and a false margin would be generated. One way around that would be to generate a margin only when an ENTER was output. Proper character handling should not result in unnecessary output; if the MARGIN were handled as I just stated, the last line of desired output would generate a spurious margin - not exactly proper. A way around that would be to maintain a flag which would be used to indicate that the last character output was an ENTER so that the current character would force the margin, reset the flag, then output the character. That would require more code and use of memory. I suppose that the person(s) who designed the FORMS filter felt entering a characters-per-line value was a reasonable compromise for accepting a MARGIN parameter.

I recollect that there is a public domain version of KERMIT available for the Model 4. I believe that Lee Rice located it on USENET and forwarded it to Vic McLung - whose address can be found in TMQ. I don't recollect if it was uploaded to CompuServe.

I am not aware of any filter or WAMMIE which would provide an audible alert that data was incoming via the serial port. But here's some complications to be aware of. You do not want to disable interrupts indiscriminately during serial I/O. A distorted tone - which may be all you need - could be generated without disabling interrupts. Another approach could be to use @VDCTL to poke the received character into the video screen like the clock time is displayed.

The Model 4 COM/DVR has only a one-character buffer. So if you are connected to an on-line service via COMM, and exit to DOS to run any other program, you will lose any incoming data stream. What captures data while you pop into PRO-WAM from the COMM program is COMM's capturing of data. Model 4 serial input generates interrupts which COMM uses to capture received characters and store them in one of its buffers until needed. The buffer could be up to 40K or so. COMM uses some excellent smart buffering of all device I/O - techniques originally developed by Randy Cook. Now if you could do your "word processing" with TED/APP - one of the MisterED applications, you are home free while using that from COMM!

You are using the wrong syntax to display/alter a flag's value using the MEMORY command. The correct syntax is to enclose the flag letter within double quotes - it is designated as a string. Thus, to display the A-flag, use: **MEMORY (Address="A")**. It is not necessary to upper-case the string.

Finally, if you have left your machine in an *idle* state, i.e. sitting at *DOS Ready* and it is in DEBUG when you return, it typically indicates a failure of memory. DEBUG is entered when the CPU program counter is at a memory location containing X'F7', the machine code value of RST 30H. Your computer is always processing code - even at DOS Ready it is constantly processing @KBD service calls. If there was a memory malfunction in any one of the locations the CPU is accessing for code and a value of X'F7' is sensed, DEBUG will be entered providing the system code used to read in a DOS overlay can still function. Even without memory failure, a program bug could result in a CALL or JP to a data area which could have a value of X'F7' resulting in a DEBUG invocation. The situation is rather difficult to run down unless it repeatedly occurs.

Glad to hear you are enjoying our 40M hard drive. Now if I can sell a few more...

PRO-WAM's Cursor

Fm MISOSYS, Inc: Speaking of PRO-WAM, Danny C. Mullen asked if there was a way to change the cursor character while in PRO-WAM? He tried by using the @VDCTL SVC, but it was a no go. In PRO-WAM, the display is handled by routines in the resident window manager. The cursor is hard coded as an underline character (5FH). One can change it only by finding its storage location in memory. Perhaps I'll consider adding a facility to deal with the cursor characgter code when PRO-WAM gets worked on.

GO:CMD- FED2

Fm Garry Howarth, Narwee, Sydney AUSTRALIA: Many, many moons ago (TMQ Vol III.i pages 2-3) you stated that you were working on a serial driver for the XLR8er. Is this still on the backburner ? I have got the hardware wired up and have modified Modem80 with limited (very limited) success and I desperately need this extra serial port. With three Eprom programmers and an IBM clone connected to the 4P (not all at the same time), the extra port would be a lifesaver.

On to another subject. I am having trouble with FED2 (GO:CMD) when used in the disk mode. I am using LS-DOS 6.3.1 and RSHARD on a 44Meg HD. The LS-DOS partition is divided up into four drives.

When reading drives 0 or 1, the program works perfectly, but when asked to read drives 2 or 3, the computer will crash in a BIG way, and I have to reset the computer.

Drive 0 203 cyls * 1 head 32 sectors/cyl 8 grans/cyl 4 sectors/gran
Drive 1 406 cyls * 1 head 64 sectors/cyl 8 grans/cyl 8 sectors/gran
Drive 2 306 cyls * 4 heads 64 sectors/cyl 8 grans/cyl 32 sectors/gran
Drive 0 203 cyls * 4 heads 32 sectors/cyl 8 grans/cyl 32 sectors/gran

The above information I have worked out from the Technical Information section in the RSHARD manual. It may or may not be correct. The DEVICE command gives the following:

:0 [LSDOS631] 5" Rigid #0 Cyls=203, Fixed
:1 [UTILITY] 5" Rigid #0 Cyls=406, Fixed
:2 [WORK1] 5" Rigid #1 Cyls=306, Fixed
:3 [WORK2] 5" Rigid #1 Cyls=306, Fixed

Any help with this problem will be much appreciated.

Fm MISOSYS, Inc: I have no good news to report on the XLR8er serial driver. I never had the time to get involved with it. I did have one user who wanted to work on it and I eventually sent him the hardware which was supposed to go along with the serial port kit: a split serial cable which connects up with the P2 connector on the XLR8er board; a split power connector needed to tap into the 12V and -12V supply which connects up to P3 - and which had to be rewired because it was made up wrong by whoever made it for HiTech; and the two serial I/O driver chips U22 (1488) and U23 (1489). But between the time of his offer and the time I sent the package, he moved and left no forwarding address.

One of the difficulties of a serial driver for the 64180 is that the two Asynchronous Serial Communication Interface (ASIC) ports need to operate using the onboard internal mode-2-type interrupts. I had originally wanted to establish a memory block for the interrupt vectors as they are needed for other types of 64180 interrupt handling (like DMA which Frank Slinkman uses). But it takes time to write a proper driver and that task is very low on the totem pole around here. I have made an attempt at narrowing down the number of different programs I am getting involved with because too many projects results in no one project getting excellent treatment. If anyone is interested in dealing with the driver, I can supply the hardware as noted above. Don't expect paid compensation.

I dug into FED2 as provided with the GO:CMD package. Your problem was caused by FED's inability to properly compute the directory size of a hard disk partition having 256 sectors per cylinder. I worked up a patch for this, GOFED2A/FIX which should cure your problem.

```
. GOFED2A/FIX - 07/17/91
- Patch to GO:FED2
. Patch corrects logging
of hard drive with 256
SPC
. Apply via, PATCH FED2
GOFED2A
D12,E8=66 69 6C 65 03 CB
00 87 C0 3D C9
F12,E8=6D 6F 64 75 6C 65
20 66 69 6C 65
D33,34=CD 8C 43
F33,34=87 CB 00
. Eop
```

MisterED and Toolkit

Fm Garry Howarth, Narwee, Sydney AUSTRALIA: Roy, I am having trouble with OOPS/APP from the MisterED package. When invoked, OOPS will clear the text buffer. A comparison of OOPS/APP and TED/APP shows that the two programs are almost identical.

```
COMP OOPS/APP TED/APP
COMP - Version 2.0.0 - File or Disk Compare program
Copyright 1982/83/84 MISOSYS, Inc., All rights reserved
Posn=X'0000,D9  OOPS/APP:7  =X'32',  TED/APP:7  =X'31'
Posn=X'0000,DB  OOPS/APP:7  =X'31,  TED/APP:7  =X'32
   2 bytes did not match
Posn=X'0000,DE  OOPS/APP:7  =X'35',  TED/APP:7  =X'34'
   2 bytes did not match
OOPS/APP:7    contains   9 sectors, EOF offset = 255
TED/APP:7     contains   9 sectors, EOF offset = 255
```

(This IMPORT and EXPORT is good stuff)

I am also having trouble with WINLINK/CMD and WINLINK/BAS from the PROWAM Programmer's Toolkit package.

1) If WINLINK is loaded without first loading PROWAM, instead of getting the error message 'Please install PRO-WAM first!' I get an 'Error 2BH' message (SVC Parameter Error). I used FED2 to have a quick look at WINLINK (the patch you sent me on the 17th 'GOFED2A/FIX' works like a dream - thanks), and found that instead of jumping to the routine that displays the error message, the program jumped to the error message itself!

The following short patch should fix that problem, I hope!

Change 00,A8 from C2 83 28 to C2 FB 26.

2) The problem with WINLINK/BAS is when the program reaches line 155, the computer will jump into DEBUG, with the PC pointing to X'F459', this address is constant. The enclosed MEMORY module map will show where PROWAM and WINLINK are loaded.

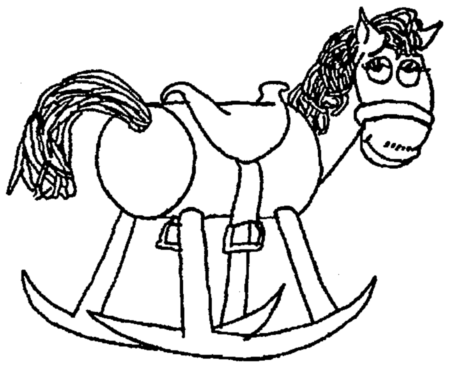
After much chewing of finger nails and pulling of hair, I decided to play around with the high memory setting and try it again [i.e. MEMORY (H=X'FFF0')]. Guess what? It works. Why? Good question. I don't know, maybe you can shed some light on this little problem.

Fm MISOSYS, Inc: Garry, The differences between OOPS and TED which you observed are simply the differences in date and time with which the two modules were assembled. But for some reason, you didn't observe the **important** difference between them - a single byte change. In record 1 relative byte X'B3', OOPS has a 00H value whereas TED has a value of 70H. I can understand why you don't see that difference as I just checked the master diskette and both TED and OOPS have a 70 at that location. Funny, the copy of OOPS on my 4P has a 00! Plug in a 00. I can't suggest why the difference had occurred as it's apparently been that way since May 25, 1988! Plug a 00 into OOPS and you'll be in business. You can, of course, use FED/APP for that.

You are absolutely correct about the wrong jump vector in WINLINK. Anyone who has examined my assembly code knows that I classically use a jump instruction to a series of LD HL,nnnn instructions for error reporting. The target of the LD HL,nnnn uses the same symbol string as the label but has a suffix character of "$". Unfortunately in WINLINK, I inadvertently added a "$" to the end of the target string in the JP to the error reporter. That's why you didn't get the "Please install PRO-WAM first!" error message. It has been that way since 1985!

Your difficulty with WINLINK/BAS crashing at line 155 was due to a bug in WINLINK/CMD. After checking the code, I found that a single RET instruction was omitted from the WEXEC code handler in the WINLINK driver. The statements immediately following where the RET should have been were data statements generating tables of vectors pointing to code blocks. These vectors would be different values at different load origins of the WINLINK driver in memory. Apparently when I originally tested the WEXEC function - and it did work(!) - the load origin of the driver must have been benign, in that the vectors treated as code resulted in an eventual RET without corrupting anything. It was almost impossible to patch in a RET statement, but as always happens, when you don't write the tightest code to begin with, there is usually a way to compress. I am providing WLINK1/FIX which corrects that problem in WEXEC, as well as the error jump target, and finally changes the version to "b". Incidentally, this problem has been around since July 11, 1988 when WEXEC was added. That's over three years ago! Thanks for bringing these things to my attention.

```
. WLINK1/FIX - Patch to
WINLINK from PROWAM Toolkit
. Corrects WEXEC, and
install error jump
. Apply via, PATCH
WINLINK WLINK1
D00,A9=FB 26;F00,A9=83 28
D01,8A=62;F01,8A=61
D04,58=77 21 09 2A C3 3E
29 21 6C 2A 7E FE 09 28
18 3C
F04,58=36 00 21 0A 2A C3
3E 29 21 6C 2A 7E FE 09
28 18
D04,68=77 85 6F 8C 95 67
71 3E 0D B9 20 0B 21 6D
2A 06
F04,68=3C 77 85 6F 8C 95
67 71 3E 0D B9 20 0B 21
6D 2A
D04,78=0C 3E 7C EF C3 14
29 AF C9
F04,78=06 0C 3E 7C EF C3
14 29 AF
D06,58=04 2A 07 2A 0A 2A
1F 2A 27
F06,58=05 2A 08 2A 0B 2A
20 2A 28
D06,69=38 2F 32 30 2F 39
31 31 34 3A 32 37 3A 30
33
F06,69=37 2F 31 31 2F 38
38 31 33 3A 33 38 3A 35
36
. Eop
```



Task Interfacing

Fm Gary W. Shanafelt, Abilene, TX: Roy, I have a question about using the LDOS interrupt task vectors, described in the technical section of my LDOS manual. What I would like to do is have the interrupts call a task routine in high memory, but I have so far not been able to figure out how to do this. I've been working with a simple routine to put a blinking character in the upper right corner of the screen, so I can easily tell if what I have is functioning properly or not (before I try writing the real routine). Anyhow, I protect HIGH$ at 4411, then add the routine with the following code:

```
        LD   DE,FE00   'Location
  of task routine
        LD   A,04 'Task slot 4
        CALL 403D '@ADTSK
        JP   402D 'Return to
  LDOS
```

The task routine in high memory looks like this, starting at FE00:

```
        PUSH AF
        LD   A,2DH  'Display <
        LD   (3C3F),A
        LD   A,3E   'Display >
        LD   (3C3F) ,A
        POP  AF
        RET
```

When I run the program, however, nothing appears in the corner of the screen. I've tried adding the @RPTSK vector to the start of the task routine: that gets the characters to display on the screen, allright, but the program never returns to DOS. I suspect what I'm missing is something very simple and obvious, and I'd appreciate any assistance you can give me.

Fm MISOSYS, Inc: Gary, when you add a task to the LDOS or LS-DOS task processor, register DE needs to contain a pointer to the Task Control Block (TCB) - this is not a pointer to the task itself. The TCB is similar to other control blocks in the system: device control blocks and drive control tables. A TCB is a block which contains data. The important data element is the task pointer which is contained in the first two bytes of the TCB. Actually, a TCB is a minimum of two bytes and needs no other data. By adding a TCB to your task, it would appear as:

```
TTCB  DW    TASK;2-byte TCB
TASK  PUSH  AF
      LD    A,2DH  ;Display <
      LD    (3C3FH),A
      LD    A,3EH  ;Display >
      LD    (3C3FH) ,A
      POP   AF
      RET
```

and instead of your initialization code loading TASK into register DE prior to the CALL @ADTSK, it would load register DE with the address of TTCB. Why is a TCB useful considering that it appears to add additional overhead? One reason is that like DCBs, on entry to a task, register IX contains the address of TTCB. If you wanted to include some data with your task, you could locate it within the TCB immediately following the task vector. Your task routine could then access the data using offsets to the index register.

Incidentally, although your routine - with a proper initialization - would toggle a character on the video screen, you would probably not be able to see it toggle. That's because the "<" character would be on the screen for only 4.25 microseconds at SYSTEM (FAST) (the time it takes to overlay it with the ">"), whereas the ">" character would be viewable for the total time between interrupts - 268 milliseconds. I'm sure the eye persistence will result in your brain sensing only the ">" character.

In order to see a toggling character, it is at a minimum necessary to change the character at each interrupt, essentially alternating between one of two characters. In that way, each is visible for the 268 milliseconds of the Model III task 4 slot.

Bug in DiskDISK

Fm Frank Slinkman: Roy, I think I may have discovered a bug in LS-DOS 6.3.1. Normally, FORMAT works fine when I format a DS/DD disk (c=40,dden,sides=2). However, if I use DISKCOPY, from then on FORMAT will only format in **single** density. Please tell me it's a bug, and not my computer.

Fm MISOSYS, Inc: Repeat after me, Frank, "There is no connection between DISKCOPY and FORMAT"... Here's something to check - you may have something corrupted in your system. Check your DCT for the drive being written to by DISKCOPY before you invoke DISKCOPY. Check DCT+4, bit 6. It should be set to a "1" - which indicates that the controller is capable of double density operation. The only thing in the system which checks that bit is FORMAT. FORMAT uses it to inhibit or permit the density query. If the bit is a "0", FORMAT will bypass double density query/activation and only single density will be allowed. The bit carries over from Model I usage. If anything changes that bit to a 0, FORMAT will then not allow DDEN. See if its your copy of DISKCOPY or some other program you run which is corrupting the bit - if it is getting corrupted.

Fm Frank Slinkman: As I mentioned on the phone, when I use DISKCOPY from a **sub-disk** to a floppy, it resets that bit. Since that conversation, I have found that DISKCOPY from floppy-to-floppy does NOT reset the bit. Therefore, it's GOT to be DISKCOPY. More specifically, it's got to be the section of DISKCOPY which permits copies from sub-disks.

I trust this information will help track down whether it's just MY copy of DISKCOPY or an actual bug. Thanks for your help and quick response.

Fm MISOSYS, Inc: Okay, I tracked down the bug. You are absolutely right and so was I as to the cause. And it happens only if the source drive is a **DiskDISK** (or subdisk - one and the same). Remember that 6.3.0's DISKCOPY wouldn't let you duplicate from a diskDISK. That was due to a test for the "ALIEN" bit in the DCT. When I got rid of that test, I neglected to determine that DISKCOPY makes the destination drive's DCT exactly the same as the source drive's DCT except for the drive select image. But a diskDISK's DCT doesn't bother to have the DCT+4, bit 6 set to a 1 because FORMAT is inhibited anyway. Thus, the diskDISK's DCT has that bit 0. When DISKCOPY strips everything from the destination DCT+4 except for bits 0-3, the source's DCT is imaged in. Thus, after the copy, that bit is reset just like in the diskDISK's DCT. The easy solution is to patch DISKCOPY to retain bit 6 of DCT+4. That's a very easy patch as follows:

```
PATCH DISKCOPY.UTILITY
(D03,89=4F:F03,89=0F)
```

After that patch is applied, you're back in business. I owe you a beer.

Fm Frank Slinkman: I'm sure you're just as happy, if not more happy, than I am that the fix is so simple. The patch to DISKCOPY, after I added "/CMD" between "DISKCOPY" and ".UTILITY," worked perfectly (of course). Now repeat after me, Roy, *"There is a connection between DISKCOPY and FORMAT!"*

Fm MISOSYS, Inc: It always makes me happy to have a one-byte patch to solve someone's problem. Now since the value changes from 0FH to 4FH - which is a single bit difference, the fix actually is a one-BIT fix - the smallest possible. But why should you have had to manually type "/CMD"? PATCH adds that extension to the patch file specification target.

Did I say there wasn't any connection between DISKCOPY and FORMAT? There isn't, of course. The problem originated in and was caused by DiskDISK. It did not properly prepare the DCT! Had absolutely nothing to do with the DOS. The 1-bit patch to DISKCOPY was an easier fix than correcting DiskDISK - in light of them both being my products, now.

SuperScripsit & DoubleDuty

Fm George Bria: I need help for a glitch in the DoubleDuty program used with SuperScripsit in my Model 4p. In Partition 3, my commands work okay, but I get a fallout of Caps interspersed with Lower Case letters when I return to Partitions 1 and 2, making it impossible to work. After executing a command, I get a jumble of upper and lower cases when I return to work on documents in Partition 1 or 2. And a "C" shows up at the bottom of my screen. The problem doesn't exist when I merely operate between Partitions 1 and 2, but only when I use commands in Partition 3. Since Caps key plus function keys are used to move from one partition to another there appears to be some connection that needs disconnecting. Appreciate any help.

Fm MISOSYS, Inc: Your problem is that you are probably trying to run SuperScripsit in both partitions 1 and 2. That won't work as SS is not capable of running two images of itself, i.e. it is not a re-entrant program package. DoubleDuty is fine if you want to run SS in one partition and some other program in partition 2. The fault is with SS. But I can't fault it too much; many programs are written in such a way as to inhibit correct operation in a task-swapping mode - which is offered by DoubleDuty. Our LBDM would be another program which could not be properly run simultaneously in two partitions.

Style or Content?

Fm David J. Kelton, Richmond, VA: Roy: I have been meaning to write and order the MS-DOS Version of LB Data Manager for some time. I have enclosed my copy of Spinnaker Software's "The Filer" in order to qualify for the reduced price on LB.

In the last issue of TMQ (V.iv), you asked for comments regarding the preference of titles. I suppose I like the "Letters to Misosys" type style best when turned sideways, but I'm really a bit more worried about something else. My current job involves the preparation, distribution, and updating of analytical test procedures for the QA Department in my company. Recently, my department has standardized on WINWORD because "it's the word processor of the future." I can't express here how deeply I hate the thing, but everyone around me seems to just love it. I notice, however, that my contributing authors seem to spend an inordinate amount of time discussing style and appearance but seem to have very little time to devote to the study of content. It seems everyone is interested in making the document look better but don't seem very interested in making the content better.

And in there lies the essence of my worry. I remember how close you were to suspending publication of TMQ with IV.iv. I, for one, would rather have the TMQ single column on 8-1/2 by 11 sheets with a staple in one corner off a Xerox machine if it would help keep the editor's time to a minimum. I've done enough of this type of thing to know it must take a pretty good bit of your time to put the TMQ together.

With that said, I will close for now and let you get on with more important things — like filling orders. With any luck, I will have put an ASCII copy of this letter on the enclosed disk.

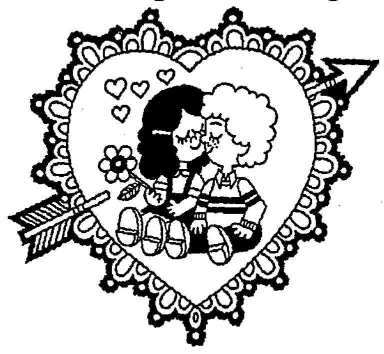
Fm MISOSYS, Inc: David, You raise a very good point. When folks who never had any exposure to page layout and design are thrown into a sophisticated word processor such as WINWORD, there is a tendency to spend more time on style than content. This is also true in the desktop publishing explosion; a good tool such as Pagemaker does not turn an *ordinary person* into a publication designer.

In my own case, I have spent some time in studying a few books on desktop publishing, so I at least have taken the time to learn about the job. I also spend time collecting and analyzing professional brochures I receive so that I may gain ideas on their layout. On the other hand, appearance does indeed matter to the receptivity of the document. This stems from the underlying psyche of most readers: a poorly presented paper will not get the attention of the reader regardless of content. Remember what we learned in school? A typewritten paper usually got a better grade because it was more legible! So it is with word processing. Granted, the content should be the only thing that matters, but we all know that is just not the case. Receive two papers: one printed on a 5 by 7 dot matrix printer using an old faded ribbon and one typeset looking very polished. Which one gets the better reception? Which presentation would you use to send out your resume? I'll surely bet that you would choose the typeset version. Why? Because it indicates to the recipient that you care to do the very best. When you are job hunting, you want your best foot forward!

In the case you presented, the problem rests with management (I was one once when I worked for AT&T). They need to ensure that the employees using WINWORD are (1) properly trained in its use, and (2) have the time to learn a little about page design. Without that level of devotion to their employees, they are only fooling themselves. Don't be upset with WINWORD, or WordPerfect, or Samna, or any other sophisticated tool which provides the ability to do professional page design. Be upset with the management who do not allow the users to become skilled with their tools. I'm sure they wouldn't hire a beginner off the street and assign that person to a sophisticated machine tool without training and education on the job. Sophisticated desktop publishing tools are no different.

As far as the questionable demise of TMQ, it was not due strictly to the amount of effort I spent in preparing each issue (and subsequently the time spent on page composition and style), but the relationship of that time to the subscriber base. When I originally set out to define TMQ back in 1986, my projection was for a subscription base of 3000. It was with that target that the 64-page issue was designed. The size exceeded 64 pages right from the outset: issues I.i and I.ii were 100 pages; issue I.iii was 106; I.iv through II.ii were 104; and so forth. I dropped the issues back to the original target of 64 pages for Volume V and will continue that through Volume VI. At its start, the subscription base was about 500. The level of subscriptions climbed slowly to a maximum of about 1000. That level lasted for a short time, only to initiate a downward trend. Issue V.iv's subscriber level was less than 500 (about 450, I recollect). Since I had worked to extend subscriptions to coincide with a volume of TMQ, most of the 450 subscribers had to renew for Volume VI; I mailed out 337 renewal notices. I am writing this on August 28th, 1991. There are currently only about 250 folks who are scheduled for Volume VI. So my time is even more uneconomical to be spent on TMQ for this volume. But I stuck with it, and I will make Volume VI as good as, if not better than, the last one. I can't, however, justify going beyond Volume VI with such a low level of subscriber participation.

Our capacity for love is infinite!
What about your disk storage?



Improve your on-line storage with 5.25" half-height 360K or 3.5" 720K floppy drives, or hard drive kits pre-assembled by MISOSYS

Floppy Drives & Accessories

5.25" 360K 1/2 ht	$75+$5S&H
3.5" 720K in frame	$85+$4S&H

Our dual vertical external floppy disk drive enclosure will hold two 5.25" half-height disk drives.

2SV5 drive case/PS	$60+$6S&H
8" 2-floppy extender	$18
4 foot 1-drive floppy cable	$12.50

Hard Drive Pre-assembled Kits

Our kits come assembled in a cabinet holding up to two half-height drives, 60 watt p/s, fan, SCSI controller, host adaptor, **hardware clock**, and cable. Software includes: driver, formatter, archive, restore, and sub-partitioning - all for LS-DOS 6.3 (LDOS software $30 additional). Drive kits work with Model III or Model 4 hardware.

20 Megabyte kit	$450+S&H
40 Megabyte kit	$575+S&H
Joystick port option	$20

Note: S&H for hard drives varies

S&H prices are UPS ground to continental U.S.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239

Inside TMQ

Pattern Matching: Another approach to fuzzy logic by Roy Soltoff

Modern day computers, as well as the bulk of those designed and manufactured over the past two to three decades, are based on bi-state logic. The phrase bi-state logic means that input to a device is in one of two states - generally classified as a logic 0 (FALSE) or a logic 1 (TRUE) - while the output also is in one of two states. A computer is simply built up from millions of these bi-state devices. That is also where we derive the term, binary.

All the processing of a computer emanates from the binary states of the devices it is built from. I'll use the term logic devices, which turns out to be an excellent term because it ties in with philosophy - the study of logic.

Binary logic can be easily used to represent bi-state conditions - i.e. conditions which can be in either one state or another. It's either raining or it's not raining; there can be no in-between. A string of characters either matches another string or it doesn't; there's no 78% match. We cannot directly use bi-state logic to represent the fact that clouds are obscurring 37% of the sunshine. We could, of course, write a computer program to run on a binary computer which could indeed accept an input of 63% sunshine and take some appropriate action. Or we could write some function to compare two character strings and tell us that they are 78% alike.

When I studied philosophy in college, the classical structure of logic was presented: conclusions are deduced from premises.

All men are mortal

Socrates was a man

Therefore Socrates was mortal

The study of philosophy also presented the use of inexact terminology. One could reach certain conclusions where premises used such inexact terms as most, some, and few. In very recent years, there were claims of new developments in an area called fuzzy logic. Fuzzy logic is simply the black box evaluation of inexact inputs to produce an output. Instead of adhering to logic devices which exist in binary states, a few companies are putting fuzzy logic into silicon. The devices themselves directly handle multistate digital values without resorting to analog operations. Personally, I don't see this as any different from the philosophy I studied back in the 60's; however, there is some inherent benefits to fuzzy logic in digital hardware.

Where is such fuzzy logic used? It's making inroads into many areas. Most of you have seen ads for the Pansonic Palmcorder - an extremely lightweight camcorder described as utilizing digital EIS stabilization, a stabilization system based on fuzzy logic. It uses fuzzy logic circuitry to eliminate the shakes caused by the inability of the hand to remain steady while holding such a lightweight object. Fuzzy logic is also being used in elevators, trains, and I suspect will be in our automobiles soon. So what does fuzzy logic have to do with us? Plenty!

I make many mistakes while typing; I'm not a touch typist, but rather a multi-finger typist. I also make character inversion errors - which are a classic type of typing error caused by two fingers stroking keys out of the proper sequence. For instance, I so frequently type "MEMROY" for "MEMORY" - perhaps because of the affinity to my name - that Logical Systems once sent me a DOS during development which had the "memory" command in

```
      TITLE SIMIL-S WRITTEN BY JOHN W. RATCLIFF AND DAVID E.
METZENER

; Original MS-DOS version: November 10, 1967

; Ported to Z80 code by Roy Soltoff: August 1991

; Uses the Ratcliff/Obershalp pattern recognition
; algorithm. This program provides a new function to C on a
; Z80 based machine. The function SIMIL returns a
; percentage value corresponding to how alike any two
; strings are. Be certain to upper case the two strings
; passed if you are not concerned about case sensitivity.


*INCLUDE MCMACS.ASM


      PUBLIC SIMIL
      EXTRN  @MULT,@UDIV

      DSEG
STR1L EQU   0          ;Offsets to low-order elements
STR1R EQU   2
STR2L EQU   4
STR2R EQU   6
; static struct {
; char *l1; contains lefts for string 1
; char *r1; contains rights for string 1
; char *l2; contains lefts for string 2
; char *r2; Contains rights for string 2
;}pstr[25];          ;Will be allocated on the stack

PSTACK      DS    2      ;Pointer to struct array
SPSAV       DS    2      ;regSP save area
STCKNUM     DS    1      ; number of elements on the stack
SCORE       DS    2      ; the # of chars in common times 2
TOTALDS     2          ; total # of chars in string 1 and 2
CL1   DS    2          ; left of string 1 found in common
CR1   DS    2          ; right of string 1 found in common
CL2   DS    2          ; left of string 2 found in common
CR2   DS    2          ; right of string 2 found in common

      CSEG

; This routine expects pointers passed to two character
; strings, null terminated, that you wish compared. It
; returns a percentage value from 0 to 100% corresponding
; to how alike the two strings are.
;
; usage: simil(char *Str1,char *str2)
; The similarity routine is composed of three major compo-
nents
; pushst — pushes a string section to be compared on the
stack
; popst  — pops a string section to be examined off of the
stack
; compare — finds the largest group of characters in common
between
;            any two string sections
;
; The similarity routine begins by computing the total
; length of both strings passed and places that value in
; TOTAL. It then takes the beginning and ending of both
```

SYS1's table patched to "MEMROY"! How many other simple inversions or mis-types do you make? Plenty, I'm sure.

Wouldn't it be neat if instead of the DOS requiring exact matches for commands, it accepted a character string with a nearly exact match? Perhaps if your entry was 90% similar or 87% similar, it would accept it. And if your string matched two or three commands, it would pop up the ones closest to your entry and allowed you to select one instead of a *Program not found* error.

Well, just how do we determine the percentage of similarity one string has to another? A fuzzy technique can help here. No, I'm not going to resort to fuzzy logic hardware, but rather employ an algorithm. In 1988, I came across an article in *Dr Dobb's Journal.* The article was entitled "PATTERN MATCHING: THE GESTALT APPROACH". Written by John W. Ratcliff and David E. Metzener, it presented an introduction to the Ratcliff/Obershelp pattern-recognition algorithm. The algorithm operated on two strings and calculated an index of similarity between the two; the index ranged from 0-100 and indicated on a percentage basis, how alike the two strings were. The original algorithm was developed in 1983 for use with educational software. It has utilization for spelling checkers, database search processes, and anywhere you would like to be somewhat forgiving with a character entry matching up with a given table of valid strings. I clipped and saved the article for consideration of the routine with a revision to my LB data base manager.

The algorithm operates on the two strings and locates the longest substring of matching characters. This operation divides the two strings and surrounds the matched substring with left and right substrings, either of which could be null if the substring was positioned extreme left or right rather than mid. The algorithm continues to scan and partition substrings until no matching characters remain. The similarity index is then calculated as the count of matching characters divided by the total number of characters multiplied by 100 to give a percentage.

Let's look at a few examples. In the strings "ABCDEF" and "ABCXEFG", the algorithm first finds "ABC" as the longest matching substring. This partitions the strings into null left strings with "DEF" and "XEFG" right substrings; the count so far is six. "EF" is the next largest matching substring partitioning the two remaining substrings into lefts of "D" and "X" with rights of null and "G". The count is now ten and the remaining substrings have no further matches. The similarity index is therefore 10/13*100=76. Using integer mathematical operations, any fractional part is truncated.

Let's take a look at my "MEMROY" versus "MEMORY". The first pass extracts "MEM" and leaves null lefts with "ROY" and "ORY" rights and a count of six. The next pass extracts "R" leaving null and "O" lefts with "OY" and "Y" rights and a count of eight. The final pass extracts the "Y" calculating a similarity index of 83 (10/12*100).

The original article, published in the July 1988 issue of *Dr Dobb's Journal*, included an 8086 assembly language SIMIL function callable from C-language programmed under MS-DOS. I took the liberty of porting the function to Z80 assembly language, which was somewhat of a struggle due to register requirements. Obviously, the program needs to operate fast if one is going to use it frequently. That dictates tight code and effective use of CPU registers.

I derived TSIMIL.CCC from the article, and adapted it to PRO-MC to (1) demonstrate use of simil() and (2) demonstrate MC's options for echoing keyboard characters and handling the BREAK key as end-of-file - options not always obvious to the casual reader of MC's User Manual.

One significant use I am making of this similarity function is in the addition of a command function to my LB data base manager, which attempts to find duplicate

```
; strings passed and pushes them on the stack. It then
; falls into the main line code. The original two strings
; are immediately popped off of the stack and are passed to
; the compare routine. The compare routine will find the
; largest group of characters in common between the two
; strings. The number of characters in common is multiplied
; times two and added to the total score. If there were no
; characters in common then there is nothing to push onto
; the stack. If there is exactly one character to the left
; in both strings then we needn't push it on the stack.
; (We already know that they aren't equal from the previous
; call to compare). Otherwise the characters to the left
; are pushed onto the stack. These same rules apply to
; characters to the right of the substring found in common.
; This process of pulling substrings off of the stack,
; comparing them, and pushing remaining sections on the
; stack is continued until the stack is empty. On return
; the total score is divided by the number of characters in
;both strings. This is multiplied times 100 to yield a
; percentage. This percentage similarity is returned to the
; calling procedure.

SIMIL:          $GA     DE,HL           ;Get pstr2, pstr1
        PUSH    IX              ;Save local registers
        PUSH    IY
        XOR     A               ;Initialize
        LD      (STCKNUM),A     ;Nothing in array stack at first
        LD      H,A
        LD      L,A
        LD      (SCORE),HL      ;Start score at 0
        LD      (PSTACK),SP     ;pstack points to array end
        LD      HL,-200         ;Allocate 200 bytes on stack
        ADD     HL,SP           ;   for the array of structures
        LD      SP,HL
        LD      H,B             ;Move arg1 to HL
        LD      L,C
        LD      B,A             ;Set BC to 0
        LD      C,A
        CP      (HL)            ;is string2 null?
        JR      Z,STRERR        ;can't process null strings
        EX      DE,HL           ;S1 -> HL; S2 -> DE
        CP      (HL)            ;is string1 null?
        JP      NZ,DOCMP        ;neither is a null string so pro-
cess them
STRERR: JP      DONE            ;exit routine
DOCMP:          PUSH    HL              ;Save L1
        PUSH    DE              ;Save L2
        CPIR                    ;scan for string delimiter in
string 1
        DEC     HL              ;point HL to '$00' byte of string
1
        DEC     HL      ;Point HL to last character of string 1
        EX      DE,HL   ;move R1 to DE where it is supposed to be
        CPIR            ;scan for string delimiter in string 2
        DEC     HL              ;point HL to '$00' byte of string2
        DEC     HL      ;point HL to last character of string 2
        PUSH    HL              ;Save R2
        LD      HL,-1           ;Calculate combined length of both
strings
        SBC     HL,BC
        LD      (TOTAL),HL      ;store string2's combined length
        POP     BC      ;move R2 to BC where it is supposed to be
        POP     IY              ;Get L2
        POP     IX              ;restore DI to what it should be
```

```
     CALL   PUSHST          ;Push values for the first call to
SIMILIARITY
MAIN:LD     A,(STCKNUM)
     OR     A               ;is there anything on the stack?
     JP     Z,DONE          ;No, then all done!
     CALL   POPST ;get regs. set UP for a COMPARE call
     CALL   COMPARE         ;do compare for this substring set
     LD     A,L             ;Get # of chars in common
     OR     A               ;if nothing in common than nothing
to push
     JP     Z,MAIN          ;try another set
     LD     HL,(SCORE)      ;SHould be *2, calc at exit
     ADD    A,L             ;add into score
     LD     L,A
     JR     NC,$+3
     INC    H
     LD     (SCORE),HL
     LD     IX,(PSTACK)     ;get entry I want to look at
     LD     DE,(CL1)        ;move CL1 into DE or R1
     LD     BC,(CL2)        ;move CL2 into BC or R2
     LD     L,(IX+STR1R-8)        ;get old R1 of of stack
     LD     H,(IX+STR1R+1-8)
     LD     (CL1),HL        ;place in CL1 temporarily
     LD     L,(IX+STR2R-8)        ;get old R2 off of stack
     LD     H,(IX+STR2R+1-8)
     LD     (CL2),HL        ;save in CL2 temporarily
     LD     L,(IX+STR2L-8)        ;move L2 into IY or L2
     LD     H,(IX+STR2L+1-8)
     PUSH   HL              ;Save twice
     PUSH   HL
     POP    IY
     LD     L,(IX+STR1L-8)        ;move L1 into IX or L1
     LD     H,(IX+STR1L+1-8)
     PUSH   HL
     POP    IX
     OR     A               ;Clear CF for subtract
     SBC    HL,DE           ;Compare CL1 to L1
     POP    HL              ;Get L2 into HL
     JP     Z,CHRGHT        ;if zero, then nothing on left
side string 1
     OR     A               ;Clear CF for subtract
     SBC    HL,BC           ;compare CL2 to L2
     JP     Z,CHRGHT        ;if zero, then nothing on left
side string 2
     DEC    DE        ;point to last part of left side string 1
     DEC    BC        ;point to last part of left side string 2
     PUSH   IX
     POP    HL              ;L1 to HL
     OR     A
     SBC    HL,DE           ;only one character to examine?
     JP     NZ,PUSHIT       ;no-'we need to examine this
     PUSH   IY
     POP    HL              ;L2 to HL (CF still reset)
     SBC    HL,BC           ;only one character in both?
     JP     Z,CHRGHT        ;nothing to look at if both only
one char
PUSHIT: CALL        PUSHST        ;push left side on stack
CHRGHT: LD  IX,(CR1)        ;move CR1 into IX or L1
     LD     DE,(CL1)        ;move R1 into DE or R1
     LD     IY,(CR2)        ;move CR2 into IY or L2
     LD     BC,(CL2)        ;move R2 into BC or R2
     PUSH   IX
     POP    HL
     OR     A
```

records. A classical problem of those involved in direct mail is the automatic elimination of duplicate records - those records which are similar, or nearly similar to others. Since mailing multiple copies of a direct mail piece to the same address wastes money, direct marketers take great pains to minimize the duplicate records in their data base. Duplicates can be caused by many factors. One record could be *Robert* with the duplicate being *Bob*. Or one could have a *Junior* after the name. Or the street could be *One Tyler* vs *1 Tyler*. I have already been using LB86 version 2.2 in-house which has this feature added. It allowed me to finally combine three distinct data bases: LSI's 6.3.0 registrations, Powersoft's TRSCROSS registrations, and our own customer base.

One technique to detect duplicates is to compare the first record against all subsequent records; then compare the second; and so forth. This can take an enormous amount of time on a micro with a database of even a few thousand records. That's one reason why direct mail houses perform list management on mainframes.

On the other hand, one can simplify the technique using knowledge of the data. Here's what I did. LB users may know that LB keeps a data structure at the beginning of a select file; one element of the structure is the key field number. My first attack was to expand the data structure of LB's select file to include the field numbers of all fields attached to the sort string present when the select or index file is generated. Thus, instead of a single key field, all fields sorted on become associate keys. This has merit for more than finding duplicates; I will also be using the multiple keys to enhance the edit module's **find** command to enable you to find the record which matches multiple key fields. For instance, when I get a call on the order line, I will pop up your record on my database. But if John Paul Jones calls, or Jeff Smith calls, I have to find the first Smith or Jones then single step through the records until I see the first name. Believe me, there are many Smiths, Jones, and Williamses in my database. How nice to be able to search for "Smith\Jeff".

Continuing on with finding duplicates, I then was able to develop an algorithm in which the database was examined based on the sequence specified in a sorted index file. The first record was compared to the second, the second with the third (could have triplicates), the third with the forth, and so forth. The record numbers of similar records would be written to another index file. I added the option of selecting the record number of either the first matching, the second matching, or both matching. Typically, if it didn't matter which similar record was deleted, you could select first or second match then use the automatic purge function of the LBMANAGE utility to automatically delete one of the matching pair of records. If you wanted to make sure which record to delete, you would select both record numbers for the output index file, then step through the edit mode with the new index file attached to make the determination after a visual inspection of the data.

Without additional intelligence, it would be possible to generate duplicate record numbers into the index file for triplicately (?) matching records. For instance, if record 5 matched 6 which matched 7, and you chose to output both matching numbers, comparing 5 to 6 would output 5 and 6 while comparing 6 to 7 would output 6 and 7; record number 6 would be generated twice! So the algorithm inhibits the record number's output if it has been immediately previously output.

Comparing adjacent records will miss lots of potential duplicates if the sorted order of the primary key places similar records farther apart. For example, my database uses a field for last name, another for first name, another for company name. I usually sort on last-first-company. But when I enter a name like *John Paul Jones, Jr.*, I put "John Paul" into the first name field and "Jones, Jr." into the last name field. A duplicate record without the ", Jr" would not sort to be adjacent. Thus, such a record would not fall out of a similarity check. But I also keep an index of the data sorted

```
      SBC     HL,DE            ;compare CR1 to R1
      JP      Z,MAIN           ;if zero, tnen nothinq on right
side string 1
      PUSH    IY
      POP     HL
      OR      A
      SBC     HL,BC            ;compare CR2 to R2
      JP      Z,MAIN           ;if zero, then nothing on right
side strinq 2
      INC     IX      ;point to last part of right side string1
      INC     IY      ;point to last part of right side string2
      PUSH    IX
      POP     HL
      OR      A
      SBC     HL,DE            ;only one character to examine?
      JP      NZ,PUSH2         ;no->examine it
      PUSH    IY
      POP     HL
      SBC     HL,BC  ;only one character to examine in both?
      JP      Z,MAIN           ;yes->get next string off of stack
PUSH2:        CALL    PUSHST           ;push right side on stack
      JP      MAIN             ;do next level of compares
DONE: LD      HL,200 ;Deallocate array from stack
      ADD     HL,SP
      LD      SP,HL
      LD      HL,(SCORE)       ;get score into HL for MUL
      LD      A,H              ;Don't bother to calculate
      OR      L                ;  if zero
      JR      Z,DONIT
      LD      DE,100*2         ;get 100 into DE for MUL (*2 for 2
strings)
      CALL    @MULT            ;multiply by 100
      EX      DE,HL            ;Result to DE
      LD      HL,(TOTAL)       ;get total characters for divide
      CALL    @UDIV            ;Divide by total, result in HL
DONIT:        POP     IY       ;Restore locals
      POP     IX
      RET             ;Leave with HL holding % similarity

; The compare routine locates the largest group of charac-
ters
;      between string 1 and string 2.
; Pass to this routine:
;     DE     - R1 (right side of string 1)
;     IX     - L1 (left side of string 1)
;     IY     - L2 (left side of string 2)
;     BC     - R2 [right side of string 2)
;
; This routine returns:
;     L        - # of characters matching
; CL1 - Left side of first string that matches
; CL2 - Left side of Second string that matches
; CR1 - Right side or first string that matches
; CR2 - Right side of second strinq that matches
; The compare routine is composed of two loops, an inner
; and an outer. The worst case scenario if that there are
; absolutely no characters in common between string 1 and
; string 2. In this case N x M compares are performed.
; However, when an equal condition occurs in the inner
; loop, then the next character to be examined in string 2
; (for this loop) is advanced by the number of characters
; found equal. Whenever a new maximum number of characters
; in common is found then the ending location of both the
; inner and outer loop is backed off by the difference
; between the new max chars value and the old max chars
; value for both loops. In short if 5 characters have been
; found in common part of the way through the search than
; we can cut our search short 5 characters before the
; true end of both strings since there is no chance or
; finding better than a 5 character match at that point.
; This technique means that an exact equal match will
; require only a single compare and combinations of other
; matches will proceed as efficiently as possible.

COMPARE: LD L,0          ;Init MAXCHARS
FOR13:       PUSH   IY   ;Save start at string 2
FOR14:       PUSH   IY   ;Save start of string 2
    PUSH    IX           ;Save start of string 1
    LD      H,0          ;Init current match count to 0
    JP      FOR16
FOR15:       INC    IX   ;Bump pointers
    INC     IY
    INC     H            ;Bump current match count
FOR16:       LD     A,(IY)      ;compare strings
    OR      A            ;End of S2?
    JR      Z,EQUAL
    CP      (IX)         ;S1 match?
    JP      Z,FOR15      ;Loop until different
EQUAL:       LD     A,H  ;Did any match?
    OR      A            ;get length of common characters
    JP      NZ,NEWMAX    ;more than 0 chars matched
    POP     IX           ;get back start of string 1
    POP     IY           ;get back start of string 2
REENT:       INC    IY   ;Do the next character always

REENT2: PUSH     IY      ;Compare IY to BC
    EX      (SP),HL      ;  without affecting HL
    OR      A
    SBC     HL,BC        ;Are we done with string 2?
    POP     HL
    JP      C,FOR14      ;No, then do next string compare
    JP      Z,FOR14      ;Must check last char
    POP     IY           ;get back start of string 2
    INC     IX           ;next char in string 1 to scan
    PUSH    IX           ;Compare IX with DE
    EX      (SP),HL      ;  without affecting HL
    SBC     HL,DE        ;Are we done with string 1? (CF
was reset)
    POP     HL
    JP      C,FOR13      ;No, then do next string compare
    JP      Z,FOR13              ;Must check last char
    RET                  ;MAXCHARS is in H; current in L
;
; We branch downwards for both newmax and newmx2 because on
; the Z80 line of processors a branch not taken is much
; one which is. Since the not equal condition is to be
; faster than found most often and we would like the inner
; loop to execute as quickly as possible we branch outside
; of this loop on the less frequent occurrence. when a
; match and or a new maxchars is found we branch down to
; these two routines, process the new conditions and then
; branch back up to the main line code.
NEWMAX: LD  A,L          ;Current MAXCHARS
    CP      H            ;New count greater than MAXCHARS?
    JR      C,NEWMX2     ;yes, update new maxchars and
pointers
    POP     IX           ;get back start of string 1
    POP     AF     ;Discard old start of string 2, save HL
```

by ZIP code and last name. A similarity check on that index would pop out the duplicate Jones'. LSI's database had about 12,000 records. TRSCROSS registrants numbered about 8,000. My database numbered about 22,000 but I extracted only about 10,000 active records. I then developed a new database definition adding some additional fields. I merged all three together using a combination of AUTO input, LBREDEF, and LBMANAGE. I then used the new in-house select module command to generate the record numbers of duplicates at about an 85% match. I used both the alphabetic name sort and the ZIP sort. The first produced about 5,000 record numbers which I manually examined in edit mode to delete those I felt were real duplicates. I had to ensure that I kept any record which had a TMQ entry in it. The ZIP check subsequently produced about 2900 record numbers. This process certainly didn't catch all the duplicates as I couldn't be sure of records with matching names but different addresses; which one was the correct one?

All in all, the process went considerably faster than if I were to attempt the job on totally a manual basis. It pays to have a fast hard drive for this operation. I keep the customer database on a 286-10MHz AST machine with a 28ms 40Meg hard drive. Running the duplicate check on 25,000 or so records took about 45 minutes. That represents a lot of bouncing around on the drive as the raw data is not in alphabetic order, and so accessing the records according to the ordered index file resulted in a great deal of seeking on the drive.

With the Z80 version of SIMIL completed and tested, and integrated into LB, I can get back to that 2.2 release once this issue of the *Quarterly* is complete. I have a few more tidbits I want to put into this next LB release besides the duplicate check. I already mentioned the multiple key find. I believe I want to investigate the feasibility of adding a control tab to the print generation. The difficulty of any enhancement is the ability to squeeze more functionality into the TRS-80 version without bumping up against memory constraints. But I will do some tweaks here and there. I also want to add another module to automatically create standard print screens and add/edit screens once the data definition is complete. After the 2.2 version, I expect to get started on an MSDOS version to run under Deskmate.

Now let me shift to the techniques I employed in writing SIMIL.ASM, as there are some techniques which may be new to my readers.

The first task you need to tackle when writing any complicated function in assembly language is to map out the use of the CPU registers. The Z80 supports two 16-bit index registers, IX and IY, which have certain advantages and disadvantages for use. There are also two banks of register pairs which can be accessed as either 8-bit or 16 bit: the regular bank is HL, DE, and BC, while the alternate bank is HL', DE', and BC'. Only one bank is accessible at a time; you switch from one to the other via the EXX instruction. There are also two accumulator/flag registers: AF and AF'; again, only one is accessible at a time with the EX AF,AF' instruction used to switch between them. Classsically, the primed registers (alternates) have been reserved for use in interrupt tasks. As such, for any properly written program to make use of the primed registers, their values should be saved on the stack - or elsewhere - then restored.

SIMIL needs to maintain pointers to the beginning and ending characters of two strings which are being compared. In addition, it needs to have access to additional byte counters. I chose to use the 16-bit index registers IX and IY as pointers to the beginning of the two strings, and register pairs DE and BC as pointers to the ends of the strings. Registers H and L were used as local 8-bit register variables. As will be explained later, the SP register is also used as a pointer to a data stack. I avoided use of the alternate or primed registers.

SIMIL is written to be a function callable

```
        JP      REENT2          ;re-enter inner loop
NEWMX2:         LD      A,L             ;Old MAXCHARS
        SUB     H               ;get delta for adjustment
        LD      L,A             ;  to ends of strings
        LD      A,H             ;save new maxchars tempy in regA
        LD      H,-1            ;Set up HL as 16-bit negative
        INC     HL      ;Adjust to point to exact character
        EX      DE,HL           ;Difference to DE, R1 to HL
        ADD     HL,DE           ;adjust end of string 1
        EX      DE,HL           ;New R1 to DE
        ADD     HL,BC           ;adjust end of string 2
        LD      B,H             ;New R2 to BC
        LD      C,L
        POP     HL              ;get back start of string 1
        LD      (CL1),HL        ;put begin of match of string 1
        POP     HL              ;get Back start of string 2
        LD      (CL2),HL        ;put begin of match of string 2
        LD      L,A             ;New MAXCHARS into regL
        DEC     IY      ;back up to last matching char string 2
        LD      (CR2),IY        ;put end of match of string 2
        DEC     IX      ;back up to last matching char string 1
        LD      (CR1),IX        ;put end at match of string 1
        JP      REENT           ;re-enter inner loop

; On entry:
;    IX - L1 (left side of string 1)
;    IY - L2 (left side of string 2)
;    DE - R1 (right side of string 1)
;    BC - R2 (right side of string 2)

PUSHST:         LD      (SPSAV),SP          ;Save stack pointer
        DI                              ;ints off for stack depth
        LD      SP,(PSTACK)             ;Point to last struct element
        PUSH    BC                      ;Save R2
        PUSH    IY                      ;Save R1
        PUSH    DE                      ;Save L2
        PUSH    IX                      ;Save L1
        LD      (PSTACK),SP     ;pointer to next free struct ele-
ment
        LD      HL,STCKNUM
        INC     (HL)    ;Add one to number of stack entries
        EI                              ;Ints back on
        LD      SP,(SPSAV)
        RET

;    IX - L1 (left side of string 1)
;    IY - L2 (left side of string 2)
;    DE - R1 (right side of string 1)
;    BC - R2 (right side of string 2)

POPST:          LD      (SPSAV),SP          ;Save stack pointer
        DI                              ;ints off for stack depth
        LD      SP,(PSTACK)             ;Point to last struct element
        POP     IX                      ;restore left side of string 1
        POP     DE                     ;restore right side of string 1
        POP     IY                     ;restore left side of string 2
        POP     BC                     ;restore right side of string 2
        LD      (PSTACK),SP          ;Save pointer to next last entry
        LD      HL,STCKNUM            ;Reduce number of stack entries
        DEC     (HL)
        EI                              ;Ints back on
        LD      SP,(SPSAV)
        RET

        END
```

```
#include <stdio.h>
#include <string.h>
#include <sgtty.h>
int simil();
char str1[80], str2[80];
main()
{
      int percent;
      static struct sgttyb sg = { 0, 0 };
      option(O_KBECHO,TRUE);              /* turn on keyboard
echo */
      ioctl(STDIN,TIOCGETP,&sg);/* get input controls */
      sg.sg_control |= IO_BREAK;          /* enable BREAK for
EOF */
      ioctl(STDIN,TIOCSETP,&sg);/* put input controls */

      printf("This program demonstrates the Ratcliff/
Obershelp pattern\n");
      printf("recognition algorithm.  Enter a series of word
pairs to\n");
      printf("discover their similarity values.\n\n");
      while (TRUE)
            {
            printf(" Enter the first string: ");
            if (!(gets(str1)))
                  break;
            printf("Enter the second string: ");
            if (!(gets(str2)))
                  break;
            percent = simil(strupr(str1),strupr(str2));
            printf("%s and %s are %d%%
alike.\n\n",str1,str2,percent);
            }
}
```

from C, specifically MC. It uses the argument passing convention employed in MC: arguments to the function are placed on the stack immediately prior to the return address and the returned integer value is passed in the HL register pair. As such, the code uses an MC macro to gain access to the arguments and library functions for math calculations which I'll explain here for those of you not yet adapted to the C environment.

The $GA macro simply pops the arguments off of the stack, then restores the stack to its entry state; i.e. it generates a series of POPs into multiple registers then PUSHes the values back onto the stack. SIMIL also uses two math functions from MC's LIBA/REL library: @MULT which multiplies two 16-bit integers, and @UDIV which divides one unsigned 16-bit integer by another.

Note that my version of SIMIL uses a storage region of 200 bytes allocated on the stack to store pointers; these are pointers to the beginning and end of left and right substrings which remain to be compared and partitioned. The 8086 version of SIMIL uses a static memory declaration and divides the region into four arrays of 25 pointers. It's easy to index the array elements in 8086 assembly language as instructions are available to access memory word values (16-bits) with 16-bit offsets to a register value. The only index instruction in Z80 code permits 8-bit memory access using an IX or IY index register coupled with 8-bit offsets. That makes it rather clumsy in Z80 code to load and store multiple pointers directly through register loads and stores. Therefore, I decided to establish the pointer arrays as an array of structures of pointers to characters. In this way, the structure contains four pointers - which are stored in sequential memory addresses. The array of structures then just repeats this sequence 25 times. I use stack instructions to capture and store data from and to this *local data stack*. Thus, I am not using the array space as a program stack but rather as a data stack. Certain processors actually provide two stack pointers: a program stack pointer and a data stack pointer. In the Z80 CPU, we have only one stack pointer - the SP register. We can use that for both data and program stack access if we take certain precautions which I'll touch on later.

One basic decision had to be addressed with the 200-byte structure array, and that was where to put it. One of the powerful aspects of the C programming language is the use of dynamic memory allocation. Library functions exist to allocate blocks of memory from a large heap and free up such blocks when a program is finished with the block(s). This has the advantage of not reserving memory for use by a function until that function was in execution. It's like sharing one long stick amongst many girl scouts toasting marshmellows over an open fire. But dynamic memory allocation also has the disadvantage of processing overhead; it makes your program run slower. With finite memory space, and competing interests in that space at various times, dynamic memory allocation may be required to fit a given program into the available memory. With 200 bytes declared statically within the SIMIL function, that amount of memory would always be reserved. If SIMIL was to be used in a program as the only function, that would not introduce extraneous memory reservations. But I was making it a part of LB's select module, which incorporated additional functionality. I hesitated to consume another 200 bytes when the duplicate checking function was not going to be used. On the other hand, using dynamic memory allocation to allocate 200 bytes from the heap on each call to SIMIL was considered to be a very negative impact on the execution time of the function. Thus, my decision was to compromise and obtain the 200 bytes as a local array allocated from the program stack rather than use alloc() to obtain the space or declare the space static within the function as part of the data segment. Obtaining memory space from the stack uses but a few additional instructions. Note also that since the array is being accessed as a stack, the first element is actually at the end of the space (higher memory location) while the 25th element is at the beginning of the space.

The first part of SIMIL - up through DOCOMP - simply performs some initialization. I chose to use the IX and IY index registers as local pointers, therefore it is essential to save these registers on entry to the program. MC uses IX and IY as the storage registers for variables declared register. Any assembly language function written which uses IX and/or IY must preserve the values of those registers.

During initialization, SIMIL uses the current stack pointer as a pointer to the next available array element in the structure and stores this pointer in PSTACK. It subsequently reserves 200 bytes of program stack space for the data array of structures by adding -200 to the current value of SP and placing the new value back into SP. SIMIL will need to de-allocate that 200 bytes on exit of the function. SIMIL also immediately exits if it finds either of the strings are null. Obviously, a null string has zero characters in common with another string - even another null string.

The next section of SIMIL needs to calculate and then store the total length of both strings. I used the CPIR instruction for this purpose. CPIR performs the following sequence of steps:

1. compares the contents of HL with A

2. increments HL

3. decrements BC

4. repeats steps 1-3 if not equal or BC <> 0

Strings in C are terminated with a zero byte value. A two-character string, such as "AB", will be stored in hexadecimal as 41 42 00 xx; "xx" just indicating the next memory location for convenience of this discussion. With register pair BC and register A initially at 0, a CPIR with HL pointing to this string will result in HL pointing to the "xx" and BC equal to 0FDH (-3). That is the result because CPIR increments HL and decrements BC before it acts on the result of the compare. If at this point we performed another CPIR on another two-character string, BC would contain 0FAH (-6). So I can calculate the length of the two strings by using the value which remains in register BC. It is two less negative than the total length of the two strings combined. If I then subtract that value from negative two, I should obtain the true length as a positive number. The Z80 supports only a 16-bit subtract with borrow (SBC) instruction. You would normally have to ensure that the carry flag is reset or cleared for an SBC - unless, of course, it was part of a multi-precision subtraction. But wait, I can take advantage of the state of the carry flag in my SBC instruction. Note that the CPIR instruction does not affect the Z80 carry flag (CF). Note also that the CF is **always** set on entry to DOCMP as the previous instruction **always** compares a non-zero character to a zero value which results in the CF being set. By using a subtract with carry (SBC) instruction, I subtract the result from **minus one** without having to reset the CF and subtract from minus two. This results in [-1 (-)-6 -CF] = [-1 +6 -1] = 4, the proper length. I thus am able to avoid including an OR A instruction to clear the carry flag in this calculation. You can sometimes capitalize on the state of a Z80 flag bit, or register value, to reduce the length of a code fragment if the flag bit or register value will always contain a known value. Make use of that for professional code.

Let's take a quick look at the subroutines used to store the string pointers into the array stack and subsequently get them back into registers.

PUSHST switches the Z80 stack pointer register, SP, from use as a program stack pointer to a data stack pointer. Under LS-DOS and LDOS, interrupt handling will require stack space. Since we don't want any other use made of our data region while we have the SP register pointing to it, the Z80 interrupts are disabled just prior to the SP alteration and are restored after SP is restored to its original value. Once SP is pointing to an array element, all it takes are four PUSH instructions to store the four pointers which need to be preserved. The POPST subroutine performs the opposite result; it saves the current SP, loads SP with the current data stack value, then pops the values back into registers. Then it restores SP. Interrupts are again disabled during the time which the SP register is accessing the data stack.

The current count of matching characters - the score - is kept as a 16-bit value in memory. If I wanted to assure myself that I wouuld never be comparing a string of greater than 255 characters, I could limit it to an 8-bit value and save some instructions when adding the new count to the previous count. However, using 16-bit additions gives SIMIL some additional flexibility. But in any given implementation, one should program according to the requirements - or limitations - of the data. 16-bit additions take more time than 8-bit and also take more storage space.

Note that SIMIL is adding an 8-bit value in the accumulator to a 16-bit value in memory. The Z80 does not support an instruction which does a direct add to memory. So the score is loaded first into register pair HL. Therein poses a dilemma. The Z80's 16-bit add instructions permit you to add only the following registers to HL: BC, DE, SP, and HL. The latter is quite useful for multiplying by two (or shifting the contents of HL left by one bit). Since BC and DE are already in use as pointers, the option is to either push DE onto the stack, load the accumulator into register E, load a zero into register D, add DE to HL, then POP DE from the stack, or add the accumulator to HL via a series of 8-bit add instructions. The following sequence easily does this:

```
        ADD     A,L     ;Add into
lo-order
        LD      L,A     ;Store
new lo-order
        ADC     A,H     ;Add in
hi-order with any carry
        SUB     L       ;Subtract
off un-needed lo-order
        LD      H,A     ;Store
hi-order
```

The third and fourth instructions of this sequence are another way of incrementing the hi-order byte by one if the result of the lo-order addition generated a carry. The following code does the same thing:

```
        ADD     A,L     ;Add into
lo-order
        LD      L,A     ;Store
new lo-order
        JR      NC,$+3 ;Skip the
INC if no carry
        INC     H       ;Inc to
add in the carry
```

Let's examine the difference in timing for these two code streams. The first has a fixed execution time of 20t. The second's execution time varies with the carry condition. On no carry, it is 20t whereas with a carry it is 19t. By examining in detail the execution of the two code streams in terms of t states, it is evident that the second edges out the first in speed in those adds which result in an 8-bit overflow and is identical in speed when there is no overflow. So I'll use the later in SIMIL. The first sequence, incidentally, would be used on an 8080 CPU which has no relative jump instruction.

The bulk of the code in main shuffles pointers from storage to registers and checks if certain pointers are equal. One thing to point out is the reason for the "-8" added into the offset calculation when accessing a set of values from the data stack. The value that needs to be accessed is the value which has just been popped off of the stack, so PSTACK is not pointing to that array structure but the next lower one (but higher in memory since the stack is accessed from the top down). So I subtract eight in the offset calculation; eight being the size of the structure element. Also at this point, I am able to use the IX register as a pointer to the data structure but am forced to access the data as individual bytes.

Another inadequacy of Z80 code is the absence of a 16-bit compare instruction. Thus, in order to compare IX with DE and IY with BC, I must transfer the contents of IX to HL and perform an SBC instruction. Similarly, I transfer IY to HL and SBC HL,BC to compare IY with BC.

The calculation of the percentage similarity uses MC's 16-bit multiply and 16-bit unsigned divide. When characters are found to match, score accumulated only the count of characters in one string. Obviously for every character in one string, there was a corresponding character in the other string. I could have doubled the count before accumulating into score, or just doubled the score at the end. This implementation doubles the score at the end since it does that but once, whereas doubling the intermediate match counts would result in four additional t-states per match (Yes, I know, multiplying by 200 is probably slower than multiplying by 100). I first multiply by 200 (100 times the 2 to double) so that the significant part of the score remains an integer larger than the total. I then divide the result by the total number of characters. This produces a result ranging from 0 to 100. If you wanted to adapt this function external to MC for LS-DOS or LDOS, you could replace the @MULT and @UDIV with appropriate @MUL16 and @DIV16 service calls with some additional register manipulation.

The compare routine is essentially composed of two nested loops which iterate through the two character strings to capture and extract matching sequences while partitioning the remaining portions. The loops continue until no further matching characters are found. In this section the code also has to compare IX with DE and IY with BC, but register HL cannot be disturbed. The following code demonstrates the technique:

```
        PUSH    IX      ;IX to
stack
        EX      (SP),HL
;Get IX's value to HL, and
HL to stack
        OR      A       ;Clear
carry flag
        SBC     HL,DE ;destruc-
tive compare HL:DE
        POP     HL      ;Restore
original HL
```

This sequence results in a zero or non-zero condition and all registers are preserved (except the flag register, of course).

USED TRSDOS USED XENIX

RADIO SHACK TANDY OWNERS!

Find the computer equipment that TANDY no longer sells.

PACIFIC COMPUTER EXCHANGE
buys and sells *used* TANDY

TRSDOS
XENIX
MSDOS
COMPUTERS &
PERIPHERALS

We sell everything from Model 3's and 4's to Tandy 6000's, 1000's to 5000's, Laptops, and all the printers and hard disks to go with them. If we don't have it in stock, we will do our best to find it for you. We have the largest data base of *used* Radio Shack equipment to draw from. All equipment comes with warranty.

PACIFIC COMPUTER EXCHANGE

The One Source For
Used Tandy Computers
1031 S.E. Mill, Suite B
Portland, Oregon 97214
(503) 236-2949

The Final Solution to the XLR8er Question

by J.F.R. "Frank" Slinkman
1511 Old Compton Road
Richmond, Virginia 23233
804/741-0205
CompuServe ID 72411,650

This is the final article in my series on the special features of the Z180/HD64180 microprocessor (which, for brevity, I shall henceforth refer to simply as the "Z180") on the XLR8er board by HiTech, now marketed by MISOSYS.

The XLR8er board adds an additional 256K to the Model 4. Obviously, it requires special software to manage this extra memory.

A number of programs have been written to accomplish this, the best so far being Michel Houdé's @PEXMEM routine (TMQ DiskNotes 3.2), which is a refinement of Roy Soltoff's @EXMEM. When used in conjuction with Houdé's bank handling routines and his ERAMDISK utility, it provides access to this extra memory, either for use as a RAMdisk or using his @PEXMEM SVC to address memory in application programs.

However, @PEXMEM uses LDIRs! Since it moves data 256 bytes at a time, it is less than half as fast as it could be if it used Direct Memory Access (DMA).

In a previous article ("How to 'Roll Your Own' on the XLR8er," TMQ V.iv.), I discussed the advantages, in both speed and flexibility, of DMA Channel 0 data transfers compared to LDIR and LDDR instructions.

The purpose of the accompanying program, FEXMEM/CMD (Fast EXtended MEMory handler), is to use the superior

```
;     FEXMEM3/ASM  23-Jun-91
;
;     New @exmem-type module using DMA channel 0 burst mode
;  transfers. This code assumes Slinkman DMA channel 0 pro-
;  tocol (i.e., SAR0B and DAR0B initialized to 0 upon boot-
;  up,  and restored to 0 after transfer).
;     Based on program created by Michel Houde',  published
;  in THE MISOSYS QUARTERLY DiskNotes 3.2.
;     Written by J.F.R. "Frank" Slinkman,  1511 Old Compton
;  Road, Richmond,  Va. 23233.  Phone 804/741-0205.  Compu-
;  Serve ID#  72411,650.
;     This program is released to the public domain.
;------------------------------------
*GET EQUATES
LBANK$      EQU   202H
@PERREQU    0DEDH ;generates `parameter error' w/NZ
LOBUF$      EQU   2300H
*LIST OFF
;------------------------------------
      ORG   2600H
;------------------------------------
STARTEQU    $
      LD    HL,HELLO$
      SVC   @DSPLY
      LD    DE,FXM$             ;module already in memory?
      SVC   @GTMOD
      EX    DE,HL        ;module entry to DE
      JR    Z,ISRES             ;found, just plug SVC's
;
NOTRES      LD    DE,'IK'             ;locate low mem ptr
      SVC   @GTDCB
      DEC   HL
      LD    D,(HL)
      DEC   HL
      LD    E,(HL)       ;DE= start of lomem
      PUSH  HL           ;HILO$ pointer on stack
      LD    HL,MODLEN    ;driver length
      ADD   HL,DE        ;end of mod + 1
      LD    B,H
      LD    C,L          ;move to BC
      LD    A,H
      POP   HL           ;HILO$ pointer
      CP    13H
      JR    NC,NOMEM     ;no room in low memory
      LD    (HL),C       ;stuff new HILO$ value
      INC   HL
      LD    (HL),B
;
;Fixup PEXMEM calls to @BANK
;
      SVC   @FLAGS
      LD    H,(IY+26)    ;SVCTAB
      LD    L,@BANK*2    ;@BANK entry
      LD    A,(HL)
      INC   HL
```

```
        LD      H,(HL)
        LD      L,A
        LD      (SVC66H),HL   ;update call
        LD      IY,RELTAB
        LD      BC,MODLEN
        LD      HL,MODBGN     ;Start of module
        PUSH    DE            ;start of relocated mod
        CALL    RELOC
        POP     DE
ISRES   SVC     @FLAGS
        LD      H,(IY+26)     ;SVCTAB
        LD      L,108*2               ;SVC 108
        LD      (HL),E
        INC     HL
        LD      (HL),D
        LD      HL,DONE$
        SVC     @DSPLY
        LD      HL,0
        RET
NOMEM   LD      HL,NOMEM$
        SVC     @LOGOT
        SVC     @ABORT
;------------------------------------------
;  Relocation routine
;  Works for low or high mem.  MODEND should be included in
;  reloc list on entry, IY contains reloc table, end of
;  table flagged as address zero on return, DE=relocated
;  end address+1
;------------------------------------------
RELOC   PUSH    HL            ;unrelocated start address
        PUSH    DE            ;relocated start address
        PUSH    BC            ;module length
        EX      DE,HL
        OR      A
        SBC     HL,DE         ;offset
        LD      C,L
        LD      B,H
$RLOOP          LD      L,(IY)
        LD      H,(IY+1)
        LD      A,H
        OR      L
        JR      Z,$RDONE
        LD      E,(HL)
        INC     HL
        LD      D,(HL)
        EX      DE,HL
        ADD     HL,BC
        EX      DE,HL
        LD      (HL),D
        DEC     HL
        LD      (HL),E
        INC     IY
        INC     IY
        JR      $RLOOP
$RDONE          POP     BC            ;mod len
        POP     DE            ;new (reloc) start add
        POP     HL            ;unreloc start add
        LDIR
        RET
FXM$    DB      '$XM',3
NOMEM$          DB 'No memory space available',CR
HELLO$          DB LF,'FEXMEM 1.0 - Fast page Extended Memory'
        DB      ' Handler for LS-DOS 6.3.1',LF,CR
DONE$   DB      'SVC_108 installed (@FEXMEM)',CR
```

DMA transfers to improve the performance of utilites such as ERAMDISK.

@FEXMEM performs the useful function of @PEXMEM—copying 256-byte pages of RAM from one location in the "normal" 128K RAM to another location anywhere above the base 32K in an XLR8er-equipped Model 4's 384K. @FEXMEM just does it faster. Unfortunately, it's 56 bytes longer than @PEXMEM.

However, to overcome the size problem, I have included a new @BANK SVC, based on Houdé's concept, adapted from my own code for Richard King's 512K hardware mod (TMQ IV.ii & IV.iv) and from the work done by David Goben (TMQ V.ii) to reduce the amount of code that spills over into low memory.

Also included is software which allows the Z180 to run in it's fastest mode (0,1,80,2) by inserting two memory waits during any keyboard access. This is my adaptation and expansion of an idea by Mel Patrick, published in CN-80.

The new @BANK routine saves 101 bytes compared to Houdé''s. The keyboard module adds 32 bytes, and the new @FEXMEM SVC is 56 bytes longer than Houdé's @PEXMEM, meaning this new configuration — the new @BANKer, the @VDCTL trap, the keyboard speedup traps plus @FEXMEM — actually takes 13 fewer bytes of precious low memory than the Houdé' utilities. Without the keyboard traps, the saving is 45 bytes.

To understand how and why @FEXMEM operates, you first need to know that the Z180 on the XLR8er has 19 address lines, enabling it to address 512K of RAM, and the XLR8er's 256K RAM is at physical addresses 40000H through 7FFFFH. (See Table 2.) Because the CPU cannot handle addresses higher than 0FFFFH, the Z180 has a Memory Management Unit (MMU) which enables the CPU to see this area of RAM at logical addresses different from the physical addresses.

The bottom 32K of stock RAM is always at 0000H through 7FFFH. The other 96K of "normal" RAM is split into three 32K banks: Banks 0, 1 and 2. Whenever one of these is resident, it has addresses from 8000H to 0FFFFH. Except on one version of Richard King's 512K hardware mod, the logical and physical addresses of this area of RAM are always the same. You also need to know how memory-to-memory DMA transfers are done.

DMA Channel 0 has several internal registers assigned to it (See Table 1). These are the Source Address, Destination Address, and Byte Count Registers. There are three other registers which are shared by the two channels, including the DMA Mode register and DMA Status register. The other common DMA register, DCNTL, is handled by the code in XLBOOTC/FIX.

DMA transfers can be done in either of two modes, "burst" or "cycle steal." In burst mode, DMA seizes control of the bus until the transfer is completed. In cycle steal mode, one normal CPU cycle is performed, then one byte transferred by the DMA channel, alternating until the transfer is complete, at which time normal CPU operation is resumed.

@FEXMEM must use burst mode transfers because it must sometimes perform "double buffering." Double buffering is necessary when both the source and destination of the data to be moved cannot be fully addressed by the DMA address registers at the same time.

To understand the need for double buffering, we need to look at the way the hardware manages banks 0, 1 and 2 (see Table 2). Because the system must always have access to the base 32K, there is no practical way the CPU can access more than one of those banks at a time. Thus data cannot be moved directly from one of these banks to another.

For example, if data to be copied from 9000H in Bank 2 to 0E500H in Bank 0, Bank 2 must be switched in, the data copied from 9000H to some area in the base 32K, Bank 0 switched in, and the data copied from the base 32K to 0E500H.

```
RELTAB          DW      RY01,RY02,RY03,RY04,RY05,RY06,RY07,RY08,0
;---------------------------------------------
;|  @FEXMEM                               SVC 108 |
;|  Entry:                                        |
;|  B =  function [3=getpage, 4=putpage]              |
;|  C =  bank number of transfer area                 |
;|  HL-> transfer-area address      [ XBUF <= 0FF00H   |
;|  DE-> 256-byte user buffer      [ UBUF in base 128K]|
;|  Exit:                                         |
;|  Z <- no error                                 |
;|  NZ <- error number in A                           |
;|  BC, DE and HL altered                             |
;|  Program assumes and honors DMA channel 0 protocol  |
;---------------------------------------------
MODBGN          JR      EXMEM@
RY01 DW         MODLST
     DB         3,'$XM'                 ;not "FXM" because ARC4V2
                                ;looks for "$XM"
EXMEM@          LD      A,C             ;p/u XBUF bank #
     CP         10+1            ;is it legal (0-10)?
     JR    .    NC,PERR                 ;error if not
     DEC        B               ;B is function code
     DEC        B
     DEC        B
     JR         Z,GETP          ;go w/B=0 if 'get page'
     DEC        B               ;is it 'put page'?
     JR         NZ,PERR                 ;ret w/err if bad function
     DEC        B               ;B=-1 if 'put page'
GETP DI
     LD         (SAVESP),SP     ;must use local stack -
RY02 EQU        $-2             ;  can't risk stack being
     LD         SP,STACK$       ;  switched out by @bank.
RY03 EQU        $-2             ;  Also, easy way to EI at
                                ;  end if no bank switching
     SET        7,H             ;force XBUF >= 8000H
;---------------------------------------------
;  Because DMA only recognizes physical addresses, straight
;DMA transfer w/o bank switching can be done only if both
;UBUF and XBUF are both currently physically addressable.
;   This code assumes UBUF is always in resident "normal"
;RAM; that no part of it is ever in XLR8er RAM; and there-
;fore UBUF is always physically addressable upon entry.
;   XBUF always exists above 7FFFH; so it must either be
;wholly in "normal" RAM or wholly in XLR8er RAM, and can
;never straddle the boundary between the two types of RAM.
;   If XBUF is in XLR8er RAM it's physical address can be
;calculated.  Thus, in this case, DMA transfer w/o bank
;switching is always possible.
;   If XBUF is in "normal" RAM, it must be assumed to be in
;a non-resident bank and therefore not currently physically
;addressable.  In this case bank switching is required, and
;double-buffering may be neccessary.
;
     CP         2+1             ;XBUF in XLR8er RAM?
     JR         NC,DO_DMA       ;if so, can do one-shot DMA
                                ;   xfer w/o bank switching
                                ;   (rets to RETSP to EI)
;---------------------------------------------
;Here if XBUF in base 128K.  Modified Houde' code:
;
     PUSH  HL
     LD    HL,7F00H
     XOR   A
     SBC   HL,DE                ;set C if UBUF > 7F00H
```

```
        POP     HL
        INC     B               ;NZ=get, Z=put, C preserved
        LD      B,A             ;B = 0 for @BANK SVC
        JR      C,DBLBUF        ;go if UBUF > 7F00H
        JR      NZ,XFER                 ;get (HL)->(DE)
        EX      DE,HL
        JR      XFER            ;put (DE)->(HL)
DBLBUF          PUSH    AF              ;here if UBUF > 7F00H
        IN0     A,CBR           ;any of it in XLR8er RAM?
        JR      Z,DBUF01        ;OK if not
        POP     AF              ;if so, clear stack and...
PERR    JP      @PERR           ;    ret w/"parameter error"
DBUF01          POP     AF              ;restore flags
        PUSH    DE              ;UBUF
        LD      DE,LOBUF$       ;always first destination
        JR      Z,DBLPUT        ;go if 'putpage'
DBLGET          PUSH    DE              ;save LOBUF$
        CALL    XFER            ;get (XBUF)->(LOBUF$)
RY04    EQU     $-2
        POP     HL              ;LOBUF$
        POP     DE              ;UBUF
        JR      DMAXFER                 ;get (LOBUF$)->(UBUF)
                                ;(returns to RETSP)
DBLPUT          EX      (SP),HL         ;XBUF on stack, UBUF in HL
        PUSH    DE              ;LOBUF$
        CALL    DMAXFER                 ;put (UBUF)->(LOBUF$)
RY05    EQU     $-2
        POP     HL              ;LOBUF$
        POP     DE              ;XBUF
                        ;code below puts (LOBUF$)->(XBUF)
XFER    CALL    BANK@           ;get requested bank
RY06    EQU     $-2
        JR      NZ,RETSP        ;exit if error
        CALL    DMAXFER
RY07    EQU     $-2
BANK@
;       LD      A,C             ;THIS LINE IS A COMMENT!
;
        ;Accompanying @BANK handler starts with LD A,C.  If
        ;using other @BANKer, you may need to insert LD A,C
        ;instruction here and add two more levels (4 bytes)
        ;to local stack.  See COMMENTed lines.
;
        JP      $-$             ;restore original bank
SVC66H          EQU     $-2             ;address of @bank SVC code
;———————————————————————————
;Stack area (6 levels) @BANK uses 2 max, @FEXMEM uses 4 max
        DC      12,0            ;local stack
;       DC      4,0             ;THIS LINE IS A COMMENT!
;
STACK$          DW      $+2             ;make RETSP the RET address
RY08    EQU     $-2
RETSP   LD      SP,$-$          ;recover original SP
SAVESP          EQU     $-2
        EI
        RET                     ;return to primary call
;———————————————————————————
;Here if XBUF in XLR8er RAM.  Low 15 bits of 19-bit address
;exist in HL.  Must calculate top 4 bits, store top 3 bits
;(A18-A16) in bits 2-0 of C; and put A15 in bit 7,H.
;
;On entry:  A = XLR8er bank number (range is 3 to 10)
;
DO_DMA          ADD     A,5             ;range now 8-15
```

The CPU and DMA "see" RAM addresses differently. The CPU uses the logical addresses determined by the MMU. DMA only works with physical addresses. For example, suppose Bank 6 is resident, and the 256-byte page of memory we want to transfer exists at logical addresses 7F80H through 807FH. Even though the CPU sees Bank 6 at logical addresses 8000H to 0FFFFH, it actually consists of physical addresses 58000H through 5FFFFH. The data in this example is not physically contiguous! The 129th byte (the one at/logical 8000H) — is actually at physical 58000H, 50001H bytes distant from the 128th byte!

If you tried to use DMA to transfer this data, the last 128 bytes would be copied from physical 8000H-807FH, not logical 8000H-807FH. In other words, it would be copied from one of Banks 0, 1 or 2 (whichever was resident when Bank 6 was switched in [see NEWBANK1/ASM to understand why]), and not from Bank 6, as intended.

Double buffering means the data must be moved twice in quick succession — from the source area to a buffer in the base 32K, and then from the buffer to the destination area. It would fail in DMA cycle steal mode because the bank containing the source data would be switched out before the DMA transfer from it was completed. Even if the source and destination banks were the same, double buffering would fail in cycle steal mode because the second transfer would be attempted before the first could be completed.

Armed with this knowledge, we are now ready to "walk through" the FEXMEM/ASM listing.

The code up to MODBGN is pretty standard installation code. In fact, I copied it directly from EXMEM. The only thing slightly unusual about it is that it obtains the address of the @BANK routine and plugs it into the relocatable module so it can invoke it via a CALL — faster than a SVC.

First, we need to look at DMAXFER, near the end of the listing.

DMAXFER assumes and honors the DMA Channel 0 Protocol I proposed in the article, "How to 'Roll Your Own' on the XLR8er." This protocol reduces the amount of code required to implement Channel 0 transfers, and helps ensure transfers go where they are supposed to go, instead of to or from unexpected areas, which would result in program failure at a minimum, and system crash at a (likely) maximum.

It is simply a requirement that the SAR0B and DAR0B registers, which hold the top three bits of the 19-bit DMA addresses, be initialized with zero upon boot up, and be restored to zero after any DMA Channel 0 operation. This causes DMA Channel 0 to default to transfers within the base 64K.

In other words, programmers should use extreme caution before changing the values in these two internal registers when @FEXMEM is installed.

The first two instructions at DMAXFER load the lower 16 bits of the 19-bit DMA Channel 0 source address register with the contents of HL. The next two instructions load the lower 16 bits of the 19-bit DMA Channel 0 destination address registers with the contents of DE.

The next two instructions load the value 01H into the top 8 bits of the 16-bit DMA Channel 0 byte counter register. The lower 8 bits of the counter, in BCR0L, are automatically initialized to zero upon RESET, and will be zero upon conclusion of any transfer. Thus the byte counter register will count 100H, or 256 bytes to transfer.

The next two instructions load the value 02H into the DMODE register. This causes both the source and destination registers to be incremented after each byte is transferred, and specifies burst mode.

The next two instructions load the value 40H into the DSTAT register, initiating the transfer by setting the Channel 0 DMA Transfer Enable bit.

```
        RRA                     ;range now 4-7 & CY
        LD      C,A             ;C = A18-A16 of XBUF addr
        RL      H               ;these two shifts replace
        RRC     H               ;    bit 7,H (A15) with CY
        INC     B               ;set Z if put, NZ if get
        LD      B,0             ;init A18-A16 of UBUF addr
;
;BDE now holds the 19-bit address of UBUF, and CHL holds
;the 19-bit address of XBUF
;
        JR      NZ,DMA020       ;go if get [(CHL)->(BDE)]
        LD      A,B             ;if put, swap BDE<->CHL to
        LD      B,C             ;    put (BDE)->(CHL)
        LD      C,A
        EX      DE,HL
;
DMA020          OUT0    SAR0B,C         ;load A18-A16 of source and
        OUT0    DAR0B,B                 ;dest addresses to DMA regs
;------------------------------------
DMAXFER         OUT0    SAR0L,L         ;load A15-A0 of source
        OUT0    SAR0H,H                 ;address to DMA source regs
        OUT0    DAR0L,E                 ;load A15-A0 of destination
        OUT0    DAR0H,D                 ;address to DMA dest regs
        LD      A,1             ;MSB value (LSB inits to 0)
        OUT0    BCR0H,A                 ;load counter regs w/100H
        INC     A               ;02H sets inc source & dest
        OUT0    DMODE,A                 ;    and burst mode
        LD      A,40H           ;40H inits DMA transfer
        OUT0    DSTAT,A
        XOR     A               ;also resets Z
        OUT0    SAR0B,A                 ;protocol:  DMA channel 0
        OUT0    DAR0B,A                 ;  must default to base 64K
        RET                     ;BC is preserved for calls
                                ;   by Houde' code
;------------------------------------
MODLST          EQU     $-1
MODLEN          EQU     $-MODBGN
;
        END     START
```

After the transfer, the XOR A instruction sets the Z flag in case this is the last code executed before return to the primary call, and SAR0B and DAR0B are loaded with 00H to conform with the protocol. This is necessary even if the transfer was within the base 64K, because if the last byte transferred was to/from 0FFFFH, the corresponding DMA address register would end up with a value of 10000H (SAR0B or DAR0B equal to 1).

Now let's go to the start of the module, at label EXMEM@.

Before beginning, keep in mind the purpose of @FEXMEM: to move data to/from the 256-byte "user buffer" (UBUF) in the currently resident 64K from/to a "transfer area" (XBUF) not currently resident. XBUF, then, will always have a logical address in the range 8000H-0FF00H.

The reason for making these transfers to/from non-resident RAM banks from/to the resident area is primarily to support RAMdisk drivers like Houdé's ERAMDISK. In the normal course of things, such transfers will always be with UBUF in the base 64K of RAM (i.e., the base 32K plus Bank 0).

Some applications probably exist where the transfer must be made to a resident 64K where the upper 32K is either Bank 1 or Bank2. As far as physical addresses go, this is no different than if it was Bank 0.

It is hard, however, to imagine an application in which UBUF would be in XLR8er

RAM. Therefore, @FEXMEM assumes this will never be the case. However, it does check for that situation, just in case.

At EXMEM@, the XBUF bank number is examined. All banks from 0 through 10 are assumed to exist. Any bank number higher than 10 will cause the function to abort with a parameter error.

The reasoning behind this is: (a) if the XBUF bank is 1 or 2 and does not exist, this error will be caught in the XFER section of the modified Houdé' code; and (b) if you only have the stock 64K or 128K RAM, and no extra 256K on the XLR8er, you don't need @FEXMEM anyway.

Now the function code is examined. The old @EXMEM codes "1" and "2", which had to do with moving single bytes, are not supported, mainly because there is no real need for them. Any code other than 3 or 4 will return a parameter error to the caller. The function code in B is replaced by a flag which will be examined later.

The interrupts are now disabled, and SP is saved, and then pointed to a local stack. This is necessary to keep the stack from being switched out in the course of bank switching. A local stack is not necessary in the case of a pure, no-bank-switch DMA transfer, but the interrupts will still have to be enabled afterwards. Treating all entries and exits the same saves code.

Now XBUF is forced to 8000H or higher by setting the high order bit in the HL register pair.

The transfer-area bank number, which is still in A, is now checked again. If XBUF is in XLR8er RAM, then no bank switching is required because both UBUF and XBUF are currently and fully physically addressable. In this case, control passes to DO_DMA.

If XBUF exists in normal RAM, then bank switching, and maybe even double buffering, will be necessary; so the Houdé' code, modified, is used to accomplish this.

Table 1: Internal Z180 DMA registers used by @FEXMEM

Name	No.	Description/function
SAR0L	20H	bits 0-7 of DMA channel 0 source address
SAR0H	21H	bits 8-15 of above
SAR0B	22H	bits 16-18 " "
DAR0L	23H	bits 0-7 of DMA channel 0 destination address
DAR0H	24H	bits 8-15 of above
DAR0B	25H	bits 16-18 " "
BCR0L	26H	bits 0-7 of DMA channel 0 byte counter
BCR0H	27H	bits 8-15 of above
DSTAT	30H	DMA Status register
DMODE	31H	DMA operation mode register

Table 2: Physical Memory Map of XLR8er-equipped Model 4

00000H	Base 32K	07FFFH	
08000H	Bank 0*	0FFFFH	"Normal"
08000H	Bank 1*	0FFFFH	128K
08000H	Bank 2*	0FFFFH	
10000H	Addressed as Banks 1 & 2	1FFFFH	
20000H	Does not exist		
3FFFFH	Does not exist		
40000H	Bank 3	47FFFH	
48000H	Bank 4	4FFFFH	
50000H	Bank 5	57FFFH	256K
58000H	Bank 6	5FFFFH	XLR8er
60000H	Bank 7	67FFFH	RAM
68000H	Bank 8	6FFFFH	
70000H	Bank 9	77FFFH	
78000H	Bank 10	7FFFFH	

* Due to the hardware management scheme RAM Banks 0, 1 & 2 exist only at physical addresses 8000H through 0FFFFH. Note the contents of bit 15 in XLR8er RAM bank addresses. In all odd-numbered bank addresses, bit 15=0. In all even-numbered bank addresses, bit 15=1.

Table 3: Function of Mem1 and Mem0 bits of OPREG$

Mem1	Mem0	Docs say	I say
0	0	Bank 0	Base 32K, forced to Bank 0
0	1	not used	Bank 0
1	0	Bank 1	Bank 1
1	1	Bank 2	Bank 2

The location of UBUF is examined to determine whether or not it straddles the boundary between the base 32K and the upper 32K of the resident 64K. If it does, the C flag will be set.

The get/put flag in B is incremented, which will set the Z flag if this is a PUT operation, and reset it if this is a GET. Then B is loaded with the 00H in A. This is the "select bank" function code for the @BANK SVC.

If UBUF is above 7F00H, as evidenced by a set C flag, double buffering is required; so control passes to DBLBUF. If not, then the Z flag determines whether control should immediately pass to XFER for a direct GET from (HL) to (DE), or if DE and HL should first be exchanged so XFER will do a direct PUT of the data from (DE) to (HL).

At DBLBUF, because at least part of UBUF is known to be in the top half of the resident 64K, a check is made to make sure it's not in XLR8er RAM. The state of the Z flag is saved by PUSHing AF, and the internal CBR register is read into A. Since the IN0 instruction manages the Z flag, Z will be set if Banks 0, 1 or 2 are resident (CBR = 0), and reset if an XLR8er bank is resident (CBR <> 0).

If NZ, then all or part of UBUF exists in XLR8er RAM, so the stack is cleared, and return made to via @PERR and RETSP, which causes return with a parameter error, with SP restored and interrupts re-enabled.

Otherwise, UBUF is in normal RAM, so control passes to DBLB01, where the POP AF restores the Z flag and clears the stack.

Because we're double-buffering, we know the first data transfer will be to the low memory buffer (LOBUF) regardless of whether UBUF or XBUF is the source; so we push UBUF, and load DE with LOBUF. Now the Z flag is tested to determine whether this is a get or put operation, with control going to DBLGET or DBLPUT as required.

At DBLGET, we push LOBUF (2300H), and call XFER to copy the contents of XBUF to it. Then we POP LOBUF into HL and UBUF into DE, clearing the stack, and JR to DMAXFER to transfer the data from LOBUF to UBUF. Return from DMAXFER is to RETSP, which is pointed to by the word at STACK$, which will be picked up as the RET address for DMAXFER.

The DBLGET sequence, then, is: (1) switch in the XBUF bank; (2) copy XBUF to LOBUF; (3) restore the UBUF bank; and (4) copy LOBUF to UBUF.

At DBLPUT, the EX (SP),HL instruction puts XBUF on the stack and loads HL with UBUF. Then we PUSH LOBUF in DE, and call DMAXFER to copy the data from UBUF to LOBUF. Then we POP LOBUF into HL and XBUF into DE, clearing the stack, and fall through to XFER to copy the data from LOBUF to XBUF.

The DBLPUT sequence, then, is: (1) copy UBUF to LOBUF; (2) switch in the XBUF bank; (3) copy LOBUF to XBUF; and (4) restore the UBUF bank.

At XFER, we first call @BANK to switch

```
        NOLOAD                          ;core image directive
                                        ;must be first in ALDS
;____________________
;       ELEMENTS...
;       A PRO-WAM APPLICATION !!
;       (c)1991  PUBLIC DOMAIN
;       Danny C. Mullen
;       6641-B Tracey Place
;       Ft Polk, LA 71459
;____________________
        PRINT  SHORT                    ;prevents excessive length
                                        ;  in listing large data areas
@CLOSE          EQU     60                      ;define equates
@OPENEQU        59
@POSNEQU        66
@READEQU        67
@SOUND          EQU     104
@WINDOW         EQU     124

;SVC    MACRO  #1                       ;remove semicolons if
;       LD     A,#1                     ;not using ALDS
;       RST    28H
;       ENDM

        PSECT  2700H                    ;may need ORG here

        DEFM   'PROWAM'                 ;standard header
        DEFM   'Elements    '               ;up to 12 char name
        DB     3                        ;  padded if needed
        DB     13%0                     ;Use DC 13,0 for EDAS/MRAS
        DW     IROW,ICOL
        DB     .HIGH.$.SHL.8-$+256%0

        IFF    $.EQ.2800H    ;check header size
        ADISP  'Something wrong in header - PC = ^$'
        QUIT
        ENDIF

STARTLD        HL,3.SHL.8+3  ;orig = row 3, col 3
IROW   EQU     $-1
ICOL   EQU     $-2
       LD      DE,20.SHL.8+72        ;20 rows by 72 cols
       LD      B,7                   ;open window
       SVC     @WINDOW
       JR      Z,OK                  ;  & go if ok
       LD      B,0.SHL.4+2   ;else error - can't open
       SVC     @SOUND
       RET                           ;  back to caller

HOME   DW      0600H                 ;cursor home position
DFLAGDB        0                     ;direction flag

OK     LD      B,5                   ;ram to window function
       LD      HL,SCREEN             ;point to ram
       SVC     @WINDOW
       LD      HL,(HOME)             ;move to home position
       CALL    SETPOS
       LD      IY,COL                ;initialize iy
       LD      IX,DFLAG              ;initialize ix
       CALL    OPNFIL                ;open data file
       CALL    ACTION                ;check for the action keys
       CALL    CLSFIL                ;close txt file
QUIT   LD      BC,8.SHL.8+0  ;close window
       SVC     @WINDOW
       RET                           ;return to caller
```

```
;--------------
;    Subroutines
;--------------
ACTION     CALL   TESTPOS      ;check if valid curs pos
     CALL  POSREAD             ;position & read a record
     LD    BC,0         ;wait for key press
     SVC   @WINDOW
     RET   C            ;break pressed ?
     RET   NZ           ;nz = error
     CP    1AH          ;down arrow ?
     JR    Z,DOWN
     CP    1BH          ;up arrow ?
     JR    Z,UP
     CP    18H          ;left arrow ?
     JR    Z,LEFT
     CP    19H          ;right arrow ?
     JR    NZ,ACTION    ;must get one of those keys

RIGHTCALL  GETPOS       ;find present pos
     LD    A,68
     CP    L            ;at end ?
     JR    Z,ACTION     ;  yes, don't move
     CALL  SETR         ;set DFLAG
     INC   L            ;  else inc
     INC   L
     INC   L
     INC   L
     INC   IY
     CALL  SETPOS       ;move to new pos
     JR    ACTION

LEFT CALL  GETPOS
     LD    A,0
     CP    L            ;at beginning ?
     JR    Z,ACTION     ;  yes - don't move
     CALL  SETL         ;set DFLAG
     DEC   L            ;  else dec
     DEC   L
     DEC   L
     DEC   L
     DEC   IY
     CALL  SETPOS       ;move to new pos
     JR    ACTION

DOWN CALL  GETPOS
     LD    A,14
     CP    H            ;lower limit ?
     JR    Z,ACTION     ;  yes - go
     INC   H            ;  no
     LD    BC,18
     ADD   IY,BC        ;  add col offset
     CALL  SETPOS       ;move to new pos
     JR    ACTION

UP   CALL  GETPOS
     LD    A,6
     CP    H            ;upper limit ?
     JR    Z,ACTION     ;  yes - go
     DEC   H            ;  no
     CALL  SETPOS       ;move to new pos
     PUSH  IY
     POP   HL
```

in the XBUF bank. If there is an error (such as an attempt to transfer to/from a non-existent Bank 1 or 2), the original stack is recovered, and return made to the calling routine with the @BANK error, which will almost certainly be "device not available."

If there is no @BANK error, DMAXFER is called to do the data transfer.

@BANK is invoked again via a JP to restore the originally resident bank. If this is the last invocation of XFER, return will be to RETSP, due to the local stack arrangement and seeding, as discussed for the JR to DMAXFER above. Otherwise, a normal return is made.

Now we get to the "fun" part: DO_DMA, which handles direct, "one-shot," DMA transfers to/from UBUF and an XBUF in XLR8er RAM.

The first thing we must do is calculate the 19-bit physical address of XBUF. The bottom 15 bits already exist in bits 14-0 of HL; so all we have to do is calculate the top 4 bits.

Upon entry, the XBUF bank number is still in the A register. A will hold a value in the range 3-10, the only legal XLR8er bank numbers. Adding five changes the range to 8-15. These are the top 4 bits of the address.

Shifting this value one bit right changes the range of values in A to 4-7, and shifts the value formerly in bit 0,A to the carry flag. The value in A — the top 3 bits of the 19-bit address — is now stored in C.

The next instruction, RL H, shifts all the bits in the H register one bit left, and puts what was in the C flag into Bit 0, H. The following RRC H instruction shifts bits 7-1 to positions 6-0, and puts what was in bit 0 into both bit 7 and the C flag. The effect of these two instructions is to throw away the original bit 7,H, and replace it with what was in the C flag when the RL H instruction was executed.

ELEMENTS: a PROWAM application

by Danny C. Mullen
6641-B Tracey Place
Fort Polk, LA 71459-3262
Danny C. Mullen
11 May 1991

Way back in the fall 1989 *MISOSYS QUARTERLY* (Vol IV.i page 64), Roy Soltoff suggested in a reply to Gary Phillips that many PROWAM applications could be written that weren't included in the basic package. One of his suggestions was a pop-up periodic chart of the elements. Since it had been some time since I'd fired up the old Model 4, and no one had done it yet, I figured I'd give it a try!

Well, I think I've succeeded in doing just that with ELEMENTS. Before going any further, however, let me just say I'm not a chemist or scientist. I say this because my chart was based on one found in the Encyclopedia Brittanica which lists 105 elements. While scanning my 6th grader's science book, it lists 109 elements and in a slightly different order than presented here. Since the data I wanted to display was in the encyclopedia, I stuck with that. I'll leave it at that.

Just as suggested by Roy in the aforementioned article, ELEMENTS is simple to operate. After PROWAM is installed, select ELEMENTS from one of the default choices (if you made it a default) or via the F3 Universal selection. Now, a file called ELEMENTS/DAT must be available on one of your drives (a ram disk is better/quicker/quieter) since this is where the data such as name, atomic #, etc. is read from. There was not enough space in the ELEMENTS/APP to include the data there; that would have made it a little faster. This

```
          LD      BC,18
          OR      A
          SBC     HL,BC          ;subtract col offset
          PUSH    HL
          POP     IY
          JR      ACTION

SETPOS            LD     B,3            ;set curs function
          JR      GETSET         ;jump around
GETPOS            LD     B,4            ;get curs function
GETSET            SVC    @WINDOW                ;execute it
          RET

SETR      LD      (IX),1         ;set right flag
          RET

SETL      LD      (IX),0         ;set left flag
          RET

;___________________
;         This routine checks cursor positions...
;            if in invalid zone, moves it.
;___________________

TESTPOS           CALL   GETPOS         ;get row in h
          LD      A,9            ;test if row 9
          CP      H
          RET     Z
          INC     A              ;test if row 10
          CP      H
          RET     Z
          INC     A              ;test if row 11
          CP      H
          RET     Z              ;rows 9-11: no action
          INC     A              ;test if row 12
          CP      H
          JR      Z,IS12
          INC     A              ;test if row 13
          CP      H
          JR      Z,IS1314
          INC     A              ;test if row 14
          CP      H
          JR      Z,IS1314       ;rows 13 & 14: same action
          LD      A,(IX)         ;must be 6,7, or 8
          OR      A              ;test direction flag
          JR      Z,IS12         ;if z, curs left
          JR      IS1314         ;  else curs right
          RET

IS12      LD      A,(IY)         ;row 12 action
          CP      1
          RET     NZ
          DEC     IY             ;dec ptr &
          DEC     L              ;  col
          DEC     L
```

file is on the DISK NOTES for this issue.

After selecting ELEMENTS, and its window opens, you move around the screen by using the shifted cursor keys. The cursor will stop on the left side of the element's symbol and show some information about it in the lower part of the screen. Blank areas on the screen will be skipped over and, of course, you're limited to the rows and columns you can move to. Hitting the BREAK key exits the application.

TECHNICAL NOTES

I used Radio Shack's ALDS package to write ELEMENTS. Since other assembler packages aren't 100% compatible, you may have to activate the SVC macro by deleting the semicolons in the first column. Also the directives NOLOAD, PSECT, PRINT SHORT, PSECT, ADISP, IFT, IFF, QUIT may also need changing to suit your assembler's syntax. 80 Micro's September '84 issue, page 66 is a fair reference for possible alternatives. Some of them are:

```
ORG     vs      PSECT
CI      vs      NOLOAD
ERR     vs      QUIT
```

The source file starts with a standard PROWAM header. I've duplicated some of the labels others have used (IROW, ICOL) and have ensured that the /APP name is 12 characters long terminated by an ETX (DB 3). This is REQUIRED if it is to be one of the DEFAULT applications; this I remembered from my past experience with NXWAM and Roy's eagle eyes/quick solution (TMQ Vol V.i, page 15).

Most of the code is documented as to functions and probably needs little explanation. The hardest part, for me, was to get that !@#% cursor to skip over any blank areas on the screen and only land on valid selections. This is where the table labeled COL came into being. Granted, there may be another way to do it, but by identifying invalid positions as 01 bytes, and the rest

```
        DEC     L
        DEC     L
        CALL    SETPOS
        JR      IS12

IS1314          LD      A,(IY)          ;row 13/14 action
        CP      1
        RET     NZ
        INC     IY              ;inc ptr &
        INC     L               ;  col
        INC     L
        INC     L
        INC     L
        CALL    SETPOS
        JR      IS1314
        RET

;――――――――――――――――――
;       File open/read/close routines
;――――――――――――――――――

OPNFIL          LD      B,73            ;open the data file
        LD      DE,FCB          ;DE ==> FCB
        LD      HL,IOBUF        ;HL ==> disk i/o buffer
        SVC     @OPEN
        RET     Z               ;go if ok
        CALL    ERROR           ;else problem
        POP     HL              ;clear stack
        JP      QUIT

CLSFIL          LD      DE,FCB          ;close the data file
        SVC     @CLOSE
        CALL    NZ,ERROR        ;something wrong
        RET

POSREAD         LD      C,(IY)          ;position to record
        LD      B,0
        LD      DE,FCB
        SVC     @POSN
        JR      Z,READ          ;go if ok
        CALL    ERROR           ;something wrong
        POP     HL              ;clear stack
        POP     HL
        JP      QUIT            ;exit
READ    LD      HL,UREC                 ;read record
        LD      DE,FCB
        SVC     @READ
        JR      Z,RD1           ;go if ok
        CALL    ERROR           ;something wrong
        POP     HL              ;clear stack
        POP     HL
        JP      QUIT            ;exit
RD1     CALL    GETPOS          ;save curs pos
        PUSH    HL
        LD      HL,17.SHL.8+0;point to data area
        CALL    SETPOS
        LD      HL,UREC                 ;data is here
        LD      B,10            ;xfer data to window
        SVC     @WINDOW
        POP     HL              ;restore old curs pos
        CALL    SETPOS
        RET

;――――――――――――――――――
```

of the positions as record numbers in the text file, I got it to work.

One small point is the use of DFLAG. When designing the program, I had to get the cursor to move all the way in one direction when it was in the top three rows. This was the only way I could come up with to do it. When you hit one of the left/right arrows, it sets the flag for any subsequently needed direction while in these rows and in the invalid areas - the cursor then jumps across any blank areas on screen.

Some astute eyes may wonder why, then, does the first position start with a byte 02. Well, I had to evade the 01 bytes (since they indicated invalid selections). The first 2 records of ELEMENTS/DAT are, therefore, skipped over. That is why you'll see 107 records listed if you do a DIR of the disk with that file present. By the way, that text file could easily be changed to show different information that I presented by editing it with any straight ASCII text editor. You will probably need to change the logical record length (LRL) by issuing the following command at DOS Ready: RESET ELEMENTS/DAT:d (LRL=256). Then, after any changes were made, repeat the same command with (LRL=73); since that's the LRL I used for it. Also, this RESET command only works under LS-DOS 6.3.1.

In order to utilize the limited memory a PROWAM application can possess, the area labeled SCREEN is recycled as a disk I/O buffer and user record buffer after the information there is loaded into the PROWAM window. These buffers are required under the @READ and @OPEN SVC's of TRSDOS/LS-DOS. I even had to use some of it for part of the file control block. I know there are probably some ways to optimize the code and gain a few more bytes of free space, but it just does fit into the PROWAM operating area as shown.

I believe this covers any explanations about the source code - most, again, is documented in the comments. Any fur-

```
;       Error handler routine
;——————————————————————

ERRORCALL   GETPOS          ;get/save curs pos
      PUSH  HL
      LD    HL,17.SHL.8+0;curs to error zone
      CALL  SETPOS
      LD    B,10
      LD    HL,ERRMSG       ;show error msg
      SVC   @WINDOW
      LD    B,0.SHL.4+2  ;beep
      SVC   @SOUND
      LD    BC,0            ;wait for a key press
      SVC   @WINDOW
      POP   HL              ;retrieve curs pos
      CALL  SETPOS          ;  and put back
      RET

ERRMSG      DEFM   ` Problem with ELEMENTS/DAT file -  `
      DB    0DH

;——————————————————————
;       Table of record numbers in ELEMENTS/DAT file
;       and invalid column positions (01 bytes)
;——————————————————————

COL DB 02,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,02,03
 DB 04,05,01,01,01,01,01,01,01,01,01,01,06,07,08,09,10,11
 DB 12,13,01,01,01,01,01,01,01,01,01,01,14,15,16,17,18,19
 DB 20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37
 DB 38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55
 DB 56,57,58,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87
 DB 88,89,90,105,106,01,01,01,01,01,01,01,01,01,01,01,01,01
 DB 01,01,01,01,59,60,61,62,63,64,65,66,67,68,69,70,71,72
 DB 01,01,01,01,91,92,93,94,95,96,97,98,99,100,101,102,103,104

;——————————————————————
;       File control block
;——————————————————————

FCB   DEFM    `ELEMENTS/DAT'        ;32 byte buffer
      DB      3                     ;shared with SCREEN area

;——————————————————————
;       Screen (after loading this to window,
;               part of it becomes disk i/o buffer)
;——————————————————————

IOBUFEQU    $+19
UREC EQU    $+256+19

; In the table, change "DB xx%" to "DC xx," for EDAS/MRAS
SCREEN      DEFM   `       WAM THE ELEMENTS  (c)1991 Danny C.
Mullen'
      DEFM    `   PUBLIC DOMAIN      `
      DB      72%' `
      DB      72%'_'
      DB      22%'‾`
      DEFM    `P E R I O D I C   T A B L E'
      DB      22%' `
      DB      72%'_'
      DB      72%'‾`
      DEFM    ` H `
      DB      62%' `
      DEFM    `H     He `
```

```
        DEFM    ' Li  Be'
        DB      42%' '
        DEFM    'B      C      N      O      F      Ne '
        DEFM    ' Na  Mg'
        DB      42%' '
        DEFM    'Al  Si  P   S      Cl  Ar '
        DEFM    ' K    Ca  Sc  Ti  VCr  Mn  Fe  Co  Ni  Cu  Zn
'
        DEFM    'Ga  Ge  As  Se  Br  Kr '
        DEFM    ' Rb  Sr  Y  Zr  Nb  Mo  Tc  Ru  Rh  Pd  Ag  Cd
'
        DEFM    'In  Sn  Sb  Te  I  Xe '
        DEFM    ' Cs  Ba  La *Hf  Ta  W   Re  Os  Ir  Pt  Au
Hg   '
        DEFM    'Tl  Pb  Bi  Po  At  Rn '
        DEFM    ' Fr  Ra  Ac**Rf  Ha'
        DB      68%' '
        DEFM    '* Ce  Pr  Nd  Pm  Sm  Eu  Gd  Tb  Dy  Ho  Er
Tm'
        DEFM    '  Yb  Lu '
        DB      15%' '
        DEFM    '**Th  Pa  U Np  Pu  Am  Cm  Bk  Cf  Es  Fm
Md'
        DEFM    '  No  Lr '
        DB      72%'_'
        DEFM    ' NAME ATOMIC #      ATOMIC WT CHARACTERISTIC '
        DEFM    'MELTING PT  BOILING PT '
        DB      72%' '
        DB      72%'_'
        DB      8%' '
        DEFM    '(use shifted cursor keys to move around -
BREAK to QUIT)'
        DB      8%' '
;------------------
;       Check for memory overflow
;------------------
        IFT     $.GT.2FFFH          ;file must fit below 3000H
        ADISP   'Memory overflow!!!!'
        ADISP   'Code too long for PROWAM...'
        ENDIF
;------------------
;       Zero last sector for patch space
;          (not mandatory)
;------------------
        IFT     $.LT.3000H
        DB      .HIGH.$.SHL.8-$+256%0
        ENDIF

        END     START
```

ther questions can be directed to me at the address above. I hope someone out there can find some benefit in using ELEMENTS or learn more about (hence write some applications for) PROWAM. If you don't have the PROWAM PROGRAMMERS TOOLKIT, get it - it's not expensive. Besides, you'll need it to write applications and it isn't hard to understand! I hereby donate ELEMENTS to the PUBLIC DOMAIN. Please include all documentation [files which appear on DISK NOTES] in any copies [of files] made.

Advertisement

Selling excess MODEL 4's, III's, and I's + BALL 14" Model I monitors with room for expansion board, etc. Have software for all. Also have a quantity of PC, XT, and AT parts and feature cards. Send wish list, offer, name, address, and phone number to Richard R, King, 808 Orleans Place, Raleigh, NC 27609. Or telephone 919-872-5846. No reasonable offer refused.

PRO-WAM Programmers' Toolkit

The **Programmers' Toolkit** provides what you need to know to write programs for the PRO-WAM environment. This includes documentation in a convenient 5.5" by 8.5" format; the same as PRO-WAM release 2. The included WINLINK device driver supports programmed invocation of applications from any programming language environment which supports device I/O. You also get the WINDOW/CCC function library for MC, and an assembler source code file for the PHRASE application to illustrate how to write a PRO-WAM application.

Programmers' Toolkit $29.95 + $3S&H

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS

```
** ELEMENTS/DAT ** This file to be used with ELEMENTS/APP in PROWAM  **
** compiled by Danny C. Mullen 6641-B Tracey Place Ft Polk, LA 71459 **
 Hydrogen        1       1.00797   Gas          -259.2  C    -252.8  C
 Helium          2       4.0026    Gas                       -168.6  C
 Lithium         3       6.941     Solid         179    C    1317    C
 Beryllium       4       9.01218   Solid        1278    C    2970    C
 Boron           5      10.81      Solid        2300    C    2550    C
 Carbon          6      12.011     Solid        3550    C    4872    C
 Nitrogen        7      14.0067    Gas          -210    C    -195.8  C
 Oxygen          8      15.9994    Gas          -218.4  C    -183    C
 Fluorine        9      18.9984    Gas           210    C    -195.8  C
 Neon           10      20.179     Gas          -248.67 C    -246.048C
 Sodium         11      22.98977   Solid          97.81 C     892    C
 Magnesium      12      24.305     Solid         651    C    1107    C
 Aluminum       13      26.98154   Solid         660    C    2467    C
 Silicon        14      28.086     Solid        1410    C    2355    C
 Phosphorus     15      30.97376   Solid          44.1  C     280    C
 Sulfur         16      32.06      Solid         112.8  C     444.6  C
 Chlorine       17      35.453     Gas          -103    C     -34    C
 Argon          18      39.948     Gas          -189.2  C    -185.7  C
 Potassium      19      39.098     Solid          63.65 C     774    C
 Calcium        20      40.08      Solid         842.8  C    1487    C
 Scandium       21      44.9559    Solid        1539    C    2832    C
 Titanium       22      47.90      Solid        1675    C    3260    C
 Vanadium       23      50.9414    Solid        1890    C    3000    C
 Chromium       24      51.996     Solid        1890    C    2482    C
 Manganese      25      54.9380    Solid        1244    C    2097    C
 Iron           26      55.847     Solid        1535    C    3000    C
 Cobalt         27      58.9332    Solid        1495    C    2908    C
 Nickel         28      58.71      Solid        1453    C    2732    C
 Copper         29      63.546     Solid        1083    C    2595    C
 Zinc           30      65.38      Solid         419    C     907    C
 Gallium        31      69.72      Solid          29.78 C    2403    C
 Germanium      32      72.59      Solid         937.4  C    2830    C
 Arsenic        33      74.9216    Solid         814    C
 Selenium       34      78.96      Solid        50/217  C     685    C
 Bromine        35      79.904     Liquid         -7.2  C      59    C
 Krypton        36      83.80      Gas          -156.6  C    -152.3  C
 Rubidium       37      85.4678    Solid          38.9  C     688    C
 Strontium      38      87.62      Solid         769    C    1384    C
 Yttrium        39      88.9059    Solid        1523    C    3337    C
 Zirconium      40      91.22      Solid        1852    C    3578    C
 Niobium        41      92.9064    Solid        2468    C    4927    C
 Molybdenum     42      95.94      Solid        2610    C    5560    C
 Technetium     43      98.9062    Artificial   2172    C    4877    C
 Ruthenium      44     101.07      Solid        2250    C    3900    C
 Rhodium        45     102.9055    Solid        1966    C    3727    C
 Palladium      46     106.4       Solid        1552    C    2927    C
 Silver         47     107.868     Solid         960.8  C    2212    C
 Cadmium        48     112.40      Solid         321    C     765    C
 Indium         49     114.82      Solid         156.61 C    2080    C
 Tin            50     118.69      Solid         231.88 C    2260    C
 Antimony       51     121.75      Solid         630.5  C    1380    C
 Tellurium      52     127.60      Solid         449.8  C     989.9  C
 Iodine         53     126.9045    Solid         113.5  C     184    C
 Xenon          54     131.30      Gas          -111.9  C    -107.1  C
 Cesium         55     132.9054    Solid          28.5  C     671    C
 Barium         56     137.34      Solid         725    C    1640    C
 Lanthanum      57     138.9055    Solid         920    C    3454    C
 Cerium         58     140.12      Solid         798    C    3257    C
 Praseodymium   59     140.9077    Solid         931    C    3212    C
 Neodymium      60     144.24      Solid        1010    C    3127    C
 Promethium     61     145         Artificial   1080    C    2460    C
 Samarium       62     150.4       Solid        1072    C    1778    C
```

Continued on page 45

Continued from page 40

The full 19-bit XBUF address is now held in the CHL registers.

Now the get/put flag in B is incremented. As within the Houdé' code, this sets Z if this is a put operation, and resets it if a get operation. Now B is loaded with 0, the top 3 bits of the 19-bit physical address of UBUF. The lower 16 bits exist correctly in DE; so the full 19-bit address of UBUF is now held in the BDE registers.

If this is a get, the source address of the transfer (CHL -> XBUF) and the destination address (BDE -> UBUF) are correct; so control passes to DMA010. Otherwise, this is a put from (BDE) to (CHL); so these registers must be exchanged before the transfer, since all transfers are from (CHL) to (BDE).

At DMA010, the top 3 bits of both the source address (in C) and the destination address (in B) are loaded into SAR0B and DAR0B, respectively, and control falls through DMAXFER to do the transfer. The return is to RETSP.

Now some comments about the programs which accompany this article.

NEWBANK1/ASM is the source listing for the new @BANK routine, which is based upon the work done by Michel Houdé', my own work on the code to support Richard King's 512K hardware mod, and on work done by David Goben.

The code assumes the existence of all Banks 0 through 10. Do not use it if you do not have a fully populated, 384K XLR8er-equipped Model 4. That is, your Model 4 must have 128K on the motherboard and 256K on the XLR8er.

This new @BANKer is substantially smaller than any of the previous versions. It actually fits in the space taken by the original LS-DOS 6.3.x @BANK routine. The only part which must go into low memory is the @VDCTL routine, which only takes 20 bytes.

One of the reasons it can be smaller is that it manages port 84H (and its image, OPREG$) differently than the Goben code, and goes against the hardware documentation, which appears to be wrong, or at least incomplete.

The difference is in the way bits 5 & 4, Memory Bit 1 and Memory Bit 0, are managed. These two bits determine which bank of normal RAM is switched in. The chart in Table 3 contrasts what the docs say about these two bits and what logic tells me must be the actual case.

I have tested this theory on my non-gate array, my standard gate array, and my Model 4D gate-array machines. On all these machines, both 0-0 and 0-1 in Mem1-Mem0 switch in Bank 0. I don't have a 4P, so have not tested my theory on one of those, but I can't see why it should be different.

In the listing you will find the correct traps to insert 2 memory wait states before keyboard access, and remove them after. I tried David Goben's alternative but, although it seems to make perfect logical sense, it did not work for me. I had "key bounce"-type problems, and worse, when running at 0,1,80,2 or 1,1,80,2.

The two patch files, XLBOOTC/FIX and XLSYS0C/FIX, implement all the code in the NEWBANK1/ASM listing, setting the Z180 to run at it's fastest speed, 0,1,80,2.

Also included are the standard XLR8BU/FIX, XLR8S2/FIX and XLR8S12/FIX patches, plus an additional patch file, XLR8S8/FIX. This patch alters the SYSGEN table to SYSGEN all SVCs from 108 through 119. I include it because I couldn't get LS-DOS 6.3.1 to SYSGEN the @FEXMEM SVC_108 vector without it. (Maybe I just don't know how, and this patch is unnecessary.)

Also included is XLR8C631/JCL, which will automatically install all the patches

Europium	63	151.96	Solid	822	C	1597	C
Gadolinium	64	157.25	Solid	1311	C	3233	C
Terbium	65	158.9254	Solid	1360	C	3041	C
Dysprosium	66	162.50	Solid	1409	C	2335	C
Holmium	67	164.9304	Solid	1470	C	2720	C
Erbium	68	167.26	Solid	1522	C	2510	C
Thulium	69	168.9342	Solid	1545	C	1727	C
Ytterbium	70	173.04	Solid	824	C	1193	C
Lutetium	71	174.97	Solid	1656	C	3315	C
Hafnium	72	178.49	Solid	2150	C	5400	C
Tantalum	73	180.9479	Solid	2996	C	5425	C
Tungsten	74	183.85	Solid	3410	C	5927	C
Rhenium	75	186.2	Solid	3180	C	5627	C
Osmium	76	190.2	Solid	3000	C	5000	C
Iridium	77	192.22	Solid	2410	C	4527	C
Platinum	78	195.09	Solid	1769	C	3827	C
Gold	79	196.9665	Solid	1063	C	2926	C
Mercury	80	200.59	Liquid	-38.87	C	356.9	C
Thallium	81	204.37	Solid	303.5	C	1457	C
Lead	82	207.2	Solid	327.5	C	1744	C
Bismuth	83	208.9804	Solid	271.3	C	1560	C
Polonium	84	209	Solid	254	C	962	C
Astatine	85	210	Solid				
Radon	86	222	Gas	-71	C	-62	C
Francium	87	223	Solid				
Radium	88	226.0254	Solid	700	C	1737	C
Actinium	89	227	Solid				
Thorium	90	232.0381	Solid	1700	C	4000	C
Protactinium	91	231.0359	Solid				
Uranium	92	238.029	Solid	1132.3	C	3818	C
Neptunium	93	237.0482	Artificial	640	C		
Plutonium	94	244	Artificial	639.5	C	3235	C
Americium	95	243	Artificial	850+	C		
Curium	96	247	Artificial	1340	C		
Berkelium	97	247	Artificial				
Californium	98	251	Artificial				
Einsteinium	99	254	Artificial				
Fermium	100	257	Artificial				
Mendelevium	101	258	Artificial				
Nobelium	102	255	Artificial				
Lawrencium	103	257	Artificial				
Rutherfordium	104	261	Artificial				
Hahnium	105	260	Artificial				

mentioned above, except XLR8S8/FIX. If you always use an ERAMDISK, as I do, then install XLS8/FIX. If you want the flexibility of keeping as much lomem free as possible (say to have both DiskDisk memory modules in lomem) then don't install it. If you use it, install it before installing FEXMEM/CMD, and then SYSGEN /immediately. If you choose not to sysgen @FEXMEM, each time you want an ERAMDISK, you must invoke FEXMEM first.

You may recall my saying earlier that I couldn't imagine an application which would cause any part of UBUF to exist in XLR8er RAM. While this, in my opinion, is extremely unlikely, it is possible.

For this reason, two additional files, FEXMEM1/ASM and FEXMEM1/CMD, are also included. FEXMEM1 allows UBUF to be anywhere in RAM. The additional code required to do this, of course, makes the module considerably longer. But if you run into an application which fails with @FEXMEM, and you still want or need the extra ERAMDISK speed, try @FEXMEM1.

Finally, for those to whom 32 extra bytes of lowmem are more important than blinding speed, are two other patch files, XLSYS0D/FIX and XLBOOTD/FIX. These two patches implement all the code in NEWBANK1/ASM except the keyboard traps, and sets the Z180 to run at the Houdé' standard speed (1,1,40) but with REFW=0; so the refresh cycle is 2T instead of 3T. They are meant to be used instead of XLSYS0C/FIX and XLBOOTC/FIX.

If you want to run at the true Houdé' settings of 1,1,40,3 instead of the 2.4% faster 1,1,40,2, change the second to last byte in the "d10,34" line in XLSYS0D/FIX from x'BE' to x'FE'.

Now, the $64 question: "Is this worth all the trouble?" If I didn't think so, I wouldn't have gone to all the trouble to do it.

We've already established that 256-byte DMA memory-to-memory transfers are about 125% faster than LDIRs when the Z180 is running in its fastest mode. But the speed increase afforded by @FEXMEM is substantially greater than that.

The real improvement is in the elimination of all double buffering when transferring data to and from XLR8er RAM banks.

When we eliminate this double buffering, we replace two 597.6 usec LDIRs (total: 1195 usec) with one 267 usec DMA transfer. This is a nearly 350% speed increase!

I would estimate that the average user should experience an effective average ERAMDISK speed increase of about 200%.

In addition to the speed increase, there's a certain satisfaction in knowing your XLR8er-equipped Model 4 is operating up to it's full potential — at least when it comes to managing extended memory and running at maximum (0,1,80,2) clock speed.

To install this software, you should start with a backup of your original LS-DOS 6.3.x disk, and reconfigure your system from scratch, applying all the XL/FIX patches (except, optionally, XLR8S8/FIX) before putting anything else in low memory.

The instructions for installing the @BANK patches are found in the patches themselves, XLBOOTC/FIX AND XLSYS0C/FIX. Or you can use XLR8C631/JCL, which requires access to those files plus the Houdé' XLR8BU/FIX, XLR8S2/FIX, and XLR8S12/FIX files.

For those who prefer the "D" patches over the "C" patches, the file XLR8D631/JCL is included to handle installation.

The big caution here is that @FEXMEM must go in low memory, since its purpose is to switch memory banks around. Make sure you configure your system so there is room for it in low memory.

Editors note: Due to the volume of XLR8er patches appearing in previous issues of TMQ; the fix and JCL files referenced in this article will not be printed herein; however, they are contained on DISK NOTES 6.1.

LDOS 5.3.1: the support continues

- ☆ 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;** time-stamping, too.
- ☆ **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.
- ☆ SYSTEM command supports removable and reusable BLINK, ALIVE, and UPDATE memory modules.
- ☆ The TED text editor now has commands to **print the entire text buffer**, or the contents of the first block encountered. Obtain directories from TED, too!
- ☆ The SPOOL command offers Pause, Resume, and Clear parameters. (OFF) attempts to reclaim memory used.
- ☆ **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-5.3 release. Specify "RESET filespec (DATE=ON)" to establish a file's directory date as that of the **current system date and time.**
- ☆ 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.
- ☆ Best of all, **a 5.3.1 diskette is available as a replacement for your 5.3.0 diskette for $15** (plus $3 S&H in US and Canada, $4 elsewhere). <u>There's no need to return your current master</u>.
- ☆ The 5.3.1 diskette(s) come(s) with a 30-day warranty; written customer support is available for 30 days from the purchase date. Versions for the Model I and Model III are available. **If you do not already have an LDOS 5.3.0, order the 5.3.1 Upgrade Kit with 30 days of customer support for $39.95** (+$4 S&H). Some features require lower case or DDEN adaptor.

Order from: MISOSYS, Inc., POB 239, Sterling, VA 22170-0239

GIF4MOD4 AND HR2GIF

for the Model 4/4P/4D

They say a picture is worth 1,000 words. This picture was converted from GIF to TRS-80 format using GIF4MOD4. Until now, Model 4 users had no way to view GIF images or to send their own hi-res graphics creations to other types of computers. GIF4MOD4 will decode any GIF image up to 640 x 480 x 256 (VGA) and put it on your hi-res screen. If you have no hi-res board, GIF4MOD4 puts it in an /HR disk file so you can dump it to your dot-matrix printer. HR2GIF converts /HR, /CHR and /BLK files to GIF format.

ORDER FORM -- Mail to
J.F.R. Slinkman, 1511 Old Compton Rd., Richmond, VA 23233
Please send me GIF4MOD4 & HR2GIF for my TRS-80 Model 4. I am enclosing $37.95 + $2.00 S&H (add $2.00 outside N. America. Virginia residents please add 4-1/2% [$1.71]).
Name____________________
Address____________________
City__________ State______ Zip______

Let our LB Data Manager solve your data storage problems

LB Version 2: A Flat File Data Manager with more powerful and easy to use features in this latest enhancement of Little Brother!

We've added many features asked for over the past few years by LB users; yet LB is still just 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. 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!

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 +, -, *, /}, *date last modified* {YYYY/MM/DD automatically maintained}, *dollar* {±dddddddd.dd}, *floating point* {±dddddddd.dddddddd, *literal* {any ASCII character}, *numeric* {0-9, -, .}, *right-justified numeric* {flush right 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; keep one alphabetic, another by address. 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.

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.

Report generation:
Report generation incorporates a great deal of 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 or strip trailing spaces. 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.

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.

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.

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 upgrade from version 1.0, send Table of Contents page and $40 +S&H. Remit to:

MISOSYS, Inc.
PO Box 239
Sterling, VA 22170
703-450-4181 or orders to 800-MISOSYS

LS-DOS 6.3.1: The latest for your Model 4

- ☆ 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)
- ☆ Customize the display of the time field in the DIR command to display **12-hr or 24-hr clock time** with SYSTEM (AMPM=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 now 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.**
- ☆ Felt uncomfortable with the *alleged* protection scheme of 6.3? **LS-DOS 6.3.1 has no anti-piracy protection!** MISOSYS trusts its customers to honor our copyrights.
- ☆ Best of all, **a 6.3.1 diskette is available as a replacement for your 6.3.0 diskette for $15** (plus $2 S&H in US). There's no need to return your current master.
- ☆ The 6.3.1 diskette comes with a 30-day warranty; written customer support is available for 30 days from the purchase date. Versions for the Model 4 and Model II/12 are available. **If you do not already have an LS-DOS 6.3.0, order the 6.3.1 Upgrade Kit with 90 days of customer support for $39.95** (+$3 S&H).

MISOSYS, Inc.
P. O. Box 239
Sterling, VA 22170-0239
703-450-4181
[orders to 800-MISOSYS (647-6797) from U.S. or CANADA]

Upgrades now available:
LB Data Manager Version 2.1
LDOS 5.3.1 Model I and III
See Details Inside!

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

Attention Postmaster: Address Correction Requested
Forwarding and return postage guaranteed