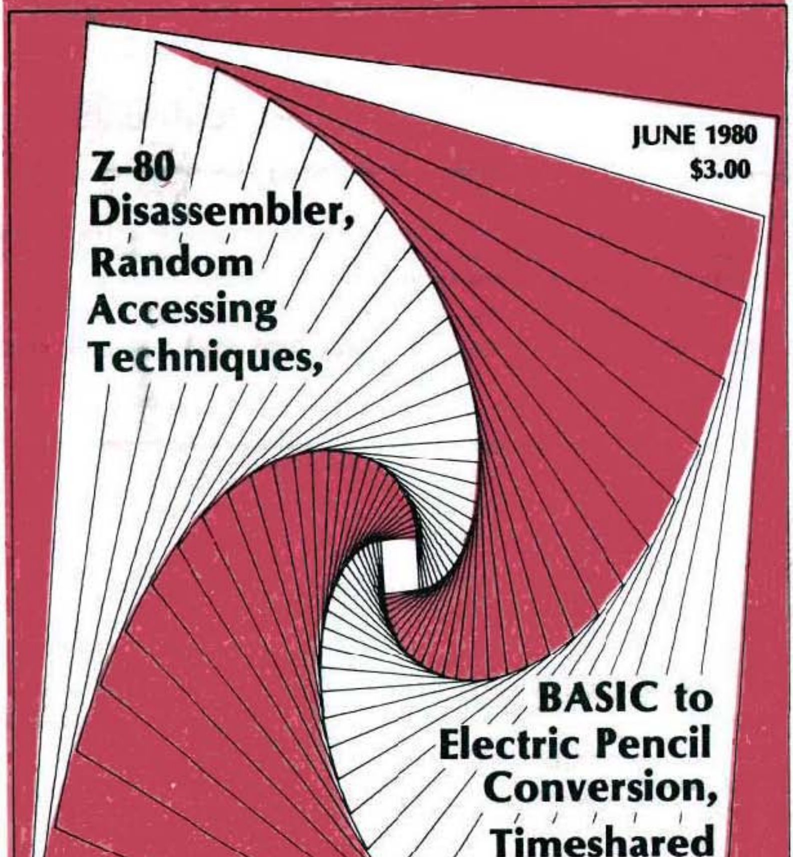
DEDICATED TO THE SERIOUS PROGRAMMER

PROG/80

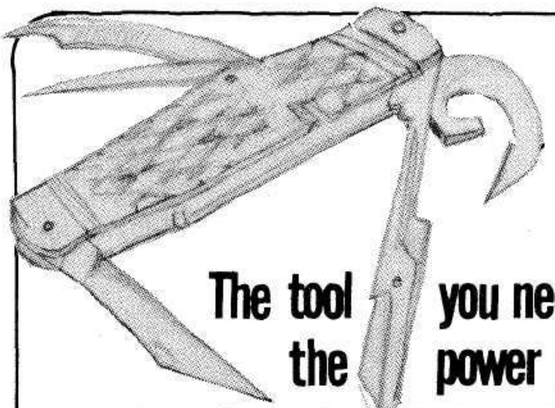


JUNE 1980
$3.00

Z-80 Disassembler, Random Accessing Techniques,

BASIC to Electric Pencil Conversion, Timeshared LPRINT via cassette

and more!



STAD

The tool you need to unlock the power of the Z-80

A powerful monitor for the TRS-80 with special ability in Tracing and Debugging. Single Step through machine language programs or set up to three breakpoints, and look at this display format!

```
AF   BC   DE   HL   IX   IY   AF'  BC'  DE'  HL'  SP   PC
0044 0000 C000 B77C 6433 FFFF 0102 0000 4000 3FC0 41FC 4400
4400    LD      A,93
```

★ ★ ★

All the power of regular monitors as well. Look at these.

A FIRST(0) LAST(FFFF)	ASCII dump
A FIRST 0	formatted ASCII dump
B	start of branch table
B VALA	display in decimal
B VALA VALB(0)	hex arithmetic
C	check system tape
D FIRST(0) LAST(FFFF)	dump hex
E FIRST(0)	edit memory
F FIRST LAST VALUE	find byte
G BRKPTS (3 max.)	set breakpoints, continue
H FIRST LAST VALUE	find word
I PORT	read port
K	keyboard echo
L	load system tape
L SECTOR MEMORY COUNT(1)	load from disk
M FIRST LAST BLOCK	move memory
N	display symbol table
N 0	symbol table to tape
N VALUE	define value for symbol table
N FIRST 0	define start symbol table
O PORT VALUE	write to port
P	initialize memory blocks
P ENTRY	write memory blocks and start
P FIRST LAST	define a memory block
Q FIRST LAST	calculate checksum
R	display / modify registers
S FIRST LAST OPTION(0)	disassembler
T COUNT OPTION(6)	trace instructions
U FIRST COUNT OPTION(0)	unformatted tape I/O
V FIRST LAST BLOCK	verify memory
W SECTOR MEMORY COUNT(1)	write to disk
X FIRST LAST BLOCK	exchange memory
Z FIRST LAST VALUE(0)	zero memory

★ ★ ★

16K, 32K, and 48K versions on one cassette for $24.95

Use from tape in Level II or as a command file from Disk.

The Software Exchange

6 SOUTH ST., MILFORD, NH 03055 (603) 673-5144

PROG/80

BASIC

DISK

BYTING OFF MORE...

A Library of Information From

The Software Exchange

LITTLE BOOK OF BASIC STYLE by John M. Nevison. Ideal reference for BASIC programmer, junior high to research scientist. Indexed, illustrated, 151 pages. $5.95 plus $1 shipping.

Z-80 INSTRUCTION HANDBOOK by Scelbi Publications. Convenient pocket-size manual describes Z-80 capabilities in easy-to-understand terms. Designed as a practical reference to mnemonics, machine codings, usage, for programmers of every level—beginner to professional—anyone working in Z-80 machine or assembler language. Appendixed. $4.95 plus $1.00 shipping.

THE BASIC HANDBOOK by Dr. David A. Lien. A definite reference/idea book. Explains over 50 favorite versions of the BASIC language in detail as used in micros, minis and mainframes. Users/Learners Manual. 360 pages. $14.95 plus $1.00 shipping.

TRS-80 INTERFACING by Jonathon Titus. What you need to know to connect your TRS-80 to the world. Assumes knowledge of some machine language programming. $8.95 plus $1.00 shipping

INTRODUCTION TO TRS-80 GRAPHICS by Don Inman. Set and reset graphics only. $9.95 plus $1.00 shipping.

Z-80 SOFTWARE GOURMET GUIDE AND COOKBOOK by Scelbi Publications. Over 100 usable subroutines, plus how to use them. $14.95 plus $1.00 shipping.

PATHWAYS THROUGH THE ROM. The definitive guide to Level II BASIC. Includes Super Map by Fuller Software, The TRS-80 Disassembled Handbook by Robert Richardson, HexMem by J. Phillip, and Z-80 Disassembler by G. Blank. $19.95 plus $1.00 shipping.

Z-80 AND 8080 ASSEMBLY LANGUAGE PROGRAMMING by Kathe Spracklen. The best introduction to assembly language we sell. You should have experience in BASIC. $7.95 plus $1.00 shipping.

SCELBI'S SECRET GUIDE TO COMPUTERS. This book will turn you into a computer expert quickly and easily. You'll learn BASIC, having fun every step of the way. The book explains how to deal with computer machinery, which buttons to press, and trains you to write many kinds of programs. The author's "underground" style of writing is sure to hold your interest. This book contains 150 sample programs, analyzes them and gives you hints on how to invent your own. Charts are given comparing the different computers. $5.95 plus $1.00 shipping.

TRS-80 DISK AND OTHER MYSTERIES by Harvard C. Pennington. If you are serious enough about disk programming to own RSM-2D or NEWDOS+, then you ought to purchase this book. It explains the organization and features of TRSDOS 2.0, 2.1, 2.2, NEWDOS, and VTOS, tells you how to use Superzap, RSM-2D, Monitor 3, Debug, Dircheck, and LMOffset, explains the directory track on the disk, file structure (including Electric Pencil files), and even gives detailed procedures for recovery of lost data. While this book is not recommended for the inexperienced user, it is essential for the serious disk programmer. The book is expensive, in 8½ by 11 format with typewritten instead of typeset printing, and only 130 pages, yet the information is worth the price. $22.50 plus $1.00 shipping.

Order
TOLL FREE
1-800-258-1790
in NH call 673-5144

PROG/80

VOL. 1 NO. 8

Publisher
Roger W. Robitaille, Sr.

Business Manager
Elizabeth J. Robitaille

Editor-in-Chief
George W. Blank

Editor
Lance Micklus

Software Editor
James Garon

Time Sharing Editor
Richard Taylor

Copy Editor
Sheila Jahnle

Creative Director
Elaine Cheever

Production
Laurie Miller
Gail Nash
Donna Bennett

Subscriptions Manager
Diana Bishop

Customer Service
Bette Keenan

PROG/80 is published bimonthly by SoftSide Publications, Milford, New Hampshire. Telephone (603) 673-5144. Subscription rates: USA regular bulk rate - $15 per year. USA first class, Canada, Mexico, APO/FPO, overseas surface mail - $21 per year. Overseas airmail - $27 per year. **Make all remittances payable in US funds.** Mail Subscription Inquiries to PROG/80 Subscriptions Manager, PO Box 68, Milford, NH 03055.

PLEASE NOTE:

Our WATS line
number is:

1-800-258-1790

(In NH call 673-5144)

TOLL FREE
9 AM to 9 PM*
(Monday - Friday)
9 AM to 3 PM
(Saturdays)

It is ONLY to be used for ordering software.

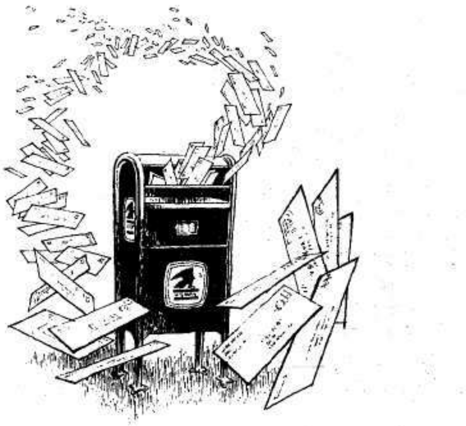
Programmers will not be available on this line.

Your Cooperation Will Be
Most Appreciated

*Eastern Standard Time

Outgoing Mail

George Blank



Lance Micklus was busy helping to put on the Vermont Educational Television auction at the time of our deadline, and asked me to do a guest editorial. Since I have just turned over the reins of SoftSide to James Garon, I am delighted to keep my hand in.

The Trenton Computer Faire was a roaring success this year. One of the highlights for Roger Robitaille, Joe Breton, and myself was an evening with Richard Taylor and Clayton Schneider after the show. Clay is a contributor to this magazine, and Richard our time-sharing editor. Now that the U.S. Snail Service is raising first class postage to 20 cents, we can't wait until must correspondence can go electronically and bypass Uncle Sam, so timesharing is an idea whose time has come.

Pathways Through the ROM, the first book from SoftSide Publications, is ready for the printer and should be available in early June. Both of the books that form the heart of the book, Supermap and the TRS-80 Disassembled Handbook, have been getting good reviews in other publications. I am already using the galley proofs as an indispensable reference. For example, I used Supermap to locate the ROM routines for last issue's article on SYSTEM tapes.

Despite all the rumors about the obsolescence of the TRS-80 Model I, we hear from a pretty good source that the system is included in the next Radio Shack catalog, to be released in August for 1981. We feel reasonably certain that the new FCC rules concerning television interference will require some changes, but do not fear the sudden obsolescence of all our programs.

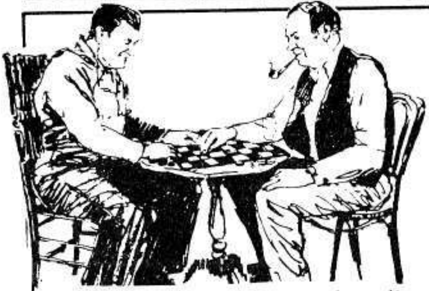
That does not mean that there will be no changes required. Clay and I were discussing the impact of the Radio Shack lower case mod on his BASIC File Utility and my automated disk directory. Since the modified computers store ASCII letters as control characters, both programs need to be changed for modified units. We do have information sheets available on the change if you care to send a self addressed stamped envelope to File Utility/Lower Case or to Disk Directory/Lower Case, P.O. Box 68, Milford, N.H. 03055.

We still can't prove that Radio Shack is coming out with a color computer, as they like to keep things under their hat. We do know that they now have a part number for a color monitor. The official word out of Fort Worth is that the monitor is for demonstsrating video games in the stores, but the part number starts with a 26, which is a computer part number, not a video game number.

I'd like to hear from readers about what you would like in PROG/80. One thing that we are considering is articles for people who are looking for income from their computer. For example, we might run comparisons of the contracts, terms, advertising, and typical payments of different software houses and magazines. This might eventually become a separate magazine, not for the Radio Shack market, but a "Writer's Guide" for programmers. Write and tell me what you would like!

APL80 has been around long enough to be noticed in the APL world, and the verdict is pretty favorable. The disk version at least is not simply a toy, especially in the latest release which adds)COPY, transposition, choice of dimension, format, and latent expressions. The price is now up to $39.95, and our upgrade policy is to provide the most recent version for the difference in price and a $5 handling and shipping fee. You must return your original. I suggest waiting a month in order to get the update manual as well.

Make plans to see us at the Philadelphia show and the Washington show in September, the Chicago Show in October, or the Boston Show in November if we are near you. We plan to attend all four. I don't promise to have time to talk at the show—we have to keep busy to pay the cost, but you can find a wide selection of our software and see some of our significant hardware products like the Busy Box, COMM-80, and the Eaton LRC Printer demonstrated.



TO ORDER CALL TOLL-FREE 800-258-1790

(in NH call 673-5144)

TRS-80 COMPUTERS:	LIST PRICE	OUR PRICE
Level II, 4K	$619.00	$559.00
Level II, 16K, no keypad		669.00
Level II, 16K, w/keypad	849.00	749.00
EXPANSION INTERFACES:		
COMM-80 Interface		179.95
Expansion Interface, no RAM	299.00	269.00
Expansion Interface, 16K RAM (NEC)	448.00	369.00
Expansion Interface, 32K RAM, (NEC)	597.00	459.00
DISK DRIVES:		
Percom, TFD-100, 40-track	399.00	389.00
Percom, Dual TFD-100's	795.00	775.00
Percom, TFD-200, 77-track	675.00	650.00
Percom, Dual TFD-200's	1350.00	1300.00
CompuThink, Dual Drive, Dual Sided		1295.00
DISK DRIVE ACCESSORIES:		
2-Drive cable for TRS-80	29.95	29.00
4-Drive cable for TRS-80	39.95	39.00
Percom Data Separator		29.95
Extender Card	15.95	15.00
PRINTERS:		
Centronics 730	795.00	719.00
Centronics 737	995.00	895.00
Centronics 753-2	3196.00	2695.00
Centronics 779-2	1559.00	995.00
RS Quick Printer II	219.00	209.00
RS Line Printer III	1960.00	1813.00
NEC 5530 SpinWriter	2995.00	2595.00
LRC 7000+	389.00	369.00
LRC 7000+ (64 col)	405.00	389.00

HARDSIDE™

PRINTER CABLES:		
QPII to Expansion Interface Cable	19.95	19.00
LRC to TRS-80 cable		20.00
LRC to APPLE cable		20.00
LRC to EXIDY cable		20.00
LRC to PET, IEEE cable		59.00
LRC to RS232C male, cable		65.00
LRC to RS232C female, cable		65.00
730 or 737 to TRS-80 cable		29.00
779 or 753 to TRS-80 cable		35.00
PERIPHERIALS:		
Novation CAT Modem	189.95	179.00
UDS 103-LP		195.00
RS-232-C Interface Board	99.00	89.00
TRS-232 Printer Interface		39.95
DATA Dubber		49.95
16K Memory Kit, Keyboard		99.00
16K Memory Upgrade Kit, E.I.		95.00
Percom Electric Crayon, w/cable		279.00
Busy Box, TRS-80		109.95
Busy Box, APPLE		114.95
BSR X-10, Starter Kit		124.95
ATARI COMPUTERS:		
ATARI 800 Computer System	1080.00	875.00
ATARI 400 Computer	630.00	515.00
ATARI 410 Program Recorder	89.95	65.00
ATARI 810 Disk Drive	699.95	565.00
ATARI 820 Printer	599.95	470.00
Joystick Controllers	19.95	18.95
Paddle Controllers	19.95	18.95

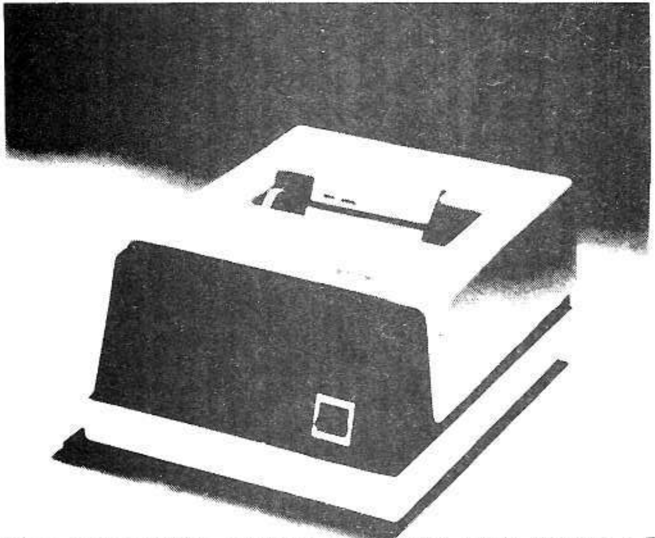
Prices subject to change without notice.

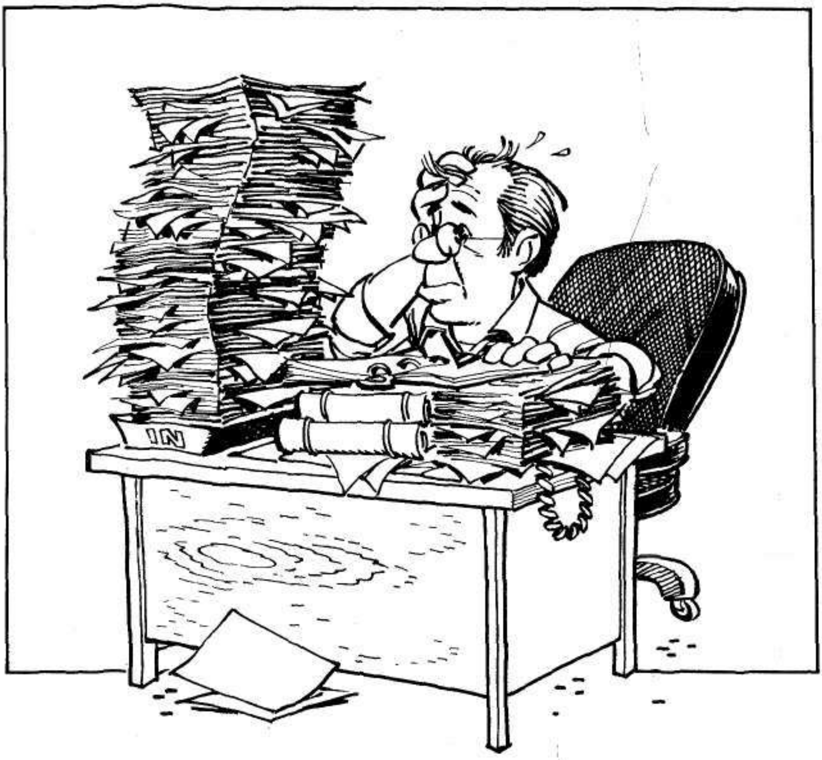
Prices do **not** include shipping

COD orders require 25% cash deposit.

*****EFFECTIVE JUNE 1st 1980*****

6 South Street,
Milford, NH 03055





Z-80 DISASSEMBLER

by George Blank

I have been astonished at the steady demand for the simple monitor and unsophisticated disassembler that I wrote for the first issue of Prog/80 (Simple Simon, March 1979). There is apparently a need for machine language programming and study tools that avoid the hassle of loading system tapes, and John Phillipp's excellent monitor in the last issue (Hex Mem, February 1980) moved me to write a complete disassembler to replace the simpler one included in Simple Simon.

While Simple Simon gave only the subset of commands that are common to the 8080 and Z-80 chips, and managed to confuse the two instruction sets at the same time, the current monitor disassembles the full Z-80 set of almost 700 instructions. In addition, the ability to construct a symbol table and reserve data blocks in the disassembled listing has been added for a truly useful listing.

Since one of the best features of basic programming is the ease with which programs may be modified for special uses, I have left in many remark statements. Particular routines that users may wish to modify include the automatic printout routine in Line 62 and the count-and-stop routine in Line 69.

The automatic printout routine works by testing the printer-ready status bit at location 14312. If it finds the value 63, indicating that the printer is on and ready for data, it sends data to the printer. If you have no printer, you may wish to remove this routine, and if your printer does not use the handshake at 14312, you may wish to change the print option.

If you want a continuous printout, you may simply delete Line 69, or you may modify the counter (C is the number of lines printed) to fit the page size on your printer. For example, you might use:

```
69 IF (PEEK(14312) 63>AND C>15) OR C>60 THEN INPUT"(PRESS
ENTER)";X$:c=0
```

if you wanted to stop after 15 lines on the screen or 60 on the printer.

The line numbers begin at 30 to make it easy to combine this program with hex mem. Then you could either patch the programs with a GOTO from HEXMEM or use the command "RUN 30" to use the disassembler. By removing all the REM statements, you may still be able to load another basic program above HEXMEM and the disassembler, as long as there are no conflicts in the numbering of the programs.

If you wish to allow for the entry of hexadecimal addresses for the start and finish of your disassembly, there is a conversion routine from hex to decimal at Lines 178-186. The routines to convert decimal to hex are located at Lines 70, 73-78, and 84. If you wish to print displacement addresses in $ or + and - form, the routine currently used to calculate hex address jumped to is in Lines 80-82.

To use the symbol table, answer "Y" when asked if you wish to construct a symbol table. Then choose hex or decimal entry and enter first the memory location and then your chosen symbol. Table entry will end when you fail to enter a value or symbol, or when your address exceeds the ending point you have chosen for your symbolic dump. If you wish to print the symbol table after your memory dump, add a flag equal to the value of S in Line 176 just before the final GOTO (SE=S), change Line 63 to:

```
63 IF M>ME THEN 192.
```

and add a routine to take the addresses in the array MS(0), convert them to hexadecimal if you wish, and print the corresponding symbol from the array AS(0) to AS(SE). If you really want to be fancy, add a routine to ignore the data blocks on the first pass through the table, then print the data blocks after the symbol table.

The symbol table routine reserves the symbols "DATA" and "EOD" for the start and finish data blocks. If the program, during execution, comes across the symbol data, it sets DA to 1 in Line 165 and the program jumps to the data routine at 188 to 191 from Line 72. Then, once it comes across the symbol "EOD", it sets DA to 2 in Line 165 and back to 0 (data flag off) in Line 188.

If you wish to eliminate the symbol table and data block routine completely, delete Lines 71 and 163-190, change 72 to:

72 GOTO 48.

and delete the last one-third of Line 46.

To delete the extended instruction set (instructions not used in the Level II ROM) delete Lines 96-126, 130-138, and 141-162. Then add the following lines:

97 REM

104 REM

106 N=N+2:GOTO140

I make no claims to efficient code. There is a wide range between testing every possible number as in Line 126 and forming 64 op codes with a single line of basic as in Line 61. Many of the Z-80 codes repeat at intervals of 8 to 16, as provided for in Lines 50 to 57. At other times it is possible to provide a series of tests with a fall through when the right op code is reached, as in Line 136. One rule that became clear was that more planning leads to less code!

```
30 REM * Z-80 DISASSEMBLER * COPYRIGHT (C) 1980 GEORGE BLANK *
31 CLEAR1000:DEFSTRA:DEFINTB-L,N-Z:DIMAH(15):DIMA(200)
32 FORB=0TO15:READAH(B):NEXT:FORB=0TO7:READAD(B):NEXT:FORB=0TO9:
READAP(B):NEXT:FORB=0TO7:READAI(B):NEXT:FORB=0TO7:READAF(B):NEXT
:FORB=0TO7:READAA(B):NEXT:AC(2)="HL":AC(3)="A"
33 REM * AH(0-15) *
34 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
35 REM * AD(0-7)  *
36 DATA B,C,D,E,H,L,(HL),A
37 REM * AP(0-15) *
38 DATA BC,DE,HL,SP,AF,(BC),(DE),(HL),(SP),AF'
39 REM * AI(0-7)  *
40 DATA ADD,ADC,SUB,SBC,AND,XOR,OR,CP
41 REM * AF(0-7)  *
42 DATA NZ,Z,NC,C,PO,PE,P,M
43 REM * AA(0-7)  *
44 DATA RLC,RRC,RL,RR,SLA,SRA,SRL,SRL
45 REM * INPUT ADDRESSES TO DISASSEMBLE *
46 C=0:CLS:PRINT"Z-80 DISASSEMBLER":INPUT"STARTING ADDRESS (DECI
MAL)";MB:INPUT"ENDING ADDRESS";ME:M=MB:INPUT"DO YOU WANT TO CREA
TE A SYMBOL TABLE";A:IFLEFT$(A,1)="Y"GOSUB169
47 A="":GOTO65:REM * CONTENTS OF 4 BYTES OF MEMORY *
48 D=PEEK(N1):D1=PEEK(N2):D2=PEEK(N3):D3=PEEK(N4):GOSUB86:D4=D/8
:D5=D/16:DR=D-8*D4:DF=(D/16-D5)*2:IFD4<8THEN50ELSEIFD4<16THEN61:
ELSEIFD5<12THEN88 ELSE90
```

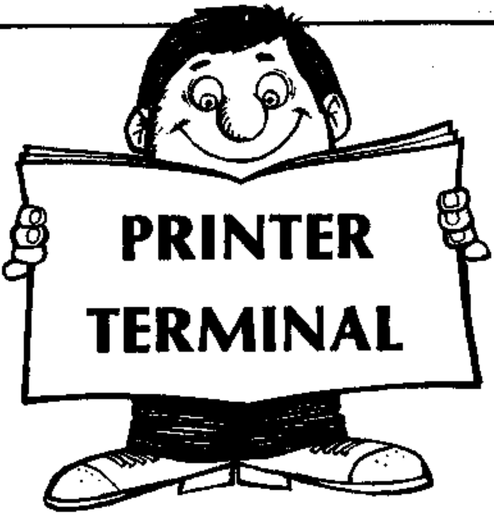
```
49 REM * OP CODES 00H TO 3FH *
50 AD=AD(D4):AP=AP(D5):IFDR<>1THEN51 ELSEIFDF=0THENAP=AP(D5):A="
LD "+AP+",":GOSUB76:GOTO62 ELSEA="ADD HL,"+AP:GOTO62
51 IFDR<>2THEN52 ELSEIFDF=0ANDD4<4THENA="LD ("+AP+"),A":GOTO62 E
LSEIFDF=0ANDD4>3THENA="LD ":GOSUB74:A=A+","+AC(D5):GOTO62 ELSEA=
"LD A,("+AP+")":IFD4>3THENA="LD "+AC(D5)+",":GOSUB74:GOTO62 ELSE
62
52 IFDR<>3THEN53 ELSEIFDF=0THENA="INC "+AP:GOTO62 ELSEA="DEC "+A
P:GOTO62
53 IFDR=4THENA="INC "+AD:GOTO62
54 IFDR=5THENA="DEC "+AD:GOTO62
55 IFDR=6THENA="LD "+AD+",":GOSUB78:GOTO62
56 IFDR=7THEN58 ELSEIFD4=0THENA="NOP"ELSEIFD4=1THENA="EX AF,AF'"
ELSEIFD4=2THENA="DJNZ"ELSEIFD4=3THENA="JR"ELSEIFD4=4THENA="JR NZ
"ELSEIFD4=5THENA="JR Z"ELSEIFD4=6THENA="JR NC"ELSEA="JR C"
57 IFD4<2THEN62 ELSE80
58 IFD4=0THENA="RLCA"ELSEIFD4=1THENA="RRCA"ELSEIFD4=2THENA="RLA"
ELSEIFD4=3THENA="RRA"ELSEIFD4=4THENA="DAA"ELSEIFD4=5THENA="CPL"E
LSEIFD4=6THENA="SCF"ELSEA="CCF"
59 GOTO62
60 REM * OP CODES 40 - 7F * LD R,R *
61 A="LD "+AD(D4-8)+","+AD(DR):IFD=118THENA="HALT"
62 N=N+1:M=M+N:AX=LEFT$(AX,N*3):N=0:PRINTTAB(6)AXTAB(20)ASTAB(30
)A:IFPEEK(14312)=63LPRINTA1TAB(6)AXTAB(20)ASTAB(30)A
63 IFM>METHENINPUT"<PRESS ENTER>";X$:GOTO46 ELSE47
64 REM * NEXT MEMORY LOCATION *
65 IFM<32768THENN1=MELSEN1=M-65536
66 IFM<32767THENN2=M+1ELSEN2=M-65536+1
67 IFM<32766THENN3=M+2ELSEN3=M-65536+2
68 IFM<32765THENN4=M+3ELSEN4=M-65536+3
69 C=C+1:IFC=15THENINPUT"<PRESS ENTER>";X$:C=0
70 H0=INT(M/4096):H1=INT((M-4096*H0)/256):H2=INT((M-(4096*H0+256
*H1))/16):H3=M-(4096*H0+256*H1+16*H2):A1=AH(H0)+AH(H1)+AH(H2)+AH
(H3)+" ":PRINTA1;" ";
71 IFS>0THEN164
72 IFDB>0THEN180ELSE48
73 REM * PRINT 2 DIGIT HEX CODE IN ( ) *
74 A=A+"(":GOSUB76:A=A+")":RETURN
75 REM * PRINT 2 DIGIT HEX CODE *
```

```
76 H=D2:GOSUB84:GOSUB78:N=N+1:RETURN
77 REM * PRINT 1 DIGIT HEX CODE *
78 H=D1:GOSUB84:N=N+1:RETURN
79 REM * CALCULATE DISPLACEMENT *
80 A=A+" ":H=D1:IFH>127THENH=H-256
81 REM * PROGRAM COUNTER = +2 * PRINT HEX ADDRESS *
82 NN=H+N+2:D2=INT(NN/256):D1=NN-256*D2:GOSUB76:N=N-1:GOTO62
83 REM * CONVERT BYTE TO HEX *
84 H1=INT(H/16):A=A+AH(H1):H1=H-H1*16:A=A+AH(H1):RETURN
85 REM * CONVERT CONTENT OF MEMORY TO HEXADECIMAL *
86 H=D:GOSUB84:A=A+" ":H=D1:GOSUB84:A=A+" ":H=D2:GOSUB84:A=A+" "
:H=D3:GOSUB84:AX=A+" ":A="":RETURN
87 REM * OP CODES 80 - BF * REGISTER ARITHMETIC *
88 A=AI(D4-16)+" "+AD(DR):GOTO62
89 REM * OP CODES A0 - FF EXCEPT C8 D0 ED FD *
90 D4=D4-24:AF=AF(D4):IFDR=0THENA="RET "+AF:GOTO62ELSEIFDR=2THEN
A="JP "+AF+",":GOSUB76:GOTO62ELSEIFDR=4THENA="CALL "+AF+",":GOSU
B76:GOTO62ELSEIFDR=7THENA="RST ":H=D4*8:GOSUB84:GOTO62ELSEIFDR=6
THENA=AI(D4)+" A,":H=D1:GOSUB84:GOTO62
91 IFDF=1THEN93ELSEIFDR=1THENAP(3)="AF":A="POP "+AP(D5-12):AP(3)
="SP":GOTO62ELSEIFDR=5THENAP(3)="AF":A="PUSH "+AP(D5-12):AP(3)="
SP":GOTO62
92 IFD4=0THENA="JP ":GOSUB76:GOTO62ELSEIFD4=2THENA="OUT ":GOSUB7
8:A=A+",A":GOTO62ELSEIFD4=4THENA="EX SP,HL":GOTO62ELSEA="DI":GOT
062
93 IFDR=5THEN94 ELSEIFDR=1THEN95 ELSEIFD4=1THEN97 ELSEIFD4=3THEN
A="IN A,":GOSUB78:GOTO62 ELSEIFD4=5THENA="EX DE,HL":GOTO62 ELSEA
="EI":GOTO62
94 IFD4=1THENA="CALL ":GOSUB76:GOTO62ELSEIFD4=3THEN104ELSEIFD4=5
THEN133ELSE106
95 IFD4=1THENA="RET":GOTO62ELSEIFD4=3THENA="EXX":GOTO62ELSEIFD4=
5THENA="JP (HL)":GOTO62ELSEA="JP M,":GOSUB76:GOTO62
96 REM * OP CODES CB XX  *
97 N=N+1:DA=D1/8:DB=D1-8*DA:IFDA>7THEN98ELSEA=AR(DA)+" "+AD(DB):
GOTO62
98 IFDA>15THEN99ELSEA="BIT "+AH(DA-8)+AD(DB):GOTO62
99 IFDA>23THEN100ELSEA="RES "+AH(DA-16)+AD(DB):GOTO62
100 A="SET "+AH(DA-24)+AD(DB):GOTO62
101 REM * ADJUST NN FOR 4 BYTE OP CODE *
```

```
102 D1=D2:D2=D3:RETURN
103 REM * OP CODES DD XX *
104 AY="IX":GOTO108
105 REM * OP CODES FD XX *
106 AY="IY"
107 REM * AY = IX OR IY * AZ = (IX+DIS) OR (IY+DIS) *
108 N=N+1:A="":H=D2:GOSUB84:AZ="("+AY+"+"+A+")":A="":D4=D1/8
109 REM * HEX HALF BYTES OF SECOND BYTE  AW=MSHB AX=LSHB *
110 H=D1:GOSUB84:AW=LEFT$(A,1):AX=RIGHT$(A,1):A="":IFD1=203THEN1
28
111 REM * XD09 TO XD39 *
112 IFD1>57THEN114 ELSEIFAX="9"THENAP(2)=AY:A="ADD "+AY+","+AP(V
AL(AW)):AP(2)="HL":GOTO62
113 REM * XD21 TO XD36 *
114 IFD1=33THENA="LD "+AY+",":GOSUB102:GOSUB76:GOTO62ELSEIFD1=34
THENA="LD ":GOSUB102:GOSUB74:A=A+","+AY:GOTO62ELSEIFD1=35THENA="
INC "+AY:GOTO62ELSEIFD1=42THENA="LD "+AY+",":GOSUB102:GOSUB74:GO
TO62ELSEIFD1=43THENA="DEC "+AY:GOTO62
115 IFD1=52THENA="INC "+AZ:N=N+1:GOTO62ELSEIFD1=53THENA="DEC "+A
Z:N=N+1:GOTO62ELSEIFD1=54THENA="LD "+AZ+",":H=D3:GOSUB84:GOTO62
116 REM * XD86XX TO XD6EXX *
117 IFD1>111THEN119ELSEIFD1<70OR(NOT(AX="6"ORAX="E"))THEN140 ELS
EA="LD "+AD(D4-8)+","+AZ:N=N+1:GOTO62
118 REM * XD70XX TO XD7EXX *
119 IFD1>117THEN120ELSEA="LD "+AZ+","+AD(D1-112):N=N+1:GOTO62
120 IFD1>133THEN122 ELSEIFD1=119THENA="LD "+AZ+",A":N=N+1:GOTO62
 ELSEIFD1=126THENA="LD A,"+AZ:N=N+1:GOTO62 ELSE140
121 REM * CULL INOPERABLE CODES *
122 IFD1>190THEN126ELSEIFAX="6"THEN124ELSEIFAX="E"THEN124ELSE140
123 REM * XD86XX TO XDBEXX *
124 A=AI(D4-16)+" A,"+AZ:N=N+1:GOTO62
125 REM * XDE1 TO XDF9 *
126 IFD1=225THENA="POP "+AY:GOTO62ELSEIFD1=227THENA="EX (SP),"+A
Y:GOTO62ELSEIFD1=229THENA="PUSH "+AY:GOTO62ELSEIFD1=233THENA="JP
 ("+AY+")":GOTO62ELSEIFD1=249THENA="LD SP,"+AY:GOTO62ELSE140
127 REM * INDEXED BIT AND ROTATE GROUP * DD CB  AND  FD CB *
128 D4=D3/8:DR=D3-8*D4:IFDR<>6THEN140
129 IFD4<8THENA=AA(D4)+" "+AZ:N=N+2:IFD4=6THEN140 ELSE62
130 D4=D4-8:IFD4<8THENA="BIT"ELSED4=D4-8:IFD4<8THENA="RES"ELSED4
```

```
=D4-8:A="SET"
131 A=A+STR$(D4)+","+AZ:N=N+2:GOTO62
132 REM * ED GROUP * CULL INOPERATIVE CODES *
133 N=N+1:IFD1<64ORD1>188OR(D1>163ANDD1<169)OR(D1>171ANDD1<175)O
R(D1>179ANDD1<184)THEN140
134 IFD1<124THEN142 ELSEIFD1<143THEN140
135 REM * EDA0 TO EDBB * BLOCK TRANSFER AND SEARCH *
136 D=D1:IFD>159A="LDI":IFD>160A="CPI":IFD>161A="INI":IFD>162A="
OUTI":IFD>167A="LDD":IFD>168A="CPD":IFD>169A="IND":IFD>170A="OUT
D":IFD>175A="LDIR":IFD>176A="CPIR":IFD>177A="INIR":IFD>178A="OTI
R":IFD>183A="LDDR":IFD>184A="CPDR":IFD>185THENA="INDR"
137 IFD=187THENA="OTDR"
138 GOTO62
139 REM * INOPERATIVE CODE * ADJUST FOR SINGLE BYTE *
140 N=N-1:A="-DATA-":GOTO62
141 REM * ED40 TO ED7BXXXX *
142 D=D1-64:D4=D/8:D5=D/16:DR=D-8*D4:DF=(D/16-D5)*2:AC="(C)"
143 REM * EDX0 *
144 IFDR>0THEN146ELSEA="IN "+AD(D4)+","+AC:IFD4=6THEN140ELSE62
145 REM * EDX1 EDX9 *
146 IFDR>1THEN148ELSEA="OUT (C),"+AD(D4):IFD4=6THEN140ELSE62
147 REM * EDX2 EDXA *
148 IFDR>2THEN150ELSEIFDF=0THENA="SBC HL,"+AP(D5):GOTO62ELSEA="A
DC HL,"+AP(D5):GOTO62
149 REM * EDX3 EDXB *
150 IFDR>3THEN154ELSEA="LD ":GOSUB150:IFDF=0THENGOSUB74:A=A+","+
AP(D5)ELSEA=A+AP(D5)+",":GOSUB74
151 REM * NO ED63 ED6B *
152 IFD5=2THEN140ELSE62
153 REM * ED44 *
154 IFDR>4THEN156ELSEIFD4=0THENA="NEG":GOTO62 ELSE140
155 REM * ED45 ED4D *
156 IFDR>5THEN158ELSEIFD5>0THEN140ELSEIFDF=0THENA="RETN":GOTO62E
LSEA="RETI":GOTO62
157 REM * ED46 ED56 ED5E *
158 IFDR>6THEN161ELSEIFD=6THENA="IM 0":ELSEIFD=22THENA="IM 1"ELS
EIFD=30THENA="IM 2"ELSE140
159 GOTO62
160 REM * EDX7 *
```

```
161 IFD=7THENA="LD I,A"ELSEIFD=23THENA="LD A,I"ELSEIFD=39THENA="
RRD"ELSEIFD=47THENA="RLD"ELSE140
162 GOTO62
163 I$=INKEY$:IFI$=""THEN163 ELSEPRINTI$:RETURN
164 AS="":IFM>=MS(SN)THENAS=AS(SN):SN=SN+1:IFSN>STHENS=0
165 IFAS="DATA"THENDB=1ELSEIFAS="EOD"THENDB=2
166 GOTO72
167 S=S-MS(S):IFS<0THENS=0
168 GOTO173
169 CLS:PRINT"SYMBOL TABLE CONSTRUCTION":PRINT:PRINT"IF YOU WANT
 A SYMBOL TABLE, YOU MUST ENTER EACH ADDRESS AND THE":PRINT"SYMB
OL YOU WANT. YOU MUST ENTER THE ADDRESSES IN NUMERICAL":PRINT"OR
DER. SYMBOLS ARE LIMITED TO SIX CHARACTERS AND 100 SYMBOLS."
170 PRINT:PRINT"TWO SYMBOLS ARE RESERVED FOR SPECIAL USE:":PRINT
TAB(6)"USE 'DATA' TO INDICATE THE START OF A BLOCK OF DATA":PRIN
TTAB(6)"USE 'EOD' TO INDICATE THE END OF A BLOCK OF DATA."
171 DIM AS(100):DIM MS(100):S=0:SN=0
172 PRINT:PRINT"PRESS <ENTER> AFTER LAST SYMBOL TO END INPUT":PR
INT"DO YOU WISH TO ENTER HEX OR DECIMAL ADDRESSES (H/D)":GOSUB16
3
173 IFI$="H"THEN177
174 PRINTS;:INPUT"MEMORY LOCATION (DECIMAL)";MS(S)
175 IFMS(S)<0THEN167ELSEIFMS(S)>MEORMS(S)<MPTHENPRINT:RETURN
176 INPUT"<SYMBOL>";AS(S):IFAS(S)=""THENPRINT:RETURNELSEMP=MS(S)
:S=S+1:GOTO173
177 INPUT"MEMORY LOCATION <HEXADECIMAL>";A
178 MS=0:L=LEN(A):ONLGOSUB183,182,181,180
179 PRINT:RETURN
180 AH=LEFT$(A,1):A=RIGHT$(A,3):GOSUB185:MS=MH*4096
181 AH=LEFT$(A,1):A=RIGHT$(A,2):GOSUB185:MS=MS+MH*256
182 AH=LEFT$(A,1):A=RIGHT$(A,1):GOSUB185:MS=MS+MH*16
183 AH=A:GOSUB185:MS=MS+MH:PRINT"DECIMAL: ";MS;:MS(S)=MS:GOTO175
184 REM * CONVERT HEX TO DECIMAL *
185 FORX=0TO15:IFAH(X)=AHTHENMH=X:RETURN
186 NEXT:MH=0:RETURN
187 REM * DATA BLOCK PRINTOUT *
188 D=PEEK(N1):H=D:GOSUB64:AX=A:IFDB=2THENDB=0
189 IF D>31 AND D<96 THEN A=CHR$(A) ELSE A=" "
190 GOTO 62
```



PROGRAM "LPRINT"

USE A STANDARD TIME-SHARING TERMINAL TO PRINT HARD COPY WITH YOUR 16K LEVEL II TRS-80

Let's face it: It's nice to be able to get a hard copy of program listings. It's also nice to be able to use the "LPRINT" command occasionally to output data. But printers are expensive. Many TRS-80 users are just not able to justify the cost of a line printer, expansion interface and/or an RS-232 port for an occasional listing or program output.

This was the problem we faced at Georgia Southern College. We have been using the TRS-80 to introduce teachers to uses of microcomputers in education. We have purposely stuck with simple hardware—generally 16K Level II Machines—since most schools do not want to invest a great deal of money initially in hardware. So we've tried to do the most we can with this basic system without buying a lot of extra equipment.

We do have standard teletype terminals for the University System of Georgia's timesharing system. These terminals, which are very common across the country, use acoustic couplers operating at 110 baud to transfer data across standard telephone lines. I began to wonder if there were some simple way of interfacing a 16K Level II TRS-80 to one of these terminals. It occurred to me that the modems provided a ready interface mechanism—if only I could get the TRS-80 to produce appropriate tones.

ACOUSTIC MODEM DATA FORMAT

I decided at this point that I needed to find out more about the format used by modems for data transmission. A little research revealed that a modem uses a 2225 HZ pulse to represent a high bit (1) and a 2025 HZ tone to represent a low bit (0). The duration of these pulses depends on the data transmission rate—at 110 baud these pulses have a duration of 9091 microseconds.

When no data is being transmitted, the data line is kept at the high frequency. A byte of incoming information is preceeded by one low (2025 HZ) pulse. The data byte then follows, one bit at a time, starting with bit 0 and ending with bit 6. Bit 7 is not required to transmit standard ASCII code, so it is used generally as a parity bit. That is, this bit is adjusted, either high or low, to make the sum of all eight transmitted bits either odd or even. (This is known as either odd or even parity.)

Finally, to separate bytes, there are two high bits (pulses) generally called stop bits. In summary, to send one byte of information, there are a total of 11 pulses transmitted-one start bit (low), 7 data bits (either high or low), 1 parity bit (either high or low), and two stop bits (high).

You can perhaps see if 110 bits (pulses) are sent per second, ten bytes are transmitted per second. It should also be apparent why each transmitted pulse is 9091 microseconds in duration:

$$\frac{\text{1 SEC}}{\text{110 BITS}} = .009091 \text{ SEC/BIT (or 9091 Microseconds)}$$

It might be good here to look at a specific example. The ASCII code for the letter "X" is 88 in decimal, or in binary, 01011000. Figure 1 illustrates how this byte would be received by a standard modem at 110 baud, assuming odd parity. Note that since the number of high bits equals three, which is already odd, the high-order bit is left unchanged for odd parity.

PRODUCING TONES ON THE CASSETTE

Now that we know how data is received by a modem, we are ready to make the TRS-80 produce these data tones on the cassette recorder. This is not actually as hard as it may first seem. There are a number of programs on the market which allow you to program music on the TRS-80. These programs all produce musical tones on the cassette recorder.

In order to understand how this is done, we need to discuss the cassette latch in the TRS-80. The cassette recorder is controlled by a 4-bit latch. This is simply another kind of memory which can hold data until it is signaled by the CPU to discard it. The cassette latch only uses the lower 4 bits of a byte. The high nibble (bits 4-7) are simply discarded. The four bits in this latch can be set using the assembly language command "OUT 255, X" where X is the number to be stored in the latch.

Bits 0 and 1 of this latch control the write head of the cassette recorder. If bit 0 is set (1) and bit 1 is reset (0), the head is turned on. If these two bits are reversed (so bit 0 is reset and bit 1 is set), the head is turned off. Bit 2 of the latch turns the cassette motor on (set) and off (reset) through a small relay. We'll ignore bit 3 of the latch in this article.

Let's see what happens if we have the following command in an assembly language program:

```
OUT   255,5
```

Since 5 (decimal) is 0101 in binary, this command will cause the cassette to be turned on since bit 2 is set. It will also cause the write head to be turned on since bit 0 is set and bit 1 is reset. The command:

```
OUT   255,6
```

will cause the number 0110 to be stored in the latch. The tape will still be on (bit 2 set), but now the record head is off.

By turning the head on and off at even intervals, we can produce a square wave, or tone. The only problem remaining is to work out the proper timing, and this requires just a little bit of mathematics and a knowledge of the execution times of Z-80 machine language instructions.

The manual for the TRS-80 Editor/Assembler gives the execution times for each Z-80 command—however, there's a slight catch! The times given are for a 4 megahertz clock. But the TRS-80 uses a clock that operates at 1.774 megahertz. In other words, we have to multiply each of the execution times given in the manual by a constant in order to get the correct execution time. This constant is determined simply as follows:

4.00 MHZ
= 2.225
1.774 MHZ

Let's take an example. The Editor/Assembler Manual gives an execution time of 1 microsecond for the command:

```
LD    A,B
```

The actual execution time for the TRS-80 would be:

1 MSEC * 2.225 = 2.225 MSECS

For half this time (225 MSEC) the write head must be on; for the other half (again 225 MSEC) the write head must be off.

We determined earlier that at 110 baud each pulse of tone must last 9091 microseconds. The final calculation involves finding the number of cycles of 225 HZ tone per pulse:

9091 MSEC/PULSE
= 20.4 CYCLES PER PULSE
449 MSEC/CYCLE

So we need about 20.4 cycles of this tone to produce one high data pulse.

Similar calculations for 2025 HZ results in a period of 494 microseconds (247 MSEC per half period) and 18.4 cycles per data pulse.

While it is not possible to produce these exact times, we can get close enough to fool the modem! Let's see how it's done.

PRODUCING THE HIGH (2225 HZ) PULSE

A portion of the source program is produced below. In this copy of the program the comments have been replaced with the total time required for each instsruction. Let's look at this carefully. (All of the times shown here have been adjusted for the correct clock time.)

```
1750        HIGH   LD     C,20       ;   3.89 MSEC
1760        CONT   LD     B,28       ;   3.89 MSEC
1770               LD     A,5        ;   3.89 MSEC
1780               OUT    255,A      ;   6.12 MSEC
1790               XOR    A          ;   2.23 MSEC
1800               XOR    A          ;   2.23 MSEC
1810          UP   DJNZ   UP         ;202.48 MSEC
1820               LD     B,28       ;   3.89 MSEC
1830               LD     A,6        ;   3.89 MSEC
1840               OUT    255,A      ;   6.12 MSEC
1850               XOR    A          ;   2.23 MSEC
1860               XOR    A          ;   2.23 MSEC
1870        DOWN   DJNZ   DOWN       ;202.48 MSEC
1880               DEC    C          ;   2.23 MSEC
1890               JR     NZ,CONT    ;   6.68 MSEC
1990               RET               ;   5.56 MSEC
```

Statement 1750 loads register C with 20. This is the number of cycles which will be output per pulse. Note in our previous calculations we found we needed 20.4 cycles to make a 9091 microsecond pulse. By using only 20 cycles, we'll be a little short on time, but this missing time is partially made up before and after the pulse. In other words, we have a few microseconds of silence between pulse, but these are not noticed by the modem.

Statement 1760 loads register B with 28. This controls the duration of the on (write) half-cycle in conjunction with the DJNZ loop found in statement 1810.

Statement 1770 puts the value 5 into the accumulator. This value is sent to the cassette latch in statement 1780—turning on the cassette motor (actually in this case, keeping it on) and turning on the write head.

Statements 1790 and 1800 are simply included to "fine tune" the tone produced. The XOR A operation zeros the contents of the accumulator, which is unimportant—but, the operation requires 2.225 microseconds. The two commands thus add 4.45 microseconds to the length of time that the write head is on.

Statement 1810 is the main delay loop for the "on" half cycle.

Statement 1820 through 1870 produce the "off" half cycle. They are nearly identical to statements 1760—1810. The only exception is that we are now loading the cassette latch with 6 (0110 binary). This keeps the motor running, but turns off the write head.

Statement 1880 decrements the value in register C. Remember, this is the number of cycles per pulse. If we havea not yet had 20 cycles, program control is transferred back to statement 1760 by the "JR NZ,CONT" command in 1890.

Now let's look carefully at the timing. If you add up the times for statements 1760 through 1810 you'll get a total of 220.84 microseconds. This is just 4 microseconds short of the required half period of 224.5 microseconds.

One more example, this one a little more complicated:

```
NXT   DJNZ   NXT      (E.T. = 3.25 MSEC)
```

This command will cause the B register to be decremented and if a non-zero results, will loop to the label NXT (in this case, back to itself). Each of these loops will take 3.25 * 2.225 microseconds. If register B contained the value 10 before this command was encountered, the total execution time would be 3.25 * 2.225 * 10 = 73.20 microseconds.

Now that we know how to calculate the execution times of Z-80 instructions, we have to determine a few more figures. Let's start with the necessary high tone—2225 HZ. One cycle of any tone consists of a peak (write head on) and a trough (write head off). For 2225 HZ, the period of one cycle is determined as follows:

1
 = .000449 SEC OR 449 MSEC
2225 HZ

Now let's find the total time of the pulse. The first thing to do is add up the times for statements 1760 through 1890.

3.89 + 3.89 + 6.12 + 2.23 + 2.23 + 202.48 + 3.89 +
3.89 + 6.12 + 2.23 + 2.23 + 202.48 + 2.23 + 6.68 =
450.59 MICROSECONDS

Keeping in mind these instructions will be executed twenty times, we now multiply by 20:

450.59 MSEC * 20 = 9011.8 MICROSECONDS

Finally, add in the times of the first and last instructions:

9011.8 MSEC + 9.45 MSEC = 9021.25 MSEC

This is 70 microseconds short of the optimal pulse duration. However, there will be other instructions that must be carried out between each pulse. These other instructions will bring the average pulse duration just about to the required 9091 microseconds. Even if this were not true, we have a pulse here which is only .76% short of standard.

We will not go through the timing mathematics of the low tone (2025 HZ) pulse here, but will leave that up to the reader. The calculations are essentially identical.

PUTTING IT ALL TOGETHER

Now that we've discussed the theory of the program's operation, we can briefly discuss the rest of the program. You will notice that we've included two listings: (1) a source listing for the Editor/Assembler and (2) a short BASIC program which "Pokes" the machine language program into high memory in decimal form. We'll first discuss the source program.

Let's begin by pointing out that there is a block of memory locations in the TRS-80 which are dedicated to the line printer. The line printer control block begins at memory location 16421 decimal and runs through 16428. Two of these bytes, 16422 and 16423, contain the address of the printer driver. When a "LLIST" or "LPRINT" command is executed, Level II BASIC loads the byte to be printed in register C and calls the address in these two bytes.

Normally this vector points to a subroutine in ROM BASIC that takes care of the printing. Among other things, it patiently waits until the line printer signals it has printed the byte. This is done by setting the Z flag. This is why the computer "locks up" if you type "LLIST" by mistake when there is no printer attached. It's waiting for the flag to be set, signaling the printer's ready for the next byte.

Program "LPRINT" makes use of this driver address vector. Indeed, if things did not work this way, it would be very difficult to make this program work. At any rate, what we do is poke the starting address of our machine language program into these two memory locations. Then, whenever an LPRINT or LLIST command is executed, Level II BASIC directs the Z-80 to our routine. Remember, the Z-80 comes to this routine with a byte to be printed in register C.

The origin of the source program is 32170. Users with 32 or 48K memories can adjust this origin as needed. The main "DRIVER" of the program begins at line 320 and runs through 760. This part of the program checks to see if the byte to be printed is a special carriage control character, keeps track of the number of lines printed, and calls a subroutine which stores the byte in a buffer.

You may wonder why we bother with this buffer. Why not just send the byte immediately to the teletype via the cassette? Originally the program was written that way, but we found that there were often pauses between bytes which were long enough to completely confuse the modem. Garbage resulted. By using a buffer, these delays between bytes are eliminated and so is the garbage.

The first subroutine, called "LDBUF" starts at line 850. This subroutine is called from the main driver program for each byte. It keeps track of where in the buffer each byte is to be stored, puts it there, and when the buffer is full, calls the subroutine PBUFF.

Subroutine PBUFF, starting at line 1010, controls the actual printing of the buffer contents. Essentially it loads the bytes one at a time from the buffer into register D and then calls subroutine OUTB2. In addition, the subroutine inserts extra carriage returns when needed if a line will exceed the maximum line length specified by the user. If it's necessary for this to be done, it also increments the line counter.

Subroutine OUTB2 begins at line 2180. This subroutine is called from PBUFF with the byte to be printed in register D. Using the RRC (right rotation into carry flag) instruction. This subroutine breaks the byte down one bit at a time beginning with the least significant bit (LSB). Each RRC instruction causes the LSB to be placed in the carry flag, all other bits are moved right one position, anda the carry flag is copied in to the most significant bit.

After each rotation subroutine select is called. This subroutine calls either subroutine HIGH (which outputs a high pulse) or subroutine LOW (which outputs a low pulse), depending on the status of the carry flag.

Subroutine OUTB2 also automatically causes a start bit to be output preceeding each byte. After 7 bits have been output, it also sets the parity of the 8th bit. In this version the parity will be odd. You can make the parity even, if you wish, by switching the words "HIGH" and "LOW" in statements 2360 and 2380. Finally, the subroutine outputs three stop bits. You may recall that two stop bits are standard. An extra is added here to make up for possible slight shortages in the timing of the data pulses.

The subroutines HIGH and LOW, which actually output the data pulses to the cassette recorder, have already been discussed in some detail above.

At the end of the program is a short data block. "SREM" defines one byte that is used to store the maximum line length desired. This value is also poked into memory location 16426 in the line printer control block. You may change this number to any value you wish. "BLEN" defines a byte which is used by the program to keep track of the number of bytes currently in the buffer.

Following "BLEN" the buffer (BUFF) is defined. Although the buffer is actually 255 bytes long, only 220 are defined in the program. This is to make room for a short initialization program which follows. Once the intialization program is executed, it is overwritten and becomes part of the buffer.

This initialization program does several things: (1) it first stores the starting address of the new driver (32170 decimal) in the line printer control block in bytes 16422 and 16423, (2) it stores the starting address for the subroutine which prints the contents of the buffer in the "USR" vector at 16526 and 16527, (3) it stores the maximum line length desired in location 16426, and finally (4) jumps to Level II BASIC.

You may wonder why we store the starting address of the "PBUFF" routine in the "USR" command. Remember that "PBUFF" is only called when the buffer is full. When an LPRINT or LLIST command is executed, it is likely that the buffer will be filled and printed several times. But, at the end, the odds are that the buffer will be only partially filled. Including a basic statement like "I=USR(0)" as the last statement in your program will cause the remaining contents of the buffer to be printed. For similar reasons, when using the LLIST command, you should enter something like this in command mode:

```
LIST:I=USR(0)
```

THE BASIC LANGUAGE PROGRAM

If you don't have the Editor/Assember, the BASIC program below will poke the machine language program into high level memory. If you use this and wish to change the maximum line length to some value other than 71, change statement 230 and the next to last data item (71).

USING THE PROGRAM

Whether you are using a "SYSTEM" tape made with the Editor/Assembler or using the BASIC program, you need to protect high memory at 32169 on power-up. After you have done this, load the program and execute. If you are using the "SYSTEM" program, execute by typing "/" and pressing ENTER. The initialization program will load the necessary values in the right memory locations and will return to "READY". Running the BASIC program will do the same thing.

After ;you have done this, you're ready to use LLIST and LPRINT commands in the normal way (see the Level II Manual). The only difference is that you must make sure you have a cassette in the recorder and the recorder must be in record mode whenever one of these commands is executed. Standard modem tones will be recorded on the cassette. Printing the contents of the cassette can be handled in any one of a number of ways.

The simplest way to print the tape requires a simple earphone that will fit the jack on the cassette recorder. Plug the earphone into the ear jack. Pick up the phone which is generally located next to teletype terminals and dial the first few numbers of the local exchange to get rid of the dial tone. Now put the earplug into the receiver end of the modem and place the telephone handset in the cradle in the normal fashion on top of the earphone. Turn on the teletype and play the tape with the level set at "4". If you find the teletype is missing characters, you've got the volume set too high. Turn it down a bit.

You can also send the information to be printed over a regular phone to some remote terminal. Simply establish a connection with someone by the remote terminal, and have them set up the terminal in the normal fashion. Then, place your telephone with the mouthpiece over the speaker of the cassette recorder. Play the tape with the volume again in the "4" position. You may need to adjust the volume up or down, depending on the quality of line you have.

It is also possible to print "LIVE" if you have the teletype handy. Just remove the black jack from the cassette, insert the earplug, and set up the modem as described for the first method above. Don't put a cassette in the recorder, but reach in on the left side and hold in the record-protect lever while you depress the record and play button. Now you can LLIST and LPRINT to your hearts desire.

IN CONCLUSION

Of course if money is no object, forget all of this and buy a good quality printer. But if you have only occasional need for hard copy and can arrange access to a teletype terminal, this program could save you a pile of money. Keep in mind that most colleges have teletypes. If you strike up a friendship with one of the faculty, you can probably use a teletype at little or no cost.

One final word. This program is designed for 110 baud. Most of the Bell teletypes operate at that speed, but many of the newer timesharing terminals operate at 300 baud. I haven't had time yet to experiment with adapting this program to that rate, but you know enough now to try it yourself. You may have to play with the timing a bit, but I see no reason why it wouldn't work. If you succeed at this, I'd really appreciate a copy of the program!

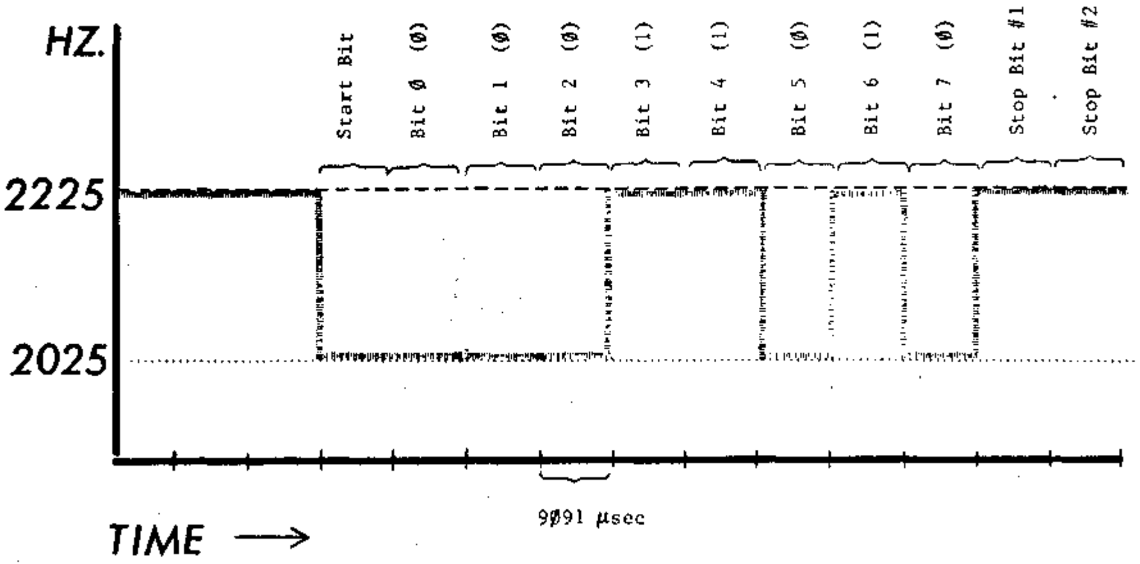


Figure 1. *Modem data transmission format for* the letter "X". (88 decimal, 01011000 binary.)

BASIC PROGRAM

```
100 REM***PROGRAM LPRINT**BASIC VERSION 1.2***MARCH 10, 1980
110 REMX   COPYRIGHT (C) 1980,  OWEN F. GREDE, PH.D.
120 REM                           GEORGIA SOUTHERN COLLEGE
130 REM                           STATESBORO, GA.  30460
140 REM
150 REM*** PUT THE FOLLOWING STATEMENTS NEAR THE BEGINNING
160 REM    OF YOUR PROGRAM
170 REM ------------------------------------------------
180 CLEAR50:DEFINTA-Z
190 POKE 16526,29:POKE16527,126  'ADDRESS OF PRINT BUFF
200 POKE 16422,170:POKE16423,125  'DRIVER ADDRESS
210 POKE 16424,66    'DESIRED LINES PER PAGE
220 POKE 16425,0     'ZERO LINE COUNTER
230 POKE 16426,71    'SET DESIRED LINE LENGTH
240 GOSUB300:END
250 REM
260 REM*** THE FOLLOWING SHOULD GO NEAR THE END OF YOUR
270 REM    PROGRAM.
280 REM ------------------------------------------------
290 REM*** THIS SUBROUTINE POKES THE PROGRAM IN HIGH MEMORY
300 CLS:FORI=32170TO32507:READA:POKEI,A:PRINTA;:NEXT:RETURN
310 REM*** THE FOLLOWING DATA STATEMENTS CONTAIN THE DECIMAL
320 REM    REPRESENTATION OF THE MACHINE LANGUAGE PROGRAM
330 DATA 197,253,229,229,121,183,40,73,254,11,32,7,62,13,205
340 DATA 0,126,24,62,254,12,32,24,175,221,182,3,40,18,221
350 DATA 126,3,221,150,4,71,197,62,10,205,0,126,193,16,247
360 DATA 24,30,87,205,0,126,122,254,10,40,9,254,13,32,21
370 DATA 62,10,205,0,126,221,52,4,221,126,4,221,190,3,122
380 DATA 32,4,221,54,4,0,225,253,225,193,201,253,33,251,126
390 DATA 6,0,253,78,0,253,52,0,253,35,253,9,253,119,0
400 DATA 58,251,126,254,255,192,205,29,126,201,33,250,126,221,33
410 DATA 37,64,78,35,70,175,184,200,205,114,126,35,86,205,187
420 DATA 126,122,35,254,13,32,5,221,78,5,24,40,254,10,40
430 DATA 36,13,32,33,126,254,13,40,28,22,13,205,187,126,22
440 DATA 10,205,187,126,221,52,4,221,126,4,221,190,3,32,4
450 DATA 221,54,4,0,221,78,5,16,199,121,50,250,126,175,50
```

```
460 DATA 251,126,211,255,201,30,255,197,205,135,126,29,32,250,193
470 DATA 201,56,4,205,161,126,201,205,135,126,201,14,20,6,28
480 DATA 62,5,211,255,175,175,16,254,6,28,62,6,211,255,175
490 DATA 175,16,254,13,32,233,201,14,18,6,31,62,5,211,255
500 DATA 175,175,16,254,6,31,62,6,211,255,175,175,16,254,13
510 DATA 32,233,201,197,205,161,126,203,10,205,125,126,203,10,205
520 DATA 125,126,203,10,205,125,126,203,10,205,125,126,203,10,205
530 DATA 125,126,203,10,205,125,126,203,10,205,125,126,203,10,234
540 DATA 236,126,205,161,126,24,3,205,135,126,205,135,126,205,135
550 DATA 126,205,135,126,193,201,71,0
```

ASSEMBLY LANGUAGE PROGRAM

```
00660 ;PROGRAM "LPRINT"----VERSION 1.2---MARCH 8, 1980
00670 ;COPYRIGHT (C) 1980     OWEN F. GAEDE, PH.D.
00680 ;                       GEORGIA SOUTHERN COLLEGE
00690 ;                       STATESBORO, GA  30460
00700 ;THIS PROGRAM ALLOWS LPRINT AND LLIST COMMANDS TO BE
00710 ;DIRECTED TO A 110-BAUD TELETYPE WITH A STANDARD ACOUSTIC
00720 ;MODEM.  MEMORY LOCATIONS 16414 AND 16415 CONTAIN THE
00730 ;STARTING ADDRESS OF THE LINE PRINTER DRIVER.  THE
00740 ;ADDRESS OF THE ORIGIN OF THIS PROGRAM SHOULD BE POKED
00750 ;INTO THESE MEMORY LOCATIONS.
00760 ;***
00770 ;***
00780 ;ENTRY  :(C)    CONTAINS BYTE TO BE PRINTED
00790 ;EXIT   :Z FLAG SET
00800 ;***
00810 ;***
00820 ;THE FOLLOWING IS THE DRIVER ROUTINE.  LPRINT AND LLIST
00830 ;COMMANDS LOAD BYTES TO BE PRINTED, ONE AT A TIME, IN
00840 ;REGISTER C.  THIS DRIVER STORES THESE BYTES IN A BUFFER
00850 ;WHICH IS 255 BYTES LONG.  WHEN THE BUFFER IS FULL, THE
00860 ;STORED BYTES ARE CONVERTED TO MODEM TONES.
00870 ;-----------------------------------------------------
```

```
7DAA          00880        ORG   32170
7DAA C5       00890 BEGIN  PUSH  BC         ;SAVE BYTE TO PRINT IN C
7DAB FDE5     00900        PUSH  IY         ;SAVE CONTENTS OF THESE
7DAD E5       00910        PUSH  HL         ;REGISTERS FOR LATER
7DAE 79       00920        LD    A,C        ;LOAD A WITH BYTE
7DAF B7       00930        OR    A          ;CHECK TO SEE IF BYTE = 0
7DB0 2849     00940        JR    Z,NEXTB    ;IF SO, GET NEXT BYTE
7DB2 FE0B     00950        CP    11D        ;IS BYTE 11?
7DB4 2007     00960        JR    NZ,NXTCK   ;IF NOT, GO CHECK FOR 12
7DB6 3E0D     00970        LD    A,13       ;IF SO, PRINT A CARRIAGE
7DB8 CD007E   00980        CALL  LDBUF      ;RETURN--NO LINE FEED
7DBB 183E     00990        JR    NEXTB      ;GET NEXT BYTE
7DBD FE0C     01000 NXTCK  CP    12D        ;CHECK FOR 12
7DBF 2018     01010        JR    NZ,OUT1    ;IF NOT, PRINT BYTE
7DC1 AF       01020        XOR   A          ;MAKE A ZERO
7DC2 DDB603   01030        OR    (IX+3)     ;IS 16424 (LINES/PG) 0?
7DC5 2812     01040        JR    Z,OUT1     ;IF SO, IGNORE 12
7DC7 DD7E03   01050 LOOP1  LD    A,(IX+3)   ;GET LINES/PAGE
7DCA DD9604   01060        SUB   (IX+4)     ;SUBTRACT PRESENT LINES
7DCD 47       01070        LD    B,A        ;LOAD B WITH # LINES LEFT
7DCE C5       01080 LOOP2  PUSH  BC         ;SAVE  BC
7DCF 3E0A     01090        LD    A,10       ;CARRIAGE FEED
7DD1 CD007E   01100        CALL  LDBUF      ;STORE LINEFEED IN BUFFER
7DD4 C1       01110        POP   BC         ;GET BC BACK
7DD5 10F7     01120        DJNZ  LOOP2      ;DO UNTIL B ZERO
7DD7 181E     01130        JR    NEXTA      ;GOTO NEXTA
7DD9 57       01140 OUT1   LD    D,A        ;PUT BYTE IN D
7DDA CD007E   01150        CALL  LDBUF      ;STORE BYTE IN BUFFER
7DDD 7A       01160        LD    A,D        ;PUT BYTE BACK IN A
7DDE FE0A     01170        CP    10         ;WAS BYTE A  10?
7DE0 2809     01180        JR    Z,INCLI    ;IF SO, INC LINE NUMBER
7DE2 FE0D     01190        CP    13         ;WAS BYTE 13?
7DE4 2015     01200        JR    NZ,NEXTB   ;IF NOT, GOTO NEXTB
7DE6 3E0A     01210        LD    A,10
7DE8 CD007E   01220        CALL  LDBUF      ;STORE 10 IN BUFFER
7DEB DD3404   01230 INCLI  INC   (IX+4)     ;INCREMENT # OF LINES
7DEE DD7E04   01240        LD    A,(IX+4)   ;LOAD A WITH # OF LINES
7DF1 DDBE03   01250        CP    (IX+3)     ;COMPARE WITH MAX #/PAGE
7DF4 7A       01260        LD    A,D        ;LOAD BYTE PRINTED IN A
```

```
7DF5 2004     01270         JR      NZ,NEXTB        ;IF LINES <> MAX LINES
7DF7 DD360400 01280 NEXTA   LD      (IX+4),0        ;SET # OF LINES TO 0
7DFB E1       01290 NEXTB   POP     HL              ;RESTORE REGISTERS & RET
7DFC FDE1     01300         POP     IY
7DFE C1       01310         POP     BC
7DFF C9       01320         RET
              01330 ;**
              01340 ;***
              01350 ;SUBROUTINE LDBUF--LOADS BUFFER UNTIL FULL.  OUTPUTS THE
              01370 ;ENTRY  (A)      CONTAINS BYTE TO BE STORED
              01380 ;EXIT:  (A)      ZEROED
              01390 ;       (ZF)     RESET
              01400 ;------------------------------------------------------
7E00 FD21F87E 01410 LDBUF   LD      IY,BLEN         ;IY POINTS TO BUFF LEN
7E04 0600     01420         LD      B,0             ;RESET HIGH BYTE IN BC
7E06 FD4E00   01430         LD      C,(IY+0)        ;C HAS #BYTES IN BUFFER
7E09 FD3400   01440         INC     (IY+0)          ;INCREMENT BYTE COUNTER
7E0C FD23     01450         INC     IY              ;POINT IY TO BUFFER START
7E0E FD09     01460         ADD     IY,BC           ;IY NOW NEXT BUF POSITON
7E10 FD7700   01470         LD      (IY+0),A        ;STORE BYTE IN THIS SPOT
7E13 3AF87E   01480         LD      A,(BLEN)        ;PUT #BYTES STORE IN A
7E16 FEFF     01490         CP      255             ;255 STORED?
7E18 C0       01500         RET     NZ              ;IF NOT, RETURN
7E19 CD1D7E   01510         CALL    PBUFF           ;IF SO, PRINT BUFFER
7E1C C9       01520         RET                     ;THEN RETURN
              01530 ;**
              01540 ;**
              01550 ;SUBROUTINE PBUFF-PRINTS BUFFER CONTENTS
              01560 ;------------------------------------------------------
7E1D 21FA7E   01570 PBUFF   LD      HL,SREM         ;SET HL TO START OF DATA
7E20 DD212540 01580         LD      IX,16421        ;MAKE SURE IX IS PROPER
7E24 4E       01590         LD      C,(HL)          ;SPACE LEFT IN LINE
7E25 23       01600         INC     HL              ;NEXT DATA
7E26 46       01610         LD      B,(HL)          ;PUT BUFFER LENGTH IN B
7E27 AF       01620         XOR     A               ;MAKE A ZERO
7E28 B8       01630         CP      B               ;BUFFER LENGTH ZERO?
7E29 C8       01640         RET     Z               ;IF SO, NOTHING TO DO
7E2A CD727E   01650         CALL    LEADER          ;PUT LEADER ON TAPE
7E2D 23       01660         INC     HL              ;HL POINTS TO BUFFER
```

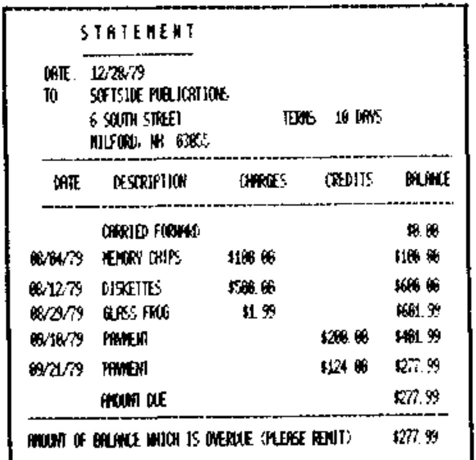
```
7E2E 56       01670 PB1   LD    D,(HL)     ;LOAD D WITH CHARACTER
7E2F CD8B7E   01680       CALL  OUTB2      ;PRINT BYTE IN D
7E32 7A       01690       LD    A,D        ;PUT BYTE IN A
7E33 23       01700       INC   HL         ;POINT TO NEXT CHAR
7E34 FE0D     01710       CP    13         ;WAS LAST CHAR CR?
7E36 2005     01720       JR    NZ,PB4     ;IF NOT, INCR LINE LGTH
7E38 DD4E05   01730       LD    C,(IX+5)   ;RESET CHAR COUNTER
7E3B 1828     01740       JR    PB2        ;JUMP TO END OF LOOP
7E3D FE0A     01750 PB4   CP    10         ;LINE FEED?
7E3F 2824     01760       JR    Z,PB2      ;IF LINEFEED, GOTO PB2
7E41 0D       01770       DEC   C          ;C HAS SPACE LEFT IN LINE
7E42 2021     01780       JR    NZ,PB2     ;JUMP TO NEXT IF NOT 0
7E44 7E       01790       LD    A,(HL)     ;LOAD A WITH BYTE
7E45 FE0D     01800       CP    13         ;IS NEXT BYTE 13?
7E47 281C     01810       JR    Z,PB2      ;IF SO, GOTO PB2
7E49 160D     01820       LD    D,13       ;OUTPUT LINE RETURN
7E4B CD8B7E   01830       CALL  OUTB2      ;OUTPUT 13
7E4E 160A     01840       LD    D,10       ;LINE FEED
7E50 CD8B7E   01850       CALL  OUTB2      ;OUTPUT 10
7E53 DD3404   01860       INC   (IX+4)     ;INCREMENT # OF LINES
7E56 DD7E04   01870       LD    A,(IX+4)   ;NUMBER OF LINE IN A
7E59 DDBE03   01880       CP    (IX+3)     ;COMPARE WITH LINES/PAGE
7E5C 2004     01890       JR    NZ,PB5     ;IF NO MATCH, GOTO PB5
7E5E DD360400 01900       LD    (IX+4),0   ;ZERO LINE COUNTER
7E62 DD4E05   01910 PB5   LD    C,(IX+5)   ;LOAD C WITH CHARS/LINE
7E65 10C7     01920 PB2   DJNZ  PB1        ;DO PB1 IF BUFF NOT EMPTY
7E67 79       01930       LD    A,C        ;C HAS SPACE LEFT IN LINE
7E68 32FA7E   01940       LD    (SREM),A   ;SAVE SPACE REMAINING
7E6B AF       01950       XOR   A          ;SET A TO ZERO
7E6C 32FB7E   01960       LD    (BLEN),A   ;SET BUFFER LGTH TO 0
7E6F D3FF     01970       OUT   (255),A    ;SHUT OFF CASSETTE
7E71 C9       01980       RET
              01990 ;***
              02000 ;***
              02010 ;SUBROUTINE LEADER--THIS PUTS A LEADER OF 255 HIGH TONES
```

```
               02020 ;ON THE TAPE PRECEEDING EACH BUFFER DUMP.  THIS IS NEEDED
               02030 ;TO ALLOW THE MODEM TO FUNCTION PROPERLY AND THE CASSETTE
               02040 ;TO COME UP TO SPEED.
               02050 ;------------------------------------------------------------
7E72 1EFF      02060 LEADER  LD      E,255           ;LOAD COUNTER
7E74 C5        02070         PUSH    BC              ;SAVE BC
7E75 CD877E    02080 LD2     CALL    HIGH            ;OUTPUT HIGH PULSE
7E78 1D        02090         DEC     E               ;COUNT
7E79 20FA      02100         JR      NZ,LD2          ;LOOP BACK IF NOT DONE
7E7B C1        02110         POP     BC              ;RESTORE BC
7E7C C9        02120         RET                     ;RETURN
               02130 ;***
               02140 ;***
               02150 ; SUBROUTINE SELECT--OUTPUTS PROPER SIGNAL DEPENDING ON
               02160 ; THE STATE OF THE CARRY FLAG
               02170 ;ENTRY. (C)     BYTE BEING OUTPUT
               02180 ;       (CF)    SET IF BIT IS 1, RESET IF BIT IS 0
               02190 ;EXIT   (C)     UNDISTURBED
               02200 ;------------------------------------------------------------
7E7D 3804      02210 SELECT  JR      C,GOHIGH
7E7F CDA17E    02220         CALL    LOW
7E82 C9        02230         RET
7E83 CD877E    02240 GOHIGH  CALL    HIGH
7E86 C9        02250         RET
               02260 ;***
               02270 ;***
               02280 ;SUBROUTINE HIGH--OUTPUTS A 9091 MICROSECOND PULSE OF A
               02290 ;2225 HZ TONE--ONE HIGH BIT.
               02300 ;------------------------------------------------------------
7E87 0E14      02310 HIGH    LD      C,20            ;CYCLES PER PULSE IN C
7E89 061C      02320 CONT    LD      B,28            ;TIMING DELAY IN B
7E8B 3E05      02330         LD      A,5             ;LOAD A WITH 00000101
7E8D D3FF      02340         OUT     (0FFH),A        ;LOAD LATCH WITH 0101
7E8F AF        02350         XOR     A               ;DELAY OF 2.225 MICROSEC
7E90 AF        02360         XOR     A               ;SAME AS ABOVE--FINE TUNE
7E91 10FE      02370 UP      DJNZ    UP              ;MAIN DELAY FOR HALF WAVE
7E93 061C      02380         LD      B,28            ;GET READY FOR NEXT HALF
7E95 3E06      02390         LD      A,6             ;LOAD A WITH 00000110
7E97 D3FF      02400         OUT     (0FFH),A        ;LOAD LATCH WITH 0110
```

```
7E99 AF       02410           XOR     A              ;FINE TUNING DELAY
7E9A AF       02420           XOR     A              ;DELAY
7E9B 10FE     02430 DOWN      DJNZ    DOWN           ;MAIN DOWN CYCLE DELAY
7E9D 0D       02440           DEC     C              ;COUNT ONE CYCLE
7E9E 20E9     02450           JR      NZ,CONT        ;IF NOT FINISHED, CONT
7EA0 C9       02460           RET                    ;IF DONE, RETURN
              02470 ;***
              02480 ;***
              02490 ;SUBROUTINE LOW--OUTPUTS A 9091 MICROSECOND PULSE OF A
              02500 ;2025 HZ TONE--ONE LOW BIT.
              02510 ;------------------------------------------------------
7EA1 0E12     02520 LOW       LD      C,18           ;18 CYCLES/PULSE
7EA3 061F     02530 CONTL     LD      B,31           ;DELAY FOR HALF CYCLE
7EA5 3E05     02540           LD      A,5            ;LOAD A WITH 00000101

7EA7 D3FF     02550           OUT     (0FFH),A       ;LOAD LATCH WITH 0101
7EA9 AF       02560           XOR     A              ;TIMING DELAY
7EAA AF       02570           XOR     A              ;TIMING DELAY
7EAB 10FE     02580 UPL       DJNZ    UPL            ;MAIN TIMING LOOP
7EAD 061F     02590           LD      B,31           ;FIX B FOR NEXT HALF
7EAF 3E06     02600           LD      A,6            ;LOAD A WITH 00000110
7EB1 D3FF     02610           OUT     (0FFH),A       ;LOAD LATCH
7EB3 AF       02620           XOR     A              ;TIMING DELAY
7EB4 AF       02630           XOR     A              ;TIMING DELAY
7EB5 10FE     02640 DOWNL     DJNZ    DOWNL          ;MAIN TIMING LOOP
7EB7 0D       02650           DEC     C              ;COUNT CYCLES
7EB8 20E9     02660           JR      NZ,CONTL       ;IF NOT DONE, CONTL
7EBA C9       02670           RET                    ;OTHERWISE, RETURN
              02680 ;***
              02690 ;***
              02700 ;SUBROUTINE OUTB2--OUTPUTS BYTE ONE BIT AT A TIME
              02710 ;ENTRY: (D)     CONTAINS BYTE TO BE PRINTED
              02720 ;EXIT:  (D)     UNCHANGED
              02730 ;-----------------------------------------------------
7EBB C5       02740 OUTB2     PUSH    BC             ;SAVE BC REGISTERS
7EBC CDA17E   02750           CALL    LOW            ;OUTPUT START BIT
7EBF CB0A     02760           RRC     D              ;GET BIT 0
```

```
7EC1 CD707E   02770        CALL   SELECT        ;SELECT PROPER TONE
7EC4 CB0A     02780        RRC    D             ;GET BIT 1
7EC6 CD707E   02790        CALL   SELECT
7EC9 CB0A     02800        RRC    D             ;GET BIT 2
7ECB CD707E   02810        CALL   SELECT
7ECE CB0A     02820        RRC    D             ;GET BIT 3
7ED0 CD707E   02830        CALL   SELECT
7ED3 CB0A     02840        RRC    D             ;GET BIT 4
7ED5 CD707E   02850        CALL   SELECT
7ED8 CB0A     02860        RRC    D             ;GET BIT 5
7EDA CD707E   02870        CALL   SELECT
7EDD CB0A     02880        RRC    D             ;GET BIT 6
7EDF CD707E   02890        CALL   SELECT
7EE2 CB0A     02900        RRC    D             ;CHECK PARITY
7EE4 EAEC7E   02910        JP     PE,EVEN       ;MARK PARITY ODD
7EE7 CDA17E   02920        CALL   LOW
7EEA 1803     02930        JR     STOPBI
7EEC CD877E   02940 EVEN   CALL   HIGH
7EEF CD877E   02950 STOPBI CALL   HIGH          ;FIRST STOP BIT
7EF2 CD877E   02960        CALL   HIGH          ;SECOND STOP BIT
7EF5 CD877E   02970        CALL   HIGH          ;EXTRA STOP BIT
7EF8 C1       02980        POP    BC
7EF9 C9       02990        RET
              03000 ;***
              03010 ;***
              03020 ;DATA BLOCK BEGINS AT THIS POINT
              03030 ;----------------------------------------------------
7EFA 47       03040 SREM   DEFB   71            ;SPACE REMAINING IN LINE
7EFB 00       03050 BLEN   DEFB   0             ;CHARACTERS IN BUFFER
00DC          03060 BUFF   DEFS   220           ;SAVE MAIN PART OF BUFFER
              03070 ;THE FOLLOWING IS AN INITIALIZATION ROUTINE.  IT SETS
              03080 ;VARIOUS MEMORY LOCATIONS TO THE PROPER VALUE.  AFTER
              03090 ;IT IS EXECUTED, IT BECOMES PART OF THE BUFFER AND IS
              03100 ;OVER WRITTEN.
              03110 ;--------------------------------------------------
7FD8 21AA7D   03120 FIRST  LD     HL,BEGIN      ;LD HL WITH START OF PGM
7FDB 222640   03130        LD     (16422),HL    ;PUT DRIVER ADDRESS AWAY
7FDE 211D7E   03140        LD     HL,PBUFF      ;START ADDRESS OF PBUFF
7FE1 228E40   03150        LD     (16526),HL    ;PUT IN "USR" POINTER
```

```
7FE4 3E47     03160        LD     A,71             ;LENGTH OF DESIRED LINES
7FE6 322A40   03170        LD     (16426),A        ;PUT LINE LENGTH AWAY
7FE9 C3191A   03180        JP     1A19H            ;JUMP TO "READY"
7FEC 00       03190 EPGM   DEFB   0
7FD8          03200        END    FIRST
00000 TOTAL ERRORS
BEGIN  7DAA 00890   03120
BLEN   7EFB 03050   01410 01480 01960
BUFF   7EFC 03060
CONT   7E89 02320   02450
CONTL  7EA3 02530   02660
DOWN   7E9B 02430   02430
DOWNL  7EB5 02640   02640
EPGM   7FEC 03190
EVEN   7EEC 02940   02910
FIRST  7FD8 03120   03200
GOHIGH 7E83 02240   02210
HIGH   7E87 02310   02080 02240 02940 02950 02960 02970
INCL1  7DEB 01230   01180
LD2    7E75 02080   02100
LDBUF  7E00 01410   00980 01100 01150 01220
LEADER 7E72 02060   01650
LOOP1  7DC7 01050
LOOP2  7DCE 01080   01120
LOW    7EA1 02520   02220 02750 02920
NEXTA  7DF7 01280   01130
NEXTB  7DFB 01290   00940 00990 01200 01270
NXTCK  7DBD 01000   00960
OUT1   7DD9 01140   01010 01040
OUTB2  7EB6 02740   01680 01830 01850
PB1    7E2E 01670   01920
PB2    7E65 01920   01740 01760 01780 01810
PB4    7E3D 01750   01720
PB5    7E62 01910   01890
PBUFF  7E1D 01570   01510 03140
SELECT 7E7D 02210   02770 02790 02810 02830 02850 02870 02890
SREM   7EFA 03040   01570 01940
STOPBI 7EEF 02950   02930
UP     7E91 02370   02370
UPL    7EAB 02580   02580
```



```
      STATEMENT
      ---------
  DATE.  12/28/79
  TO     SOFTSIDE PUBLICATIONS
         6 SOUTH STREET              TERMS   10 DAYS
         MILFORD, NH  03055
  ------------------------------------------------------
   DATE     DESCRIPTION        CHARGES   CREDITS   BALANCE
  ------------------------------------------------------
            CARRIED FORWARD                          $0.00
 08/04/79   MEMORY CHIPS       $100.00             $100.00
 08/12/79   DISKETTES          $500.00             $600.00
 08/29/79   GLASS FROG           $1.99             $601.99
 09/10/79   PAYMENT                      $200.00   $401.99
 09/21/79   PAYMENT                      $124.00   $277.99
            AMOUNT DUE                             $277.99
 --------------------------------------------------------
 AMOUNT OF BALANCE WHICH IS OVERDUE (PLEASE REMIT)   $277.99
```

SINK THE

BISMARK!

Now there's a true historical wargame for your home computer. Computer Bismarck accurately simulates the epic battle between the awesome German battleship and the British Home Fleet. Best of all, the computer program eliminates the drudgery of paper and pencil wargames — remembering all the rules and details while keeping track of the battle on a North Atlantic map on your video display.

Play the Computer

It maneuvers the Bismarck and Prinz Eugen so well that you'll have to command the British ships brilliantly to avoid losing your vital merchant convoys.

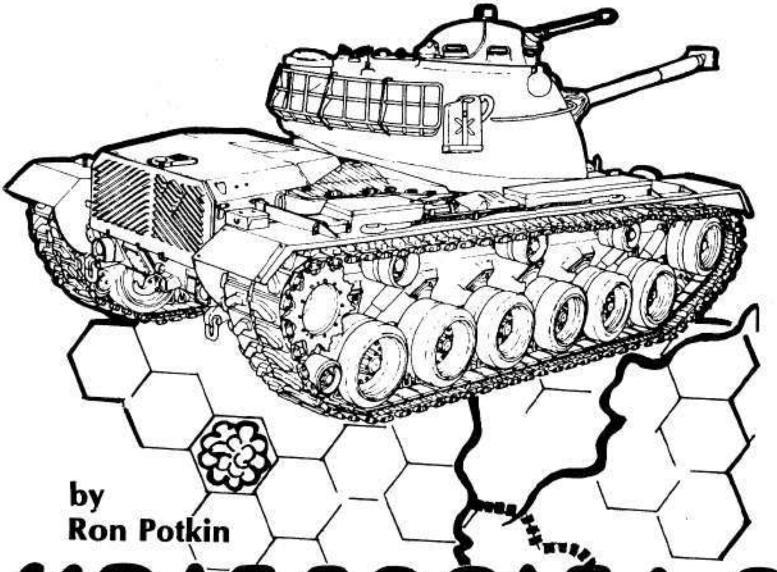
Play a Human

The two of you plot your strategies in grease pencil on an off-screen mapboard while the battle is fought on the video screen (monochrome or multi-color depending on your display capabilities). You deploy battleships, cruisers, carriers — each with unique and realistic operating parameters. You must deal with all the variables which challenge an actual battle commander: firepower and damage; shadowing ability (better in radar-equipped vessels); and visibility — which depends on weather, which varies with geography and time.

Level II, 16K TRS-80 cassette.................................... $49.95

The Software Exchange

6 SOUTH ST., MILFORD, NH 03055 . 673-5144



GARON'S GOODIES

by James Garon

BASIC TO ELECTRIC PENCIL CONVERSION PROGRAM

How would you like to be able to use the powerful editing features of the ELECTRIC PENCIL on your BASIC programs? You could, for example, change all PRINTs to LPRINTs with a few keystrokes — you could quickly locate a phrase in a long program — you would have an auto repeat on all keys — the list is endless....

Readers of Mr. Harvard C. Pennington's runaway best seller, "TRS-80 Disk and Other Mysteries", know that this is indeed possible. All that is required is to save the program as an ASCII file with the "/PCL" extension and (Super) zap the final byte from 0D to 00.

Sound like Heaven? It is! But there is a small catch. ELECTRIC PENCIL requires a space every 30 characters or so; otherwise it gets indigestion. Many of us "byte-misers" wouldn't put a space in a long line if our lives depended on it.

BPENCIL will put in the necessary spaces and place the 00 byte at the end for you. It will also create a PENCIL-readable file (NAME/PCL where "NAME" is the name of the original ASCII-saved BASIC program).

BPENCIL will search each line of the program in groups of thirty characters. If a space is found, BPENCIL moves its attention to the thirty characters following the space. If no space is found, BPENCIL looks for any of 7 special characters: :=,.,(or). Upon encountering one of these, BPENCIL inserts a space immediately following. In almost all cases, this will leave the syntax and the logic of the line intact (but check!).

In the rare event that none of the above special characters are found in a group of thirty characters, BPENCIL will plop in a space anyhow. Since this could cause problems later, all such lines are listed at the end of the conversion process.

You should also be aware that any down-arrows (line feeds) in the original program will be converted to the letter Z.

Enjoy!!!!

```
 10 CLEAR 1E3:DEFINT C-Z:DEFSTR A,F:DIM Z(99):N=30:Z=1
 20 CLS:PRINT"BASIC TO PENCIL CONVERSION --- BY JAMES GARON
 30 PRINT:INPUT"WHAT IS THE FILENAME OF THE BASIC PROGRAM";F
 40 OPEN"I",1,F:X=INSTR(F,"/"):IF X=0 THEN X=9
 50 F1=LEFT$(F,X-1)+"/PCL"
 60 LINEINPUT#1,A:Y=ASC(A)
 70 IF Y>57 THEN PRINT:PRINT F" IS NOT SAVED UNDER THE ASCII OPTI
ON ":END
 80 IF Y<48 THEN PRINT:PRINT F" IS NOT A BASIC PROGRAM. ":END
 90 PRINT@192,STRING$(64,143):PRINT@576,STRING$(64,188)
95 OPEN"O",2,F1
 100 L=LEN(A)-1:C=1
 110 PRINT@256,STRING$(4,255);:PRINT@256,A:GOSUB 600
 120 D=INSTR(MID$(A,C,N)," ")
 130 IF D>0 THEN C=C+D:IF C>L-N THEN 500 ELSE 120
 200 D=INSTR(MID$(A,C,N),":"):IF D=0 THEN 250
 210 C=C+D:GOSUB 700:GOTO 120
 250 D=INSTR(MID$(A,C,N),"="):IF D=0 ELSE 210
 260 D=INSTR(MID$(A,C,N),">"):IF D=0 ELSE 210
 270 D=INSTR(MID$(A,C,N),"<"):IF D=0 ELSE 210
 280 D=INSTR(MID$(A,C,N),","):IF D=0 ELSE 210
 300 D=INSTR(MID$(A,C,N),"("):IF D=0 ELSE 210
 400 D=INSTR(MID$(A,C,N),")"):IF D=0 ELSE 210
 410 IF C>L-N THEN 500
 450 Z(Z)=VAL(LEFT$(A,INSTR(A," "))):IF Z(Z)>Z(Z-1) THEN Z=Z+1
 460 D=N:GOTO 210
 500 PRINT#2,A:IF EOF(1) THEN PRINT#2,CHR$(0):CLOSE
    ELSE LINEINPUT#1,A:GOTO 100
 510 PRINT@640,"THE PENCIL FILE IS NOW SAVED UNDER THE NAME "F1
 520 IF Z=1 THEN END ELSE PRINT TAB(20)"***CAUTION***
 THE FOLLOWING LINE(S) MAY CONTAIN SPACES IN ILLEGAL PLACES:
 530 FOR I=1TOZ-1:PRINT Z(I);:NEXT:END
 600 L=L+1:PRINT@512,"LENGTH ="L;:IF L<255 THEN RETURN
    ELSE PRINT@640,"SORRY - THE LINE IS TOO LONG;
    PLEASE BREAK IT INTO TWO LINES AND TRY AGAIN.":END
 700 IF C>L THEN 500 ELSE C=C-1:A=LEFT$(A,C)+" "+RIGHT$(A,L-C):
    PRINT@256,A:GOSUB 600:C=C+9:RETURN
```

TSE

NOTE: ST-80 is a trademark of Lance Micklus, Inc.

by Lance Micklus

The ST-80 family of smart terminal packages

Now you can have the right terminal package for your communication needs! From ST-80 UC — the simple, pre-set, inexpensive terminal program designed for users of THE SOURCE, MICRONET, and FORUM 80 — to the power and sophistication of ST-80 III, The Software Exchange has the package for you.

All four programs include the ability to use an unmodified TRS-80 keyboard to produce RUB, ESC, and other control characters for time sharing, software control of the RS-232-C board, repeat key, bell, software support for the three most common upper/lower case hardware conversions, and line printer output.

ST-80 UC 4K Level II cassette, **$24.95**

This universal communicator program is an easy-to-use timesharing program for the beginner. Special features include preset parity, word length, and baud rate (regardless of switch settings on the RS-232-C board) for THE SOURCE, MICRONET, and FORUM 80, automatic testing of the RS-232-C board, and even spooling of prepared messages on tape directly into FORUM 80 using a basic program supplied as a line listing.

ST-80 4K Level II cassette **$49.95**

The original smart terminal program for the TRS-80, ST-80 lets you reprogram your RS-232-C board from the keyboard, and run at different baud rates. Does not have auto testing of RS-232-C or tape spooling.

ST-80 D 32K disk program, **$79.95**

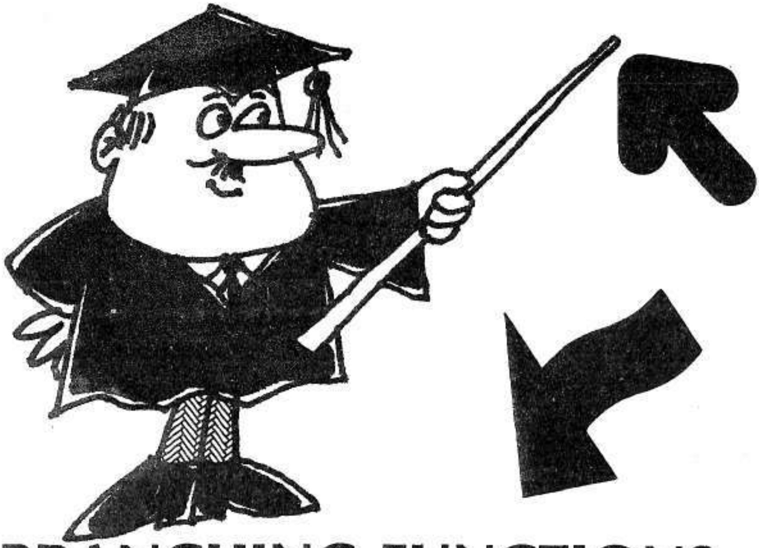
Special features include connection time clock, option of user-created translation tables for keyboard, gathering and pre-formatting data to be sent directly from disk to host computer, spooling of received files to disk or printer, editing of received files, and auto logon. If you use it with VTOS 3.1, you also get device driven I/O, job logging, and chaining.

ST-80 III 32K disk program, **$150.00**

ST-80 D with extra utility programs

The Software Exchange

6 South Street, Box 68, Milford, NH 03055 603-673-5144



BRANCHING FUNCTIONS IN APL80

by Phelps Gates

One of the aspects of APL which can be confusing at first is the way it handles branching: the right arrow (→), equivalent to GOTO in BASIC. A non-conditional branch is no problem — it works just like BASIC. For example:

```
)DEF INFINITELOOP
   1: 'PRESS BREAK TO STOP'
   2:  →1
```

Or you can use labels:

```
)DEF SQUAREROOT; X
   1: GETMORE: 'ENTER NUMBER'
   2: X ← q
   3: 'THE SQUARE ROOT OF';X;'IS';X*.5
   4: → GETMORE
```

Note the use of X as a local variable in this function: this allows us to call the function without affecting any value which we may have stored in a variable called X. The shifted q (for "quad") prints a prompt and waits for input (APL ⎕).

Conditional branching is a little trickier, since APL doesn't have operators which correspond directly to IF or THEN in BASIC (or FOR...NEXT). Of course, you don't have to use branching as much in APL as in BASIC. Since APL can operate directly on arrays, a single APL line can often do the work of a whole program in BASIC. For example:

```
(18)″.*18
```

does exactly the same thing as the following BASIC program:

```
10 FOR X = 1 TO 8
20 FOR Y = 1 TO 8
30 PRINT X ♠ Y;
40 NEXT Y
50 PRINT
60 NEXT X
```

But sometimes you do have to branch conditionally. Consider the following function, which generates Pascal's triangle of binomial coefficients:

```
)DEF TRIANGLE
   1: K ← -1
   2: ANOTHER: K ← K+1
   3: 1,(iK)!K
   4:  → ANOTHER
```

This will print Pascal's triangle, all right, but it won't stop...it just keeps printing lines until the numbers get too big and you get a DOMAIN ERROR. Is there any way to print only the first ten rows? We need to test K and loop back only if K is less than 10, and to do this, we have to know the rules for branching in APL. There are three cases:

1. The line number (or label) exists in the current function. Control simply passes to that line: this is the simplest case, and all the examples so far have been like this.

2. The line number doesn't exist in the function (→ 0 or → 999 or → -1 or just →). The function terminates: this is equivalent to RETURN in BASIC.

3. The expression to the right of the arrow is an empty vector (i0). No branch occurs: control just passes to the next line.

In the TRIANGLE function, for example, we could change line 4 to:

```
4:  → ANOTHER x K < 10
```

Now if K is less than 10, the logical expression K < 10 will have the value 1 (true), and the expression in line 4 will have the value ANOTHER x 1, which equals ANOTHER. On the other hand, if K is not less than 10, the expression K < 10 has the value 0 (false), and the function branches to ANOTHER x 0, which equals 0, and so the function terminates, by rule 2.

We could also write line 4 as:

```
4:  → ANOTHER x i K < 10
```

If K is less than 10, this branches to ANOTHER x i 1, and since i 1 equals 1, this is equivalent to →ANOTHER. If K is not less than 10, we get

```
ANOTHER x i 0
```

and since multiplying an empty vector by anything still gives an empty vector, this is equivalent to → i 0. No branch occurs (rule 3), but since there aren't any more lines in the function, execution terminates anyway.

This works fine mathematically, but it doesn't make your programs any easier to read, and it's a nuisance to have to figure all this out every time you write a function. One solution would be to define a function to figure it out for us. For example:

```
)DEF BRANCH ← LABEL IF CONDITION
   1: BRANCH ← LABEL x i CONDITION
```

Now we can write line 4 of the TRIANGLE function as

```
   4: → ANOTHER IF K< 10
```

which makes a little more sense. The function IF computes the proper destination for the branch, depending on the value of K. Note that we don't have to call it IF: you could call the function PROVIDED.THAT or SI or whatever. It's a dyadic function, requiring both a left operand (the label) and a right operand (the condition), and it returns a value (the appropriate destination).

You can also do it backward, and define a function called UNLESS:

```
)DEF BRANCH ← LABEL UNLESS CONDITION
   1: BRANCH ← LABEL x i n CONDITION
```

and rewrite the TRIANGLE function as:

```
   4: → ANOTHER UNLESS K > 9
```

Instead of using multiplication, you can define the IF function using the compression operator:

```
   1: BRANCH ← CONDITION/LABEL
```

This is actually the simplest (and fastest) way of doing it: 0/LABEL is the empty vector and 1/LABEL is identical with LABEL. Or you can use the Take operator (↑):

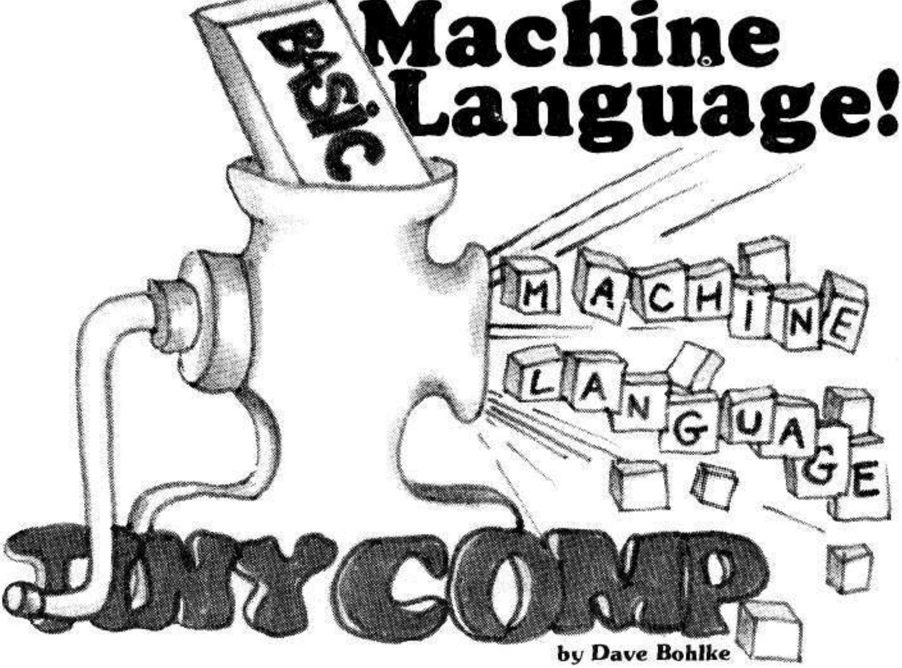
```
   1: BRANCH ← CONDITION ↑ ,LABEL
```

If the condition is false, this selects zero elements from LABEL (and therefore gets an empty vector); if the condition is true, it selects one element (and therefore gets just LABEL). Note that we have to use the ravel function (,) to convert LABEL from a scalar to a one-element vector, since Take won't work with scalars in APL80.

Here's another example of an APL80 function which makes use of the IF function for conditional branching: it prints a list of all prime numbers less than 100. It's actually possible to do this without branching, using a single outrageous line (can you figure out how?), but the solution using branching makes it a little clearer what's going on. It uses a technique known to the ancient Greeks called the "sieve of Eratosthenes", in which you start with a list of all integers in the range, and then throw out the ones which are evenly divisible by primes less than or equal to their square root. Eventually only primes are left. Note the use of the APL80 j function (Residue, also called Remainder or Modulo) as a test of divisibility:

```
)DEF PRIMES; DIVISOR; LIST; INDEX
   1: DIVISOR ← 2  3  5  7
   2: LIST ← 1 ↓  i 100
   3: INDEX ← 1
   4: NEXTTEST: LIST ← ((DIVISOR(INDEX)jLIST) > 0)/LIST
   5: INDEX ← INDEX+1
   6: → NEXTTEST IF INDEX <  5
   7: DIVISOR;LIST
```

The Lazy Man's Shortcut to Machine Language!



by Dave Bohlke

A BASIC **Compiler** in BASIC! Run your source program in BASIC, compile it into FAST Z-80 Code and execute the compiled version — all without reloading. 26 integer variables, GOTO, GOSUB, END, REM, RND, LET, +, *, /, IF, THEN, ELSE, <, =, >, INKEY$, CLS, PRINT@, CHR$, PEEK, POKE. Compiled programs may be saved via TAPEDISK.

Supplied with game program, "3D TIC TAC TOE", which uses all of the TINY COMP Statement set and is ready to compile.

Manual includes several sample programs as well as thorough documentation of the Compiler for those who like to know "how things work" and for those who might even wish to EXPAND on TINY COMP's capabilities.

Tape version: **$19.95**
Disk version: **$24.95**

The Software Exchange

6 South Street, Box 68, Milford, NH 03055 603-673-5144

FORM LETTER

by Richard Taylor

This program will allow you to use letters generated with the Electric Pencil in conjunction with mailing lists created the same way for form letters:

Instructions:

There must be at least one blank line between each name and address entry in the Pencil file.

MLIST/PCL is the default file name for the mailing list.

There must be a C/R after each line of the letter in the Pencil file. Program will check this and inform you of error.

LETTER/PCL is the default name for the letter file.

Please be sure to enter your name and address using upper and lower case to be consistent with your letter. Lowercase is entered by holding the shift key down while hitting the character.

Unless you are using a lowercase driver, lowercase will not be displayed.

```
 10 ' *** FORM LETTER PRINTER   ***
        BY RICHARD TAYLOR
        THIS PROGRAM USES
        FILES CREATED BY THE
  ELECTRIC PENCIL

60 CLEAR5000:DIM A$(120)
70 CLS
 80 ' ***  READ THE LETTER INTO MEMORY
 90 PRINT@388,"*** INSERT PENCIL FILE DISK ***";
 100 PRINT@512,"";:INPUT"PLEASE ENTER THE NAME OF THE PENCIL FILE
THAT THE LETTER IS STORED IN (C/R IF 'LETTER')";L$
 110 IFL$=""THENLE$="LETTER/PCL" ELSE LE$=L$+"/PCL"
120 ONERROR GOTO 960
130 OPEN"I",1,LE$
140 ONERROR GOTO 970
150 CLS
160 X=X+1
170 LINEINPUT#1,A$(X)
 180 ' ** ADJUST FOR DIFFERENT PRINTERS **
 190 IF A$(X)="" THEN A$(X)="   "
 200 PRINTA$(X)
 210 IFX>45 CLS:PRINT@512,"THIS LETTER WILL TAKE TWO PAGES  !!!

 HIT <ENTER> IF YOU WISH TO GO ON, OR TYPE 'ABORT'";:INPUTU$:IFU$
="ABORT" THEN CLOSE: END
 220 IF EOF(1) THEN GOTO 240
230 GOTO 160
```

```
240 CLOSE1:FORS=1TO900:NEXT:CLS:PRINT@512,"LETTER OK.  NOW STORED
 IN MEMORY";:FORI=1TO600:NEXT:CLS
250 PRINT@512,"":INPUT"PLEASE ENTER THE MAXIMUM NUMBER OF CHARAC
TERS PER LINE
THAT YOUR PRINTER CAN HANDLE  (C/R FOR 80)  ";M
260 IFM=0 THEN M=80
270 FORR=1TOX:IF LEN(A$(R))>M GOSUB1020
280 NEXTR
290 CLS:PRINT@512,"";:INPUT"PLEASE ENTER FILE NAME OF MAILING LI
ST
 (C/R IF FILE IS NAMED 'MLIST')";M$
300 IF M$="" THEN ML$="MLIST/PCL" ELSE ML$=M$+"/PCL"
310 ONERROR GOTO 980
320 OPEN"I",2,ML$
330 CLS:PRINT@384,"** FILE FOUND, PLEASE LEAVE DISK IN DRIVE **"
;
340 PRINT@512,"";:INPUT"SENDER'S NAME";NAM$
350 LINEINPUT"SENDER STREET ADDRESS ";ADD$
360 LINEINPUT"STATE, CITY ZIP ";STA$
370 LINEINPUT"DATE OF THE LETTER? ";DA$
380 CLS:PRINT@384,NAM$:PRINTADD$:PRINTSTA$:PRINTDA$:INPUT"IS THI
S CORRECT";Y$:IF LEFT$(Y$,1)="N" GOTO 340
390 ONERROR GOTO 0
400 CLS:PRINT@512,"";:INPUT"LETTERS ADDRESSED WITH LAST NAME (Y/
N)";N$
410 IFN$<>"Y" AND N$<>"N" GOTO 400
420 INPUT"DO YOU WISH SELECTIVE PRINTING";G$:G$=LEFT$(G$,1):IF G
$<>"Y" AND G$<>"N" GOTO 420
430 IFG$="Y" GOTO 470
440 INPUT"DO YOU WISH TO STOP BETWEEN LETTERS TO ADJUST PAPER (Y
/N)
    (A 'FORM FEED' WILL BE SENT TO PRINTER)";T$
450 IFT$<>"Y" AND T$<>"N" GOTO 440
460 CLS:IF T$="Y" THEN PRINT@512,"";:INPUT"HIT ENTER TO START LE
TTER";Z
470 '** GET NAME AND ADDRESS FROM 'MLIST/PCL'
480 ONERROR GOTO 990
490 B=0
```

```
500 B=B+1:IFB>7 GOTO 990
510 LINEINPUT#2,B$(B)
520 IF EOF(2) THEN CLS:PRINT@532,"*** END OF FILE ***":PRINT:PRI
NT:PRINT"HIT ENTER TO RERUN PROGRAM";:INPUTO:RUN
530 ONERROR GOTO 0
540 IFB$(B)="" GOSUB 560  :GOTO 460
550 GOTO 500
560 ' *** TEST FOR MORE THAN ONE BLANK LINE BETWEEN NAMES
570 IF B<2 GOTO 490
580 '*** FIND OUT IF IT IS A PERSON OR A COMPANY
590 MR$=LEFT$(B$(1),3):MISS$=LEFT$(B$(1),5):MS$=LEFT$(B$(1),4)
600 IFLEFT$(B$(1),1)<>"M" THEN C$="DEAR SIRS:":GOTO 740
610 IF MR$="MR " OR MR$="MR. " OR MR$="MR " OR MR$="MR. " OR MR$="
MS " OR MR$="MS " OR MR$="MS. " OR MR$="MS. " THEN TL$=MR$:GOTO650

620 IF MISS$="MISS " OR MISS$="MISS " THEN TL$=MISS$:GOTO650
630 IFMS$="MRS " OR MS$="MRS " OR MS$="MRS. " OR MS$="MRS. " THEN
TL$=MS$:GOTO650  ELSE C$="DEAR SIRS:":GOTO 740
640 '**  FIND FIRST AND LAST NAMES
650 I=INSTR(B$(1)," "):Y=INSTR(I+1,B$(1)," "):Q=INSTR(Y+1,B$(1),
" ")
660 ' ** ELIMINATE MIDDLE INITIAL **
670 IF Q<>0 THEN Y=Q
680 ONERROR GOTO 1000
690 F$=MID$(B$(1),I+1,Y-I-1)
700 LN$=RIGHT$(B$(1),LEN(B$(1))-Y)
710 ONERROR GOTO 0
720 IF N$="N" THEN C$="DEAR "+F$+"," ELSE C$="DEAR "+TL$+" "+LN$
+"."
730 ' ** CHECK TO SEE IF PRINTER IS ON **
740 IFPEEK(14312)=127 THEN CLS:PRINT@532," ** PRINTER NOT READY
**";:PRINT@662,"";:INPUT"HIT ENTER WHEN READY";O:GOTO740
750 ' ** PRINT LETTER **
760 IFG$="Y" THEN CLS:PRINT@256,"READY TO PRINT LETTER TO:":FORR
=1TOB:PRINTB$(R):NEXT:ELSE GOTO 800
770 PRINT@768,"HIT <ENTER> TO PRINT
OR ANY OTHER LETTER FOR NEXT NAME";
780 E$=INKEY$:IFE$=""GOTO 780
790 IF E$<>CHR$(13) GOTO 490
```

```
 800 CLS PRINT@384,"NOW PRINTING LETTER TO:":FORR=1TOB:PRINTB$(R)
:NEXT
810 LPRINTTAB(50);NAM$
820 LPRINTTAB(50);ADD$
830 LPRINTTAB(50);STA$
840 LPRINTTAB(50);DA$
850 LPRINTSTRING$(4,32)
860 FOR R=1 TO B
870 LPRINT B$(R):NEXTR
880 LPRINTSTRING$(2,32)
890 LPRINT C$
900 LPRINTCHR$(32)
 910 FOR R=1 TO X:IF R=46 THEN LPRINT CHR$(12)
920 LPRINT A$(R):NEXTR
930 LPRINTCHR$(12)
940 RETURN
 950 ' ***  ERROR TRAPPING ROUTINES ***
 960 CLS:L$="":PRINT@320,"  *** FILE NAME INCORRECT PLEASE RESTAT
E ***":RESUME100
 970 CLS:PRINT@512,"LETTER FORMAT INCORRECT CHECK LINE LENGTH


        AND RERUN PROGRAM ****";:END
 980 CLS:M$="":PRINT@320,"  *** FILE NAME INCORRECT PLEASE RESTAT
E ***";:RESUME290
 990 CLS:PRINT@521,"*** FILE FORMAT ERROR - PLEASE CHECK MLIST **
*
        ***       FILE AND RERUN THIS PROGRAM           ***":END
 1000 CLS:PRINT@256,"NAME DOES NOT CONFORM TO FORMAT  SKIPPING OV
ER":FORR=1TOB:PRINTB$(R):NEXTR:FORE=1TO1000:NEXTE:RESUME 490
 1010 ' ** SUBROUTINE FOR RETYPING LINES THAT ARE TOO LONG **
 1020 CLS:PRINT@256,"** LINE NUMBER";R;"IS";LEN(A$(R));"CHARACTER
S LONG"
 1030 PRINT"IT READS:":PRINTA$(R):INPUT"DO YOU WISH TO RETYPE (Y/
N)";V$:V$=LEFT$(V$,1):IF V$<>"Y" AND V$<>"N" GOTO 1030
 1040 IFV$="N" THEN RETURN
1050 ONERROR GOTO0
 1060 PRINT:PRINT"PLEASE ENTER CORRECTED LINE:":INPUT A$(R):IFLEN
(A$(R))>M THEN PRINT"STILL TOO LONG !!! ":GOTO1060
1070 RETURN
```

The new word processor from Radio Shack!

Scripsit

SCRIPSIT word processing system training program
Radio Shack TRS-80 MICRO COMPUTER SYSTEM
Scripsit
MINIDISK
Radio Shack TRS-80 Business Software
Scripsit Disk Word Processor

FEATURES:

● Lower case capability ● Total document formatting ● Error correction by deletion, insertion, overwriting and exchanging ● Automatic headers, footers and page numbers ● Global find, replace and delete ● Operator-defined blocks for hyphenation and editing

SCRIPSIT will produce letters or any text material. The documents can be stored on disk for revision and update.

REQUIREMENTS:

● 32K Level II TRS-80; one or more disk drives; line printer

If you've seen or used the Electric Pencil, you'll be delighted with SCRIPSIT's versatility and extra features. Rapid typists will appreciate the fact that SCRIPTSIT does NOT lose letters at the end of each line. Titles can be automatically centered. The Screen width can be changed to match your printer. Tabs can be set, etc...

If you are a BASIC programmer, you'll want to utilize SCRIPSIT's ability to work directly with BASIC programs.

SCRIPSIT comes on disk with thorough documentation and sample text files. Includes three audio cassette tapes which cover all aspects of this exciting Word Processing Package.

The Complete SCRIPSIT Package (with Binder, Manual, 2 Disks, 3 Cassettes, Instruction Summary Card and special press-on key labels) is **only $95.00 + $2.00 shipping.**

TSE *The Software Exchange*
6 SOUTH ST., MILFORD, NH 03055 (603) 673-5144

RANDOM ACCESSING TECHNIQUES DISK FILE STRUCTURES AND PROGRAMMING AIDS FOR THE TRS-80 PROGRAMMER

by Will Hagenbuch

As we all know from even a cursory glance at our Disk Operating System instruction manual, there are two methods for storing information on diskette — *sequential and random. Sequential* does not seem to pose a problem for most readers since it is simply a carry-over from the methods employed to write cassette tape files. However, the employment of random file techniques is another story.

The Disk Operating System manual lists a variety of advantages offered to the programmer who uses randomly accessed record files. Some of these advantages are not readily visible to the programmer; such as the space-saving features of a single Input/Output (I/O) buffer. What is apparent is the time savings achieved from the direct accessing of a desired record. After titillating the reader with the many time-saving virtues of random accessing, the manual leaves the reader with the promise that once you have set up the file structures, random I/O becomes quite simple; however, "this is the hard part — it takes a little thought!" (Amen.)

While there is no intent to take a "cheap shot" at the Authors of the Disk Operating System manual (in fact, it is an excellent condensation of a very involved subject), it seems that somewhere among the next ten and one-half pages many readers lose the bubble. Be it on the first page, or somewhere around the middle, many readers tend to adopt the attitude that "Oh well, I know how to write sequential files, I'll just stick with that until someone explains all of this to me." The problem with this laissez faire attitude is that seldom does one achieve his goal or satisfy his thirst for knowledge from among his peer group. After all, they probably have the same problems as you do!

To summarize the information provided by the Disk Operating System manual, we might say that it provides you with the basics for using random files. It tells you that you must "OPEN" a file with the access code "R" to specify random files, that you must assign a buffer number (1-15), and that you must assign a file name.

For the moment, let's stop right there. Trust me that, as we proceed through this article, you will find out that this information on how to "OPEN" a random file is really all you need to know! What we will be doing, further along in this article, is introducing you to File Manager-80, an effective alternative for disk I/O handling. However, before we get involved with File Manager-80, let's discuss some of the other aspects of systems development using disk files.

Random Accessing

Let's assume for the moment that you have, or will have, the ability to use random accessing techniques to fetch and write records from and to a file. It would naturally follow that you would need to know "which" record. This is quite easy if you have a listing of the file and it contains the number of each record; but, what if you don't have such a listing? Well, you might consider accessing every record on the file with a FOR...NEXT loop and inspecting the data content of the file to see if that is the one you wanted. However, this is no more efficient than a sequential file, is it?

The determination of "which" record is best done by the use of an Index. An Index might be described as a pointer to "which" record on your random access file contains the information that you want. There are two methods of creating such an Index; either by using the FOR...NEXT loop to read all of the records from your random file one time, or by creating and maintaining a separate "Index" file which is normally a sequential file. In either case, the file which is the basis of your Index will be read into a memory array which we will call your Index Table. Naturally, you must have included a dimensioned array in your program to accommodate this Index Table which must include, as a minimum, the Data Element which is to be your "key" for locating your record. This "key" may be either an alphanumeric or numeric element and you must dimension accordingly — and you will need a sufficient array size so that there is one bucket for every record in your file, including any file size expansion that you may have in the future!

As an example, let's say that you have a random file of customers. At the time each of these customers was placed in the file, a unique number was assigned to each customer. This Customer Number is the Data Element that will be used as the "key" for accessing the customer record. If the Customer Number is a part of the customer file it will need to be split off (using the FOR...NEXT loop) to create an Index Table each time a program requiring random access of the file is run. Similiarly, if the Customer Numbers had been maintained in a separate file they would also be required to be accessed each time the program is run, but this would appear to be the more prudent way since fewer bytes of data would need to be accessed in order to build the Index Table. In either case you would place the Customer Number into a one-element array that had been dimensioned to accept it.

Now that the Index Table of Customer Numbers has been established, it is a simple procedure to relate the Customer Number to the physical record location for that customer's record. By searching the table with the desired Customer Number and finding a match (equal comparison) on the Nth record, we know that the customer record is the Nth record in our random access file.

The searching of the Index Table is, however, an art unto itself. If we were to insure, in our program which creates customer records, that all Customer Numbers were assigned sequentially in ascending sequence, we could use a Binary Search to locate the desired Customer Number very quickly. A Binary Search is a table search which "bisects" the table entries. In other words, if we were to compare the desired Customer Number to the "middle" entry in the Index Table of customer numbers and that comparison told us that the desired number was less than the middle entry, we would have effectively reduced further searching by one-half. Obviously, the Customer Numbers between the middle and the top (last number) of the array will not match the desired number. So, let's look at the bottom half of the Index Table.

Our next comparison would be at the first Index Table entry (or the last entry if the key we are searching was "high" to the middle). If our key is "low" to the first entry (or "high" to the last entry) we know immediately that the number we seek is not on the table. Obviously, whenever an "equal" condition occurs, we have found our number and, consequently, our pointer for random accessing of the customer file.

If the second comparison (the one to the bottom of the Index Table) was "high", we know that our key number ranges somewhere between the first and middle numbers on the index table. Therefore, we would compute the middle location between the first and middle Index Table entries and do the third comparison against that entry. Again, if unmatched we would continue the bisecting until the number was either found by an equal comparison or determined to not be on the Index Table. The "not found" condition is determined when no more bisecting is possible and an equal condition has not been found.

The following code is an example of a Binary Search routine included as a program sub-routine:

```
 20 DIM AR(500) 'Dimension statement for Index Table array
 '
 '
 '
500 'Binary Search Sub-routine
          Enter with search argument in variable "XA"
510 XL=1 'Set Low Limit'
520 XH=n 'Set High Limit to number of total table entries
530 XM=INT((XH+XL)/2) 'Compute Middle Entry number
540 IF XM=XH OR XM=XL THEN XM=0 : RETURN 'Not Found
550 IF XA    AR(XM) THEN XL=XM : GOTO 530 'Bisect
560 IF XA    AR(XM) THEN XH=XM : GOTO 530 'Bisect
570 RETURN 'Found! Exit with record pointer in "XM"
```

However, if your Index Table was not in ascending sequence, you could not use the Binary Search and would be relegated to the slower method — the Sequential Search.

In the Sequential Search, we must look at every entry in our Index Table; or at least until we find our match. We do this with the FOR...NEXT loop routine. The fastest method of performing this search is to break out when (and if) a match is found. If you are going to employ the "break-out" search loop, be sure that it is not a nested FOR...NEXT loop (inside of another FOR...NEXT loop) since you will leave an "unsatisfied NEXT" and that's a "no-no". An example of the "break-out" Sequential Search is as follows:

```
 20 DIM AR(500) 'Dimension array for Index Table
 '
 '
 '
500 'Sequential Search
       Enter with search argument in variable "XA"
       XH=Number of Index Table entries
510 FOR I=1 TO XH
520 IF XA=AR(I) THEN RETURN 'Found! I=Pointer to record in file.
530 NEXT
540 I=0 : RETURN 'Set I=0 to indicate not found.
```

Of course, in either case, your program will have to handle the situation of a return with a "not found" condition as indicated by a zero value in the Index Table pointer; "XM" in the first example, "I" in the second.

Optional File Structures

To get the maximum use out of your disk storage space, you should have a working knowledge of how you might organize files to best serve your needs. It often happens that if you are limiting yourself to a Fixed Format file structure, you might need to create several files, and consequently require several buffer allocations, when one Multi-Format file would do the job.

A Multi-Format file, for the purpose of this article, is a file that contains more than one format of records, or records which have more that one format. As an example, you may have reason to require that a file contains a "header" record of one format and some variable number of "trailer" records in another format. We will be using the term "header" to define any format which is the first of a group of related records; the term "trailer" will refer to the remainder of the records in that group. In this example, we will refer to the structure as a "Dual Format" file.

STRUCTURE A. — The "Dual Format" file

Header	Batch Number	Date of Batch	Keyed by:
Trailer 1	Date	Customer Number	Amount
Trailer 2	Date	Customer Number	Amount
⋮			
Trailer n	Date	Customer Number	Amount

In this example, we see a typical application of transaction entry employing the "Dual Format" file structure concept. The first record on the file contains such one-time information as the Transaction Batch Number, Date of the Batch, and Identification of the Operator who entered the information. This format appears only one time on the file.

The second format is that of the transactions which make up the batch. One record is used for each transaction and the number of this transaction format that may be placed on the file is limited only by the physical size of your recording media (disk or tape).

The second example of Multi-Formatted records will be called the "Variable Format". In this type of structure, we use two different formats in the same record; and, of course, the number of records that can be stored in a file is limited only by your physical storage capacity.

In the "Variable Format" structure, the number of trailers for each header must be specified. Because both the header and the specified number of trailers must be contained in a single sector of disk (255 or 256 bytes, depending on the DOS you are using), the aggregate size (bytes) of the header plus the number of specified trailers cannot exceed the physical restriction placed upon you by your DOS.

Let's take a look at this structure:

STRUCTURE B. — The "Variable Format" record

Header	Name	Address	City State	Zip
Trailer 1	Date	Invoice Number	Amount	
Trailer 2	Date	Invoice Number	Amount	
Trailer 3	Date	Invoice Number	Amount	

In this example, we see a typical application of transaction detail (or trailer records) applied to a customer record (or header). In this case, we have established a maximum of three (3) detail records per header record. It could have been more or less depending upon our requirements — as long as the aggregate size of the header and the maximum number of trailers for that header does not exceed the physical block size (255 or 256, depending on the DOS we use). Note that this file structure is called "Variable Format" because the trailers may or may not contain information. This does not mean that unused trailers do not take up space — they do. But, whether or not they actually contain information is a matter for your program to determine.

It might be noted, also, that in order to optimize file storage space, if the aggregate records size is less than one-half of the block size (255 or 256) the block may contain two (or more) "Variable Format" structure records. However, this "blocking" of records is only rhetorical since, if you are using File Manager-80, this is an automatic feature.

Again, if you are ready to cash it all in and go back to your cassette tape, do not despair! Help is on the way as we shall see further along in this article. First, however, let's discuss another aspect of data processing systems development — one that is often neglected but extremely important if your system is to survive the test of time!

System Documentation

Take it from one who has been doing this data processing thing for the past 20 years, the common denominator among systems, without fail, is that they will require change. In order that changes to a program can be made without chaos, it is imperative that the files, upon which the programs are based, be fully documented.

Today, you know perfectly well what you mean when you call a Data Element "CUT-OFF DATE"; but, will you remember next week? — How about next year? Now that you remember that "CUT-OFF DATE" means the date on which you are scheduled for the barber shop, I'll wager that you can't remember the variable name that you assigned to it in your program!

If these typical problems, normally incurred in program maintenance, do not ring a bell, well then:

a. You maintain outstanding documentation; or,

b. You have one fantastic memory; or,

c. You haven't written more than about one program — but you probably will or you would not have read this far!

As a minimum, "good" program documentation consists of a reference to all Data Elements (fields of information) used for files structures and a Record Layout (the arrangement of these Data Elements) for every file. In addition to the explicit definitions of any vague or ambiguous Data Element names, each Data Element should be cross-referenced to the variable name assigned to it in the program.

Proper program documentation consists of these products, whether you are programming for yourself or for someone else; and, whether you are using a Micro-Computer or an IBM 370:

a. A catalog of all Data Elements used in data files.

b. A Record Layout for each record file used.

c. A "snapshot" of each video screen display.

d. A sample of each printer output product.

e. A narrative description of the program operation.

Products c. and d. are easy. Most Operating Systems that are in general use today provide the facility to automatically print the video screen display and your text data print-outs will suffice for sample reports. The other products pose more of a time problem and are the ones that generally get pushed aside until later — whenever "later" is!

With your indulgence, we will be providing some viable alternatives to completing documentation products a. and b. as a by-product of FM-80 operations.

File Manager-80

This article has, thus far, posed many problems and few solutions. Well, from here on out, it is solution time — and the solution is File Manager-80.

If your Disk Operating System manual left you cold, I am sure that this article has been of little help. Early on, you were advised not to despair. The reason is that File Manager-80 can make all of those file structuring and random accessing problems go away because it writes your file I/O instructions for you! And, as a by-product, *will* produce a *Dictionary* of Data Elements and the Record Layouts that are so vital to your system support documentation.

We have previously discussed several file structures (Dual Format and Variable Format) that might be employed when a plain vanilla Fixed Format record is not suitable. File Manager-80 will accommodate all three of these file structures. In addition, your worries (if you have any) about optimum record blocking and those pesky little algorithms you need to find sub-records are taken care of automatically by File Manager-80.

Of course, File Manager-80 will not do everything. You must still "OPEN" and "CLOSE" your files at the proper time. You still have to determine "which" record you want by using one of the indexing methods previously described. And, you still have to pour your own coffee. But it will do almost everything else.

How File Manager-80 Works

As we all know, or are soon to find out, the first thing you need to consider in the development of a program (or system) is what problem you are attempting to solve. (Programmers, including Ish, can be characterized as persons with solutions who run around looking for a problem.) After identifying the problem, it is our job to reduce that problem into meaningful pieces of information (we'll call them Data Elements). These Data Elements consist of Input (information that we start with), Output (information that we end with), and perhaps Intermediate (information that needs to be massaged in between Input and Output).

The second logical step is the arrangement of these Data Elements into logical Record Layouts which will serve our Input, Intermediate, and Output requirements. Naturally, we strive to determine all of the required Data Elements at the start. However, we sometimes miss one or two and this is where File Manager-80 really does its thing. It will allow graceful recovery from difficult situations such as adding a couple of new Data Elements to an existing file structure.

Once our Data Elements are defined into Record Layouts (or at least our first cut at them), we will assign an arbitrary value to each different Data Element. Any value from 1 to 999 may be used but you will soon acquire a "pet" (no pun intended) method of assignment. We are now ready to use File Manager-80.

We begin by entering each of the Data Elements for our proposed system according to the prompts offered by the program. You will be prompted for the number you assigned to the Data Element (hereinafter called the Data Element Number, or DEN), the Name that you want to call the Data Element, and a single character (I, S, D, or $) to identify the attributes (Integer, Single- or Double Precision, or $tring) of the Data Element. Optionally, you may also enter any comments you feel necessary in the future to refresh your memory as to the intent or content of the Data Element. This information serves as the data base for the printed Data Element Dictionary; the Record Layouts; and, as we shall see later, the creation of the actual I/O instructions.

After all Data Elements have been cataloged in the data base, we may use File Manager-80 to prepare the individual Record Layouts for each of the file structures that we plan to use. This requires that you enter the File Name that is to be used in the creation of the data file to which the Record Layout applies, and the following file attributes:

a. Is it to be a Disk or Tape storage file?

b. Will it be a Sequential or Random Access file?

c. Is is to be a Fixed Format, Dual Format, or Variable Format file?

You will then be prompted to supply all of the Data Elements which are to be contained in the Record Layout. The prompting will be based on the type of record you specified. For example, Fixed Format records, Dual Format records consist of Format #1 (header record) and Format #2 (trailer record), and a Variable Format record contains both header and trailer formats as well as a specification of the number of trailer records that are in each logical record (remember that a Variable Format has a definite number of trailers?). Whatever the type of record, you need only supply the DEN (Data Element Number) and the variable name you wish to use for the Data Element in your program. You may then print the Record Layout whenever you wish.

Once you have catalogued your Data Elements and created your Record Layouts, File Manager-80 will create the actual I/O instructions required by your program to read (get) or write (put) records from or to Disk or Tape files. If your record size permits blocking (multiple records per disk sector), File Manager-80 will do this automatically and will create those pesky sub-record computation algorithms and imbed them in your coded instructions along with the necessary "FIELD" statements.

When I/O instructions are to be created, File Manager-80 will prompt you to provide the file buffer number you wish to assign to the file, the program line number on which you wish to instructions to begin, and the line number increment you wish to use (default=10).

The instructions will then be created and stored so that you can later "MERGE" them with your program. In addition, File Manager-80 will prepare a Cross-Reference Listing of the line numbers to be used to access the records from any file you have catalogued. As we shall soon see, this Cross-Reference Listing is invaluable when you are writing your mainline code for a program.

Working With File Manager-80

File Manager-80 has supplied you with several tools which you now use in program development and will retain for permanent documentation of your programs. The Data Element Dictionary; the Record Layouts; and, of course, the actual I/O instruction code and Cross-Reference Listing will materially assist in program development. However, you must know how to use them.

As we have said previously, File Manager-80 will not "OPEN" or "CLOSE" files — you must do that. It will, however, perform all I/O functions for you once you have OPENed the file. File handling for sequential files is easy, you simply perform a "GOSUB" to the appropriate line number as shown on your Cross Reference Listing.

Random files are a bit more work. Before you can access a record from a random file you must know the physical record number of the record you want. It is suggested that one of the two indexing methods described earlier be used. You will note that both methods provide an exit with the "found" record number in a variable (XM" in the first example, and "I" in the second). Just remember that if the returning value of the variable you might use is zero, you have not found the record pointer.

To access (read or write) the proper record, you simply set the variable "UR" to the value of the "found" record pointer (either UR=XM or UR=I in the examples) and do a "GOSUB" to the line number shown on the Cross Reference Listing. Voila! You have either read or written your record to or from the variable locations you specified when creating your Record Layouts! Isn't that nice?

Please note that the variable "UR" was specified for File Manager-80's use. In fact, variables "UR", "UT", "UP", "US", "U$", and "US$( )" are used by the I/O instructions created by File Manager-80 and it would be most prudent to avoid the use of these variables in your program, except when "UR" or "UT" are specifically called for.

When you are working with the Variable Format record type (a header followed by some specified number of trailers), you will use the variable "UT" to identify which trailer record you wish to access. It works like this: You access the header record in the same manner described above, by setting variable "UR" and doing a "GOSUB" to the appropriate line number. This will, of course, bring in all of the associated trailer records to the file buffer area because they are part of the logical record. You will now have the header record setting in the variables you have specified on your Record Layout.

To access the trailer records (move them to your variable locations), you simply set "UT" to the physical number of the trailer you want (1, 2, or 3 if you have specified three trailers) and do a "GOSUB" to the appropriate line number from your Cross Reference Listing. One of the best ways to handle trailers is to set up a FOR...NEXT loop after you have read the record into the file buffer (with "UR") which will bring all of the trailers into an array for whatever processing is necessary. Assuming that we might have a trailer record with one variable ("A"), the routine might look something like this:

```
500 UR=nn 'set variable UR to pointer number
510 GOSUB nnnn 'Get header record
520 FOR UT=1 TO 6 'Assumes 6 trailers per record
530 GOSUB nnnn 'Get trailer record
540 A(UT)=A 'Move data to array
550 NEXT UT 'Loop for all trailer records
```

So this is what File Manager-80 can do for you. As we said earlier, if your Disk Operating System manual has left you cold as far as random accessing is concerned, you might find that File Manager-80 is the solution. Or, even if you have mastered file accessing, File Manager-80 may appeal to you as a time-saving programming tool.

Postscript

In this article we have attempted to do several things not the least of which was to make you cognizant of the versatility and usability of File Manager-80 as a programming aid. In addition, we have:

- attempted to provide an amplification of available information on Disk File Handling from what is currently available in our Disk Operating System manual;
- covered, albeit cursorily, the accessing of random file records by a couple of techniques;
- provided a couple of alternatives to the plain vanilla Fixed Format file structure;
- presented a viable alternative to the "ho-hum" of writing file I/O instructions and program documentation.

LEARNING LEVEL II
David A. Lien

ALL PURPOSE PRINT ROUTINE

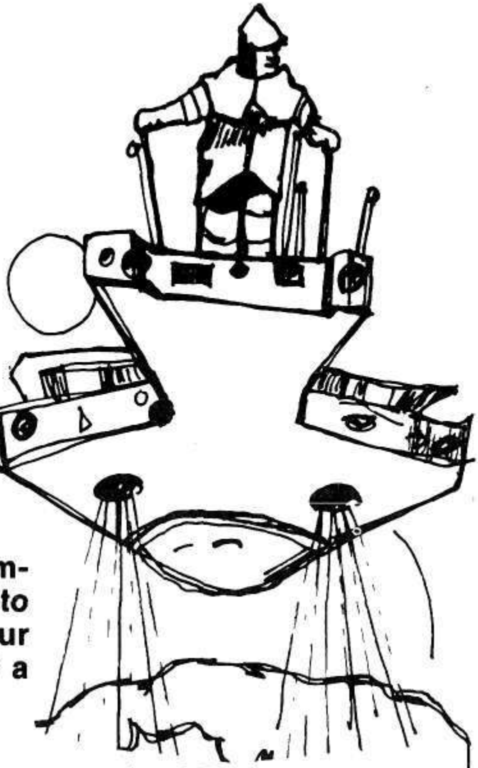
by C. E. Laidlaw

Here is a handy routine to control the output of your printer. Just follow the instructions to set the number of characters per line, lines per page, spaces in the left margin, and lines of text on each page. In addition, you can program a line feed after each carriage return if your printer needs one, print a tear line between each page, and double space.

```
100 CLS:PRINT
"                           = > APPR 1.1 < =

                           BY C. E. LAIDLAW
110 PRINT"
                       ALL PURPOSE PRINT ROUTINE

FOR USE WITH THE TRS-80 AND PARALLEL INTERFACED PRINTERS SUCH AS
THE CENTRONICS MODEL 730 (RADIO SHACK LINE PRINTER II).
120 CLEAR 100:DEFINT C-Z:DEFSTR A,B:BK=STRING$(50," ")
130 CM=80      ' MAX CHARACTERS PER LINE (INCLUDING MARGIN)
140 CS=10      ' NORMAL MARGIN - SPACES
150 CL=64      ' NORMAL LINE LENGTH (LESS MARGIN) - CHARACTERS
160 PM=100     ' MAX PAGE LENGTH - LINES PER PAGE
170 PL=66      ' NORMAL PAGE LENGTH - LINES PER PAGE
180 PT=60      ' NORMAL TEXT LENGTH - LINES PER PAGE
190 LF=0       ' LINEFEED AFTER C/R FLAG
200 DS=0       ' DOUBLE SPACE AFTER C/R FLAG
210 TL=0       ' TEAR LINE FLAG
```

```
 220 PRINT:PRINT"PROVIDES: 1  VARIABLE PAGE AND TEXT LENGTH (";PM
;"LINES/PAGE MAX)"; PRINTTAB(10)"2  VARIABLE MARGIN AND LINE LEN
GTH (";CM;"CHAR/LINE MAX)";
230 PRINTTAB(10)"3. TEAR LINE BETWEEN PAGES WHEN USING ROLL PAPE
R":PRINTTAB(10)"4. PRINTS PROPER  ^  VICE SQUARE BRACKET
 240 PRINT@896,"ENTER SYSTEM SIZE (1=16K, 2=32K, 3=48K) ";:INPUT
SS:IF SS<1 OR SS>3 PRINT@896,BK;:GOTO 240
 250 SS=SS+1:MM!=SS*16*1024-1
 260 MS!=PEEK(16561)+256*PEEK(16562):IF MS!<=MM!-188 THEN 290
 270 CLS:PRINT@256,"
S Y S T E M   N O T   P R O P E R L Y   I N I T I A L I Z E D

  MEMORY SIZE MUST BE SET AT LEAST 188 BYTES LESS THAN MAXIMUM

       START AGAIN AND RESERVE MEMORY AT";MM!-187;"OR LOWER
 280 IF PEEK(16396)-201 THEN CMD"S" ELSE GOTO 730
 290 CLS:PRINT"NORMAL SET-UP: ";PL;"LINES/PAGE        ";PT;"LINES
 TEXT/PAGE":PRINTTAB(15) CS;"SPACE MARGIN      ";CL;"CHARACTERS/
LINE
 300 PRINT@214,"NORMAL SET-UP (Y/N) "; GOSUB 600:IF TE=2 PRINT@19
2,BK;:GOTO 300 ELSE IF TE=1 THEN 360
 310 PRINT@326,"NOTE: TEXT LENGTH < PAGE LENGTH < =";PM;"LINES/PA
GE",:PRINT@396,"MARGIN + LINE LENGTH < =";CM;"CHARACTERS/LINE
 320 PRINT@536,"PAGE LENGTH ";:INPUT PL:IF PL>PM PRINT@512,BK;:GO
TO 320
 330 PRINT@664,"TEXT LENGTH ";:INPUT PT:IF PT>=PL PRINT@640,BK:GO
TO 330
 340 PRINT@790,"MARGIN LENGTH ";:INPUT CS
 350 PRINT@920,"LINE LENGTH ";:INPUT CL:IF CL+CS>CM PRINT@896,BK;
:PRINT@768,BK;:GOTO 340
 360 CLS:PRINT@20,"LINEFEED AFTER C/R (Y/N) ";:GOSUB 600:IF TE=2
THEN 360 ELSE LF=TE
 370 PRINT@148,"DOUBLE SPACE AFTER C/R (Y/N) ";:GOSUB 600:IF TE=2
 PRINT@128,BK;:GOTO 370 ELSE DS=TE
 380 PRINT@276,"TEAR LINE (Y/N) ";:GOSUB 600:IF TE=2 PRINT@256,BK
;:GOTO 380 ELSE TL=TE
390 PRINT@406,"I N I T I A L I Z I N G"
 400 MB=INT((MS!+1)/256):LB=MS!+1-256*MB
 410 IF MS!>32767 THEN MS=MS!-65536 ELSE MS=MS!
 420 FOR I=MS+1 TO MS+188:GOSUB 700:NEXT
```

```
430 ' SET UP LINE PRINTER CONTROL BLOCK
440 POKE 16422,LB+3:POKE 16423,MR:POKE 16424,PL:POKE 16425,0:POK
E 16426,PT
450 ' SET MARGIN AND LINE LENGTH
460 POKE MS+1,CS:POKE MS+2,CL+1
470 IF LF=0 THEN FOR I=0 TO 2:POKE MS+130+I,0:NEXT
480 IF DS=0 THEN FOR I=0 TO 8:POKE MS+112+I,0:NEXT
490 IF TL=0 THEN FOR I=0 TO 13:POKE MS+52+I,0:NEXT
500 PRINT@515,"BE SURE PRINTER IS TURNED ON AND READY - THEN PRE
SS ENTER ";:INPUT A
510 IF TL=0 THEN 560
520 IT$="SQR!%@SQR^SQRTRONNOT?<D+RESTORE.OCINKEY$"
530 J=VARPTR(IT$):K1=PEEK(J+1):K2=PEEK(J+2):J1=K1+256*K2
540 IF PEEK(16396)-201 DEFUSR0=J1 ELSE POKE 16526,K1:POKE 16527,
K2
550 FOR I=J1 TO J1+17:GOSUB 700:NEXT:J=USR(J)
560 PRINT@642,"THE ALL PURPOSE PRINT ROUTINE IS INITIALIZED AND
READY TO GO":GOTO 730
600 INPUT A:TE=0:IF A<>"Y" AND A<>"N" THEN TE=2:RETURN
610 IF A="Y" THEN TE=1
620 RETURN
700 READ D
710 IF D<0 THEN D!=MS!-D:D1=INT(D!/256):D2=D!-D1*256:POKE I,D2:P
OKE I+1,D1:I=I+1:GOTO 720 ELSE POKE I,D
720 RETURN
730 PRINT:PRINT:PRINT:END
1000 DATA 0,0,1,243,121,254,10,40,74,254,11,40,4,254,12,32,87
1010 DATA 221,126,5,61,221,150,4,40,6,71,205,-89,16,251,221
1020 DATA 126,3,  221,150,5,79,203,57,145,71,205,-89,16,251,205
1030 DATA -80,62,80,71,62,45,205,-178,16,251,205,-80,12,65,205
1040 DATA -89,16,251,205,-80,221,54,4,0,201,62,13,24,94,205
1050 DATA -89,24,4,62,10,24,85,221,52,4,221,126,5,221,150,4,40
1060 DATA 185,201,254,13,32,46,205,-125,245,221,126,4,183,196
1070 DATA -84,241,205,-135,201,62,13,205,-178,205,-89,24,214,33
1080 DATA -2,70,35,112,58,-1,183,200,71,62,32,205,-178,16,251
1090 DATA 201,254,91,32,2,62,94,33,-3,70,16,10,245,205,-125,205
1100 DATA -135,241,24,234,112,245,205,209,5,32,251,241,50,232
1110 DATA 55,201
1120 DATA 221,33,37,64,221,126,3,221,150,5,203,63,60,79,205
1130 DATA -49,201
```



GHOST TOWN

by Scott Adams

Scott's long-awaited 9th adventure is here! Thirteen treasures are hidden somewhere in the GHOST TOWN. You'll need more resourcefulness than ever to find them before sunset. Be ready for a town full of surprises and some REAL ghosts! Bring along a sense of humor, but be alert for danger as well. If you live so long, you MAY learn the answer to the tantalizing question: "What goes on in the saloon after dark?"

Level II, 16K Cassette $14.95

TSE The Software Exchange
6 SOUTH ST., MILFORD, N.H. 03055 (603) 673-5144

Outstanding Utilities from RACET Computes

INFINITE BASIC

Adds more than 70 commands to BASIC that can be merged in any combination to make efficient use of memory. Includes matrix read, inverse, transpose, identity, simultaneous equations, scalar, vector, and multidimensional array arithmetic, dynamically reshape, expand, and delete arrays, change arrays in mid program, read and write arrays on tape, copy elements, zero and move arrays. String functions include left and right justify, truncate, rotate, text justification, string centering, delete and insert substring, pack string, convert upper and lower case, translate characters, reverse strings, verify function, test number of occurences, masked string searches, encrypt and decrypt string, compress and uncompress string characters. High speed sort routines for strings and arrays, including multikey sorts, are also part of this package.

For business users, an add on package includes multiple precision packed decimal arithmetic, with up to 127 digits of accuracy, binary search of sorted arrays, insert elements in sorted arrays, automatic page headings, footings, and pagination including forced end of page, and automatic hash for record retrieval. **Infinite BASIC $49.95. Infinite business (add on) $29.95**

COMPROC COMMAND PROCESSOR

Chain multiple steps in disk BASIC upon power up, relocatable key debounce, allows pauses for data entry at specified steps during execution (on cassette for disk systems only) for **$19.95.**

DOSORT

BASIC control program with high speed machine language sort for disk users. Merge and sort files on more than one disk if you have 2 or more drives. Self prompting with manual. Specify 32K or 48K version **$34.95.**

GSF GENERALIZED SUBROUTINE FACILITY

18 machine language subroutines with easy access for BASIC users. Sort 1000 element arrays in 9 seconds, read and write arrays to tape, compress and uncompress data, move arrays in memory, duplicate memory, fast horizontal and vertical lines, 5 routines for screen control. Specify 16K, 32K, or 48K version, for **$24.95.**

REMODEL PROLOAD

BASIC program utility allows you to renumber portions of a program, move portions from one location to another, delete, merge, save and verify combined and changed programs, and create your own library of programs, subroutines, and even data statements. Works on tape or disk systems. Two programs on one tape. Specify 16K, 32K, or 48K (unspecified orders receive 16K) tape **$34.95**

TIMSER

Time series analysis program fits data to 9 different 1st, 2nd, and 3rd order curves. Goodness of fit data, tables, projected data, confidence limits, curve fit, variance, correction factor, seasonal and cyclical variations, inflation corrections are all implemented. Detailed user manual with illustrated examples. Order tape at **$14.95.**

Y-YBAR

Optical system design program allows manipulation of ray heights at lens surfaces using Y-Y Bar diagram method. With documentation on tape for **$14.95.**

DISK SORT/MERGE

DSM IS POWERFUL!!!!!!

- Sorts large multiple diskette files on a minimum two drive Mod-I disk system.
- All records are physically rearranged — no key files are required.
- Sorts random files created by BASIC, including files containing sub-records spanning sectors.
- Sorts on one or more fields in ascending or descending order. Fields may be character, binary integer, or floating point.
- The sorted output file may optionally have fields deleted, rearranged, or padded.
- Sort commands can be saved for reuse in production applications.
- Single sort, merge, or mixed sort/merge operations may be performed in a single DSM application.
- Sorted output may be written to a new file, or replace the original input file.

DSM IS FAST!!!!

DSM is written entirely in machine language for fast sorting. **$75.00**

TSE The Software Exchange
6 South Street, Box 68, Milford, NH 03055 603-673-5144

LANGUAGE SALE:

FRENCH	GERMAN	SPANISH
"BONJOUR"	"GUT MORGAN"	"BUENAS DIAS"

LANGUAGE TEACHER SERIES FROM ACORN SOFTWARE

These programs provide a whole disk full of vocabulary and conjugation drills to be used in conjunction with a course in the language. We have decided not to offer educational software, so while supply lasts, you can have them for **$15.95 each**. Comes on diskette, requires 32K memory.

TSE The Software Exchange
6 SOUTH ST., MILFORD, NH 03055 673-6144

NEW NEWDOS COMMAND

by Carl William MacKey

While going through the Level II manual (as I do at least once a month), I found a command that might be useful to other programmers who use Disk BASIC. The command is the "LOC" command. It is documented in NEWDOS+ but not in the TRSDOS 2.3 manual. After a call to Fort Worth, Texas to try to find out what it was, I was told that it is used ONLY in the Model II; I tried it out anyway, and here is what I've found out:

(1) In DOS 2.2 and 2.3 the command generates an error.

(2) In NEWDOS the command is active, but not too well documented. I discovered that its format is LOC(X) where X is equal to the buffer in use (from 1 to 15) for Random files. The LOC command will count the number of times the buffer has been used by GET or PUT statements. So by using LOC you are able to limit the number of times the buffer is used at one time.

LOC(X) will return a value from 1 to 32767.

Below is a small program I used to check the command:

```
   5 CLEAR 2000
  10 OPEN "R",1,"TESTDATA:0"
  20 GOSUB 1000
  30 X=1
  40 LINEINPUT A$
  50 LSET T$=A$
  60 IF LOC(1)=3 THEN PRINT "BUFFER #1 ACCESSED"; LOC(1); "TIMES":
       CLOSE:GOTO 200
  70 PUT 1,X
  80 X=X+1
  90 IF X   20 THEN 40
 100 END
 200 OPEN "R",1,"TESTDATA:0"
 210 GOSUB 1000
 230 X=1
 240 GET 1,X
 250 PRINT T$
 260 IF LOC(1)=2 THEN PRINT "BUFFER #1 ACCESSED"; LOC(1); "TIMES":
       CLOSE:END
 270 X=X+1
 280 IF X   20 THEN 240
 290 END
1000 FIELD 1,255 AS T$
1010 RETURN
```

I hope that some of you will be able to make use of this new command.

The Best Has Gotten Better!

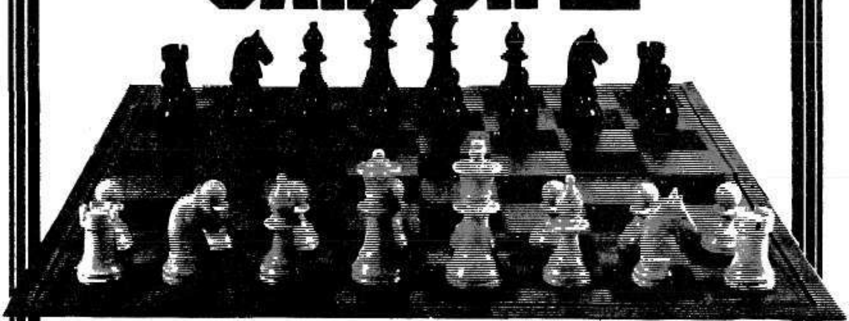
Sargon, the program that came in first in the Creative Computing Microcomputer Chess Tournament, has become Sargon II - **$29.95**

- A vastly improved game
- Faster response time
- New level 0 for beginners
- Easier to pre-set board
- Hint mode - What does the computer suggest

Sargon II is the program that took on the maxi-computers in the West Coast tournament, and finished in the money! More thinking power than you ever expected .

Now in stock for immediate delivery!

SARGON II



TRS-80 (Tape only - 16K)	
SARGON	**$19.95**
SARGON II	**$29.95**

Apple Tapes	
SARGON	**$19.95**
SARGON II	**$29.95**
SARGON II on disk	**$34.95**

TSE

The Software Exchange

6 South Street, Box 68, Milford, NH 03055 603-673-5144

Call (603) 673-5144 for immediate shipment.

MICROSOFT

DISK EXPANSION PACKAGE

The HARDSIDE DISK EXPANSION PACKAGE allows you to upgrade your TRS-80 Level II with 16K RAM to the power of Disk.

- **Expansion Interface, 16K RAM**
- **Percom Data Separator**
- **Percom TFD-100 Disk Drive**
- **Two Drive Cable**
- **NEWDOS Disk Operating System**
- **Box of BASF Diskettes**

This expansion package saves you $125 off of regular list prices.

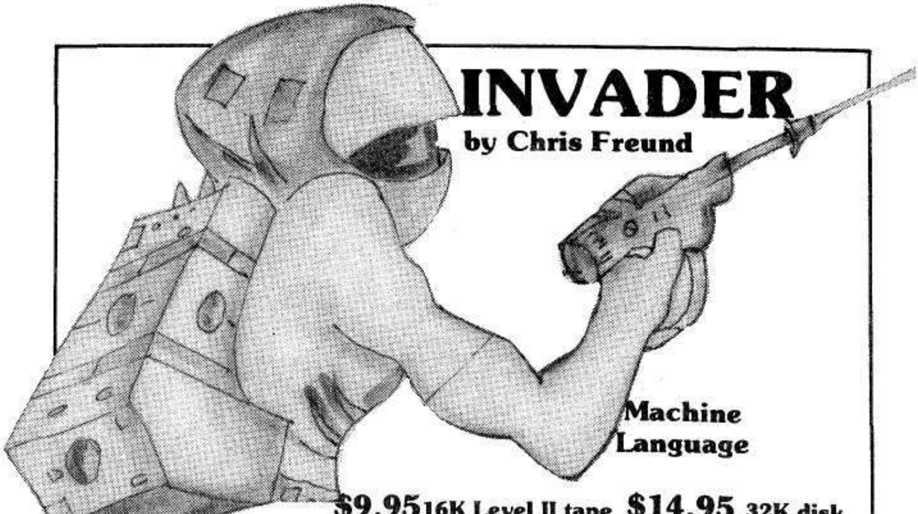
# PKG12 (22 lbs) $899.95

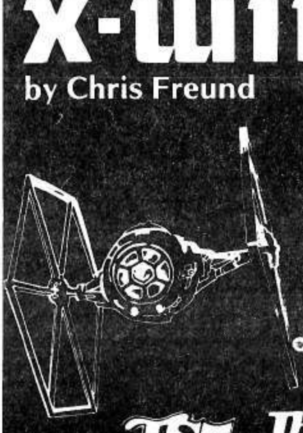
Options:

Additional Percom Disk Drive	*$389.00*
Additional 16K RAM (Exp. Int.)	*$ 90.00*
Four Drive Cable	*$ 10.00*

HARDSIDE™

6 SOUTH STREET, MILFORD, NH 03055 (603) 673-5144





USING YOUR ELECTRIC PENCIL WITH THE RADIO SHACK LOWER CASE MODIFICATION

The adaptation of the Electric Pencil to the Radio Shack computer requires a control key to access the command table. Since the Radio Shack lower case modification does not include a control key, this makes it impossible to use Electric Pencil with the Radio Shack modification.

For many people, this is not a problem, as Radio Shack's Scripsit is better than the Electric Pencil for most uses and costs less. However, there are still some uses for Pencil, and some people prefer the more expensive, less powerful system. These modifications to the Electric Pencil will allow you to use the BREAK key for a control key, and the CLEAR key for the functions that the BREAK key is normally used for.

These modifications were provided by Jeff Brown on THE SOURCE, tested by Clay Schneider, and verified by George Blank. This article was written using the modified PENCIL. The modifications also skip over the upper/lower case question, the title, change the default print margin to 5, and start you in the "Control K" sub command mode.

Use the DFS function of Superzap 3.0 to make the following changes to PENCIL/CMD: (Make these changes on a copy of your program to protect the orginal.)

This change will:
- At start-up, skip title card
- Allow for start-up default print margin of greater than 0
- Start-up with keyboard in lowercase-entry mode
- Start-up in control "K" subcommand mode

```
F00743 from  5A  21  C9  3D  11  6E  59  CD  CF  67  CD  79  65
         to  5A  3E  05*  32  2A  5A  CD  6D  65  C3  D6  61  65
F000A4 from  22  81  5C  21
         to  22  00  00  21
```

*The "05" is the start-up default margin and may be changed to whatever value you like.

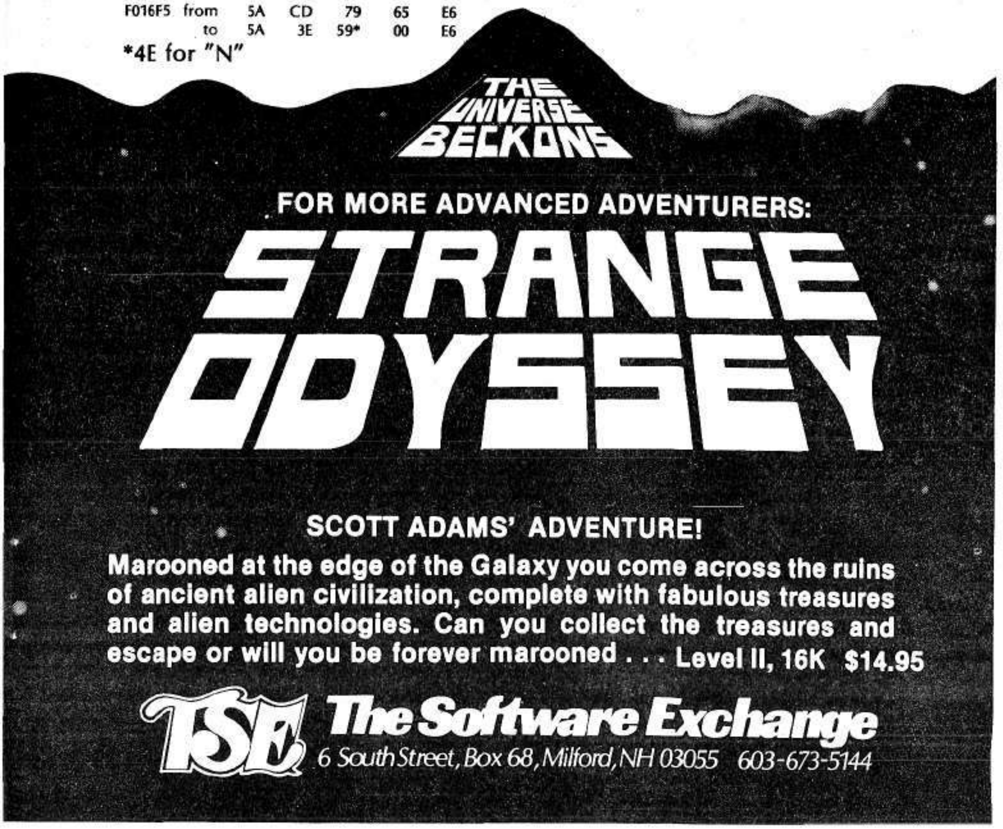
This change will make the "BREAK" key into the Control key, instead of that obnoxious hidden little button. Use "CLEAR" whenever BREAK is normally called for (exiting control—"K" subcommand mode, ending scrolling). IMPORTANT: These added routines use space in the Pencil title message, therefore, ZAP #1 MUST also have been made.

```
F01047  from   3A   7F   38   B7   28   28   D9   CD
          to   3A   3F   38   B7   C3   CA   59   CD
F01BC   from   C0   CB   61   20   F6
          to   C0   C3   BD   59   F6
F01D9   from   5D   1B   17
          to   5D   00   17
F010E1  from   5F   1E   5B
          to   5F   00   5B
F010F0  from   FE   1B   CA
          to   FE   5D   CA
F0069F  from   00   54   48   45   20   45   4C   45   43   54   52   49   43   20
          to   00   E5   21   40   38   CB   56   E1   C2   FC   65   C3   06   66
(cont)  from   50   45   4E   43   49   4C   20   20   28   43   29   20   31   39
          to   28   04   D9   C3   93   65   3A   40   38   E6   FB   20   F5   C3
(cont)  from   37   39   20
          to   BA   65   20
```

This modification will skip over "LOWERCASE KIT INSTALLED?" question, automatically answering "Y" (or "N", for that matter):

```
F016E6  from   6C   CD   CF   67   21
          to   6C   00   00   00   21
F016F5  from   5A   CD   79   65   E6
          to   5A   3E   59*  00   E6
```

*4E for "N"



ASSEMBLY
LANGUAGE
PROGRAMMING
AIDS

FOR TSE Order Form SOFTWARE ONLY

Accts Rec II Disk 32K	69.95
Adv Pers Fin Disk	24.95
Adventure Sample = 0	5.95
2 Adventures on Disk	24.95
☐ Land & Pirate	
☐ Mission & Voodoo	
☐ Count & Odyssey	
☐ Fun House & Pyramid	
3 Adventures Disk	39.95
☐ Land Pirate Mission	
☐ Count Voodoo Odyssey	
Adventures on tape	14.95
☐ Count	
☐ Fun House	
☐ Land	
☐ Mission Impossible	
☐ Pirate's Cove	
☐ Pyramid of Doom	
☐ Strange Odyssey	
☐ Voodoo	
Adventure (Microsoft)	29.95
Airmail Pilot	7.95
Air Raid	9.95
Alien Invasion (sound)	cass. 9.95
Alien Invasion (sound)	disk 14.95
Amateur Astronomy Handbook	14.95
Amateur Radio Disk	24.95
Amazin' Mazes	7.95
Android Nim (sound)	14.95
APL	disk 34.95
APL	w/book 49.95 + $3
APL	tape 14.95
APL	book only 15.50 + $3
Appointment Log	9.95
Automated Disk Directory	14.95
Barricade	9.95
Basic Handbook	14.95 + $1
Basic Statistics	9.95
Basic Styles Handbook	5.95 + $1
Bee Wary (sound)	14.95
Binders	4.95 + $1
Biorhythms	4.95
Bismarck	49.95
Bridge Challenger	14.95
Cards of Fortune	7.95
Cassettes	
☐ C-10	10 for 6.50 + $1
☐ C-20	10 for 7.50 + $1
Casino Anthology	7.95
Challenge (sound)	9.95
Chess Companion	7.95
COMPROC Command Processor	tape for disk only $19.95
Cribbage	7.95
Data Management System (CCA)	74.95 + $2
Datestones of Ryn	
Tape	14.95
Disk	19.95
Diskettes	
BASF	☐ 5 for 24.95 + $1
	☐ 10 for 39.95 + $1
	☐ 20 for 69.95 + $2
	☐ 100 for 299.00 + $3
Dysan	box of 5 29.95 + $1
Dynamic Data Base	39.95
Diskette Storage Box	5.00 + $1
Dosort 32K or 48	tape 34.95
DSM Sort Utility for disk	75.00
Editor/Assembler Plus 16K	tape 29.95
Electric Pencil	tape 100.00
Electric Pencil	disk 150.00
Educator Assistant	9.95
8080-Z80 Conversion	14.95
Electronics Assistant	9.95
End Zone II	9.95
Fastgammon	19.95
File Manager 80	49.95
Floppy Armour	box of 5 4.95 + $1
Floppy Disk Diagnostic	24.95
Fortran/Assembler	150.00 + $5
Fortran	80.00 + $2.50
Assembler	80.00 + $2.50
Galactic Empire	14.95
Galactic Empire/Trader	disk 29.95
Galactic Revolution	14.95
Galactic Trader	14.95
Galactic Trilogy	disk 39.95
GSF 16K,32K,48K	24.95
Ham Radio	9.95
Histograph/Scattergram	9.95
Home Financial Management	9.95
I Ching Level II 16K	7.95
Infinite BASIC by Racet	49.95
Business add-on	29.95
Intro TRS-80 Graphics	7.95 + $1
Invasion of Orion	
Tape	19.95
Disk	24.95
Inventory II.3 Disk	79.95
Inventory S	tape 24.95
Inventory S disk	w/invoicing 59.95
	w/o/invoicing 39.95
Investment Portfolio	49.95
IRV	
Tape	24.95
Disk	29.95
Journey Center Earth	7.95
Kamikaze	7.95
Keyboard	9.95
Kriegspeil	7.95
KVP	24.95
KVP on Disk	29.95
KVP 232	24.95
Learning Level II	15.95 + $1
Level I in Level II	14.95
Level III BASIC	49.95
Life Two (sound)	14.95
Loan Amortization	19.95
Lost Dutchman's Gold	9.95
Lower System	9.95
Machine Lang. Mon. (RSV.2)	26.95
Machine Lang. Mon. (RSV.2D)	29.95
Magic Paper Calculator	14.95
Mail List II Disk	99.95
Marlock's Tower	
Tape	14.95
Disk	19.95
Mastermind	7.95
Mean Checkers Machine	tape $19.95
Mean Checkers Machine	disk 24.95
Microchess 1.5	19.95
Microtext Editor	9.95
Moving Signboard	9.95
Mu-Math	74.95 + $3
NEWDOS	49.95
NEWDOS +	99.95
NEWDOS 80	149.95
9 Games/Preschool Child	9.95
Numerology 32K	disk 14.95
On-Line Invoicing	disk 39.95
Pathways Through the ROM	19.95
Payroll Disk 32K	39.95
Pencil Pal	35.00
Pentominoes (sound)	7.95
Periodical Cross Reference	tape 14.95
Periodical Cross Reference	disk 19.95
Personal Finance	9.95
Pigskin	9.95
Pork Barrel	9.95
PR Dogfight	7.95
Preflight	20.00
Print Spooler	disk 24.95
Remodel & Proload 16K,32K,48K	34.95
Renumber	7.95
Renumx	24.95
Rescue at Rigel	
Tape	19.95
Disk	24.95
Roots	disk 19.95
RPN Calculator	9.95
RSM-2	26.95
RSM-2D	29.95
RX (disk)	24.95
Santa Paravia & Fuimaccio	7.95
Sargon Chess	19.95
Sargon II	29.95
Sargon Handbook	15.95 + $1
Scripsit	
Tape/Disk	95.00 + $2
Secrets of the Tarot	9.95
Small Business Bookkeeping	tape 24.95
Small Business Bookkeeping	disk 29.95
Small Business Bookkeeping	journal 7.00
Snake Eggs (sound)	14.95
Space Battles	tape 14.95
Space Battles	disk 19.95
STAD	24.95
Star Fleet Orion	
Tape	19.95
Disk	24.95
Star Trek III.4	14.95
ST80 Smart Terminal	49.95
ST80D Smarter Disk	79.95
ST80 III	150.00
ST80UC	24.95
Super Simon	$9.95
System Copy	9.95
T Short	9.95
Taipan	9.95
Tape Recorder Alignment Kit	9.95

Temple of Apshai	
Tape	24.95
Disk	29.95
Text 80 Disk	59.95
Time Series Analysis	14.95
Time Trek	14.95
Ting Tong Level II 16K	9.95
Tiny Comp	tape 19.95
Tiny Comp	disk 24.95
Troll's Gold	4.95
TRS-80 Assy. Lan Pro.	3.95 + $1
TRS-80 Disk & Other Mysteries	22.95 + $1
TRS-80 Interfacing	8.95 + $1

TRS-80 Opera Theatre (sound)	9.95
with challenge on disk	19.95
Tycoon	7.95
Typing Tutor	19.95
Warfare	7.95
Wordo	14.95
X-Wing Fighter II	9.95
XREF	19.95
Y-Y Bar by Racet	14.95
Z-80 Chip Poster	3.99 + $1
Z-80 Gourmet Cookbook	14.95 + $1
Z-80 Handbook	4.95 + $1
Z-80 and 8080 Assembly	7.95 + $1

The Software Exchange

6 SOUTH STREET MILFORD, NH 03055 Order Number: 603-673-5144

Level II software available on disk for a $5.00 (per order) medium charge. This extra fee is for any number of programs transferred to disk from tape when you order. If the order exceeds the capacity of a single disk, we absorb the extra cost.

Please state level and memory size on order form . . . otherwise, we automatically ship Level II cassettes.

Be sure to include handling charge and any additional charges when figuring your total. All orders shipped with 24 - 72 hours. Prices subject to change without notice.

☐ **UPS** ☐ **1st Class** ☐ **Charge** ☐ **COD**

Charge card account number

Signature ....

Expt. Date .... **Inter.#** ....

Charge customers: please fill in account information above and below

Name ....

Address ....

City .... **State** .... **Zip** ....

ALL SOFTWARE SOLD ON AN AS-IS BASIS WITHOUT WARRANTY. TSE assumes no liability for loss or damage caused or alleged to be caused directly or indirectly by equipment or products sold or exchanged by them or their distributors, including but not limited to any interruption in service, loss of business or anticipatory profits or consequential damages resulting from use or operation of such equipment or software. We are not responsible for typographical errors anywhere in this publication.

DO NOT FORGET SHIPPING CHARGES

Shipping Charge - Add $1 Disk Charge - Add $5

All C.O.D. and overseas orders must add $5.

SoftSide™

PO Box 68
Milford, NH 03055
"your BASIC software magazine"

Rush me the next 12 issues of **SoftSide**.

☐USA bulk $18 1 yr. ☐$32-2 yrs. ☐CANADA/MEXICO $25 1 yr.
☐USA first class $25 1 yr. ☐OVERSEAS airmail $30 1 yr.
☐APO/OVERSEAS surface $25 1 yr.

Telephone your charge card order! Call our Subscription office Monday through Friday 9:30 to 5:30 (Eastern time) at **1-800-258-1790 (in NH call 673-5144)**

Credit Card VISA

Exp. Date________Interbank # (M/C only)________

Signature________________

Name________________

Address________________

City__________State__________Zip________

Please remit in US funds ONLY

Ready to get serious? SUBSCRIBE TO PROG/80 the magazine dedicated to serious programmers...beginners to professionals

PROG/80

SUBSCRIPTION RATES - 6 issues per year

USA	
Bulk mail - $15.00	Canada
First Class Mail - $21.00	Mexico $21.00
Overseas airmail - $27.00	APO/FPO
	Overseas surface mail

☐ **Check/Money Order enclosed** ☐ **Master Charge** ☐ **VISA**

SIGNATURE________________

ACCOUNT #________________

EXP. DATE__________INTER. #__________

NAME________________

ADDRESS________________

CITY__________STATE__________ZIP________

Telephone orders accepted for Master Charge or VISA accounts. Call Monday through Friday, 9:30 to 5:30 EST at **1-800-258-1790 (in NH call 673-5144)**

PO BOX 68, MILFORD, NH 03055

PROG/80
A SoftSide Publication
P.O. Box 68, Milford, NH 03055