TRSTIMES

Keeping Models 3 & 4 Alive

Volume 1. No. 1. - January 1988 - $3.00

michael shrayer's
ELECTRIC pencil
Dennis Bathory Kitsz
THE CUSTOM TRS-80 & OTHER MYSTERIES
William Barden, Jr.
HOW TO DO IT ON THE TRS-80 & OTHER MYSTERIES
H.C. Pennington
TRS-80 DISK & OTHER MYSTERIES
James Farvour
TRSDOS 2.3 DECODED & OTHER MYSTERIES
James Farvour
MICROSOFT BASIC DECODED & OTHER MYSTERIES for the TRS-80
Lewis Rosenfelder
BASIC FASTER AND BETTER & OTHER MYSTERIES
Michael J. Wagner
MACHINE LANGUAGE DISK I/O & OTHER MYSTERIES

Basic and Assembly Language type-in programs.

Discussion of public domain software

Programming hints & tips, review/tutorial, humor, question & answer column and more...

LITTLE ORPHAN EIGHTY

We all knew it would eventually happen. But it seemed such a long time off, so we didn't worry about it. Now the writing on the wall is so large and clear that it can be ignored no longer.

What am I talking about?

I am talking about the abandonment of our dear friend, the TRS-80.

The first blow came when Tandy officially announced that it would no longer support the TRS-80 line. Though very discouraging, we could live with that. Tandy never really did much for us anyway. Sure, they sold us some great machines and included some decent manuals, but then they left us alone to figure out what to do with it all.

Well, we did figure it out, and in spite of their continued neglect, the TRS-80 became a popular machine. Hardware and software was developed by third-party vendors. Utilities, databases, word-processors, spreadsheets, general ledgers and other serious business applications saw the light of day. There were arcade games, text adventures, and Leo Kristophersen made our otherwise silent machines play beautiful music.

I know of no other computer that has been blessed with as many alternative DOS'es as the Model 3. Standard TRSDOS 1.3 had very heavy competition from Newdos/80 2.0, and several versions of LDOS, DOS PLUS and MULTIDOS.

At one time the TRS-80 world was supported by such magazines as Kilobaud, Softside, 80 U.S., the TRS Newsletter, Computronics, Computer User, 80 Micro, the Alternate Source, Northern Bytes, DOS PLUS newsletter, LDOS Quarterly, and a handful of cassette-based magazines, with CLOAD being the most prominent. Even magazines such as Byte and Creative Computing featured articles and programs for Model I and III.

Boy, that was, indeed, the good old days. Information was plentiful and flowing freely. The TRS-80 was Top Dog.

In contrast, today we find ourselves weak. The vendors and software houses, if still in business, all have switched to the profitable world of MS-DOS and on October 16, 1987 came the one blow that we may not survive.

Our flagship, the last remaining shining light of the glory-days, 80 Micro, made the following announcement in the November SIDE TRACKS column by Eric Maloney:

'After several years of agonizing soul-searching, we at 80 Micro have concluded that we can no longer straddle the fence. Starting with the (1988) January issue, the magazine will devote its entire coverage to the Tandy MS-DOS systems.'

With 80-Micro abandoning us, we are almost beyond help. The TRS community will have no central rallying point.

The equation is frightingly simple. No regular flow of information = **OBLIVION** = **DEATH.**

Thinking about this turn of events long and hard, doing some very serious soul-searching of my own, I decided to stick my neck out and do something about it.

What I wanted was a newsletter whose roots and enthusiasm would be firmly planted in the Wayne Green tradition of 80 Micro from 1981-83; one that would be primarily oriented towards the TRS-80 hobbyist: Fun and useful type-in programs, both in Basic and Assembly Language, tutorials, hints & tips, reviews (with a different slant), questions & answers, definitely something for the BBS world and, if possible, a hardware column.

I wanted to create something that would give me, and hopefully others, the same kind of excitement that was felt when each new issue of 80 Micro came out in those days.

After much hard work and many sleepless nights, it has finally become a reality and you are looking at issue no.1 of:

TRSTimes

I hope you are as pleased with it as I am.

Lance W.

TRSTimes - Volume 1. No. 1. - January 1988

CONTENTS:

• TRSTimes is published bi-monthly by TRSTimes publications. 20311 Sherman Way #221. Canoga Park, Ca. 91306.

HUNTING FOR BURIED TREASURE

Peeking and Poking Model 4

by Lance Wolstrup

If you are a Model 4 user, your DOS is probably either TRSDOS 6.2 or LS-DOS 6.3. The author of both DOS'es, Roy Soltoff, has been very careful to isolate us from the actual addresses of important data areas and DOS routines.

Instead Assembly Language programmers were given SuperVisor Calls and the Basic programmer was given the clumsy SYSTEM command. He explains the reason for this:

"Trying to keep the memory locations data constant across all implementations of the system is quite restrictive and usually becomes limiting to the healthy growth of the system. Keeping portability in mind, the designers of the system have provided SuperVisor Calls which return pointers to data that may be useful to a program. Thus, there should usually be no need to access data areas by memory address."

As Tandy is no longer supporting TRSDOS 6.2 and most likely, LS-DOS 6.3 will never see any upgrades, we need not concern ourselves about 'healthy growth' or 'portability'. Instead we can go hunting for the treasures that is buried deep within DOS and use the ones we find interesting.

Unlike Models I and III whose low memory consists of unchangeable ROM, the Model 4's ROM is switched out upon boot-up and is replaced with the DOS which is ALL RAM. This means, not only can we read the information there, but we can also change it.

Although the information presented here is intended primarily for the Basic programmer, the addresses can certainly also be used from within Assembly Language programs, so let us look at some of the things we can do by talking directly to low memory.

TIME

2DH (45) seconds
2EH (46) minutes
2FH (47) hours

Time can be set from Basic by POKEing the desired values into these addresses. TIME$ goes here for information.

DATE

33H (51) year
34H (52) day
35H (53) month

The date has two purposes: First it is used to stamp the directory entry of new and updated files. Secondly, it is used from within programs as a default value for date prompts. The Model 4 community is painfully aware that TRSDOS 6.2 is not able to accept dates after 12-31-87. Even with the available patches, the directory entry date stamp will not work correctly.

If your main use of the date is prompts from within programs, simply disable the DOS DATE prompt by typing SYSTEM (DATE=N) from DOS. Then write your own date routine in the program, POKEing and PEEKing locations 33H, 34H and 35H. DATE$ looks to these addresses for the values, and it has almost none of the limitations of the DOS DATE routines. (see POKEDEMO/BAS program listing-lines 221-231 for date routine written in Basic).

FLAG$

Starting at address 6AH (106) is the FLAG table. This table consists of 26 consecutive memory locations, most of which hold an abundance of information. Here are some of the discoveries:

74H (116) KFLAG$
Bit 5.- 0=lower case. 1=upper case.
Force upper case with:
POKE &H74,PEEK(&H74) OR 32
Force lower case with:
POKE &H74,PEEK(&H74) AND 223

7CH (124) SFLAG$
Bit 3.- 0=indicate 2 mhz - slow. 1=indicate 4 mhz - fast.

This flag does not change the speed. It is only a check point for DOS. The actual speed must be set by writing 72H (114) to PORT &HEC (236) for fast or writing 8H (8) for slow.

4 mhz - FAST:
POKE &H74,PEEK(&H74) OR 8:OUT &HEC, &H72

2 mhz - SLOW:
POKE &H74,PEEK(&H74) AND 247:OUT &HEC, &H8

Bit 4.- 0=BREAK key enabled. 1=BREAK key disabled.

Disable BREAK key:
POKE &H74,PEEK(&H74) OR 16

Enable BREAK key:
POKE &H74,PEEK(&H74) AND 239

7DH (125) TFLAG$
If PEEK(&H7D) = 4 then Model 4.
If PEEK(&H7D) = 5 then Model 4P.

7FH (127) VFLAG$
Bit 4.- 0 = clock is not displayed. 1 = clock is displayed on screen.
Display clock:
POKE &H7F,PEEK(&H7F) OR 16
Display off:
POKE &H7F,PEEK(&H7F) AND 239

Bit 6.- 0 = cursor blinks. 1 = cursor is solid.
Force solid cursor:
POKE &H7F,PEEK(&H7F) OR 64
Force blinking cursor:
POKE &H7F,PEEK(&H7F) AND 191

There are many more fun things in the FLAG table, but for now let us move on to something that is sorely missed in the Model 4: SCROLL PROTECTION

I have seen several letters in the pages of 80 Micro, either looking for help in accomplishing scroll protection or offering ingenius solutions to that problem. Here is the EASY way:

Bits 0-2 of memory location &HB94 holds the amount of lines to scroll protect.

0B94H (2964) Bits 0-2.- Scroll protect from 0 to 7 lines at top of screen.

Scroll protect 5 lines:
POKE &HB94,PEEK(&HB94) OR 5
Remove scroll protect:
POKE &HB94,PEEK(&HB94) AND 248

Bit 3.- 0 = chr$(192-255) are space compression characters. 1 = chr$(192-255) are the special characters.
Force spec. characters:
POKE &HB94,PEEK(&HB94) OR 8
Force space compress.:
POKE &HB94,PEEK(&HB94) AND 247

0B97H (2967)
This location holds the visibility status of the cursor. If PEEK(0B97H) = 0 then cursor is invisible.
Any non-zero value in B97H makes the cursor visible.
Cursor off: POKE &HB97,0
Cursor on: POKE &HB97,1
(or any other non-zero value)

0B98H (2968)
The value of the cursor character is stored here.
If you have the normal underline cursor, PEEK(&HB98) will return 95.

Changing cursor to full graphic block:
POKE &HB98,191

This is just the surface of what is buried in low memory. There are many more goodies to be explored and I hope that this information will be an incentive for readers to join me and uncover the power that can be ours.

POKEDEMO/BAS
TRSDOS 6.2 OR LS-DOS 6.3

Start up with:
Break key disabled
Cursor = chr$(176)
Cursor = non blink
Cursor off
Uppercase on
Speed = fast
Time display on
Special characters on

```
5 POKE &H7C,PEEK(&H7C) OR 24:OUT
&HEC,72: POKE &HB98,176: POKE
&H7F,PEEK(&H7F) OR 80: POKE
&H74,PEEK(&H74) OR 32: POKE &HB97,0: POKE
&HB94,PEEK(&HB94) OR 8

10 CLS: PRINT@(1,0), CHR$(143); CHR$(244);
CHR$(245); CHR$(246); " "; CHR$(239); " 1987
Lance Wolstrup **** POKE DEMO **** peeks &
pokes for Model 4"; STRING$(80,131): POKE
&HB94, PEEK(&HB94) AND 247

20 PRINT@(3,0), "Break key: "; CHR$(13); "Cur-
sor chr: "; CHR$(13); "Cursor blink: "; CHR$(13);
"Cursor status:";

30 PRINT@(3,50), "Caps:";: PRINT@(4,50),
"Speed:";: PRINT@(5,50), "Time display:";:
PRINT@(6,50), "Basic Date:";

80 GOSUB 50000: PRINT@(7,0),
STRING$(80,140)
```

100 PRINT@(9,29), "1. Toggle Break key": PRINT@(10,29), "2. Set cursor character": PRINT@(11,29), "3. Toggle blink": PRINT@(12,29), "4. Toggle cursor status": PRINT@(13,29), "5. Toggle caps lock": PRINT@(14,29), "6. Toggle speed"

110 PRINT@(15,29), "7. Toggle time display": PRINT@(16,29), "8. Toggle basic date": PRINT@(17,29), "9. Set time": PRINT@(18,29), "Q. Quit - back to Basic"

120 PRINT CHR$(13); STRING$(80,131); CHR$(13);

This routine is just for fun -

Screen action is caused by alternately writing 70 and 72 to PORT 236

125 FOR Y = 1 TO 14:FOR X = 1 TO 10: OUT &HEC,70: NEXT: OUT &HEC,72: FOR Z = 1 TO 50: NEXT: NEXT

129 PRINT@(21,0), STRING$(80,32);

130 PRINT@(21,0), "Make your selection (1 - Q) ";

140 I$=INKEY$:IF I$="" THEN IF PEEK(&H2D) = 59 OR PEEK(&H2D) = 0 THEN GOSUB 50000: GOTO 140 ELSE 140

145 IF I$ = "Q" OR I$ = "q" THEN CLS: END

150 I = VAL(I$):IF I < 1 OR I > 9 THEN 140 ELSE ON I GOTO 160, 170, 180, 185, 190, 200, 210, 220, 240

Reverse bit 4 of SFLAG$

Bit 4 = 1 - Break key disabled
Bit 4 = 0 - Break key enabled

160 POKE &H7C,PEEK(&H7C) XOR 16: GOSUB 50000: GOTO 130

Change Cursor character routine

Erase selection prompt on line 21
Prompt for cursor chr value
Poke valid input into &HB98
DOS gets cursor chr from this address

170 PRINT@(21,0), STRING$(80,32): PRINT@(21,0), "Enter value for cursor character (1-255) ";: I$="": INPUT I$: IF I$ = "" THEN 129 ELSE I = VAL(I$): IF I < 1 OR I > 255 THEN 170 ELSE POKE &HB98,I: GOSUB 50000: GOTO 129

Reverse bit 6 of VFLAG$

Bit 6 = 1 - solid cursor
Bit 6 = 0 - blinking cursor

180 POKE &H7F,PEEK(&H7F) XOR 64: GOSUB 50000: GOTO 130

Toggle cursor status

&HB97 = 0 - cursor is invisible
&HB97 < > 0 - cursor is visible

185 POKE &HB97,PEEK(&HB97) XOR 32: GOSUB 50000: GOTO 130

Toggle bit 5 of KFLAG$

Bit 5 = 1 - uppercase
Bit 5 = 0 - lowercase

190 POKE &H74,PEEK(&H74) XOR 32: GOSUB 50000: GOTO 130

Toggle speed

Write 72 to PORT 236 and set bit 2 of KFLAG$ = Fast speed
Write 8 to PORT 236 and reset bit 2 of KFLAG$ = Slow speed

200 POKE &H7C,PEEK(&H7C) XOR 8: GOSUB 50000: IF PEEK(&H7C) AND 8 THEN OUT &HEC,72: GOTO 130 ELSE OUT &HEC,8: GOTO 130

Toggle bit 4 of VFLAG$

Bit 4 = 1 - display time clock
Bit 4 = 0 - turn off time clock

210 PRINT@(0,0), STRING$(80,32);: POKE &H7F,PEEK(&H7F) XOR 16: GOSUB 50000: GOTO 130

Toggle date on or off

If &H34=0 - no date (day)
If &H35=0 - no date (month)
else date is present - so poke 0 into both locations to disable date
221-231 is the enter date routine

220 IF PEEK(&H34) = 0 OR PEEK(&H35) = 0 THEN PRINT@(21,0), STRING$(80,32) ELSE POKE &H33,0: POKE &H34,0: POKE &H35,0: GOSUB 50000: GOTO 130

221 PRINT@(21,0), "";: PRINT"Enter date (MM/DD/YY)";: DT$ = "": INPUT" ",DT$: IF DT$ = "" THEN 129
'input date - if just ENTER is pressed - back to selection prompt

222 M$ = LEFT$(DT$,2): M=INT(VAL(M$)): IF M < 1 OR M > 12 THEN 220
'extract month from input

223 IF MID$(DT$,3,1) < > "/" OR MID$(DT$,6,1) < > "/" THEN 220
'check if slashes are in correct positions

224 Y$ = RIGHT$(DT$,2): Y = INT(VAL(Y$)): IF Y < 0 OR Y > 99 THEN 220
'extract year from input

225 D$ = MID$(DT$,4,2): D = INT(VAL(D$)): IF D < 1 THEN 220
'extract day from input and make sure it is at least 1

227 IF M = 1 OR M = 3 OR M = 5 OR M = 7 OR M = 8 OR M = 10 OR M = 12 THEN IF D > 31 THEN 220 ELSE 231
'make sure 31 day months do not exceed 31

228 IF (M = 4 OR M = 6 OR M = 9 OR M = 11) THEN IF D > 30 THEN 220 ELSE 231
'make sure 30 day months do not exceed 30

229 IF Y = 0 AND D > 28 THEN 220 ELSE IF Y/4 = INT(Y/4) AND D > 29 THEN 220 ELSE IF Y/4 = INT(Y/4) AND D < 30 THEN 231 ELSE IF D > 28 THEN 220
'check february -
if year=0 (2000 is NOT a leap year)
do not allow more than 28 days -
else if leap year do not allow
more than 29 days otherwise do
not allow more than 28 days

231 POKE &H33,Y: POKE &H34,D: POKE &H35,M: GOSUB 50000: GOTO 129
'poke year,day & month into &H33,&H34 and &H35 - date now accessible from DATE$

Time input routine

240 PRINT@(21,0), STRING$(80,32): PRINT@(21,0), "Enter time (HH:MM:SS) ";: T$ = "": INPUT " ",T$: IF T$ = "" THEN 129

241 H$ = LEFT$(T$,2): H = INT(VAL(H$)): IF H < 0 OR H > 23 THEN 240
'extract hours from input - make sure range is correct

242 IF MID$(T$,3,1) < > ":" OR MID$(T$,6,1) < > ":" THEN 240
'make sure colons are in correct position

243 MI$ = MID$(T$,4,2): MI = INT(VAL(MI$)): IF MI < 0 OR MI > 59 THEN 220
'extract minutes from input - make sure range is correct

244 S$ = RIGHT$(T$,2): S = INT(VAL(S$)): IF S < 0 OR S > 59 THEN 220
'extract seconds from input - make sure range is correct

245 POKE &H2F,H: POKE &H2E,MI: POKE &H2D,S: GOTO 129
'poke hours,minutes,seconds into &H2F, &H2E and &H2D

Subroutine to update screen

Display status of Break key

50000 PRINT@(3,16), "";: IF PEEK(&H7C) AND 16 THEN PRINT "Disabled"; ELSE PRINT "Enabled ";

Display number of cursor character

50010 PRINT@(4,16), USING "###"; PEEK(&HB98);

Display status of cursor blink

50020 PRINT@(5,16), "";: IF PEEK(&H7F) AND 64 THEN PRINT "Off"; ELSE PRINT "On ";

Display status of cursor visibility

50025 PRINT@(6,16), "";: IF PEEK(&HB97) < > 0 THEN PRINT "On "; ELSE PRINT "Off";: PRINT@(21,28), " ";

Display status of Caps lock key

50030 PRINT@(3,66), "";: IF PEEK(&H74) AND 32 THEN PRINT "On "; ELSE PRINT "Off";

Display status of speed mode

50040 PRINT@(4,66), "";:IF PEEK(&H7C) AND 8 THEN PRINT "Fast - 4mhz"; ELSE PRINT "Slow - 2mhz";

Display status of time clock

50050 PRINT@(5,66), "";:IF PEEK(&H7F) AND 16 THEN PRINT "On "; ELSE PRINT "Off";

50051 Y = PEEK(&H33): D = PEEK(&H34):M = PEEK(&H35):IF Y > 0 THEN IF Y/4 = INT(Y/4) AND M = 2 AND D = 28 AND PEEK(&H2F) = 23 THEN LY = 1
'set LY to change to feb 29 in leap year

50052 IF LY = 1 AND PEEK(&H35) = 3 AND PEEK(&H34) = 1 THEN POKE &H35,2: POKE &H34,29: LY = 0
'DOS goes from feb 28 to mar 1 in all years after 1987 - so if leap year change it back to feb 29 and reset LY

Display status of date

50060 PRINT@(6,66), "";: IF PEEK(&H34) = 0 OR PEEK(&H35) = 0 THEN PRINT "No date "; ELSE IF PEEK(&H33) > 99 THEN POKE &H33,0: PRINT DATE$; ELSE PRINT DATE$;

50080 PRINT@(21,28), "";

51000 RETURN

INSIDE

POKEDEMO/BAS

POKEDEMO/BAS is, as the name implies, a demonstration program. It is not intended to do anything particularly useful other than reveal some Model 4 BASIC techniques that will add speed and power to programs.

The SYSTEM command can accomplish most of the same things found in this program, however in order to use SYSTEM, Basic has to load in an overlay from the DOS disk. This takes time. The POKEs alter memory instantly. No overlays. No waiting.

POKEDEMO/BAS uses something that is absolute GREEK to a lot of Basic programmers: BOOLEAN LOGIC, so let us try to demystify this concept.

Boolean logic is, among other things, used for 'bit-fiddling'. That is, comparing or changing individual bits in a byte.

Each byte in the Model 4 has 8 bits that can be turned ON or OFF. That gives us 256 different possible ON/OFF combinations for each byte. (0-255).

The bits are numbered from right to left: 0-7

7 6 5 4 3 2 1 0

X X X X X X X X

Bit 0 when turned ON has the value 1, bit 1, ON, has the value 2, bit 2, ON, has the value 4, bit 3, ON, has the value 8 etc.

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X
128	64	32	16	8	4	2	1

The logical operators used in POKEDEMO are: AND, OR and XOR. They do three different, very useful things, all comparing two binary numbers bit by bit.

AND works this way:
If both bits are 1, the result is 1. All other times the result is 0.
For example:

```
      0000 0101    05H     5 decimal
AND
      0000 1100    0CH    12 decimal
      ----------------------------------
      0000 0100    04H     4 decimal
```

OR does this:
If either or both the bits are 1, the result is 1. Only if both bits are 0 will the result be 0.

For example:

```
      0000 0101    05H     5 decimal
OR
      0000 1100    0CH    12 decimal
      ----------------------------------
      0000 1101    0DH    13 decimal
```

XOR does the following:
If one or the other of the bits is 1, the result is 1. If both the bits are 1 or both the bits are 0, the result is 0.

For example:

```
      0000 0101    05H     5 decimal
XOR
      0000 1100    0CH    12 decimal
      ----------------------------------
      0000 1001    09H     9 decimal
```

AND is used to 'mask' out (turn OFF) one or more individual bits. When you wish to turn off a particular bit in a memory location, simply POKE that location with what is already there ANDed with everything ON except the particular bit.

For example let us say we want to turn off bit 4 of memory location &H7C (124). Remember that the value of bit 4 is 16, so our AND number is the sum of ALL the bits EXCEPT bit 4. In other words: 1+2+4+8+0+32+64+128. The AND number is 239.

Now, in order not to change anything except bit 4, we POKE &H7C,PEEK(&H7C) and 239.
(You just disabled the BREAK key.)

OR turns one or more individual bits on. To turn bit 4 back on in location &H7C, we POKE the location with what is already there ORed with the value of bit 4: POKE &H7C,PEEK(&H7C) OR 16.
(BREAK key is now enabled again.)

XOR can be used to toggle a bit to the alternate state. That is, if it is 1 it will be 0, and if 0 it becomes 1.

For example, if we want to change the status of the BREAK key to whatever it is not doing now, we would POKE &H7C with the value that is already there XORed with the value of bit 4: POKE &H7C, PEEK(&H7C) XOR 16.

Now that we have a general understanding of Boolean logic let us take a look at what happens in POKEDEMO/BAS:

Line 5 sets the start up conditions in the program. ORing the contents of &H7C with 24 (16 and 8) sets both bit 4 (disables BREAK key) and bit 3 (indicate FAST-4mhz). Writing 72 (64 and 8) to PORT &HEC sets both bit 6 (puts Model 4 in FAST mode-4mhz) and bit 3 (enables alternate character set). POKEing 176 into &HB98 makes the cursor character CHR$(176). ORing the contents of &H7F with 80 (64 and 16) sets bit 6 (makes cursor solid) and bit 4 (displays clock on screen). ORing &H74 with 32 sets bit 5 (upper case). Then turn cursor off by POKEing location &HB97 with 0. Finally, set bit 3 of location &HB94 (select special characters).

Line 10 prints the copyright and title of the program. At the end of the line we turn off bit 3 of &HB94. (special characters are now switched off again).

Lines 20-30 prints the status display on the screen.

Line 80 gets the actual condition of the selected display items from subroutine in 50000.

Lines 100-120 prints the menu on the screen.

Line 125 is pure folly. The action of the screen will certainly get your attention. Writing 70 to PORT &HEC sets bits 6,2 and 1 (0100 0110).

Bit 6 controls the speed of the Model 4. We set this bit. (FAST). Bit 3 controls the alternate character set. This bit is 0. (alt. set disabled). Bit 2 selects the screen mode (80 or 40 chrs). Setting bit 2 makes it 40 character mode. Setting bit 1 turns the cassette motor on.

Writing 72 to PORT &HEC sets bits 7 and 3 while all other bits are turned off (0100 0100). Comparing the bits of the two numbers it is evident that bits 7,6,5 and 4 remain constant, while bit 3 is toggled off/on and bits 2 and 1 are toggled on/off. The toggling of bits 3 and 2 causes the shaking of the screen while togging bit 1 causes the clicking sound. (Model 4P owners will not hear the clicking sound as you do not have the cassette port.

Line 129 erases screen line 21.

Line 130 prompts to make a selection from the menu.

Line 140 is the INKEY$ routine. Because POKEDEMO/BAS will keep the date current, several things happen while waiting for a key press. Since DOS will change the day when TIME reaches 00:00:00, we PEEK(&H2D) to see if we have reached 59 seconds or 00 seconds. If either of these values are TRUE we update the screen by jumping to subroutine in 50000. Then we jump back to line 140 and continue to wait for a key press.

Line 145 ENDs the program if Q or q is pressed. Do be careful here as the options selected are still valid. In other words, if the program has the BREAK key disabled, it is STILL disabled after you exit POKEDEMO.

Line 150 checks for valid selection and then jumps to appropriate routine.

Line 160 uses XOR to toggle bit 4 of SFLAG$ to enable or disable the BREAK key then update the screen in subroutine 50000 and finally back and wait for key press.

Line 170 erases selection prompt and replaces it with the cursor value prompt. Pressing ENTER without a value returns the selection prompt. An INPUT in the range of 1-255 is POKEd into &HB98. Screen is updated and the selection prompt is returned.

Line 180 uses XOR to toggle bit 6 of VFLAG$ to make the cursor either solid or blinking. Again, screen is updated and we are prompted to make a selection.

Line 185 uses XOR to toggle the cursor status between ON (visible) and OFF (invisible). Memory location &HB97 controls the visibility of the cursor. If the value is 0, the cursor is invisible. Any non-zero value makes the cursor visible and uses the value for the character behind the cursor. Therefore 32 or 128 (both blank characters) are the logical choices.

Line 190 controls whether we will be in upper- or lower case. We use XOR to toggle bit 5 of KFLAG$. We update the screen and display the selection prompt.

Line 200 does several things. First it uses XOR to toggle bit 2 of SFLAG$ to indicate whether we are in 4 mhz mode or 2 mhz mode, then it jumps to the subroutine in 50000 to update the screen display. Finally we PEEK SFLAG$ to see what the speed status should be. If it indicates SLOW (bit 2 set) we write SLOW mode to PORT &HEC: OUT &HEC,72 (sets bit 2 and bit 6). If it indicates FAST mode (bit 2 reset), we OUT &HEC,8 (set bit 2 and reset bit 6). Bit 2 of KFLAG$ is used from several DOS subroutines, so it does need to be handled correctly. However, the value we write to PORT &HEC is what actually controls the speed.

Line 210 toggles bit 4 of VFLAG$ which controls the display of the real-time clock. First we erase the entire top line of the screen (in case we are turning off the clock display) then we either turn on the clock by setting bit 4 of VFLAG$ or we turn if off by resetting bit 4. Screen is updated and selection prompt is returned.

Line 220 toggles the date on and off. We check to see if &H34 (day) or &H35 (month) contain 0. If either of these locations contain 0 it means that the date is disabled. We will then turn it on in lines 221-231. If, on the other hand, both locations contain non-zero values, it means that the we have a valid date and that we need to turn it off. We do this by POKEing 0 to locations &H33, &H34 and &H35. We then update the screen and go back to the selection prompt.

Line 221 starts the actual date routine by erasing the selection prompt on screen line 21 and replacing it with the Enter date prompt. If we simply press ENTER without a date we will be returned to the selection prompt.

Line 222 checks to see if the Month has been entered correctly. We extract the 2 leftmost characters of DT$ and convert them into numbers that we store in M. We check to see if M is in the range 1-12. If not in valid range we are sent back to date prompt. If we are in range, we know that only if M=2 do we need to check for leap year

Line 223 checks if the slashes are in the correct position. If not, we are sent back to the date prompt.

Line 224 extracts the Year from DT$, converts it into a number and stores it in Y. We then check that the number is in valid range (0-99). If not, we are sent back to the date prompt in line 220.

Line 225 extracts the Day from DT$, converts it into a number and stores it in D. If D is 0 or less, we are returned to the date prompt.

Line 227 checks the validity of the day in a 31 day month. If not valid, back to the date prompt. If valid, skip to line 231.

Line 228 checks the validity of the day in a 30 day month. If not valid, back to the date prompt. If valid, we skip to line 231.

Line 229 checks for leap-year. First we need to take care of the exception. Since year 2000 is NOT a leap-year, we jump back to the date prompt if Y=0 AND D>28. Next, if Y is divisible by 4 AND D>29 we jump back to the date prompt, because even though it is a leap-year, the numeric value of the day is too high. Then, if Y is divisible by 4 AND D<30 we have a valid date in a leap-year and we jump to line 231. Finally, if not a leap-year and D>28 we jump back to the date prompt.

Line 231 processes the date. We POKE the Year into &H33, POKE the Day into &H34 and POKE the Month into &H35. The screen is updated and we jump back to the selection prompt.

Line 240 is the beginning of the time routine. First we erase the selection prompt on screen line 21 and replace it with the time prompt. Pressing ENTER without a value returns the selection prompt.

Line 241 extracts the Hour from T$, converts it into a numeric valueand stores it in H. H is then checked for valid range (0-23). If range is not valid we return to the time prompt.

Line 242 checks if the colons are in the correct positions. If not, we go back to the time prompt.

Line 243 extracts the Minute from T$, converts it into a numeric value and stores it in MI. MI is then checked for valid range (0-59). If range is not valid, we go back to the time prompt.

Line 244 extracts the Second from T$, converts it into a numeric value and stores it in S. We check for range (0-59) and if not valid we return to the time prompt.

Line 245 POKEs the Hours into &H2F, the Minutes into &H2E, and the Seconds into &H2D. Then we return to the selection prompt.

Line 50000 checks the status of the BREAK key. If SFLAG$ (&H7C) has bit 4 set, we print Disabled. If bit 4 is not set, we print Enabled.

Line 50010 displays the value found in &HB98. This is the value of the cursor character.

Line 50020 checks the status of the cursor blink. If VFLAG$ (&H7F) has bit 6 set, the cursor is solid and we print Off. If bit 6 is not set, the cursor blinks and we print On.

Line 50025 checks if the cursor is visible or invisible. If &HB97 has the value 0, the cursor is invisible, so we print Off. Any value other than 0 makes the cursor visible, so we print On.

Line 50030 checks the status of the Caps lock key. If KFLAG$ (&H74) has bit 5 set, we are in UPPER CASE and we print On. If bit 5 if not set, we are in lower case and we print Off.

Line 50040 checks the speed of the CPU. If SFLAG$ (&H7C) has bit 3 set, we are in 4 mhz mode and we print Fast - 4mhz. If bit 3 is not set, we print Slow - 2 mhz.

Line 50050 checks the status of the time clock. If VFLAG$ (&H7F) has bit 4 set, the clock is visible in the top right hand corner and we print On. If bit 4 is not set, we print Off.

Line 50051 is an addition to the date routine. DOS changes the the date when time goes from 23:59:59 to 00:00:00. TRSDOS 6.x changes all 02/28/YY to 03/01/YY after 1987.

In order to change 02/28/YY to 02/29/YY in leap-years we PEEK &H33 (Year), &H34 (Day) and &H35 (Month). If the Year is not 0 (remember 2000 is not a leap-year) then if Year is divisible by 4 (leap-year) and Month is 2 (february) and Day is 28 and Hour is 23 (PEEKing &H2F) then we set LY=1. LY is our flag to indicate that we have the condition that warrants the change from 02/28/YY to 02/29/YY.

Line 50052 checks if our leap-year flag (LY) is 1. When Time reaches 00:00:00 the Date changes to 03/01/YY regardless of leap-year, therefore, if LY=1 we check if this has occured by PEEKing &H35 (month) to see if it is 3 and &H34 to see if it is 1. If so, we simply POKE &H35 with 2 and &H34 with 29 and then reset LY=0.

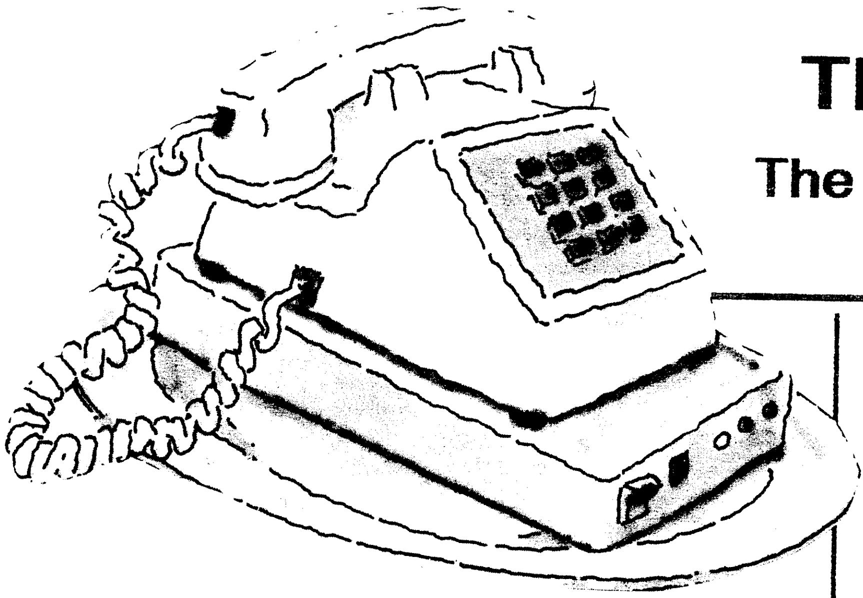
Line 50060 checks whether or not the Date is present. If &H34 (day) is 0 or &H35 (month) is 0, no date is present so we print No date, otherwise we display the date with DATE$.

Line 50080 simply makes sure that the cursor will be positioned immediately after the selection prompt.

Line 51000 returns subroutine to the caller.

The date routine will allow you to handle the Date correctly in Basic from 03/01/1900 to 02/28/2100. This ought to be sufficient.

If anyone needs correct leap-year dates after 02/28/2100, my great-grandson will write an article about it. I PROMISE!



TIM's PD EXPRESS

The Compression Confusion

By Timothy Sewell

File compression is not new to the computer world. Anybody who uses CP/M is bound to be familiar with the SQUEEZE and UNSQUEEZE programs, as well as the LIBRARY utilities readily available in the BBS world.

MS-DOS users have the popular ARC utilities. These programs are used to compress a file into a new file smaller than the original, thus saving disk storage space and especially, saving transfer time over the phone lines.

There are many different ways to compress a file. We won't go into those details here, as they are entirely too numerous. Instead this article is about the COMPRESSION UTILITIES available for the TRS-80 Models 3 & 4, and how to identify them in your BBS travels.

The TRS-80 community was not introduced to file compression until fairly recently, however we now have utilities available to us that will allow us to compress files, combine several files into one file, and even un-ARChive files created on an MS-DOS machine.

One of the first compression programs available for the TRS-80 was SQ3 and SQ4. These utilities were adapted from the original CP/M SQUEEZE source codes by David Huelsmann.

Until recently, these along with their counterpart UNSQUEEZE, were THE programs to use. They performed their job well and compression as much as 52% on ASCII files was easily obtained.

The first version was crude and was a little tedious to use due to the fact that you had to type in the name of every single file you wanted to Squeeze. If you had several files, this could seem like an eternity.

In addition, there was a flaw in the compression technique itself. Very small files could actually become larger and the original Squeeze program did not check to see if this problem had occurred.

Later versions solved these problems by adding Wildcard capabilities and adding a routine that compares the Squeezed file with the original and removes the Squeezed file if it did not benefit from compression.

A companion program USQ3 and USQ4 were also developed to UNSQUEEZE the compressed files. After all, why compress a file if you can't bring it back to its original state? They were very similar to SQ3 and SQ4 in operation and went through basically the same evolutions.

David Huelsmann also developed a program, based on the CP/M LIBRARY utility, that allows us to combine several files (Squeezed or Unsqueezed) into one single file. This was perfect for Uploading and Downloading to a BBS.

The practice is to Squeeze all the files first before adding them to the library. One file consisting of several compressed files equals time and money saved on the phone lines. Instead of Downloading several related files of one program, you could now Download the entire program package in one file.

There are several versions of LU available depending upon the DOS you use:

LU4 - Works in TRSDOS 6.2, DOSPLUS and MULTIDOS.

LU3 - Works in LDOS, DOSPLUS and MULTIDOS. It will also work in TRSDOS 1.3 with an additional file to allow Wildcards.

LUN3 - Works in NEWDOS/80.

LUM3 - A Library maintenance utility that is used to take apart Libraries created with other older versions of LU.

Getting confusing? Just wait!

A few weeks later, another LU utility appeared on the BBS circuits that would work the same as CP/M's Library utility. This program has no credits so I cannot identify the author.

It is commonly renamed LBR4/CMD so as not to confuse it with LU4/CMD. If David's programs won't take apart a Library you downloaded, it was most likely created with this program.

Around the same time as all of this was going on, a company called System Enhancement Associates (SEA) developed a program for MS-DOS computers that would not only Squeeze files, but also put them into a Library all at the same time.

Instead of several programs and several steps to reach your goal of one Library, it could now be done all in one step with one program. The program is called ARC, and it was a major breakthrough for file compression.

ARC even combined several different types of compression and determined which type would compress the file in the most efficient manner.

This was all fine and dandy for the MS-DOS people, but what about the TRS-80?

"Can't be done" some said. "Memory is too limited".

NEVER say CAN'T to a TRS-80 Hacker!

Along comes Chuck Harper who is best known for a Program called DISKCAT6 written for the Model 4. It is a nice Disk Catalog program that will read TRSDOS 6.2, DOSPLUS 4 AND Montezuma Micro's CP/M diskettes.

Chuck decided that he needed a program similar to SEA's ARC program so he studied some of the books available on file compression and decided to write one.

Easier said the done.

After several months he developed a program called ARCHIVE4/CMD. He wrote his own compression technique and made the program menu driven. The program also interacted directly with another program called SHELL. (Shell will be the subject of this column in a future issue.).

The first release was crude but it did the job, with a few major drawbacks.

1 It did not check to see if the compressed file was larger then the original.

2 Wild card capabilities were not available. You had to type in the name of every file you wanted to add.

3 You could not add or extract single files to or from the Archive.

4 The program was limited to run in TRSDOS 6.2 only.

The concept was great, but using the Squeeze/Unsqueeze method still saved an average of 20% more space.

Chuck continued hacking and his later versions of ARCHIVE4/CMD solved most of the problems while adding new functions, such as getting a directory of a disk, "tagging" the files to be added to the Archive, and even executing a program directly from the Archive.

The program became the standard for those who wanted to Archive programs, and versions for the Model 1 and 3 were developed for use in LDOS.

But the Squeeze/Unsqueeze Library method STILL saved more space, and besides, people wanted compatibility with SEA's ARC program.

Chuck's code is his own and can not be adapted to the code that SEA uses.

"Not good enough!",
the people said,
"We want MS-DOS compatibility!".
"Why?"
"Well....JUST BECAUSE!"

In steps David Huelsmann again. He obtains permission to directly modify SEA's source code to ARC for use on the Model 4.

The result of this work is ARC4/CMD and it allows adding, moving, deleting, extracting, printing, encrypting and decrypting of the files in the Archive.

ARC4 analyzes a file and chooses between 3 different types of compression in order for the file to be stored in the most efficient manner.

Files created by ARC4, the Model 1/3 version ARC31, as well as the MS-DOS version, could be read interchangeably by the other.

This program was just what the people were begging for but there was still one problem. TRS-80 people still could not take apart MS-DOS created Archives.

As it turned out, they COULD take apart MS-DOS ARC's. That is, if they wanted to use CP/M. However, not everybody owned CP/M for the Model 4, and besides, it was more work than most wanted to do.

On February 15, 1987, David Huelsmann released XARC4/CMD.

XARC4 will take apart MS-DOS created ARC's while in the Model 4 native mode. XARC4 allows us to direct the drive output, remove carriage returns and line feeds from files, and list a file in the archive directly to the screen. ARC's created with ARC4 and ARC31 can also be taken apart by XARC4.

A few other authors have entered their compression/library utilities into the BBS world, but none of them have made the impact of the programs mentioned here.

MY FAVORITE

So, with all of these Squeeze/Library/Archive utilities available, which ones should you use?

My personal favorite is David Huelsmann's ARC4/XARC4 programs.

Since I call a lot of Long Distance BBS systems, my biggest concern is my Long Distance bills, and after several hours of testing, I have determined that ARC4 has the best file compression of all of the programs available. An average of 20-25% more compression over ARCHIVE4.

ARC4 doesn't have as many fancy features and is a bit slower then ARCHIVE4, but the extra compression capabilities of ARC4 far outweigh the features of ARCHIVE4.

I also find that I put the MS-DOS capability of XARC4 to use quite frequently as many text files are available for use in MS-DOS created ARC's. High Resolution Graphic files are also usually found in Arc'ed format.

These utilities are invaluable to any TRS-80 owner. If you BBS frequently you will find that more and more ARC's are appearing, as opposed to lists of related files.

The following is a list of ways to identify various files and the compression programs that were used on them:

/xQx - This is a SQUEEZED file. Use USQ3 or USQ4 to Unsqueeze it.

/LBR - This is a LIBRARY file. LU4, LU3, LBR4, LUN3, or LUM3 will be needed.

/LQR - This is a SQUEEZED LIBRARY file.

/ACH - This is an ARCHIVE created with Chuck Harper's ARCHIVE4 program.

/ACC - This is an extra compressed ARCHIVE4 program.

/ARC - This is an ARC created with ARC4, ARC31, or possibly MS-DOS's ARC. Note that some early files created with ARCHIVE4 have the /ARC extension but they are not around much anymore.

/ESS - This is a STACKED file. STACK is a utility by D&D Software and it is very unlikely that you will come across a STACKED files, as the program wasn't particularly good.

David Huelsmann is currently working on newer versions of ARC that will allow greater file compression, and I am looking forward to them.

These programs are available on most TRS-80 BBS systems, and certanly ALL are available on GEnie.

They are also in The File Cabinet's Mail Order Public Domain Software Catalog.

The catalog is available on 2 disks for $5.00 which includes postage and can be obtained by writing to:

The File Cabinet.
PO Box 4295
San Fernando, Ca., 91342

Tim is the Sysop of the TRS-80 section at GE's nationwide BBS, GEnie. Under his supervision the TRS-80 is now in the top 10 sections of that board and its popularity is growing constantly.

Tim has what is probably one of the largest collections of TRS-80 Model 4 Public Domain software in the world and, as one of his three Model 4's has the Radio Shack High Resolution board installed, he is naturally a great fan of the MACpaint graphic files. Being a music buff, his collection of ORC-90 files is quite impressive. He is also a very active member of the Los Angeles based 'Valley TRS-80 Hackers Group'.

Tim can be contacted by writing to him directly:
P.O. Box 4295
San Fernando, Ca. 91342

Please enclose a Self Addressed Stamped envelope for a reply.

COMPUTING AT CHAOS MINOR

with Jerry Pellmelle

by Eric Bagai

Housekeeping here at Chaos Minor is a never-ending task. Old and new disks reviewed and discarded by the bushel basket, hard drives erased at random, eproms exposed to polarized ectoplasm, and all amidst the usual expansion and reconstruction. Sometimes I just need to relax and think of some of the great classics that have passed over my heads.

For example, there was that beta version of ELIZA that showed up in the mail. When I finally got around to firing it up on Agnes D (my Z80 friend) it looked like a serious bit of hacking, not just the toy program that eventually wound up being sold to us masses. For one thing, it really worked! While this first Eliza still did a lot of "yes, go on," and "what does that remind you of?" it would also give direct advice, make suggestions, save data for a later session, and give opinions. It worked so well that I used it on a drooling problem I had that only appeared when I talked to my friend Bob Heinlein, and by golly all I have now is just this little thing about military uniforms. But what the hey, the fans all love me for it.

But there was a serious problem here. If Eliza could really fix our little psychic kinks and bugs, if it really provided enough mental grease to smooth the jagged edges and help us "adjust to society," then what kind of a society was it adjusting us to? The more I thought about it, the more it seemed, well, sinister. Almost liberal. I'd boot it up and it would start asking stupid questions about why I felt I needed my own tactical nukes (to protect my family, of course), and how much a space-based system like L5 might cost (who cares when you can tap the power of the sun!) It even had problems with my proposal that only people who code in Modula4 should have the vote.

There is only so much limp-wristed, pacifist, anti-technocratic pablum that I can take in a piece of software before my American blood begins to boil. So I called up the author of Eliza and explained why his product could not be marketed. (It's really very simple to show a liberal where his interests lie.)

Of course we know how it all turned out. I won and Eliza was crippled beyond recovery. Unfortunately, the Great Chaos Minor Renovation of '79 swallowed up my only copy of the original, full-bore ELIZA. No big loss, but it was an interesting hack.

Programming Technique

Multiple Keystroke INKEY$ Routine for Model 3 & 4

By Bill Harrison

Basic's INPUT command has a couple of severe limitations.

First, when the ENTER key is pressed, everything to the right of the cursor on the input line is erased. Normally this is not noticeable, but if you are trying to pretty up your program with, for example, a border, it certainly presents a problem. The moment the ENTER key is pressed in response to an INPUT, the border will have a hole in it. Definitely not a professional touch.

Second, you have no control over how many characters the user will type. For example, you may have prompted the user to type in his or her name, and positioned the INPUT on line 2, column 40. If the user types too long a response, it will wrap around to the next line. Not professional either.

To solve the above two problems, I wrote a short subroutine using INKEY$ in place of INPUT.

It has the further advantage of being able to lock out undesirable keys. To demonstrate, I locked out the two 'killer' keys: RIGHT ARROW and DOWN ARROW.

The subroutine stretches from line 10 to line 55.

Line 1 jumps over subroutines at the start of the program. In this case the real program starts at line 100.

Line 100 erases screen

Line 110 sets up the position of the prompt. PO = 128 (model 4 use PO = 160). Then it sets up the actual prompt in PR$. PO is then altered to reflect the length of the prompt and the maximum characters allowed for the user to type is stored in ML. In our case we will allow up to 16 characters. (ML = 16). Then we use the subroutine in line 10.

Line 10 makes sure that A$ is blank and the current length of the input is 0. (L = 0). Then, as many periods as allowable input characters are placed at the input position.

Line 20 is the standard INKEY$ routine. Waits for a keypress. If no keypress, back to line 20.

Line 30 RETURNs only if ENTER is pressed.

Line 35 locks out RIGHT ARROW and DOWN ARROW.

Line 40 locks out LEFT ARROW (backspace) if no input has taken place.

Line 50 checks if LEFT ARROW (backspace) was pressed. If so, decrement length of string by 1 (L = L-1). Overwrite previous character with a period. (PRINT@PO + L,CHR$(46)) and make A$ contain the altered string. Then go back to the INKEY$ routine in line 20.

Line 55 checks if the character count (L) has reached the maximum (ML). If so, do not accept current keystroke. Instead go back to INKEY$ routine. NOTE: At this point, the routine will only accept 2 keys: LEFT ARROW or ENTER.

Line 60 By default the keypress is allowed, so print the character at the current input position. Expand A$ to include the character. Increment the length of the string by 1 and go back to the INKEY$ routine to get more keystrokes.

```
 1 GOTO 100
 10  A$ = "":L = 0: PRINT@PO,STRING$(ML,46)
 20 I$ = INKEY$: IF I$ = "" THEN 20
 30 IF I$ = CHR$(13) THEN RETURN
 35 IF I$ = CHR$(9) OR I$ = CHR$(10) THEN 20
 40 IF I$ = CHR$(8) AND L = 0 THEN 20
 50 IF I$ = CHR$(8) THEN L = L-1: PRINT@PO + L,
CHR$(46): A$ = LEFT$(A$,L): GOTO 20
 55 IF ML = L THEN 20
 60 PRINT@PO + L, I$;: A$ = A$ + I$: L = L + 1:
GOTO 20
```

Your program goes here. For example:

```
 100 CLS
 110 PO = 128:PR$ = "TYPE YOUR NAME: ":
PRINT@PO,PR$;:  PO = PO + LEN(PR$): ML = 16:
GOSUB 10
```

Hope you find it useful.

Bill H.

CATTING around with LDOS 5.1.4

by Lance Wolstrup

Model 3 - LDOS - Editor Assembler

Model 4 provides two DOS commands to obtain disk directories. DIR and CAT.

DIR, of course, produces the long listing with all the specific information about the files.

CAT gives only the filenames. They are displayed 5 across, allowing you to view most, if not all, of the disk files in one screenful. It is a very convenient command.

Model 3's LDOS 5.1.4 has only the DIR command, and since I use that operating system frequently, I concocted a very short program to provide me with the CAT function.

Get out your trusty old EDTASM, or other editor assembler, and type in the program listing. Then assemble it as CAT/CMD.

Whenever you now need just a listing of the filenames, simply type **CAT :d** where **d** is number of the drive you wish to access.

If your chosen disk contains more files than the screen can display, it will fill the screen and then stop for your perusal. Pressing any key will continue the file listing.

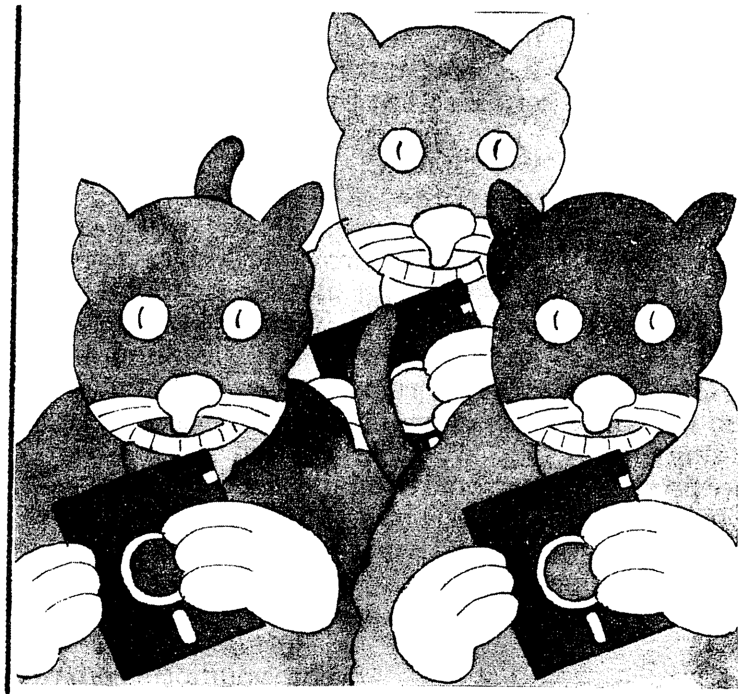
CAT is capable of addressing 8 drives (0 to 7), however, if you do not have that many and you try to access a nonexistant drive, the error message **Drive is not available** will be displayed and you will be returned to LDOS. This same message will greet you if the chosen drive is not ready.

Errors resulting from incorrect syntax will not produce a message, instead an immediate return to LDOS will occur.

To keep the program short and simple, I chose to allow only the drive number to be passed to the program. Therefore it is not able to diplay invisble or system files. The available disk space information also has purposely been left out.

The other tradeoff I had no control over. Because Model 3's screen width is 64 as opposed to 80 on the Model 4, only four filenames will be displayed per line.

Try CAT, I think you'll like it.



CAT/CMD

```
00100          ORG   7000H
00110 START    LD    HL,4228H
00120          LD    A,(HL)
00130          CP    32
00140          JR    NZ,QUIT
00150          INC   HL
00160          LD    A,(HL)
00170          CP    58
00180          JR    NZ,QUIT
00190          INC   HL
00200          LD    A,(HL)
00210          SUB   30H
00220          CP    8
00230          JR    NC,QUIT
00240          LD    C,A
00250          CALL  4209H
00260          JR    NZ ,ERROR
00270          LD    B,0
00280          CALL  4419H
00290 QUIT     RET
00300 ERROR    LD    HL,ERMSG
00310          CALL  4467H
00320          JR    QUIT
00330 ERMSG    DEFM  'Drive is not available'
00340          DEFB  13
00350          END   START
```

Here's looking at:

ALLWRITE

GIANT OF TRS-80 WORD PROCESSORS

Review/Tutorial by PAUL R. VEST

The TRS-80 market has seen a number of word processors come and go. ALLWRITE, for Models III, and IV, stands head and shoulders above them all.

Some years ago I paid $195 for my copy of Allwrite and it has been worth every dollar. Today, at a cost of less than half the original price, it is a great investment for your Model III or IV.

Installing ALLWRITE

Installation is easy. Have two formatted disks ready and insert the master disk in Drive 0. The destination disk goes in Drive 1.

Type ALINSTAL and hit <ENTER>.

The program takes over and prompts you whenever you need to do something. Just carefully follow the directions on the screen.

Looking through the list of more than 60 printers can be confusing to a newcomer, but take your time and go through the list one or more times until you find your printer.

By hiting the <ENTER> key you can return to the top of the list and set the program for a second and a third choice of printer.

Hitting the asterisk <*> will exit the printer portion of the program and proceed to the next section.

If you are not satisfied, you may at any time press the <BREAK> key. This cancels the installation procedure and returns to DOS.

Select printer or printers by typing the associated number followed by <ENTER>.

You will then be prompted to tell ALLWRITE whether the selected printer is attached to a parallel or serial port.. Normally you will choose the default setting, which is the parallel port. You can do this by merely pressing <ENTER>.

Next you will be asked if you wish printing pause after each page. Answer No if you use continuous paper, and Yes if you use single sheet paper.

You are then asked about linefeeds. The answer depends on your printer and its dip-switch settings. If it provides its own linefeeds, answer No, otherwise answer Yes.

This ends the printer installation portion and you will now be asked about the EDITOR characteristics.

Most of these choices are not really critical, so choose the default setting by pressing <ENTER>, however, at the screen--width prompt, Model III owners will want a 64 character screen and Model IV owners will be able to make a choice between 64 and 80 character screen-widths.

The AUTOSAVE feature allows you to set the number of changes you will make to your document before the program will automatically save your file. This updating of your files, as you write, can save you much grief if you have a power failure or some other problem.

By setting the AUTOSAVE-number to 100 or less you would lose very little of your document.

If you are new, or a bit slow at typing on the keyboard, keep the repeat speed low. On the other hand , if you are a fast typist, you can set the repeat key speed to a faster rate.

To the DISK WRITE VERICATION and CLICK prompts, choose the default values.

Using an OPTIONAL KEYBOARD DRIVER depends on the DOS. Type No if you use TRSDOS 6.x. and Yes if DOSPLUS 4 or any Model III DOS.

DOS selection is next. Enter your choice by typing the number next to the DOS you are using, then press <ENTER>.

The final prompt allows you to save your answers. Type Yes to save, No to restart or press <BREAK> to return to DOS without saving the answers.

Once you have finished the installation program, be sure to make a least one backup copy of the disk. Put your master disk away safely in another room.

GETTING DOWN TO BUSINESS

DOSPLUS 4 and Model III users can enter the program by typing: **ALK AL filename** and press <ENTER>. TRSDOS 6.x users can simply type **AL filename..**

If the chosen file exists, ALLWRITE will load it into memory. If not, a message will appear telling that it is a new file, and asking if you want to create this file. Answer Yes and a blank screen will appear with a cursor in the upper left corner.

First of all, keep in mind that the <CLEAR> key is always used as the CONTROL key.

You can now type your text and need only hit the <ENTER> key at the end of a paragraph.
To open up a line anywhere within the text, type <CLEAR> N.

ALLWRITE lets you write a file with about 24K capacity for a Model III and in a Model IV with 128K you can have a file with about 34K in memory. However, by APPENDING program files together, you can actually write a book.

The clearly written, well-organized manual will help you to quickly start using ALLWRITE in a productive manner. But you will find, even after a year or so of experience, that you still have not used all of the possible features of this versitile word processor.

EDITING

The three basic editing commands, REPLACE, INSERT and DELETE are extremely easy to use.

REPLACE (overtype existing text) is the default. You need only move the cursor (by pressing the appropriate arrow keys) to where you wish to replace text, and then type the replacement text.

To INSERT, move the cursor to where you wish to add text, press <CLEAR> I, and type away. To escape INSERT mode, press <CLEAR> I again.

Should you need to DELETE a character or series of characters, move the cursor to where the deletion is desired and press **<CLEAR> D**.

FORMATTING COMMANDS

Two letter code commands are placed on **separate lines** from the text to allow the formatting program to, for example, correctly set margins and other miscellanous goodies:

;lm3 means left margin starts at column 3.
;tm2 means top margin starts at line 2.
;ce3 means center the next 3 lines.
;in5 means indent five spaces.
;pp means skip line and start new paragraph.

All two letter code commands must start with a semi-colon (;).

Because the program's authors used logical choices for formatting code commands, it is just a short time until you are doing them from memory. For those who need it, however, there is a HELP FILE available listing many of the possible commands.

ALLWRITE allows the user to print the text in columns. To do so, you enter the two-letter code command **;CB** and the program will be formatted with two columns equally divided on whatever line length you set at the top of your document. The number of columns are only limited by the line length. ALLWRITE, by itself, will first format in memory and then print the columns without having to reverse feed the paper.

BLOCK MOVES are a snap. Hit <CLEAR> E and a special cursor shape will mark the beginning of the section you wish to move. Place the cursor at the end of the section to be moved or copied, hit <CLEAR> C, move the cursor to your destination and hit <CLEAR> H (for COPY HERE) and your paragraph or line is moved very quickly.

ALLWRITE can be interfaced with a spelling checker which can save the writer a large amount of time, energy and embarrassment.

PRINTING THE DOCUMENT

Before actually sending the text to the printer, it can be previewed on the screen by hitting SHIFT <CLEAR> 2.
You will be asked to enter the filename. Do so and to the prompt 'ENTER OPTIONS', answer: ;VI <ENTER>
The **;VI** command (video) causes the document to be printed to the screen. The Model IV version allows you to see underlined words and other print time options you might have included. Any errors will be noted and you can back out of the preview and make corection (by pressing the <BREAK> key).

Printing your file easy. Simply press SHIFT <CLEAR> 1. The formatter program will load, ask for the filename and then proceed to print.
Should more than one printer be installed, the alternate printer can be selected by hitting the <BREAK> key and typing PR2 (for PRINTER # 2).
Hitting the <BREAK> key again and than typing **ST** (for status) will give you a screen which tells you the size of your file, the program name, the printer the program is set for, the screen width and more. This is one way you can be sure you have the program set for the correct printer.
For example, I use a daisy wheel printer frequently so I simply check the STATUS SCREEN before printing to make sure that the correct printer is set.

Hitting the SHIFT <CLEAR> 1 keys will begin the printing of your document. ALLWITE will use whatever features your printer has, providing you have placed the proper code commands on a separate line at the point where you want them used.

A short sample file.

;ll65
(set a Line Length of 65 characters.)
;lm8
(set Left Margin at 8/10 inch at 10 CPI)
;tm6
(Top Margin set at 6 lines down from top)
;bm6
(Bottom Margin 6 lines from bottom)
;ce on
(turn on centering)
@@$DAISY WHEEL PRINTERS@%
(@$ = turn on underline - @% = turn off underline)
;sk2
(SKip 2 lines
;CE OFF
(Turn off CEntering)
;pp
(new paragraph - also indents 5 characters.)

Here is where the body of the text would go. It would on and on, using many lines and then end.
;EN
(ENd the file)

The above commands, and all others for that matter, can be typed in any combination of upper and lower case. ALLWRITE will understand them regardless.

Hanging indents, line numbering, column printing - all is possible, depending on how the CODE COMMANDS are used within the file.

Form letters are a breeze when using ALLWRITE. Typing the same information repeatedly, is a waste of time. By using a small file with the information you need to use again and again, you can imbed (;IM) a file within your larger document. ALLWRITE will read the ;IM code, search out the file to be imbedded, and print the information within that file. At the end of the imbedded file, ALLWRITE continues with the original file.

If you are writing a document larger than about 34K with Model IV, (approx. 20K for Model III), you can chain files together by using the APPEND (;AP) command at the end of the first file.
ALLWRITE will then just continue reading in as many files as you command. It will keep chaining the specified files until it encounters an ;EN code command.

If you are a beginner in computing, ALLWRITE will allow you to organize and speed up your word-processing skills.
For the intermediate, advanced and especially the 'SUPER' user, ALLWRITE will allow you to use your talents to the fullest and create printed files that you can view with pride..

Perhaps the truest test of a program is the support that its authors give you if there is a need. The talented people who answer the phones at Prosoft are a wonderful source to the properly registered owners
I have never had better service from any other company.

With its capactiy to expand to your needs (special drivers can be ordered for laser printers), ALLWRITE would be a good investment on anyone's part. I highly recommend it.
It is truly the GIANT of TRS-80 word processors.

P V

ALLWRITE is a product of:

PROSOFT
Dept. C.
Box 560
No. Hollywood, Ca. 91603

Do your TRS-80
a favor

Subscribe to TRSTimes

Get all six 1988 issues
for $15.00 (US)

or

Single issues available
for $3.00 (US)

Send to:

TRSTimes
20311 Sherman Way #221
Canoga Park, Ca. 91306

Teaching the kids to spell

By Bill Harrison

Model 3 - Basic

Being a parent, I face the same situation every Monday evening while school is in session. My kids bring home their weekly spelling words, which they have to learn by Friday.

I wish I had a Dollar for each time I missed Monday Night Football listening to them stammer through one and two syllable words.

Finally I got smart. I decided to take advantage of the fact that they love to play on my computers, and I wrote a program to help them learn their words. I call it 'SPELLING TUTOR' and it works this way.

When the spelling words are brought home, the parents should use option 1 from the menu to input the words. Tell the program how many words you wish to enter and then proceed to do so. Don't worry if you make a mistake. When all the words have been entered you will have an opportunity to correct any words you wish. Type 0 when you feel they are correct. At this point the spelling words will be written to a disk file named 'WORDLIST.DAT' and you can now turn it over to the kids. They should use option 2 from the menu.

The words will be loaded in from the disk and they will be presented with a word selected at random on the screen for a brief moment. It will then be erased and they will be asked to type it. The program keeps track of how well they do, and when they decide to quit, a report card screen will be presented.

Have fun with it, my kids did.

SPELLING TUTOR

```
0 'SPELLING TUTOR - <c> 1987 Bill Harrison
1 CLEAR 500: DEFINT A-Z: DIM WD$(20): QA = 0: CO = 0: CU$ = CHR$(176)
2 FOR X = 1 TO 10: READ WR$(X): NEXT: DATA You can do better than that!, That was a bad answer!, "No good, that is wrong!", You must try harder!, Are you trying to guess the answers?, "Nope, that's not the right one!"
3 DATA Wrong answer - try to get the next one right!, C'mon now - is that the best you can do?, Poor spelling!, You blew it!
4 FOR X = 1 TO 10: READ CO$(X): NEXT: DATA Bravo - that was right!, Sensational answer!, Very good - you got that one, Dynamite - You are a good speller, "Hey, Hey, Hey, perfect!", Great answer - are you cheating?
5 DATA You are good at this - the answer is correct!, Nice job - you got it!, That was a good answer!, Wonderful - you are getting good
8 GOTO 100
10 LO = PO * 64 + INT((64-LEN(A$))/2): PRINT@LO, A$;: RETURN
20 I$ = INKEY$: IF I$ = "" THEN R = RND(W): GOTO 20 ELSE I = VAL(I$): RETURN
30 WD$(X) = "": L = 0: ML = 15
31 I$ = INKEY$: IF I$ = "" THEN PRINT@PO + L, CU$;: GOTO 31 ELSE IF I$ = CHR$(13) THEN IF L < ML THEN PRINT@PO + L, CHR$(46);: RETURN ELSE PRINT@PO + L, CHR$(32);: RETURN
32 IF I$ = CHR$(9) OR I$ = CHR$(10) THEN 31
33 IF I$ = CHR$(8) AND L = 0 THEN 31
34 IF I$ = CHR$(8) THEN L = L - 1: WD$(X) = LEFT$(WD$(X),L): IF L + 1 < ML THEN PRINT@PO + L + 1, CHR$(46);: GOTO 31 ELSE PRINT@PO + L + 1, CHR$(32);: GOTO 31
35 IF ML = L THEN 31
36 PRINT@PO + L, I$;: WD$(X) = WD$(X) + I$: L = L + 1: GOTO 31
```

MAIN MENU

```
100 CLS: PO = 0: A$ = " Spelling Tutor": GOSUB 10: PRINT@0, "TRSTimes Presents:"; @48, "TRS-80 Model III"; STRING$(64,131)
110 PO = 5: A$ = "1. Input Spelling Words": GOSUB 10: PO = 7: A$ = "2. Spelling Tutor": GOSUB 10: PO = 9: A$ = "3. Exit       ": GOSUB 10: PO = 12: A$ = "Select 1, 2, or 3": GOSUB 10
120 GOSUB 20: IF I < 1 OR I > 3 THEN 120 ELSE ON I GOTO 1000, 2000, 3000
```

PARENTS MODULE

```
1000 PRINT@0, STRING$(64,32);: PO = 0: A$ = "Input Spelling Words": GOSUB 10
1010 F = 0: PRINT@192, CHR$(31): PRINT@192, "How many words will you input (20 is maximum) ";: W$ = "": INPUT W$: IF W$ = "" THEN 100 ELSE W = VAL(W$)
1020 IF W < 1 OR W > 20 THEN 1010 ELSE PRINT@192, CHR$(31)
1030 PO = 192: FOR X = 1 TO W: PRINT@PO, "Word #"; USING "##"; X;: PRINT " "; STRING$(15,46);: PO = PO + 64
1040 IF F = 1 THEN 1050 ELSE IF X > 9 THEN F = 1: PO = 224
1050 NEXT
```

```
  1060 F = 0: PO = 202: FOR X = 1 TO W:
GOSUB 30: PO = PO + 64
  1070 IF F = 1 THEN 1080 ELSE IF X > 9 THEN
F = 1: PO = 234
  1080 NEXT
```

EDIT THE SPELLING WORDS

```
  1100 PRINT@0, STRING$(64,32);: PO = 0: A$
= "Edit Spelling Words": GOSUB 10
  1110 PRINT@192, CHR$(31)
  1120 F = 0: L = 192: FOR X = 1 TO W:
PRINT@PO + L, USING "##"; X;: PRINT "- ";
WD$(X);: L = L + 64
  1130 IF F = 1 THEN 1140 ELSE IF X > 9 THEN
F = 1: L = 224
  1140 NEXT
  1150 PO = 14: PRINT@PO * 64, CHR$(31): A$
= "Type number of word to edit - type 0 when cor-
rect": GOSUB 10
  1160 INPUT X$: IF X$ = "0" THEN 1200 ELSE X
= VAL(X$): IF X < 1 OR X > W THEN 1150
  1170 PRINT@PO * 64, CHR$(31): PRINT@PO *
64, "Change: "; WD$(X);: PRINT@PO * 64 + 68,
"To: ";: WD$(X) = "": PO = 15 * 64 + 8: L = 0:
ML = 15: PRINT@PO, STRING$(15,46);: GOSUB
31
  1180 IF X <10 THEN L=128+X*64 ELSE
L=160+(X-10)*64
  1190 PRINT@L + 4, STRING$(15,32);:
PRINT@L + 4, WD$(X);: PRINT@PO - 64,
CHR$(31): GOTO 1150
  1200 PRINT@0, STRING$(64,32);: PO = 0: A$
= "Saving Spelling Words to disk": GOSUB 10:
PRINT@14 * 64, CHR$(31)
  1210 OPEN"O",1,"WORDLIST/DAT"
  1220 PRINT#1,W
  1230 FOR X = 1 TO W:PRINT#1, WD$(X): NEXT
  1240 CLOSE#1
  1250 GOTO 100
```

CHILD'S MODULE

```
  2000 ON ERROR GOTO 2800
  2010 PRINT@0, STRING$(64,32);: PO = 0: A$
= "Loading Spelling Words from disk": GOSUB 10:
PRINT@192, CHR$(31)
  2020 OPEN"I",1, "WORDLIST/DAT"
  2030 INPUT#1,W
  2040 FOR X = 1 TO W: INPUT#1, WD$(X):
NEXT
  2050 CLOSE#1
  2060 ON ERROR GOTO 0
  2070 PRINT@0, STRING$(64,32);: PO = 0: A$
= "Spelling Tutor": GOSUB 10: PRINT@1, "Words:
"; USING "###"; QA;: PRINT@51, "Correct: ";
USING "###"; CO;
  2100 PO = 14: A$ = "Press any key when
ready": GOSUB 10:GOSUB 20: PRINT@192,
CHR$(31)
  2110 R = RND(W): RR = RND(10): PO = 6: A$
= "The word is: " + WD$(R): GOSUB 10
  2120 FOR X = 1 TO 1500: NEXT
  2130 PRINT@PO * 64, STRING$(LEN(A$),32):
A$ = "Now you spell it: ": GOSUB 10
  2140 INPUT AN$:IF AN$ = WD$(R) THEN 2200
  2150 PO = PO + 2: A$ = WR$(RR): GOSUB
10: PO = PO + 2: A$ = "The word was " +
WD$(R): GOSUB 10: GOTO 2300
  2200 PO = PO + 2: A$ = CO$(RR): GOSUB
10: PO = PO + 2: CO = CO + 1
  2300 QA = QA + 1: PRINT@8, USING "###";
QA;: PRINT@60, USING "###"; CO;
  2310 PO = 14: A$ = "Press ENTER when
ready - or Q to quit": GOSUB 10
  2320 GOSUB 20: IF I$ = CHR$(13) THEN
PRINT@192, CHR$(31): GOTO 2110 ELSE IF I$ =
"Q" OR I$ = "q" THEN 2400 ELSE 2320
```

REPORT CARD & END

```
  2400 PRINT@0, STRING$(64,32);: PO = 0: A$
= "Report Card": GOSUB 10: PRINT@192,
CHR$(31)
  2410 PO = 4: A$ = "You answered" +
STR$(CO) + " correctly out of" + STR$(QA):
GOSUB 10
  2420 PE = INT((CO/QA) * 100 + .5)
  2430 PO = 6: A$ = "You scored" + STR$(PE)
+ "%": GOSUB 10
  2440 IF PE = 100 THEN GD$ = "A": CM$ =
"You are a genius": GOTO 2500 ELSE IF PE > 90
THEN GD$ = "A -": CM$ = "Almost perfect -
Good job": GOTO 2500 ELSE IF PE > 84 THEN
GD$ = "B": CM$ = "You did well": GOTO 2500
  2450 IF PE > 80 THEN GD$ = "B -": CM$ =
"Not too bad - try again": GOTO 2500 ELSE IF PE
> 74 THEN GD$ = "C": CM$ = "That was just
average": GOTO 2500 ELSE IF PE > 69 THEN
GD$ = "C -":CM$ = "Below average - you must try
harder": GOTO 2500
  2460 IF PE > 64 THEN GD$ = "D": CM$ =
"You did not do well - tr to improve": GOTO 2500
ELSE IF PE > 59 THEN GD$ = "D -": CM$ =
"That was terrible - Can't you do better than that?":
GOTO 2500
  2470 GD$ = "F": CM$ = "You flunked - prepare
to go back to reform school"
  2500 PO = 8: A$ = "Your grade is: " + GD$:
GOSUB 10: PO = 10: A$ = CM$: GOSUB 10: PO
= 14: A$ = "Press any key to quit": GOSUB
10:GOSUB 20: GOTO 3000
  2800 PRINT@0,STRING$(64,32);: PO = 0: A$ =
"*** Error has occurred ***": GOSUB 10: PO = 4:
A$ = "The file named WORDLIST/DAT was not
found": GOSUB 10
  2810 PO = PO + 2: A$ = "It can be created by
using option 1 from the menu": GOSUB 10
  2820 RESUME 2830
  2830 ON ERROR GOTO 0
  2840 PO = PO + 4: A$ = "Press any key for
menu": GOSUB 10: GOSUB 20: GOTO 100
  3000 CLS: END
```

The Hackers' group is made up of dedicated TRS-80 enthusiasts from the Greater Los Angeles area. They come from varied backgrounds, have different skills, interests and even different ideas about what to do with their machines. But they have one important thing in common: They are willing and eager to share their knowledge.

So if you have a question about Model 3 or 4,

Ask the Hackers.

Q. I don't understand the use of Macros in assembly language. Would you lead me through creating and using a Macro?

Jim Savage
Clinton, MS.

A. First let's make it clear that a Macro is NOT a Z-80 instruction. It is an instruction to the program you are using to create machine language, the editor/assembler. It is known as a pseudo-op and like all other pseudo-ops, it does not produce Z-80 code, instead it tells the editor/assembler to do something special.

EDAS, ALDS, and a few others are capable of Macros. EDTASM is NOT.

Many people confuse a Macro with a subroutine, as they at first glance seem very much alike. Let's look at how they differ.

A subroutine is a complete piece of Z-80 code already written into the program by YOU. When it is needed you CALL it from the main body of the program. The code produced by the assembler will be CD, followed by the address of the subroutine. The subroutine never changes.

A Macro is not CALLed. Whenever the assembler sees a reference to the Macro, it will write the entire series of instructions you defined as being the Macro. The assembler will do this each and every time it sees the reference.

You are probably thinking, "Why put a series of instructions in a Macro when using the same instructions in a subroutine would save memory?"

Good question! The answer, of course, is that the Macro has an added ability. It is able to pass certain parameters so that the code it writes is actually changeable. This is very powerful and can save the programmer an immense amount of time.

To illustrate let us write a small program that will:

1.) Erase the screen.
2.) Position the cursor at 10th line, 15th column.
3.) Display the message 'TRSTimes'.
4.) Position the cursor to 15th line, 8th column.
5.) Display the message 'January 1988'
6.) Return to DOS.

First let's write the program using subroutines:

```
@VDCTL EQU    15
@DSPLY EQU    10
@EXIT  EQU    22
       PSECT  3000H
START  CALL   0545H    ;undocumented CLS
       LD     H,10     ;vertical pos of cursr
       LD     L,15     ;horiz pos of cursr
       CALL   LOCATE   ;position cursor
       LD     HL,MSG1;point to msg1
       CALL   DSPLY    ;put msg1 on screen
       LD     H,15     ;vertical pos of cursr
       LD     L,8      ;horiz pos of cursr
       CALL   LOCATE   ;position cursor
       LD     HL,MSG2;point to msg2
       CALL   DSPLY    ;put msg2 on screen
       LD     HL,0     ;indicate no error
       LD     A,@EXIT  ;and return
       RST    40       ;to DOS
LOCATE LD     B,3      ;VDCTL function #
       LD     A,@VDCTL ;SVC #
       RST    40       ;position cursor
       RET             ;return to caller
DSPLY  LD     A,@DSPLY ;SVC #
       RST    40       ;msg to screen
       RET             ;return to caller
MSG1   DEFM   'TRSTimes'
       DEFB   13
MSG2   DEFM   'JANUARY 1988'
       DEFB   13
       END    START
```

Now the same program using a Macro that passes the correct SVC number each time it is written.

```
@VDCTL EQU    15
@DSPLY EQU    10
@EXIT  EQU    22
SVC    MACRO  #0       ;define macro and
       LD     A,#0     ;pass the equated
       RST    40       ;parameter to SVC
       ENDM            ;end of macro
       PSECT  3000H
START  CALL   0545H    ;undocumented CLS
       LD     B,3      ;@vdctl function #
       LD     H,10     ;vertical pos of cursr
       LD     L,15     ;horiz pos of cursor
       SVC    @VDCTL;use macro
       LD     HL,MSG1;point to msg1
       SVC    @DSPLY;use macro
       LD     H,15     ;vertical pos of cursr
       LD     L,8      ;horiz pos of cursr
       SVC    @VDCTL;use macro
       LD     HL,MSG2;point to msg2
       SVC    @DSPLY;use macro
       LD     HL,0     ;indicate no errors
       SVC    @EXIT    ;use macro
MSG1   DEFM   'TRSTimes'
       DEFM   13
MSG2   DEFM   'JANUARY 1988'
       DEFB   13
       END    START
```

CLOSE#1

This brings issue 1 of TRSTimes to a close and it has been some kind of a learning experience. Wearing the hat of publisher and editor was quite a thrill, and I would like to voice a few observations from each position.

Many people have intended to provide information for the TRS-80. Some of these people let us down by distributing ads for magazines or some other media, took our subscription money, and then never delivered the product. Some gave us a few issues and then, when they could not continue, covered the balance of our subscription by sending us some magazine we didn't really want. This hurt us all.

As publisher, I do not want TRSTimes to emulate this shoddy way of doing business. Therefore this newsletter is keeping promises down to a level that CAN and WILL be met:

1. TRSTimes WILL be published bi-monthly in 1988. That means 6 issues. No more and NO LESS. No matter when you subscribe, you'll get all six issues for that year.

2. A new issue WILL be mailed the first Post-Office-business-day of January, March, May, July, September and November.

3. IF there is enough interest and IF it is FINANCIALLY feasible, TRSTimes WILL continue with 6 issues in 1989 following rules, 1 and 2. In essence, TRSTimes will be run using a revolutionary concept: 'OLD FASHIONED INTEGRITY.'

Enough said.

As editor, let me sound a cry for help.

TRSTimes NEEDS YOUR articles, programs, hints & tips, letters, criticism, suggestions, ideas, etc.

The guidelines are flexible: Anything pertaining to the TRS-80 Model 3 or 4 will be considered.

At this time, renumeration for submissions is not possible nor can unused material be returned. The budget simply does not allow it.

What I can guarantee though, is that each and every submission will be read and carefully evaluated. When a submission is used, FULL credit will be given to the author.

The copyright of unused submissions will be respected. In other words, it will not appear in its original form or modified, with someone else getting the credit for it.

The reason your input is desperately needed is not hard to understand. For TRSTimes to survive (or for that matter, the TRS-80 itself), the remaining loyal inhabitants of the TRS community must band together and help one another. Enough articles and programs to fill six small 1988 issues are sitting on my desk. However, that material is the product of only a handful of people, and while all of it will be used eventually, my preference is to balance each issue with as much variety as is possible. To accomplish this, it is imperative to have the input of more than a few.

With this in mind, let me introduce the ones who birthed this first issue:

The BBS column was written by, none other than 'the Guru of BBS', Timothy Sewell.

Those of you who frequent the TRS-80 section on GEnie know Tim already as the Sysop of that board. Under his guidance it has become the largest TRS-80 board in the country and probably the world. When Tim speaks about BBS and public domain software, you can be sure, I listen. He has promised regular input and involvement in these pages and we will all be the richer for it.

Bill Harrison is a friend from the old Model I days in Tampa, Fla. He is a collector of computers.

As of the last count, he informs me, 21 of them are sitting in his basement computer room. They include 7 Model 1's (all with very low serial numbers), Model 3's, 4 & 4P's, a couple of CoCos, a Model 16, some PC clones and even, heaven forbid, an APPLE IIe and a C-64.

His 'Multiple Keystroke INKEY$ Routine' and the Model 3 'Spelling Tutor' are examples of how he likes to program. Bill has a subroutine for just about anything. We will be seeing more of his work in future issues.

'COMPUTING AT CHAOS MANOR' was written by Eric Bagai who has been a librarian, systems analyst, special education diagnostician, publisher, information retrieval consultant and president of the Valley TRS-80 Hackers' Group. He now edits psychophysical and psychological tests

The author of the Review/Tutorial column is Paul Vest. He is a schoolteacher and a devoted TRS-80 user and enthusiast. He will be talking about software that never received the media attention it deserved.

Lastly, you may blame me for 'HUNTING FOR BURIED TREASURE'. It is the first installment of a continuing series of tinkering with TRSDOS 6.2. I also admit to 'CATTING AROUND WITH LDOS 5.1.4.

My background is one of pursuing a multitude of interests. From professional soccer player, full time musician, to being on the Board of Directors of a nationwide motel chain, I got hooked on computers in the late seventies and this obsession has stayed with me to this day, where I teach programming languages in the Greater Los Angeles area.

Next issue will bring more PEEKS and POKES for Model 4, a program that will allow easy copying back and forth between NEWDOS/80 and the other DOS'es, another observation from Eric and much more.

Until March, keep those TRS-80's humming.

LANCE W.