TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

MARCH 2026 - VOLUME 20 ISSUE 1

PAGE 02 A BIG THANK YOU FROM THE EDITOR

PAGE 04 FIXING THE ALTERNATE ASTEC PSU
PATRICK BUREAU

PAGE 08 CODEX-80
E.T. FONEHUME
2 BASIC ONE-LINERS HOT ON THE HEALS OF ENIGMA `
(JUST)!!

PAGE 12 MAV'S QUARTERLY T.A. UPDATE
IAN MAVRIC
MAV WET OUR APPETITE FOR THE NEW VENUE FOR 2026

PAGE 18 THE POWER OF THE ROM
VINCE OTTEN
PART 5 OF VINCE'S BRILLIANT SERIES 'CUT!, PRINT!'

PAGE 46 IN MAV'S HISTORY WORKSHOP
IAN MAVRIC
WHO BOUGHT A MODEL 2000, PART 3

PAGE 56 PROGRAMING LANGUAGES FOR THE MODEL1 - LISP
VINCE OTTEN
VINCE LOOKS AT EARLY ATTEMPTS, WITH AN EMULATOR
IN BASIC

PAGE 66 NEWS FROM FRANCE
PASCAL HOLDRY
NEW_JFD-CP

PAGE 69 THE WAY WE WERE
A LOOK 'BACK AT THE DAY'
LOWERCASE FOR $5 ? + CGP115 REVIEW

PAGE 77 'COUNTDOWN'
A 30 SECOND PUZZLE TO GET THE OLD GREY MATTER
WORKING

PAGE 78 IN MAV'S WORKSHOP
IAN MAVRIC
KEYBOARDS ROUNDUP FOR THE MODELS 1, 3 AND 4

PAGE 85 SURFIN' THE NET
WOW! TRS-IO OVERVIEW

PAGE 86 BASQUE TRS-80 TOURNAMENT
EGOITZ CAMPO
DETAILS OF THE LATEST BASQUE TRS-80 COMPETITION

THANK YOU !



I COULDN'T LET THE START OF TRS8BIT'S 20TH YEAR PASS WITHOUT SAYING A VERY BIG, MINION STYLE, THANK YOU TO EVERYONE WHO HAS CONTRIBUTED IN MAKING THE NEWSLETTER SUCH A SUCCESS!

I'M AWARE THAT TECHNICAL DIFFICULTIES WITH ARTICLE CONVERSION, FOR M/S PUBLISHER, IS CREATING PROBLEMS WITH LANGUAGE TRANSLATION ON SOME OF THE PAGES THAT ARE IMAGES, (VIZ. .JPG ETC.). IT REALLY IS A BIT OF A NIGHTMARE WITH DETAILED TECHNICAL ARTICLES.

SO IF YOU ENCOUNTER SUCH PROBLEMS, PLEASE CONTACT ME, AT THE USUAL EMAIL ADDRESS AND I'LL SEND YOU A COPY OF THE ORIGINAL .PDF FILE.

I FEEL THE QUALITY OF THE ARTICLES I RECEIVE DEMAND THE BEST POSSIBLE DISPLAY ON PUBLICATION.

DUSTY - MARCH 2026

LATEST UPGRADE
VERSION 2.5.6

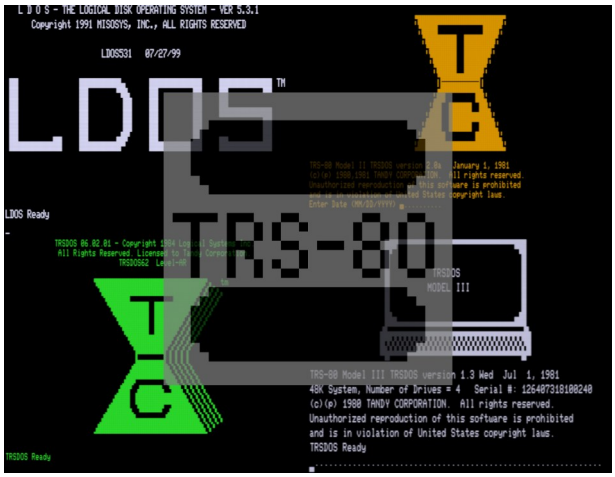
DON'T HAVE A TRS-80?

CRAZY PRICES ON EBAY?

TIRED OF RIFA CAPS BLOWING?

OF FLOPPIES SHEDDING OXIDE?

OR JUST PLAIN DEAD CHIPS?





THEN WHAT YOU NEED IS TRS80GP.
THE ULTIMATE TRS-80 EMULATOR



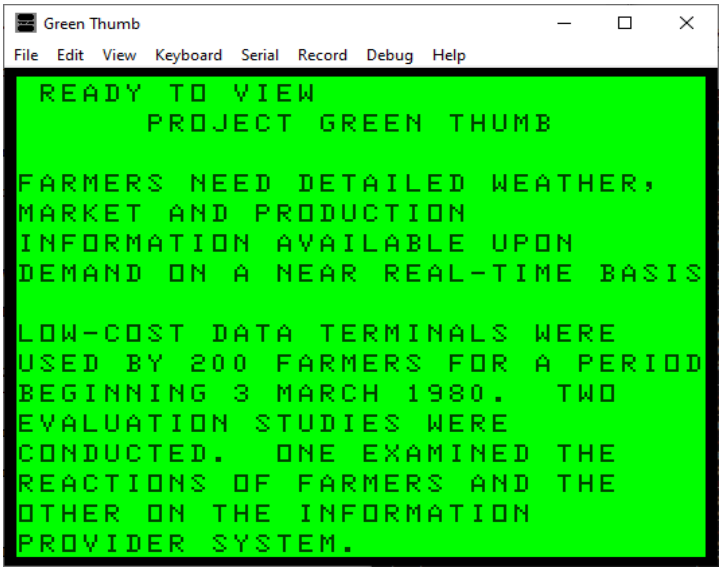
RUNS ON WINDOWS, MACOS, LINUX & RASPBERRY PI.
EMULATES MODELS 1, 2, 3, 4, 12, 16, 6000, MC-10,
COLOR COMPUTER, DT-1, VIDEOTEX, TANDY 10 AND NOW 'GREEN THUMB'.

JUST LIKE A REAL TRS-80 WITH NONE OF THE HASSLE.
BUILT-IN DEBUGGER AND OTHER FEATURES, JUST GREAT FOR PROGRAMMERS.

GET IT NOW AT HTTP://48K.CA/TRS80GP.HTML

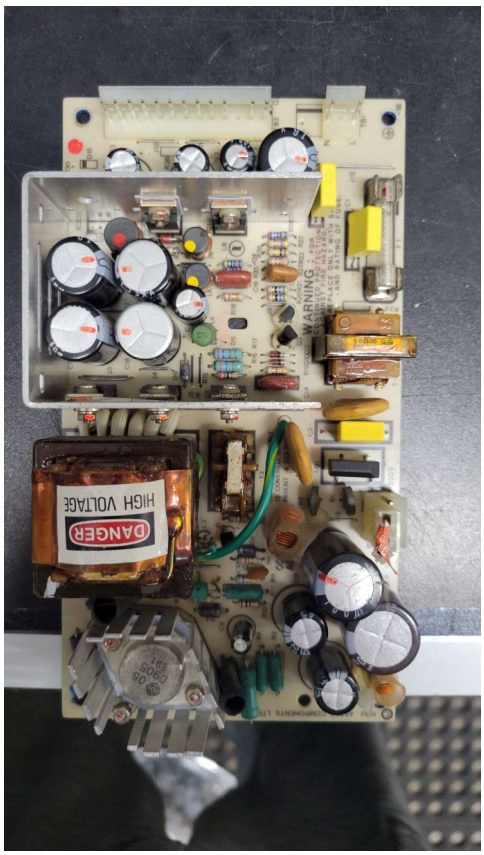
UPDATED VERSION, 2.5.6, ADDITIONALLY OFFERS :

'GREEN THUMB' EMULATION
NEW SCRIPTING COMMANDS



Fixing the Alternate Astec AA12090 R/S PSU 87790043, on a White PWA

by Patrick Bureau



I will start by saying I am not an electro technician at all, but a simple Hobbyist that wants to share information that is deeper than the usual YouTube videos about "replace all the caps and Rifa's PSU repairs" out there.

I "dig" (yes an 1970 expression) troubleshooting things, especially when I do not know them very well. The unknown is what I seek, and this PSU was a lot of things, and took several days, digging for information to get it going, with someone to help me get over the finish line in the end (yes, thank you Amardeep!)

Here is a picture of the said PSU, it is different, let me show you the easy to spot the differences.

The caps (C2/C23 etc.) below the DC power strip are aligned in a straight line. also the big power transistor D905 bottom left with heat sink , doesn't exist on Rev C (green board PWA PSU)

The hard to find schematic to this PSU is also found only in the **Sam's technical Manual of theTRS-80 Model 4GA on page 47**
(also attached the schematics as the last page of this document)

Hope you enjoy the reading of my tribulations and may it help someone to solve their issues in the future.

Tools: I pull the PSU on my workbench, I use a old 2 prong cassette player to provide the 110v to the main power in connector, a Old Seagate ST238 Hard drive to be my load power on device, a multimeter, and a capacitor tester. I used basic tools because many do not own a scope, would it make this easier, possibly, but I did not use one. (there are 50 ways do to things this was how I did it, good or bad...)

I use a Hand held soldering Iron, a hand held manual solder suction, and braided copper to desolder, and a roll of rosin lead solder to put it back together.

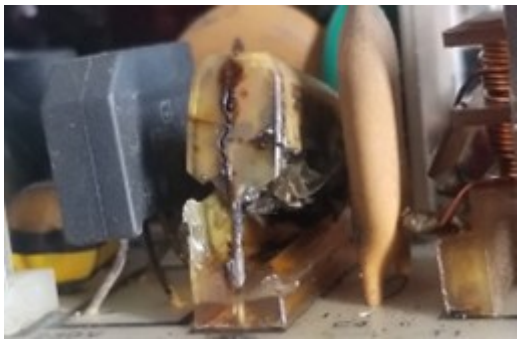
Now I have to put a warning because if I don't, someone will blame me for not being careful

ELECTRICITY CAN HURT YOU, THE LOADED CAPACITORS CAN CARRY HUNDREDS OF VOLTS, AND HURT YOU. YOU'RE WARNED!

Let's begin,

Before applying power I like to dust and look for obvious " issues" like broken wires, burn marks, cracked resistances, leaking capacitors, often those are the first thing you should find and fix to save you time to find the real issue , and will often shorten your time troubleshooting it entirely.

My first inspection, working from top of board down, observed C22 and C23 capacitors were leaking, so I replaced them with equivalent capacitors, making sure I replaced them in the same electrical directional (+ marked on the board helped a great deal to keep this straight as it is only visible when the cap is pulled off the board), I did test both caps and both where half valued to the marking on them.



My second visual inspection spotted the usual culprits, Rifas, smoke lost a long time ago, so I took my time and replaced C1, C2, C13 with their corresponding modern equivalents. At this point I could not see any visual defects anymore, so I plug it in and turned it on. and immediately a few things happened, the PSU started “clicking” like a time bomb .. Tiktiktiktiktik, about 2 tiks per seconds, the hard drive did not start up spinning, lastly the voltages on the main DC output connector were non-corresponding to expected values.

I took it offline again, and looked at the schematics. The sound was clearly loud in the area about the Inductor L3, so I probed with my multimeter on L3 and found the resistance on the inductor was 0.001 Ohms, so I took it off the board to double-check and yes it was short. so I replaced it with an 2.2uH 5% 1W inductor, and tested power again, the sound remained. So I returned to probing, and next was the capacitors on C5/C6/C7/C8 cluster, on a hunch, removed them from the board, and it turned out C6/C8 marked 47 uF 250V both read over 55uF (so more than 10%) so I replaced those and the 100uF 250v capacitors, the C5 capacitor read 227uF and the other one at C7 was only 108uF (well inside the 10% margin, so I left that one alone) and only replace C5 with a new cap.

So back to plug, turn on , test... the ticking now beats at over 6 beats per second.... much, much faster... so I turned my attention to the component marked DB1 (rectifier bridge that converts AC/DC) this can easily be done with a multimeter look up how to on YouTube but basically +- = value, =~ and -~ should be half values of +- tested together. Simple!. and it checked out so I put it back on the board.

After trying to track what could be the problem and staring at various video about “repairs” I found this 7-year-old video and midway through the trouble shooting, they suggested testing the Q1 D467 as they can die without warning
Even though their issue wasn't the same as mine...I pulled the soldering gun and pop it off the board, “in for a pound” as they say... and after using my multimeter to test the bipolar D467 Transistor, it was found shorted as well.



A new problem Arises; the 2SD467/D467 is actually hard to find as they are old tech, after speaking with a friend on the discord Tandy channel, he suggested I try a 2N2222 making sure the Emitter, Collector and Base pins be properly inserted back into the board, and that was easy as the board is clearly marked for this... of course in my excitement double-checking the pinout of the 2N2222, I did not realise I was looking at the 2SD467 papers and soldered the 2N2222 incorrectly....

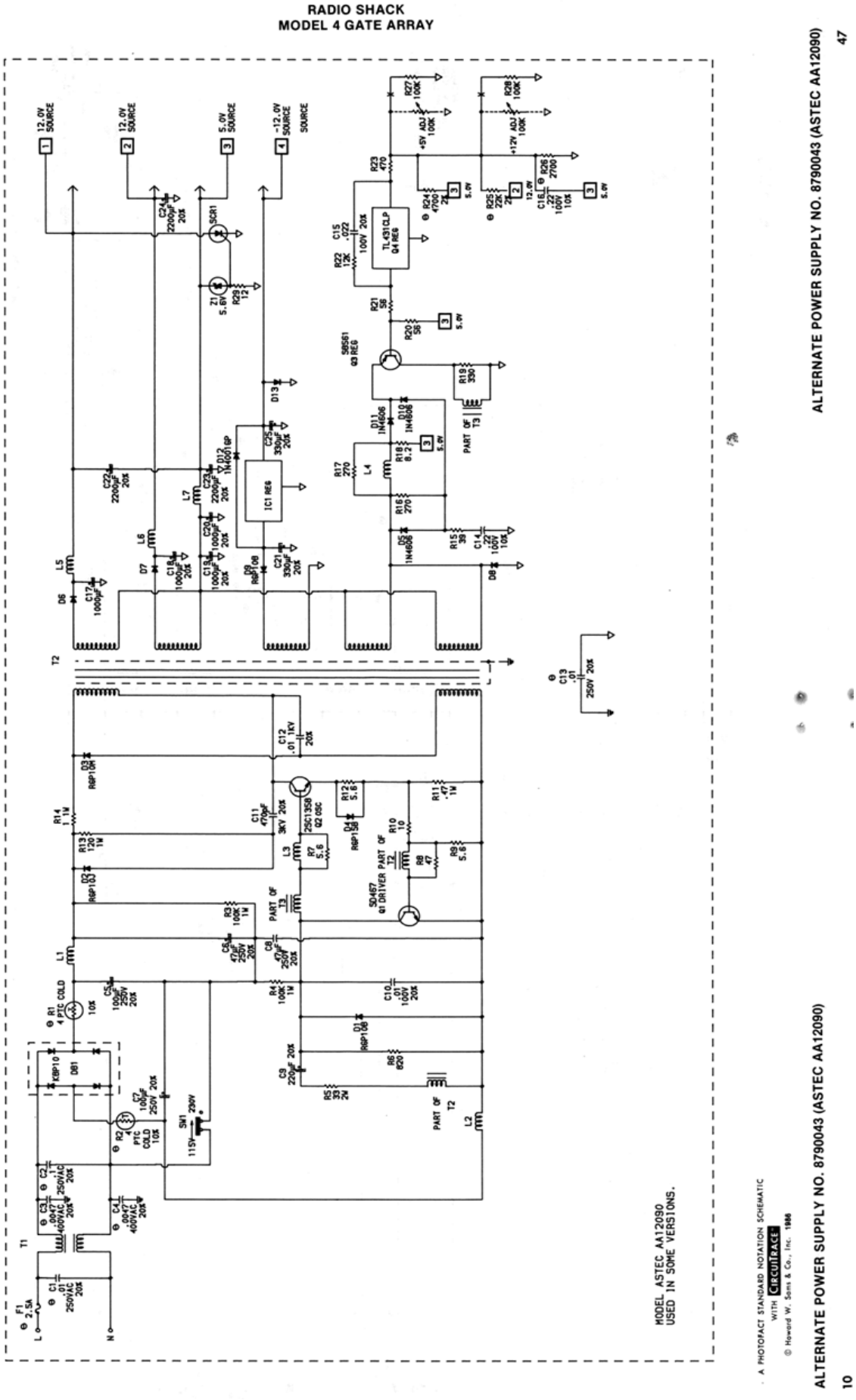
A few moment after going to test it , no power at all coming out I realise I had inverted the emitter and collector pins on the 2N2222.... back with unsoldering, triple checking and resoldering it back...

Restart testing for the X’s time, the old drive MFM drive start spinning up in a thunder rumbles like a train through a town.. It's alive !!!

After testing the main connector pins all voltages are on part with less than 0.09v difference from put 12v. -12, and 5v, Perfect! So I plug in a second ST238 drive and let it runs for an hour..... the inductor at L3 I replace is cool to the touch so nothing is stressed out now.

I took sometimes after all this to reflow almost all solder points on the board as they were showing some cracking all over....

Hope you enjoyed and maybe learned where to check on a non-working Alternate Model 4 PSU AA12090 Power Supply.



Please note that this is the short version, in the end, I tested the following components trying to track down the issue. but all were fine and in working condition: D905, R13, TB2, R12, C4.

Floppy Days
Vintage Computing Podcast
with Randy Kindig

Pull out a vintage computer and compute as if it were yesterday.

Subscribe Today!

Tandy-specific Episodes:

- TRS-80 Model I
- TRS-80 Model II
- Tandy Pocket PC
- Steve Leininger and Don French interviews
- The MISE with Peter Bartlett
- CoCo
- Boisy Pitre and Bill Loguidice book interview
- CoCoFest and Tandy Assembly coverage
- and much more...

https://www.floppydays.com

https://www.soigeneris.com/universal-backpack-drive



Universal Backpack Drive

The Backpack is a small SD card 'disk drive' for vintage computers that runs from a single AA battery

Choose Options

- ☐ Preloaded 8GB uSD card [+$5.00]
- ☐ Spare CR1025 RTC Battery [+$1.00]
- ☐ Tandy WP-2 DB9 to DB25 adapter [+$5.00]
- ☐ Sinclair Z88 DB9 t0 DB25 adapter [+$5.00]
- ☐ Epson PX-8 / HX-20 adapter [+$10.00]

$65.00



E.T. FONEHUME

HI DUSTY.
AT CHRISTMAS I RECEIVED A COPY OF "THE HUT 6 STORY" BY GORDAN WELCHMAN. IT'S ALL ABOUT BREAKING THE 'ENIGMA' CODES IN WW2. A MOST INTERESTING READ, AS UP UNTIL THE 1980'S IT WAS STILL, ALL VERY SECRET.

WELL, AS PER USUAL, I STARTED TO WONDER IF SOME SORT OF CODING (AND, ABOVE ALL, DE-CODING) BASIC ONE-LINER PROGRAM WOULD WORK ON MY (TRS80GP) MODEL 1. AS PER THE ORIGINAL 'ENIGMA' MACHINE, ONLY THE 26 LETTERS OF THE ALPHABET WOULD BE USED. THE MAIN LIMITATION, AS YOU SAID, IS THE ONE-LINE LIMIT, BUT ALTHOUGH THIS SOLUTION IS PRIMITIVE COMPARED TO 'PGP' (JUST USING LETTER TRANSPOSITION), I CAME ACROSS A UNIQUE WAY TO CRACK THE USUAL PROBLEM OF KEEPING THE KEY SECRET. VERY 21ST CENTURY!

THE SENDER AND THE RECEIVER MUST BE ABLE TO SHARE AND ACCESS THE SAME EMAIL ACCOUNT. THE SENDER WOULD PLACE THE CODED MESSAGE (WITH THE UN-CODED KEY) INTO THE 'DRAFT' FOLDER, AND INFORM THE RECEIVER IN AND TRIVIAL WAY. THE RECEIVER WOULD THEN LOG INTO THE SHARED EMAIL ACCOUNT AND EXTRACT THE KEY AND CODED MESSAGE FROM THE 'DRAFT' FOLDER (CTR/C IN WINDOWS); THEN DELETE, **PERMANENTLY**, THE MESSAGE FROM THE FOLDER.

THEN RUN THE DECODING ONE-LINER PROGRAM BY ENTERING THE TWO KEYS AT THE FIRST QUESTION AND ENTER THE MESSAGE INTO THE SECOND QUESTION. EASY IF YOU'RE RUNNING UNDER 'TRS80GP'. BINGO, THE ORIGINAL DE-CODED MESSAGE IS DISPLAYED. BECAUSE A SIMPLE LETTER SUBSTITUTION IS EMPLOYED, IT IS BEST NOT TO ENTER ANY DOUBLE LETTERS AND ABBREVIATE WORDS TO KNOWN SHORTHAND WHEREVER POSSIBLE.

FOR EXAMPLE, TO SENT THE MESSAGE "THE PERSON OF INTEREST ARRIVES AT MANCHESTER AIRPORT AT 14:00 HOURS GREENWICH MEAN TIME"

```
READY
>RUN
 5  21 ? POIARIVEMCONEFOURHZ
UTNFWNAJRHTSJKTZWME
READY
>_
```

CODED MESSAGE WITH KEYS FOR THE 'DRAFT' FOLDER

5 21 UTNFWNAJRHTSJKTZWME

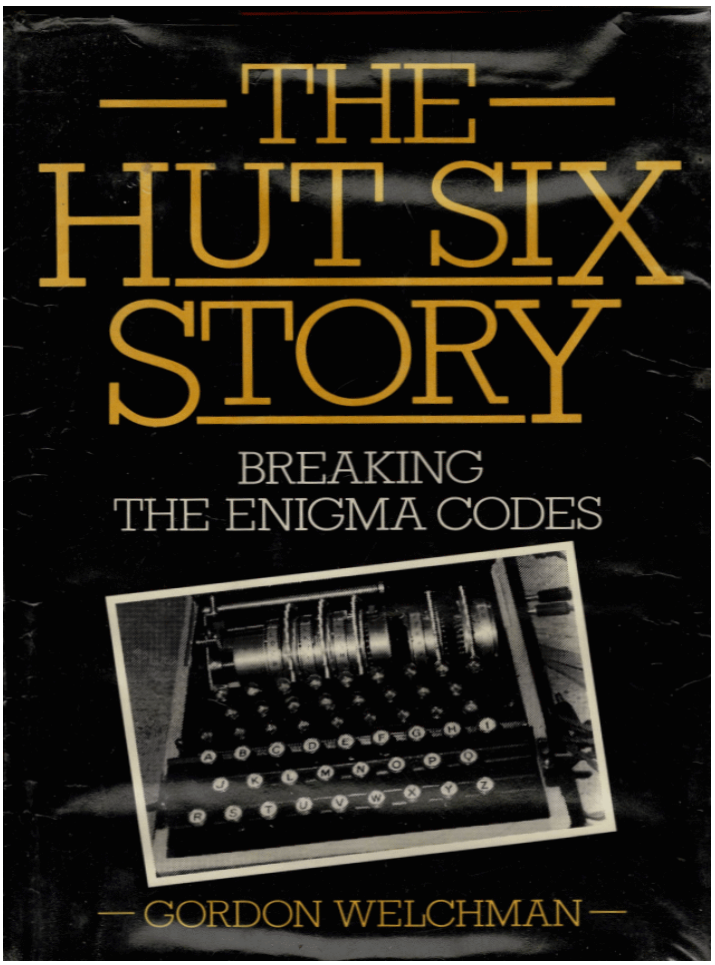
```
READY
>RUN2
? 5,21
? UTNFWNAJRHTSJKTZWME
POIARIVEMCONEFOURHZ
READY
>_
```

OF COURSE, IN ENGLISH..
POI 'PERSON OF INTEREST'
ARRIVE MC 'MANCHESTER',
ONEFOURHZ 'FOURTEEN HUNDRED HOURS ZULU (I.E. GMT)'

Letter	Count		Letter	Frequency
E	21912		E	12.02
T	16587		T	9.10
A	14810		A	8.12
O	14003		O	7.68
I	13318		I	7.31
N	12666		N	6.95
S	11450		S	6.28
R	10977		R	6.02
H	10795		H	5.92
D	7874		D	4.32
L	7253		L	3.98
U	5246		U	2.88
C	4943		C	2.71
M	4761		M	2.61
F	4200		F	2.30
Y	3853		Y	2.11
W	3819		W	2.09
G	3693		G	2.03
P	3316		P	1.82
B	2715		B	1.49
V	2019		V	1.11
K	1257		K	0.69
X	315		X	0.17
Q	205		Q	0.11
J	188		J	0.10
Z	128		Z	0.07

OBVIOUSLY, THE TWO PROGRAMS ABOVE ARE JUST FOR FUN. HOWEVER, THE BASICS ARE THERE FOR ALL TO SEE, (HENCE THE LETTER FREQUENCY LIST) AND WITH A BIT MORE CODING, I THINK, BACK IN THE DAY, IT WOULD HAVE MANAGED TO CREATE A HEADACHE OR TWO FOR SOME UNLUCKY PERSON.

```
READY
>LIST-5
0 RANDOM:DIMA$(26):S=RND(19)+3:X=26-S:Y=ABS(S-X):FORC=1TOX:A$(C)
=CHR$(C+64+S):NEXT:FORC=X+1TO26:A$(C)=CHR$(C-X+64):NEXT:PRINTS;X
;:INPUTM$:FORZ=1TOLEN(M$):T$=MID$(M$,Z,1):PRINTA$(ASC(T$)-64);:N
EXT
1 END
2 CLEAR500:DIMA$(26),B$(26):S=0:X=0:FORZ=1TO26:B$(Z)=CHR$(Z+64):
NEXT:INPUTS,X:FORC=1TOX:A$(C)=CHR$(C+64+S):NEXT:FORC=X+1TO26:A$(
C)=CHR$(C-X+64):NEXT:INPUTM$:FORZ=1TOLEN(M$):T$=MID$(M$,Z,1):FOR
Y=1TO26:IFT$=A$(Y) THEN PRINTB$(Y);:NEXTZELSENEXTY:NEXTZ
5 END
READY
>_
```



BetaGamma Computing

Specialists in all things that compute







Based in Corfu Greece

- Qualified IT Engineers
- Repair of all PC's and Laptops
- Software and Hardware Upgrades and Installation
- Broadband and Network Installation
- Supply of Custom built to order PC solutions

RETRO RESTORATION SPECIALISTS

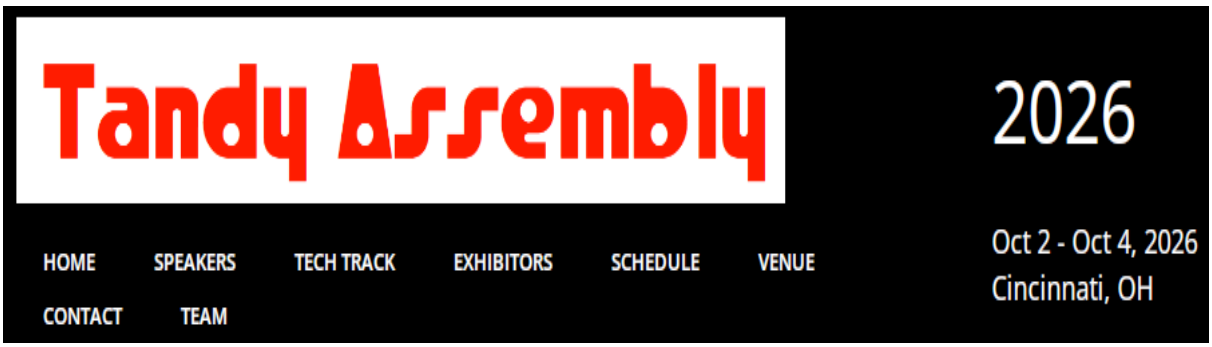


Call Corfu 26610 26358 Mobile 698 755 8427

Email: bgcompute@aol.com • www.betagammacomputing.com

MAV'S TANDY ASSEMBLY QUARTERLY COLUMN
TA26 NEWS

by Ian Mavric



Bringing you the latest news you need to know about the greatest Tandy TRS-80 show on the planet. This time I talk about the 2026 show, with a new announcement!

WE ARE MOVING!



Since 2018 we have been at Springfield Ohio which has served us well and we are grateful to Marriott who looked after us all these years and allowed us to run successful Tandy Assembly shows.
We do listen to the feedback from people who have attended shows and those who, for various reasons, were unable to attend, and a common response was that the Springfield venue was not close enough to an airport.

Almost as common as the airport problem was the general lack of other things to do around Springfield, other than Tandy Assembly.

The airport problem stems from people needing, after a long, sometimes multiple flights to get into Dayton or Columbus either hire a car, get a cab, or organise someone to drive you to Springfield.
Both Dayton and Columbus were roughly the same distance from Springfield I can tell you from first hand experience if you need to get a taxi from Columbus it came to an eye-watering $US178, something I was not need to repeat.

A decision was therefore made to put the show much closer to an international airport.

Getting families or even WAGs (wives and/or girlfriends) to accompany people to Tandy Assembly often fell flat due to a lack of 'draw' for anything to do which may occupy and amuse those not really into what TA was offering.

Simply put, there wasn't enough 'touristy sh!t' for people to entice their WAGs and families to attend. Thusly a decision was made to put the show in a bigger city with more to offer and even better nightlife for those to enjoy such things.

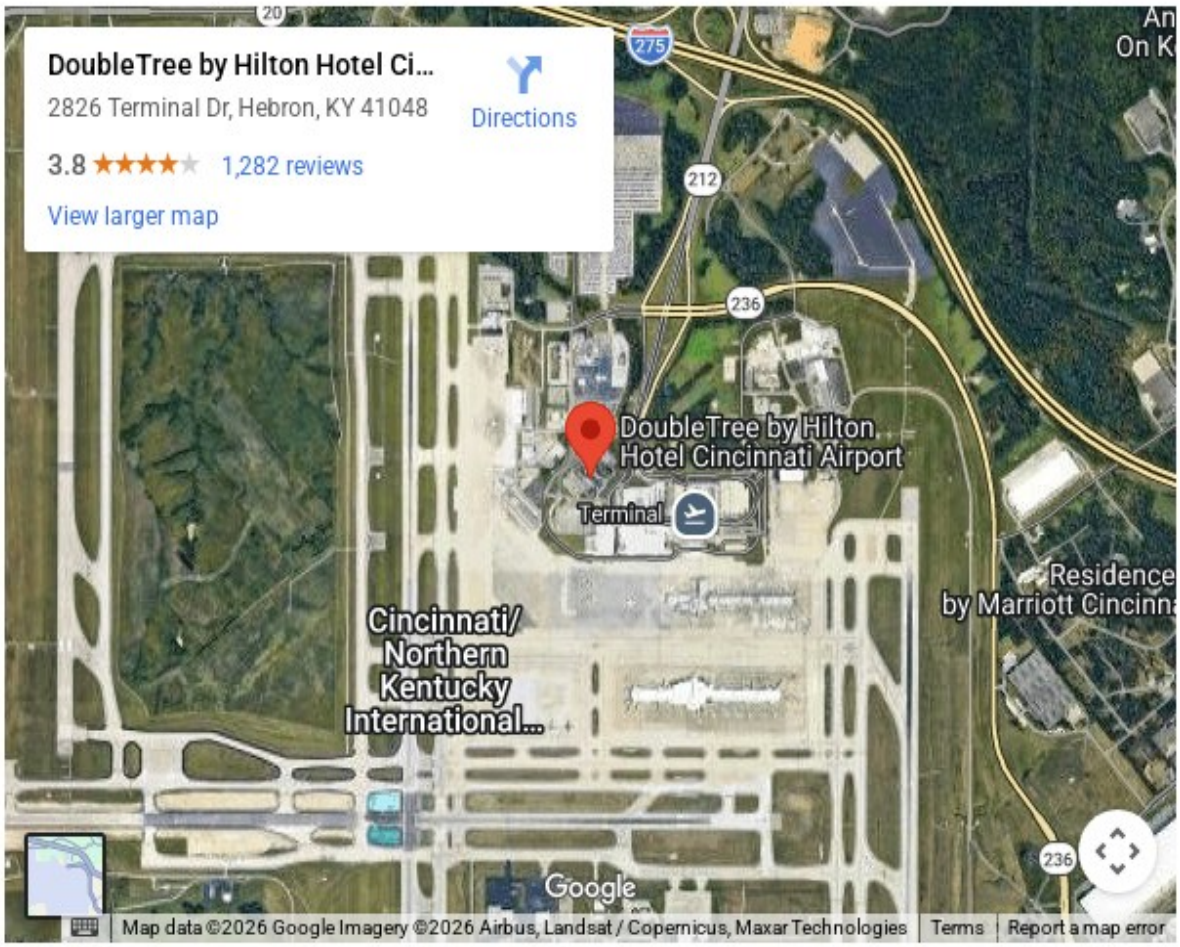
The TA committee discussed this over a few of our monthly meetings, and we unanimously decided that Cincinnati offered most of what we were looking for: a venue for the show, very close to an international airport, and a large modern city with lots of attractions for tourists.

DOUBLE TREE BY HILTON HOTEL
CINCINNATI AIRPORT

The address of the hotel is:

2826 Terminal Drive, Hebron, Kentucky, 41048, USA

Phone: +1 859-371-6166



To avoid any confusion, the airport (IATA code: CVG) is actually on the border of Kentucky and Ohio, minutes away from Cincinnati. The venue itself is within the airport precinct, you don't even need to rent a car if you don't want to.

SPECIAL ROOM RATES

Like the previous place we have special room rates for people attending Tandy Assembly. How to book a room is on the Tandy Assembly website, under the Venue tab. It looks like this:

VENUE

Tandy Assembly 2026 will be held at the DoubleTree by Hilton Hotel Cincinnati Airport.

We have worked with the hotel to get a group rate discount for Tandy Assembly.

To reserve a room:

- Click on the link above.
- Select the dates you would like to stay. Tandy Assembly will take place on October 2, 3 and 4.
- Click the "Special Rates" button and enter group code **902**
- Click on "Check Rooms & Rates" and proceed with the reservation.
- If you have problems booking, you can call Laura Tupman (859) 817-2613 and she can assist.

CALL FOR PRESENTERS, SPEAKERS, AND TRIVIA GAME CONTESTANTS

The March issue of this column always turns out to be the shortest one as we have only just begun putting this year's show together.

We have some great presenters that we are already in contact with and hopefully signed up soon. I can talk more about it in the next issue.

Of course I always need contestants to play the trivia games. Keen spotters of last year's show saw two trivia games: one for the ladies and one for the men, so I'm looking for anyone to play. Reach out to me if you like: ianm@trs-80.com

2026 WILL CELEBRATE 40 YEARS OF THE COLOR COMPUTER 3

Each year we pick a system which has released some significant milestone and try to do it justice with presentations by knowledgeable individuals. July 30th, 1986 marked the release of the Coco 3, the last and best Coco. Stay tuned for the presentation, commemorative t-shirt, and cake!



NEXT TIME: The first of the presenters will be announced.

Ian Mavric
ianm@trs-80.com



Ian Mavric is an IT Specialist who also restores and collects TRS-80's and classic cars. He lives with his wife and kids in Melbourne, Australia.


ACTUAL REALITY
TRS-80 MODEL I/III/IV
DREAMLAND SOFTWARE


ROCK STAR!
TRS-80 MODEL I/III/IV
DREAMLAND SOFTWARE

Power of the ROM

Part 5 - Cut !, Print !

Vince Otten

It's the very early 1980s, and my fledgling programming business "*Morningstar Computer Consultants*" is trying to get off the ground. I've taken possession of an Expansion Interface and two disk drives for my Model I. I'm delving into the wonders of Disk BASIC and developing some of my first "business" programs. But it's getting tiresome to have to write out my program listings by hand. It's time for a "Line Printer"! I need one that uses "tractor feed" to keep the paper straight, and which has a wide carriage to hold the business forms I hope to program for.



This issue's "Power of the ROM" article explores Assembly Language programming for the early printers that Radio Shack sold. I'll also cover an A/L version of BASIC's PRINT TAB(), and a rewrite of the "trailing comma" PRINT specifier, as in

```
100 PRINT A, B, C
200 LPRINT X$, Y$, Z
```

Let's get right to the Printer. Just like writing to the Screen, there's a ROM routine which does the job nicely.

CALL 003B — Print Character

The character contained in the C-register is sent to the printer. A line count is maintained by the driver in the DCB. When a full page has been printed (66 lines), the line count is reset and the status register returned to the caller is set to zero. Control codes recognized by the printer driver are:

CODE	ACTION
00	Returns the printer Status in the upper two bits of the A-register and sets the status as zero if not busy, and non-zero if busy.
0B	Unconditionally skips to the top of the next page.
0C	Resets the line count (DCB 4) and compares its previous value to the lines per page (DCB 3) value. If the line count was zero, no action is taken. If the line count was non-zero then a Skip to the top form is performed.
0D	Line terminator. Causes line count to be incremented and tested for full page. Usually causes the printer to begin printing.

There's a problem with just jumping in and using this ROM call though, and there's a hidden challenge as well. First of all, the problem: the Printer routine expects the character in register A, not C: by the time it gets to address 03CDH, it will overwrite C with what's in A! So when you CALL 003BH, be sure to put the character you wish to print into register A.

Then there's the *challenge*: writing to the Printer is not like writing to the Screen. The Screen is always ready to receive your output, but the Printer may be turned off; it may be out of paper; it may even be jammed. I remember those frustrating days of BASIC program development when I had written some remarkable lines of code, and wanted to list them to the Printer. So I fired the LLIST command off, but nothing happened. Oops! The Printer was out of paper, and I had no more in stock. But what was worse was that my TRS-80 was now hung up, and I'd forgotten to SAVE the program first. There was no other option but to reset the machine, lose all my hard work, and start over again.

What has to be done is to check the Printer's "status": is it ready to print? The BASIC ROM code does that—but it just waits until the Printer reports "Ready", so when the Printer is turned off or is out of paper, the processor just endlessly loops, waiting for that to be resolved. What

would be nice is if we had a routine that waited a preset period of time for the Printer to get ready, and if it never did, to report as an error and return to our program so it could "decide" whether to go on or quit or fix the problem. Here's what I came up with to do that. The calling program loads the B register with a multiple of 300 milliseconds to wait for the Printer to get ready (ie, load B with 2, and this subroutine will test the Printer for "ready" up to .6 seconds.

```
;
;     LPRWTR -- LINE-PRINTER-WAIT-READY()
;          WAIT N X .30 SECONDS FOR
;          PRINTER TO BE READY
;          ARGUMENT:
;               B     N (300 MSEC. INCREMENTS)
;          REPORTS:
;               BERROR      "UNPRINTABLE"  IF NOT READY;
;                           0              IF READY
;               Z FLAG      NZ             NOT READY
;                           Z              READY
;
LPRWTR      EQU     $
            XOR     A               ; INITIALIZE
            LD      (BERROR),A      ; ERROR CODE 0 (NONE)
LPRWT1      EQU     $
            LD      DE,10H          ; WHILE (NOT READY):
LPRWT2      EQU     $
            CALL    05D1H               ; GET PRINTER STATUS
            RET     Z                       ; IF "READY", EXIT
            DEC     DE                  ; ELSE KEEP TRYING--
            LD      A,D
            OR      E
            JR      NZ,LPRWT2
            DJNZ    LPRWT1          ; :WEND
            LD      A,20            ; IF NEVER READY:
            LD      (BERROR),A          ; REPORT "UNPRINTABLE"
            OR      A                   ; & SET NZ FLAG
            RET                     ; :ENDIF
```

"LPRWTR" and some of the low-level Printer subroutines that will use it report and test success/failure results through BERROR. We discussed this variable (stored at 409AH) which is used by such BASIC statements as ERR and ON ERROR in the June 2025 issue of TRS8bit. Instead of "hanging up" waiting for the Printer to report that it's ready, LPRWTR will only repeat testing the status until either the Printer *is* ready, or until its "time", measured in 300 msec. maximum durations, is up, when it will report the BASIC error code 20, "UNPRINTABLE", in BERROR.

Alright: we're ready to print to the Line Printer. Here's a typical application:

```
;
;    LPRT1C – WRITE 1 CHARACTER IN A TO THE LINE PRINTER
;                IF PRINTER NOT READY, REPORT ERROR IN "BERROR"
;    ARG:        A           CHARACTER TO PRINT
;    REPORTS:    BERROR      0                   SUCCESS
;                            20/"UNPRINTABLE"    PRINTER NOT READY
;
    LPRT1C      EQU     $
                PUSH    AF              ; PRESERVE CHAR TO PRINT
                LD      B,1             ; MAX WAIT FOR PRINTER READY:
                CALL    LPRWTR          ;       300 MSEC.
                JR      NZ,LPRT1E       ; IF (PRINTER READY):
                POP     AF                  ; RETRIEVE CHAR TO PRINT
                CALL    003BH               ; PRINT CHAR IN A
                RET                     ; ELSE:
    LPRT1E      EQU     $
                POP     BC                  ; RESTORE STACK
                RET                     ; :ENDIF
```

If the Printer's not ready, we don't "hang up" in an indefinite loop, but simply report the failure in BERROR and carry on. Code that calls LPRT1C should check BERROR before continuing.

By this point, the discerning reader might ask, "Weren't these articles supposed to mimic the Level I BASIC code that Dr. Lien introduced in his book?"



Another reader will reply, "That's right! Dr. Lien didn't even *mention* the Printer!"

I confess—you're right. But I have a sneaky rationale for mentioning the Printer. Firstly, it allows for including some extra ROM calls. Secondly, it's a useful extra application for my implementation of the two BASIC PRINT modifiers I introduced above: PRINT TAB(*n*) and the "trailing comma". I think it would be nice to have one central subroutine that can implement TAB() for either the Screen or the Printer, and then use it as the basis for the comma. BASIC does this by either prefixing PRINT with an "L" for the Printer, or omitting the "L" for the Screen, ie

```
10  LPRINT TAB( 30) X
20  PRINT TAB( 40) Y
```

In pseudocode, TAB() might look like this:

```
Tab( Position, Device) {
/* TAB() up to Position on Device, ie Printer or Screen          */
    if  (Position < Device.Width) {
        /* may not TAB() past last column on this device         */
        While (Device.Current < Position) {
            /* TAB() up to Position                              */
            write( Device, ' ');
            Device.Current++;
            }
    else
        /* Decide what to do with TAB() exceeding line length   */
        AdjustForExcessiveTab( Device.Current);
    }
```

For the TRS-80 Model I Screen, Device.Width is 63 (Columns 0..63). For the Printer, however, it can be anywhere from 20 columns (the lowest setting on the DMP-400/500) to 132 columns (Line Printers I and IV-VI, highest setting on DMPs 200 through 500). Notice as well Device.Current, which records the current position of the cursor (for the Screen) or the column number for the print head on the Printer. This gets incremented every time we print a character, and reset to 0 whenever a "newline" (Carriage Return, ASCII 13/0DH on the TRS-80) gets printed. There's a separate Device.Current value for each device, and there's 3 ways in which it might get updated:

1. Whenever a character other than a "newline" is printed and we haven't reached the last (rightmost) column number for this device: Device.Current is incremented; or
2. Whenever the character printed was a "newline": Device.Current is set to 0; or
3. Whenever the TAB() "Position" argument is greater than Device.Width: it must be adjusted according to the method we choose, something like
    a. Print a newline and set Device.Current to 0; or
    b. Divide "Position" by Device.Width and print a newline repeatedly until the result is 0; then TAB() by the remainder. This is what BASIC does, at least for the Screen: 'PRINT TAB(129) "HERE!"' will print two "newlines" (Carriage Returns/ASCII 13/0DH), then TAB over 2 spaces before printing "HERE!".

"3.b" implies that Level II BASIC does indeed keep track of what we call Device.Current:

- For the Screen, 40A6H
- For the Printer, 409BH

Unfortunately, it only does that while running BASIC. TRSDOS commands and programs don't update (or presumably reference) these values. That means that we're going to have to track them ourselves. That means our primitive "Write A Character To The Screen" and "Write A Character To The Printer" subroutines have to be updated to track Device.Current for each.

```
VIDCUR    EQU   40A6H          ; "CURSOR'S CURRENT COLUMN #"
PRTCUR    EQU   409BH          ; "CURRENT TAB POSITION OF LINE PRINTER"

VIDW      DEFB  64             ; SET SCREEN WIDTH HERE
PRTCW     DEFB  80             ; SET PRINTER CARRIAGE WIDTH HERE
```

First, we simply print the character. Then, to properly update our current cursor position, we need to check if the character is a newline (0DH) or a linefeed (0AH) and, if so, set the current position to 0. As well, if we've reached the far right column for our device, and if so, possibly print a newline (depending on the device) and reset the current position to 0. Other than that, we just increment the current cursor position. Pseudocode for the Screen looks like this:

```
VideoPrint1Character( char) {
    if (char != NEWLINE or char != LINEFEED) and (Device.Current < Device.Width)
        Device.Current= Device.Current + 1;
    else
        Device.Current= 0;
    )
```

(C programmers would probably like to write something like Device.Current=(((char != NEWLINE) || (char != LINEFEED)) && Device.Current < Device.Width) ? Device.Current + 1 : 0 , but that doesn't help me code this in assembler.)

Here's the code for the Screen:

```
;
;     VPR1CH -- PRINT 1 CHAR TO VIDEO SCREEN,
;                     UPDATING CURSOR POSITION
;     ARG:            A       CHARACTER TO PRINT
;     REPORTS:
;                     ANY ROM ERRORS          BERROR
;                     CURSOR SET TO 0         Z FLAG
;
VPR1CH      EQU     $
            PUSH    AF              ; PRESERVE CHAR TO PRINT
            CALL    33H             ; PRINT CHARACTER
            POP     AF              ; RESTORE CHAR
            CP      0DH             ; IF (NOT A NEWLINE
            JR      Z,VPR1C9
            CP      0AH             ; OR A LINEFEED)...
            JR      Z,VPR1C9
            LD      A,(VIDW)
            LD      C,A
            LD      A,(VIDCUR)
            CP      C
            JR      NC,VPR1C9       ; AND (CURSOR < WIDTH):
            INC     A                       ; CURSOR POSITION++
            LD      (VIDCUR),A
            RET
VPR1C9      EQU     $               ; ELSE:
            XOR     A
            LD      (VIDCUR),A              ; CURSOR POSITION= 0
            RET                     ; :ENDIF
```

Code for the Printer is almost identical, with one addition. If the print head reaches the far right end of the carriage, the last printable column, what happens? I last used my Line Printer VI around 40 years ago, but I think that if it was fed a 200+ character BASIC program line to LLIST, it would automatically return the head to column 0 ("Carriage Return") and scroll the paper up one line ("Line Feed"). But what if we're using a printer that doesn't inherently do that? The "Printer/Model" menu on the version 2.5.5 trs80gp emulator I use simply keeps printing on the same line: it doesn't have a fixed carriage width. So I've added a little "Fixup" subroutine that optionally adds that action when the print head reaches the far right. As well, my Line Printer version of "VPR1CH", LPR1CH, first checks that the Printer is ready before calling the actual "print 1 character" subroutine, "LPRINC". There's also a little sub to check if there's an error (non-zero value) in BERROR, which gets used in several places. Here's the set of 4 subroutines:

```
;
;     LPR1CH -- LINE-PRINT-1-CHAR()
;
;     ARG:          A             CHAR TO PRINT
;     REPORTS:      BERROR        SUCCESS          0
;                                 FAILURE          20
;                                                  ("UNPRINTABLE")
;
LPR1CH  EQU           $
          PUSH        AF              ; PRESERVE CHAR TO PRINT
          LD          B,1             ; MAX WAIT FOR PRINTER READY:
          CALL        LPRWTR          ;      30 MSEC.
          JR          NZ,LPR1C1       ; IF (PRINTER READY):
          POP         AF                    ; RETRIEVE CHAR TO PRINT
          CALL        LPRINC                ; PRINT A: CHAR &
          RET                               ; UPDATE CARRIAGE POS'N
LPR1C1  EQU           $               ; ELSE:
          POP         AF                    ; RESTORE STACK
          RET                         ; :ENDIF

;
;     LPRINC -- LINE PRINT A CHARACTER
;
;     ARG:          A             CHAR TO PRINT
;     REPORTS:      BERROR        ANY ROM ERRORS
;
LPRINC  EQU           $
          PUSH        AF              ; PRESERVE CHAR TO PRINT
          CALL        003BH           ; PRINT CHARACTER
          CALL        BERTST          ; CHECK "BERROR"
          JR          NZ,LPRN09       ; IF (PRINTER WAS READY):
          POP         AF                    ; RESTORE CHAR
          CP          0DH                   ; IF (NOT NEWLINE OR
          JR          Z,LPRN07              ;    OR
          CP          0AH                   ;    LINEFEED)
          JR          Z,LPRN07              ;    AND
          LD          A,(PRTCW)             ;    (CARRIAGE POS'N
          LD          C,A                   ;     <
          LD          A,(PRTCUR)            ;    CARRIAGE WIDTH):
          CP          C
          JR          NC,LPRN08
          INC         A                     ; HEAD POSITION++
          LD          (PRTCUR),A
          RET
LPRN08  EQU           $               ; ELSE:
                                            ; IF (POS'N >= WIDTH):
```

```
        CALL    LPRFIX                  ; "FIX" (PRINT CR)
LPRN07  EQU     $
        XOR     A
        LD      (PRTCUR),A              ; HEAD POSITION= 0
        RET                             ; :ENDIF
LPRN09  EQU     $               ; ELSE:
        POP     AF                      ; RESTORE STACK
        RET                             ; & REPORT BERROR
                                ; :ENDIF

;
;   LPRFIX -- SUBROUTINE FOR LPRINC TO "FIX"
;       END-OF-LINE CONDITION WHERE HEAD REACHES
;       RIGHT-HAND END OF CARRIAGE. IF PRINTER
;       DOESN'T AUTOMATICALLY WRAP, USE THE
;       FOLLOWING CODE TO PRINT A NEWLINE
;
LPRFIX  EQU     $
        LD      A,0DH       ; NEWLINE/CARRIAGE RETURN
        CALL    003BH       ; PRINT IT
        RET

;
;   BERTST -- TEST "BERROR" & RETURN Z FLAG
;
;       REPORTS:    Z FLAG      SET     NO ERROR
;                               RESET   ERROR
BERTST  EQU     $
        LD      A,(BERROR)
        OR      A
        RET
```

If your printer does the "automatic" thing at the far right column, just comment out or delete LPRFIX's 2 lines of code ("LD A..CALL 003BH"). BERTST is an example of "factoring": it makes the code that uses it, like LPRINC, a little easier to read. It'll also get used in higher-level procedures that call LPR1CH. This may be the occasion to mention a possible addition to INIT subroutine that we CALL at the beginning of each user program in this series. (INIT was introduced in the second article, TRS8Bit June 2025.) I propose inserting the following:

```
;
;     INIT -- INITIALIZE PROGRAM THAT USES BASIC ROM
;
INIT    EQU     $
        XOR     A               ; ZERO VALUE
        LD      (BERROR),A      ; BASIC ERROR CODE 0
        LD      (411BH),A       ; TURN "TRON" OFF
        LD      (PRTCUR),A      ; CARRIAGE POS'N= 0
        LD      (VIDCUR),A      ; CURSOR POS'N= 0
        RET
```

Now we've got everything we need to implement our version of PRINT TAB(*x*). (Side note: did you know that when you use TAB in BASIC, the "opening parenthesis" must immediately follow TAB, or the interpreter will consider TAB (*x*) to be an array reference? You're normally not allowed to use "reserved words" like PRINT or INPUT in variable names, but this is a tricky exception.) Here's a reminder of how TAB works in pseudocode:

```
Tab( Position, Device) {
/* TAB() up to Position on Device, ie Printer or Screen          */
    if  (Position < Device.Width) {
        /* may not TAB() past last column on this device         */
        While (Device.Current < Position) {
            /* TAB() up to Position                              */
            write( Device, ' ');
            Device.Current++;
            }
    else
        /* Decide what to do with TAB() exceeding line length    */
        AdjustForExcessiveTab( Device.Current);
    }
```

This is an algorithm that's common to both the Screen and the Printer. What varies for each "device" is its Width (the rightmost Screen column or Printer Carriage position), its Current cursor or print head position, and the actual subroutine to print one character. What I decided to do was to write separate PRINTB and SCPRTB subroutines which call TAB with their own set of properties, as in

```
PrinterTAB( Position) {
/* TAB() to new Position on Printer */
Tab( Position, PrinterCursor, PrinterWidth, LPR1CH);
}
```

and

```
ScreenTAB( Position) {
/* TAB() to new Position on Screen */
Tab( Position, ScreenCursor, ScreenWidth, VPR1CH);
}
```

which means that the generalized TAB now takes 4 arguments instead of 1:

```
Tab( Position, Cursor, Width, DeviceSubroutine) {
    /* TAB() up to Position on Device, ie Printer or Screen          */
        if  (Position < Width) {
            /* may not TAB() past last column on this device         */
            While (Cursor < Position) {
                /* TAB() up to Position                              */
                DeviceSubroutine( ' ');
                /* ie, write( Device, ' ');                          */
                Cursor++;
                }
        else
            /* Decide what to do with TAB() exceeding line length   */
            AdjustForExcessiveTab( Cursor);
        }
```

This sort of "reverse factoring" is where a general procedure like TAB is invoked by specialized "wrappers" like PrinterTAB and ScreenTAB. The general procedure then calls on details specific to each caller, such as Cursor, Width, and the actual "write" subroutine. This brings up the issue of "parameter passing". How do we let Tab() and the individual DeviceSubroutine()s know what they're supposed to work with?
In higher-level languages like C, the actual "arguments" to a function like

```
PrintAstringAtTAB( position, string)
```

are "passed" via the stack. Perhaps the small integer value of position is in A, and a pointer to the string is in HL, so we

```
PUSH    AF
PUSH    HL
```

but the challenge is that once we CALL PrintAstringAtTAB()

```
CALL    PRSTAB
```

the stack looks like this:

(lower memory address)

+0	- return address (low byte)
+1	- (hi byte)
+2	- address of "string" (HL)
+3	- (hi byte)
+4	- "position" (A)
+5	- (Flags)

(higher memory)

so we can't just POP HL within the PRSTAB subroutine to access the address of the "string". We end up having to do something like

```
LD      IY,0
ADD     IY,SP           ; IY NOW HAS THE ADDRESS OF THE STACK
;
LD      L,(IY+2)
LD      H,(IY+3)        ; HL NOW HAS THE ADDRESS OF "STRING"
```

and that's how some C compilers for the TRS-80 actually did it. This method is good for recursion, because each call to a subroutine has its own "stack frame"—in this example, the section containing the return address, the string address, and the position byte. Its drawback is that it's complex and slower than passing arguments through the registers.

So why don't we just use registers, like we have so far? "Position" can go into A; the address of "Cursor" into DE (not just the *value* of "Cursor", because TAB will need to alter it); the "Width" into C, and the address of the "DeviceSubroutine" into HL. That works, although we'll probably need to juggle HL and DE and probably PUSH and POP a few times. Here's a different technique that I first developed in BASIC for my "Software Tools" articles.

I set up a group of small arrays in memory: one pair for "arguments", and another pair for "return values".

Arguments		Return Values	
8-bit	16-bit	8-bit	16-bit
BARGS "Byte Args"	WARGS "Word Args"	BRETS "Byte Returns"	WRETS "Word Returns"
+	+ 2 x	+	+ 2 x
0	0	0	0
1	1	1	1
2	2	2	2
...	...	...	...
8	8	8	8
9	9	9	9

To access byte-sized argument 1, I write BARGS+1, as in "LD A,(BARGS+1)". To return 16-bit value 3, I write WRETS+3*2, as in "LD (WRETS+3*2),DE" (16-bit or "word size" entries are, of course, 2 bytes

long). Altogether, BARGS, WARGS, BRETS, and WRETS take up only 48 bytes. But one of their advantages to me is that their values "last", so that I can re-use the value of BARGS+1 that I sent to Subroutine2 back again in the Subroutine1 that called it if needed. I do this in TAB and COMMA, for example.

So what does the actual code for the general-purpose TAB look like?

```
;     TAB( POSITION, &CURSOR, MAX, &ROUTINE) -- TAB TO
;          EITHER "POSITION" OR START OF NEW LINE
;          USING "ROUTINE" (LPRACH FOR PRINTER, OR
;          VPR1CH FOR SCREEN)
;     ARGS:  POSITION                BARGS(0)
;            MAX                     BARGS(1)
;            &CURSOR                 WARGS(0)
;            &ROUTINE                WARGS(1)
;     REPORTS:
;            ANY ROM ERRORS          BERROR
;            (CURSOR >= POSITION     C FLAG)
;            CURSOR SET TO 0         Z FLAG
TAB     EQU     $
        LD      A,(BARGS+0)     ; BARGS(0)= POSITION TO TAB TO
        LD      HL,BARGS+1      ; BARGS(1)= MAX COLUMN #
        CP      (HL)
        JP      NC,TAB002       ; IF (TAB POS'N < MAX):
        LD      C,A                 ; TAB POS'N -> C
TAB001  EQU     $
        LD      HL,(WARGS+0)        ; WARGS(0)= &CURSOR
        LD      A,(HL)              ; CURSOR -> A
        CP      C                   ; CURSOR - TAB POSITION
        RET     NC                  ; WHILE (CURSOR < TAB POS'N):
        PUSH    BC                      ; PRESERVE TAB POS'N
        LD      HL,(WARGS+1*2)          ; &ROUTINE -> HL
        LD      A,' '                   ; SPACE CHAR -> A
        CALL    HLSUB                   ; CALL ADDRESS IN HL
                                        ; (PRINT A ' ' TO
                                        ; PRINTER OR SCREEN)
        POP     BC                      ; RESTORE TAB POS'N
        RET     Z                       ; IF (NEW LINE):
                                            ; EXIT
        CALL    BERTST                  ; IF (PRINTER !READY):
        RET     NZ                          ; EXIT
        JP      TAB001              ; :WEND
TAB002  EQU     $               ; ELSE:
        CALL    TABEXC              ; CORRECT FOR TAB POS'N
        RET                         ; EXCEEDING WIDTH
                                ; :ENDIF
```

Here's how it implements the pseudocode for TAB above. Notice that we address "BARGS[0]" as "BARGS+0", and "BARGS[1]" as "BARGS+1". The "CP" instruction is like a non-destructive subtraction: we're "subtracting" the maximum column number in BARGS[1], ie "Width", from the position we want to TAB to in BARGS[0]. Normally this will result in a negative number: if we want to TAB(20) and Width is 63, then 20-63 is -43, so the Carry flag is set. "JP NC,TAB002" only bypasses the whole TAB procedure if that flag is *not* set because for some reason the TAB argument is >= the maximum column for this device.

Next at TAB001 we LD HL with the *address* of the current Cursor position, which has been stored in WARGS[0]. The value *at* that address, the actual Cursor, gets transferred to A. We compare that with the TAB Position saved in C. Once again, if the Cursor position is >= Width, the Carry flag will *not* be set, so we simply exit TAB. Otherwise we can now attempt to actually TAB up to Position. A blank character gets stored in A. HL now gets loaded with the *address* of the device's "Write One Character" subroutine (ie, VPR1CH or LPR1CH). We use a trick to CALL the address in HL by CALLing "HLSUB". Here's what that little subroutine looks like:

```
;     HLSUB -- CALL SUBROUTINE IN HL
;           (LOAD HL WITH ADDRESS OF SUBROUTINE,
;           THEN "CALL HLSUB". WILL RETURN TO
;           INSTRUCTION AFTER "CALL HLSUB" IN
;           YOUR CODE.)
;
HLSUB   EQU     $
        JP      (HL)
```

By CALLing HLSUB, the processor stores the address after this CALL on the stack. Then when HLSUB jumps to the address in HL, that subroutine (VPR1CH or LPR1CH in our case) will eventually RETurn back to our code above. It's an easy way to "CALL HL".

Both VPR1CH and LPR1CH will return a 0 if the Cursor position is reset (since adjusting that position and storing it in VIDCUR or PRTCUR is the last thing each procedure does). That's why we have a RET Z instruction after CALLing our "Write One Character" subroutine: we're done TAB-ing. We also confirm that the character actually got PRINTed by CALLing BERTST "BERROR Test": if BERROR is non-zero, the device wasn't ready, so we might as well exit TAB. (It's probably wise to check BERROR after using TAB in your user program.) Otherwise, we can simply loop until our Cursor reaches the TAB Position.

TAB002: we arrived here if the TAB Position specified was >= the Width setting for this device. Provision is made to decide what to do in this case via the TABEXC subroutine. Level II BASIC adjusts the Position by dividing it by Width, printing that many newlines, and then TAB-ing over by the remainder. For example, PRINT TAB(129) will print two newlines and then TAB over 3 spaces.)

In my code, I simply write a newline to the device, and its xPR1CH routine will automatically adjust the Cursor position to 0.

```
;
;     TABEXC -- CORRECT CONDITIONS WHEN TAB() POSITION
;          EXCEEDS SCREEN/PRINTER WIDTH
;     ASSUMES:       BARGS(0)        POSITION
;                    BARGS(1)        MAX
;                    WARGS(0)        &CURSOR
;                    WARGS(1)        &ROUTINE
;
TABEXC      EQU     $
            LD      A,0DH                ; CARRIAGE RETURN/NEWLINE
            LD      HL,(WARGS+1*2)       ; PRINT ROUTINE
            CALL    HLSUB                ; PRINT A NEWLINE
            RET                          ; FOR NOW...
```

(Here's an example of leaving values accessible in BARGS and WARGS. We can easily use HLSUB to call the xPR1CH subroutine and print a newline to the device we're TAB-ing on.)
Of course, it would be counterproductive to have to specify the "MAX" Width, address of the Cursor position variable, and xPR1CH subroutine each time we wanted to TAB on the Screen or on the Printer. So here's a pair of "wrapper" procedures which allow the user to simply specify the TAB Position in BARGS[0] and automatically use TAB to adjust the Cursor accordingly.

```
;     PRINTB -- PRINTER TAB: USES TAB SUBROUTINE
;          TO TAB PRINTER CARRIAGE TO
;          "POSITION" OR START A NEW LINE IF
;          THAT WILL PUT HEAD PAST CARRIAGE
;          WIDTH (SET IN "PRTCW").
;     ARG:           POSITION             BARGS(0)
;     REPORTS:       ANY ROM ERRORS       BERROR
;
PRINTB  EQU     $
          CALL      PREPRT               ; PREPARE TAB ARG LIST
          CALL      TAB                  ; TAB TO POSITION
          RET
```

```
;     SCPRTB -- SCREEN TAB: USES TAB SUBROUTINE
;           TO TAB TO "POSITION" ON SCREEN,
;           OR START A NEW LINE IF "POSITION" IS
;           BEYOND SCREEN WIDTH (SET IN "VIDW").
;     ARG:          POSITION          BARGS(0)
;     REPORTS:      ANY ROM ERRORS    BERROR
;
SCPRTB EQU    $
       CALL   PREPSC                  ; PREPARE TAB ARG LIST
       CALL   TAB                     ; TAB TO POSITION
       RET

;     PREPRT -- PREPARE TAB ARGUMENT LIST FOR PRINTER
;
PREPRT EQU    $
       LD     A,(PRTCW)               ; PRINTER CARRIAGE WIDTH
       LD     (BARGS+1),A
       LD     HL,PRTCUR               ; PRINTER HEAD POSITION
       LD     (WARGS+0*2),HL
       LD     HL,LPR1CH                ; "PRINT 1 CHAR"
       LD     (WARGS+1*2),HL          ;     SUBROUTINE ADDR
       RET

;     PREPSC -- PREPARE TAB ARGUMENT LIST FOR SCREEN
;
PREPSC EQU    $
       LD     A,(VIDW)                ; SCREEN WIDTH
       LD     (BARGS+1),A
       LD     HL,VIDCUR               ; &CURSOR
       LD     (WARGS+0*2),HL
       LD     HL,VPR1CH               ; "PRINT 1 CHAR"
       LD     (WARGS+1*2),HL          ;     SUBROUTINE ADDR
       RET
```

SCPRTB "Screen Print TAB()" uses PREPSC to set up its BARGs and WARGs, and then CALLs TAB to PRINT TAB(n) on the Screen. PRINTB does the same with its values and uses TAB for the Printer. Here's how to PRINT TAB(20) on the Printer:

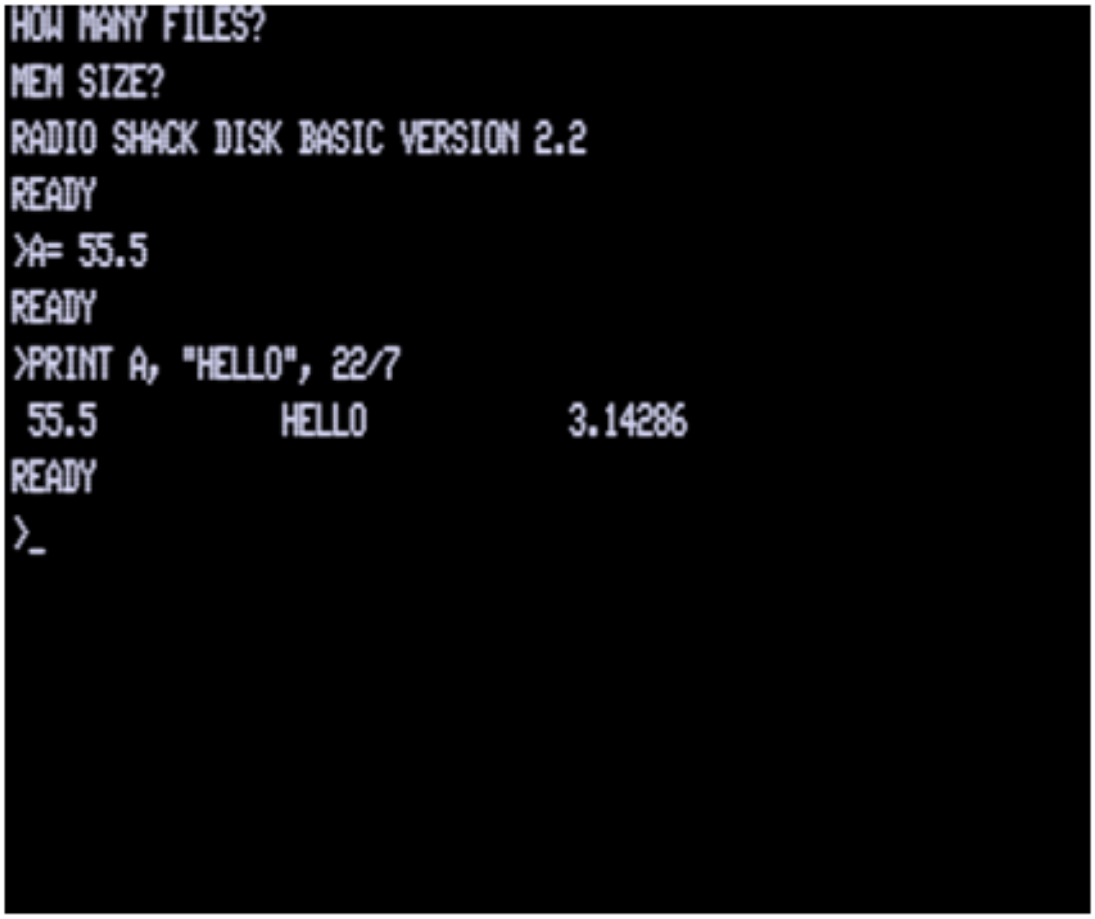
```
       LD     A,20
       LD     (BARGS+0),A
       CALL   PRINTB                  ; MOVE PRINT HEAD TO COLUMN 20
```

To do the same on the Screen, simply CALL SCPRTB instead.

TAB also serves well to implement the “trailing comma”. Remember how in BASIC we could

```
PRINT A, "HELLO", 22/7
```

and the output would look something like



The “trailing comma” after A and "HELLO" causes PRINT to TAB over to a preset column which is a multiple of 16. Here’s how I implemented a generalized “COMMA” subroutine which uses TAB to move the Cursor.

```
;     COMMA( COMMAWIDTH, CURSOR, WIDTH, ROUTINE)
;     EMULATE BASIC'S "TRAILING COMMA" IN A PRINT
;          STATEMENT, AS IN
;          "PRINT X, Y, Z"
;     WHERE THE COMMA ADVANCES TO THE NEXT TAB STOP
;          ADVANCE CURSOR TO NEXT TAB STOP
;          SPECIFIED BY THE COLUMN# CALCULATED
;          AS THE NEXT GREATEST MULTIPLE OF
;          COMMAWIDTH, OR PRINT A NEWLINE IF
;          THAT EXCEED WIDTH, ON EITHER THE
;          PRINTER OR SCREEN BY "ROUTINE".
;     ARGS:      COMMAWIDTH          BARGS(2)
;                WIDTH               BARGS(1)
;                &CURSOR             WARGS(0)
;                &ROUTINE             WARGS(1)
;     REPORTS:   ANY ROM ERRORS      BERROR
;
COMMA     EQU     $
          LD      HL,(WARGS+0*2)
          LD      C,(HL)              ; CURSOR POS'N -> C
          XOR     A                   ; COMMA COLUMN= 0
COMMA1    EQU     $
          CP      C                   ; WHILE (COMMA-COLUMN <=
          JP      NC,COMMA9           ;      CURSOR POS'N):
          LD      HL,BARGS+2
          ADD     A,(HL)                   ; COMMA-COLUMN +=
                                           ;    COMMA-WIDTH
          JP      COMMA1              ; :WEND
COMMA9    EQU     $
          LD      (BARGS+0),A         ; TAB POSITION
          CALL    TAB
          RET
```

This procedure lets us define our own "Comma Width", which is fixed at 16 columns in BASIC. (It's even possible to adjust Comma Width "on the fly", before each CALL to COMMA.) The subroutine works by starting "Comma Column" at 0 in the accumulator. As an example, let's assume we've already printed "HELLO WORLD", so the Cursor position is at column 11, counting from 0. As well, let's say we set the Comma Width to 16, just like in BASIC. The Comma Column is compared with the Cursor Position, and as long as it is less than that, the Comma Width is added to it. In our example 16 is added to the Comma Column, and we loop back to compare it with the Cursor Position again. This time 16 is > 11, so we jump out of the loop to COMMA9, where the Comma Column in A is stored in the TAB Position at BARGS[0]. Then we call TAB to move the Cursor there.

Once again we have individual "wrapper" subroutines to make using COMMA easy for the Screen and for the Printer:

```
;     SCOMMA -- EMULATE BASIC'S "TRAILING COMMA"
;           FOR SCREEN.
;
SCOMMA    EQU     $
          LD      A,(VIDCMW)      ; SCREEN "COMMA WIDTH"
          LD      (BARGS+2),A
          CALL    PREPSC          ; PREPARE TAB ARG LIST
          CALL    COMMA           ; TAB TO NEXT COMMA COL.
          RET

;     PCOMMA -- EMULATE BASIC'S "TRAILING COMMA"
;           FOR PRINTER.
;
PCOMMA    EQU     $
          LD      A,PRTCMW)       ; PRINTER "COMMA WIDTH"
          LD      (BARGS+2),A
          CALL    PREPRT          ; PREPARE TAB ARG LIST
          CALL    COMMA           ; TAB TO NEXT COMMA COL.
          RET
```

Let's test it out, shall we? In the chapter where he introduces the trailing comma, Dr. Lien also teaches FOR…NEXT loops. Did you know that BASIC's programming language "predecessor" FORTRAN had DO loops that could only use *integers* as the index variable, and could only count *up*? BASIC was a great improvement. You can write

```
10 E= 1 + 1E-1: REM "1.1"
20 FOR X= (22 / 7) TO E STEP 3.14
30        PRINT X
40 NEXT X
```

and use floating-point variables and even expressions to count up *or* down. Did you also know that Level II BASIC also processed each FOR…NEXT loop at least once, regardless of the limits (x to y)? For example, try

```
10 FOR X= 11 TO 10 STEP 1
20        PRINT X
30 NEXT X
```

An "11" will be PRINTed. Ah, trivia. (Microsoft corrected this with 8086 QBASIC.)

Anyway, we'll develop our test code from pages 49-50 of Dr. Lien's book:

```
10 REM * TIME VS RATE FLIGHT CHART *
20 CLS
```

```
30 D = 3000
40 PRINT "    B O S T O N  T O  S A N  D I E G O     "
50 PRINT
60 PRINT "RATE (MPH)", "TIME (HOURS)", "DISTANCE (MILES)"
70 PRINT
80    FOR R=200 TO 1000 STEP 100
90     T = D/R
100    PRINT R,T,D
110   NEXT R
```

Enter the program and RUN.

It is really solving the problem from Chapter 3 nine times in a row, for different values, and printing out the result. Your screen should look like this:

```
          B O S T O N  T O  S A N  D I E G O

RATE (MPH)       TIME (HOURS)     DISTANCE (MILES)

  200               15               3000
  300               10               3000
  400               7.5              3000
  500               6                3000
  600               5                3000
  700               4.28571          3000
  800               3.75             3000
  900               3.33333          3000
 1000               3                3000
```

50

This is my Assembly Language version:

```
MAIN    EQU         $
        CALL        INIT
        CALL        I00CRM          ; USES MATH
```

We covered CALL INIT and CALL 100CRM in the June 2025 issue of TRS8Bit. These initialize RAM locations related to the ROM calls we're going to use, especially the MATH routines.

```
;
;   "10 REM * TIME VS RATE FLIGHT CHART *"
;   "20 CLS"
;
        CALL        01C9H           ; CLS
;
;   "30 D = 3000"
;
        LD          HL,N3000        ; "3000" S.P. CONSTANT
        LD          DE,DVAR         ; "D" VAR
        LD          BC,4
        LDIR                        ; "D" = "3000"
```

In the September 2025 issue, I listed and demonstrated a BASIC program which has you type in a number like "3000", and which will display the 4 bytes it's stored as in Single Precision mode. For this program, we have four constants: 100, 200, 1000, and 3000, stored in that format. Here's "N3000", which stores the constant 3000 as a Single Precision value:

```
N3000   DEFB        0           ; 3000 IN S.P.
        DEFB        128
        DEFB        59
        DEFB        140
```

You'll find this and the rest of the constants at the end of our listing, below. Next, we PRINT some caption lines.

```
;
;   '40 PRINT "   B O S T O N  T O  S A N  D I E G O"
;
        LD          HL,TITLE
        CALL        PRINT
;
;   "50 PRINT"
;               (2 NEWLINES INCLUDED IN "TITLE",
;                   USER DATA BELOW)
```

```
;
;     ' 60 PRINT "RATE (MPH)", '
;
          LD        HL,SUBTL1
          CALL      PRINT
          LD        A,16
          LD        (VIDCMW),A
          CALL      SCOMMA
```

Here's where we begin to use the Screen version of the trailing comma, CALL SCOMMA.

```
;
;                   '           "TIME (HOURS)", '
;
          LD        HL,SUBTL2
          CALL      PRINT
          CALL      SCOMMA
;
;                             '                   "DISTANCE (MILES)"
;
          LD        HL,SUBTL3
          CALL      PRINT
;
;     "70 PRINT"
;               (2 NEWLINES INCLUDED IN "SUBTL3",
;                    USER DATA BELOW)
;
```

Here I've implemented a FOR…NEXT loop, trying to keep as faithful to the original interpreter's actions as possible. Is this the fastest or best way? Probably not: all the constants and the variable in this FOR…NEXT are integers, and it might be a tad quicker to only convert them to S.P. when necessary. But these articles are designed to demonstrate ROM calls after all…

```
;
;     "80 FOR R=200 TO 1000 STEP 100"
;          PART 1: "R=200"
;
          LD        HL,N200             ; "200" S.P. CONSTANT
          LD        DE,RVAR             ; "R"
          LD        BC,4
          LD        A,C                 ; NTF "4"
          LD        (NTF1),A            ;     IE, S.P.
          LDIR                          ; "R" = "200"
```

```
;
;     "90 T = D / R"
;
FOR001  EQU         $                   ; (FOR LOOP)
        LD          HL,RVAR
        LD          DE,WRA1
        LD          BC,4
        LDIR                            ; WRA1= "R"
        LD          DE,(DVAR)
        LD          BC,(DVAR+2)         ; BC/DE= "D"
        CALL        GETSPC              ; "ON ERROR RESUME NEXT"
        CALL        08A2H               ; WRA1= D / R
        LD          HL,WRA1
        LD          DE,TVAR
        LD          BC,4
        LDIR                            ; "T" = WRA1 ("D"/"R")
        CALL        BERTST
        JP          NZ,ERRORS           ; "ON ERROR GOTO ERRORS"
```

If you've been following along, you know from the June 2025 issue that "GETSPC" is our subroutine to recover from ROM errors by fixing up the stack and resuming execution at the next instruction. Since division is particularly susceptible to "divide by zero" errors, we also use BERTST to check BERROR before continuing.

```
;
;     "100 PRINT R, T, D"
;
;          PART 1: PRINT R;
;
          LD          HL,RVAR
          CALL        PRINSP              ; PRINT S.P. VAR
```

Because I had to print the contents of 3 Single Precision variables in a row here, I decided to "factor out" the job to "PRINSP".

```
;
;     PRINSP -- PRINT S.P. VARIABLE (4 BYTES)
;          WHOSE ADDRESS IS IN HL
;          ARG:        VAR ADDR        HL
;
PRINSP  EQU         $
        CALL        SP2STR              ; CONVERT S.P. -> STRING
        CALL        PRINT               ; PRINT IT
        RET
```

PRINSP calls on SP2STR, below, to convert a Single Precision value pointed to by HL to a 0-terminated string located in BASIC's internal buffer.

```
;
;     SP2STR -- CONVERT S.P. VARIABLE (4 BYTES)
;          WHOSE ADDRESS IS IN HL
;          TO STRING
;     ARG:          VAR ADDR          HL
;     RETURNS:      ADDR OF STRING    HL
;
SP2STR  EQU         $
        LD          DE,WRA1
        LD          BC,4
        LDIR                          ; WRA1= S.P. VAR VALUE
        LD          A,4
        LD          (NTF1),A          ; SINGLE PRECISION
        LD          BC,0505H          ; NO COMMAS OR DEC. POINT
        XOR         A                 ; NO EDITING
        CALL        GETSPC            ; ON ERROR RESUME NEXT
        CALL        0FBDH             ; CONVERT S.P. TO STRING
        RET                           ; HL= ADDR OF STRING
```

Then PRINSP calls on PRINT, the general purpose Print-0-terminated-String-To-Screen subroutine we developed in the March 2025 issue. Back to our main program code:

```
;
;          PART 2: TRAILING COMMA
;
        CALL        SCOMMA
;
;          PART 3: PRINT T;
;
        LD          HL,TVAR
        CALL        PRINSP
;
;          PART 4: TRAILING COMMA
;
        CALL        SCOMMA
;
;          PART 5: PRINT D
;
        LD          HL,DVAR
        CALL        PRINSP
        LD          HL,RETURN         ; NEWLINE
        CALL        PRINT
```

```
;
;     "110 NEXT R"
;
;          PART 1: IS R > 1000?
;
          LD        DE,RVAR
          LD        HL,N1000
          CALL      SPCMP           ; IS R >= 1000?
          JP        C,FOR009
          JP        Z,FOR009
```

True to Level II BASIC, we test our index value R at the *end* of the loop instead of doing it the *proper* way, like C. SPCMP, "Single Precision Compare", sets the C flag if the S.P. value pointed to by HL is *less than* the S.P. value pointed to by DE, sets the Z flag if they are *equal*, and sets neither if the value pointed to by HL is *greater*. SPCMP was developed in the September issue.

```
;
;          PART 2: NO: INCREMENT R
;
          LD        HL,RVAR
          LD        DE,WRA1
          LD        BC,4            ; NO: "R" -> WRA1
          LDIR
          LD        DE,(N100)
          LD        BC,(N100+2)     ; "100" -> BC/DE
          CALL      GETSPC          ; ON ERROR RESUME NEXT
          CALL      0716H           ; WRA1= "R" + "100"
          LD        HL,WRA1
          LD        DE,RVAR
          LD        BC,4
          LDIR                      ; "R"= "R + 100"
;
;          PART 3: REPEAT FOR LOOP
;
          JP        FOR001
;
;          PART 4: EXIT FOR LOOP
;
FOR009    EQU       $
          JP        FINISH
```

"FINISH" is where we restore the Math vectors we set up with CALL I00CRM at the beginning. Finally, here's what the "Data Division" of our program looks like:

```
TITLE   DEFM    '  B O S T O N  T O  S A N  D I E G O'
        DEFB    0DH
        DEFB    0DH
        DEFB    0
SUBTL1  DEFM    'RATE (MPH)'
        DEFB    0
SUBTL2  DEFM    'TIME (HOURS)'
        DEFB    0
SUBTL3  DEFM    'DISTANCE (MILES)'
        DEFB    0DH
        DEFB    0DH
        DEFB    0
;
;       SOME CONSTANTS:
;
N100    DEFB    0               ; 100 IN S.P.
        DEFB    0
        DEFB    72
        DEFB    135
N200    DEFB    0               ; 200 IN S.P.
        DEFB    0
        DEFB    72
        DEFB    136
N1000   DEFB    0               ; 1000 IN S.P.
        DEFB    0
        DEFB    122
        DEFB    138
N3000   DEFB    0               ; 3000 IN S.P.
        DEFB    128
        DEFB    59
        DEFB    140
;
;       S.P. VARIABLES
;
RVAR    DEFS    4               ; "R" S.P. VAR
TVAR    DEFS    4               ; "T" S.P. VAR
DVAR    DEFS    4               ; "D" S.P. VAR
```

(Why didn't I just use "R", "T", and "D" as my variable names, per the original BASIC program? The assembler sees R as the name of the Refresh register, and similarly D as the name of one of the general purpose registers.
Let's try it out.



What's left to cover in the book "BASIC: It's Easier Than You Think"? Graphics, Cassette I/O, and the remaining Math functions like RND, INT(), SQRT(), etc.



LEVEL 2.5 (TWOHAF) BASIC ALLOWS CASSETTE AND ACULAB FLOPPY TAPE USERS MANY OF THE TANDY DISK BASIC FEATURES, NAMELY -

"&", "DEF FN", "DEF USR", "MID$=", "INSTR", "MKD$", "MKI$", "MKS$", "CVD", "CVI", "CVS"

AVAILABLE, AS A **FREE DOWNLOAD**, FROM

WWW.TRS-80.ORG.UK

IN MAV'S HISTORY WORKSHOP

You Bought a Tandy Model 2000! What Now?



Part 3, by Ian Mavric

This time I talk about some of the best software for the Tandy 2000. Having the best hardware of the time is no use unless there is good software to run on it. While software was slow to arrive when the computer was released, over time more and

more software became available as the machine found it's particular niches. Here are some titles you really should try out if you have a T2K.

FIVE PROGRAMS EVERY TANDY 2000 *SHOULD* OWN

5. Games programs: Planetfall and The Witness

With the T2K being such an expensive computer games were likely hardly on the top of the list of things to do with it, but Tandy made some ports of popular games available to buyers.

Text adventures from Infocom's top programmers of the time: Planetfall, and The Witness, were available at the launch of the T2K. Text-only, so that they would run on the most basic T2K with monochrome screen and no graphics adapters installed. Each package came with a quantity of paraphernalia required to complete each game. These are as close as 'big box' game titles came to the Tandy 2000.



Photo: Game programs the Tandy 2000 (source: RSC-11)

When you look for these two games on eBay make sure all the contents are included with each game.

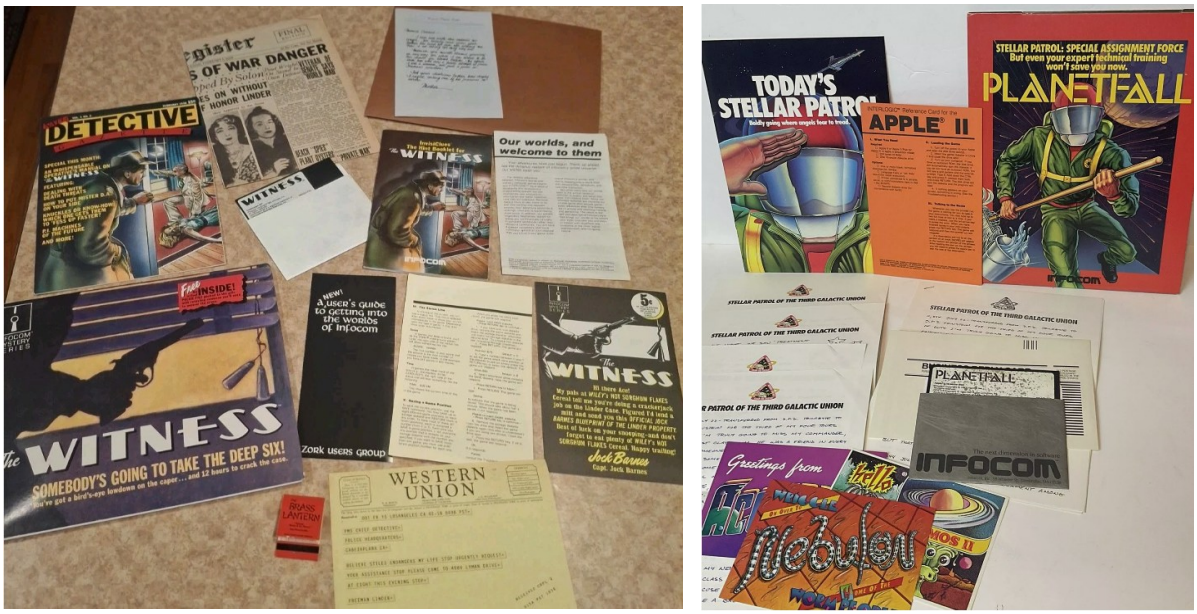


Photo: Contents of The Witness and Planetfall adventure game packs

4. DeskMate and DeskMate II

People mainly associate DeskMate with the Tandy 1000 series, as that was their initial exposure to the quaint-but-useful apps package which came with that machine.

On later T1Ks it was even included in ROM for rapid startup of those machines. History will show that after the T1K release versions were written for the TRS-80 Model 4 (as a stand alone package and included with the Tandy Model 4D) and two Coco versions for the CC1/CC2 and CC3.

In 1985 Radio Shack did a port to the T2K and as previously stated, the high performance of the machine made a version which ran far better on the T2K than where it originated on the T1K.

A few years after DeskMate was released an updated version called, appropriately, DeskMate II was released and there was a T2K version which carried catalog number 25-1164.



Photo: DeskMate for the Tandy 2000 cat. no. 26-5316

3. Graphics programs: DR DRAW and DR GRAPH

Now we are getting seriously into the zone of what the T2K goes best, it's graphics. If you have the CM-1 monitor and associated high-resolution graphics board you can now start producing some striking colour displays. DR Graph is a charting program, a very popular app at the time, and DR Draw is like an early version of MacPaint or CorelDraw.

You paid a kings ransom for your CM-1 monitor and card, now you can get a good use out of them.



Photo: Graphics programs the Tandy 2000 (source: RSC-14)

2. MS-WINDOWS 1.01

The Windows we know and love (or for some, love to hate) had to start somewhere and in this instance it was released in 1985 as MS-Windows 1.01 for the IBM PC/AT and compatibles.

History will show that Windows had a humble start that really ramped up from Windows 3.1, and early versions found it hard to find early adopters. Bill Gates knew he was onto something and his latest creation, known as MS-Windows before the "MS-" was dropped, performed best on computers with fast CPU and large amounts of memory, and preferably a hard disk. The T2K was ideal.

He even featured in an advert for Radio Shack to help sell the T2K, and no doubt get people talking about MS-Windows.

To get MS-Windows running on your Tandy 2000 you can find all you need at this website:

https://github.com/Tandy2K/Tandy2000/tree/master/Software/Windows%201.01



"Designing a revolutionary concept in software demanded a computer with extraordinary performance. The Tandy 2000 delivered." —Bill Gates
Chairman of the Board, Microsoft

Bill Gates has been at the leading edge of personal computing from the very beginning. His company is a leading producer of microcomputer software.

"Our newest software product, MS-Windows, is an integrated windowing environment. It will let personal computer users combine individual programs into a powerful, integrated system.

"When we set out to design MS-Windows in color, we knew that the Tandy 2000 computer would let us turn an extraordinary product into a work of art. The graphics are sharp and crisp, and gave us a degree of creativity like nothing before.

"Our engineers were quite impressed with the processing speed of the Tandy 2000's 80186 microprocessor, too. And while the finished product will utilize the 2000's Digi-Mouse, the well-laid out keyboard has helped us speed through the design stage.

personal computer? Go ahead, watch how much faster today's most sophisticated programs run on the high-technology Tandy 2000.

You can choose from the hot-

Available at over 1200 Radio Shack Computer Centers and at participating Radio Shack stores and dealers.

Radio Shack®
COMPUTER CENTERS
A DIVISION OF TANDY CORPORATION

1. MICROSOFT FLIGHT SIMULATOR 1.0

Microsoft Flight Simulator can be traced back to SubLogic who wrote the original T80-FS1 flight simulator for the TRS-80 Model I. Microsoft's Flight Simulator 1.0 was released for the PC/XT but it was also found to be easily ported to take advantage of the special PCjr graphics mode of the Tandy 1000.

While at it a version was completed for the T2K and was sold through Radio Shack stores as pictured below, for the Tandy 1000, 1200HD and 2000. While the artwork for the non-RS version is the same, the non-RS version only works in CGA mode on IBM PCs and compatibles - bear this in mind if you are buying it off eBay.

While the simulation runs at a playable rate on the IBM PC/XT and T1K at their 4.77MHz CPU speed, on the Tandy 2000 it runs at almost double the FPS (frames per second) making for a much more pleasant user experience. CGA on the XT plays in 320x200 4 colours, and in PCjr graphics on the T1K (320x200 in 8 colours) on the Tandy 2000 with the CM-1 monitor you get the full 640x200 experience.

Flight Simulator, like The Witness and Planetfall were inexpensive compared to other T2K software; initially $US49.95 so they sold in large volume and come up regularly on eBay, just keep your eyes open for it.



Photo: Tandy version of MS Flight Simulator 1.0 specifically for their machines.

2. HONOURABLE MENTION: VersaCAD - ENTRY LEVEL

This is the biggie and I have left it for last and numbered it zero because realistically even the most ardent Tandy 2000 enthusiast is going to struggle to achieve getting this program running. Computer Aided Design (CAD) package VersaCAD came out fairly late in the T2Ks life and was an expensive program with system requirements to match. The VersaCAD package itself wasn't overly expensive at $US399.95, but the system requirements to get the most out of it totalled a hefty sum. A full graphics system T2K with CM-1 monitor, expanded memory (preferred) beyond 256K, a hard disk drive (though it could be run on floppy disks), a Digi-mouse and mouse controller card, and a GT-2000 graphics input tablet. Printing needed a Radio Shack or Houston Instruments flatbed plotter.



Picture: a Tandy 2000 running VersaCAD - Entry Level.

Very few people at the time had the system requirements to run VersaCAD, or had an occupation which required a PC running CAD software, so very few copies were sold. It's almost impossible to find on eBay or other online forums. You can, however, like MS-Windows 1.01 find the T2K version on GitHub if you would like to attempt a setup or run it in a limited capacity:
https://github.com/Tandy2K/Tandy2000/tree/master/Software/Versacad%20Entry%20Level%2026-5282

Ian Mavric/

ianm@trs-80.com



Ian Mavric is an IT Specialist who also restores and collects TRS-80's and classic cars. He lives with his wife and kids in Melbourne, Australia.

LOOKING FOR FAST, INEXPENSIVE, UNLIMITED MASS STORAGE FOR YOUR TRS-80 MODEL I/III/4/4P/4D?

The amazing

"FreHD"

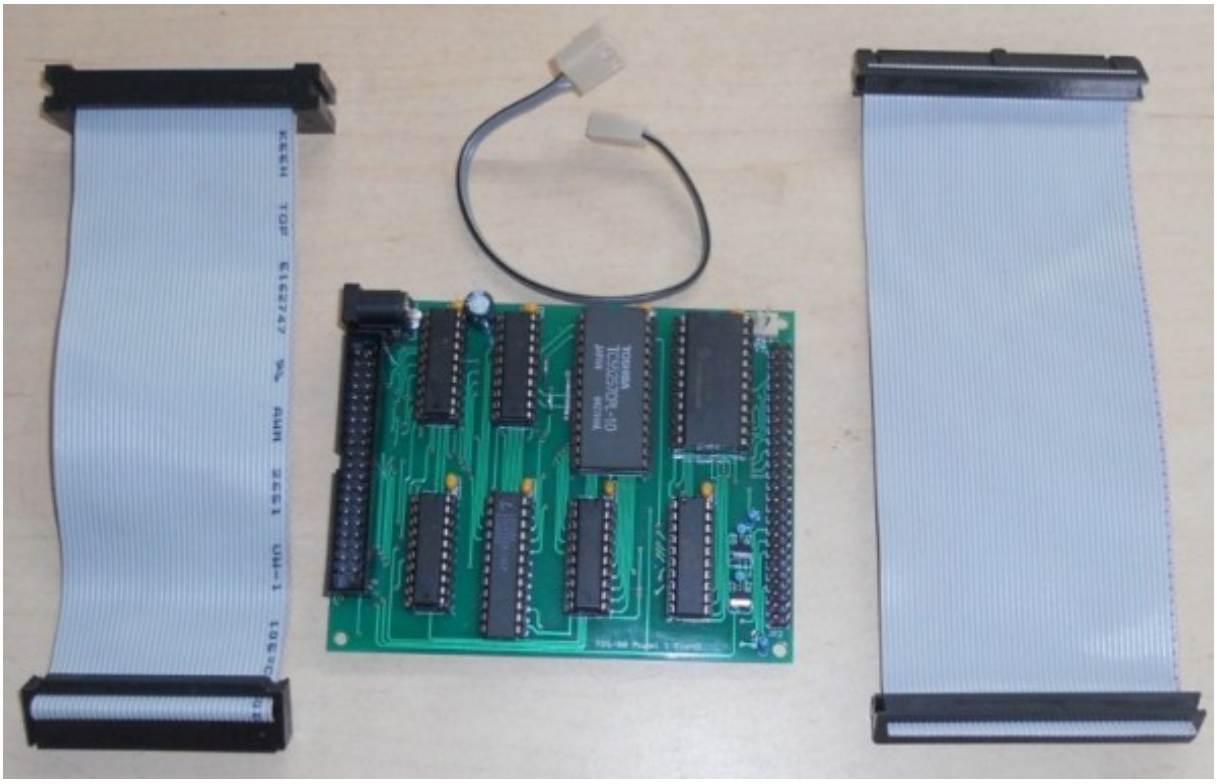


-Emulates a TRS-80 hard drive, but faster than any hard drive!
-Works with your favourite DOS (LS-DOS, LDOS, CP/M, Newdos/80 2.5)
-Uses SD card for storage medium
-Bonus free Real Time Clock function!
-Designed in Belgium and proudly built and shipped from Australia
-Kit form or fully assembled

Order yours today
http://trs-80.com.au/trs80/emulator.htm

“Quinnterface” Mini Expansion Interface for 16K Model 1 FreHD users.

- COMPLETLY ASSEMBLED AND TESTED -



This is THE perfect device for all Model 1 users who own a 16K Level II unit, (which is most of us!), but no Expansion Interface or disk drives, especially if you don't want to modify your M1 with upgraded boot ROM or memory upgrade.

The ‘Quinnterface’, developed by J. Andrew Quinn from New Zealand, adds 32K RAM and auto-boot functionality to you FreHD.

U.K. And Europe, contact Bas. at BetaGamma Computing.
U.S.A. And rest of the world, contact Mav. At “The Right Stuff”

Programming Languages for the Model I

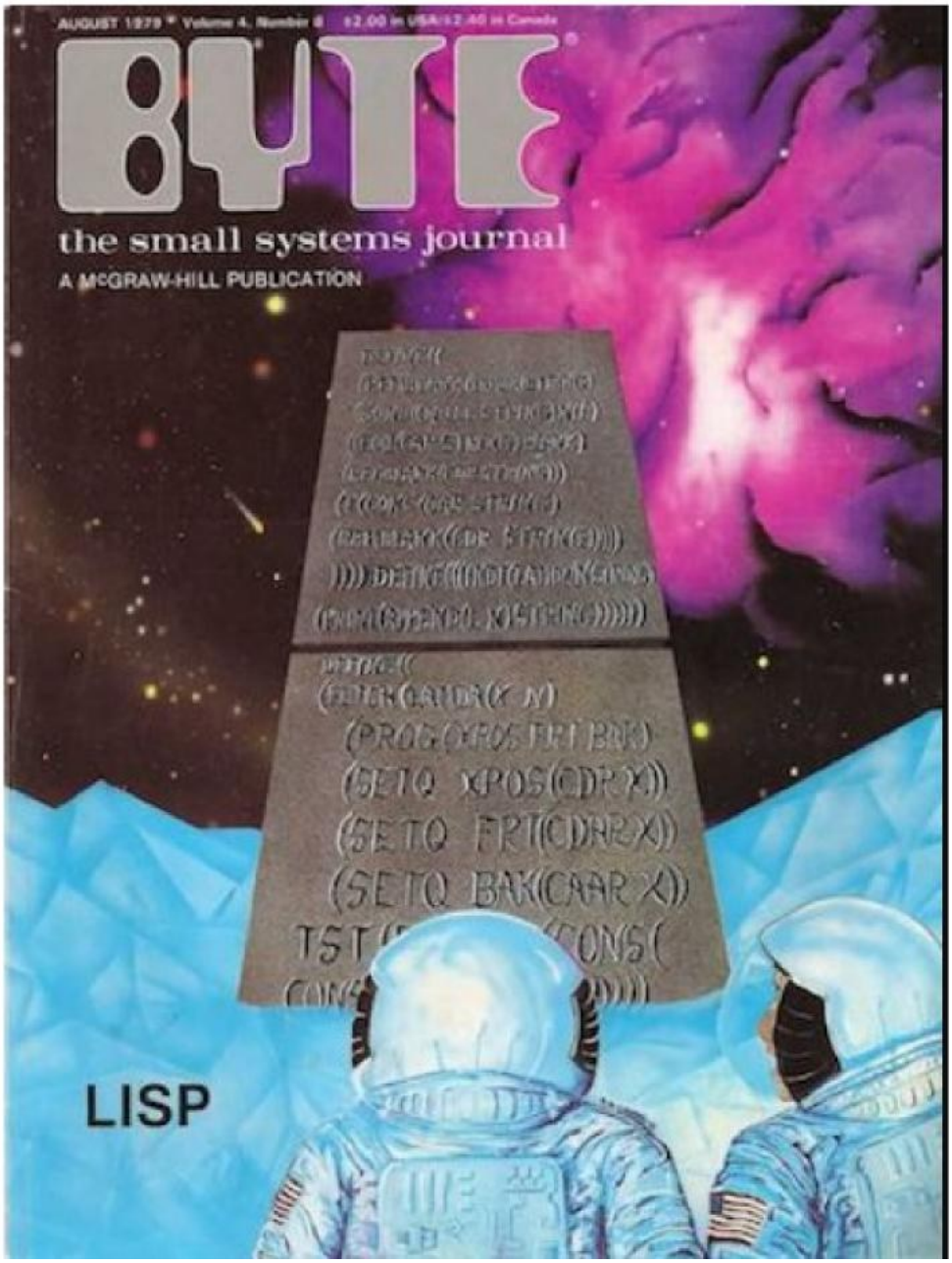
Making a LISP, Checking it Twice

Vince Otten

LISP? Oh, no. Not LISP. Not “Lots of Irritating Stupid Parentheses”?!? Or should I say,

```
(MAKE (LIST '(CHECK-IT (TIMES 2 X))))
```

Blame it on *Byte Magazine* yet again. Following their lead, I’ve been looking at some early programming languages that were available for my Model I in the late 1970s and early 1980s. We’ve covered APL and Pascal, and now there’s...



I think artist Robert Tinney (who sadly just passed away within the past month) summed up LISP well: it's a foreign language, created by otherworldly intelligences. So why would it pique my interest in those early days when I was only familiar with BASIC and programmable calculators?

I'd wanted to make a Z80 assembler in BASIC. It's that simple—and that difficult. I'd already created a sort-of-8080-dissassembler on my 4K TRS-80 so I could decode the BASIC ROM. This was at least a year before *Microsoft BASIC Decoded* by James Farvour, and decades before Ira Goldklang's thorough trs-80.com provided annotated listings of the ROMs. But I couldn't afford an assembler yet, and I'd read an article (in Byte, of course) on how to "roll your own", so I wanted to try.
I soon realized that an assembler needs one or more "symbol tables": records of items like instruction codes, variable names, etc. along with information on what to do with them.

Symbol	Opcode	Address	Value	Pointers
RET	C9			
MYVAR		3012		
		407F		
		1234		
SIZE			15	
$				---
(IX+4)			---	

...or something like that. I wasn't sure. I wanted to figure it out as I went along. But if I was coding this in BASIC, I'd have to create an array to store all this in. How about

```
DIM SY$(1000,5)
```

Wait. "Opcode", "Address", "Value", and "Pointers" aren't BASIC Strings. Okay, I'll just make two arrays:

```
DIM SY$(1000), OT%(1000, 4)
```

"Symbol" and "Other". But I soon realized that the combination of BASIC assembler program, assembly source ("SR$(500)", perhaps), SY$(), and OT%() might take more space than my by-now-16K Model I had. How big did I have to make my arrays? If only they could grow—and shrink!— as needed.

Once I read a bit about LISP, I realized that it provided for exactly those kinds of requirements. It let you create "lists"—kind of like C's "structs" or Pascal's "records"—variables that could contain many different values: strings, integers, even more lists, which could even contain LISP

code themselves! Why, you could make an item called LINE-OF-CODE, which might contain elements like LABEL, INSTRUCTION, ARGS, LINK-TO-LABEL, WHAT-TO-DO, where each part might be a string, or an address, or even some LISP code that might be executed. All these lists could be created, expanded, contracted, and deleted.

So you can imagine that I got excited about this unique programming language. In 1979's Canada, I didn't see anything other than BASIC and machine code programs for my TRS-80, though. My first encounter with a LISP interpreter was on an Apple][in one of the first computer stores in London, Ontario. I was such a regular—I'd almost bought a Commodore PET there before I got my TRS-80—that they kindly printed off the BASIC source code for me. I rushed it home and entered it into my 16K machine.

No luck. I don't remember if it was because of the Apple-specific details or typos, but I never got it to work. I'd bought a copy of Laurent Siklossy's *Let's Talk LISP*, but between that text and the BASIC program, I gave up and turned my attention to other languages like Pascal and APL. Recently, I've wondered if I might "get it" after learning a bit about more modern LISPs and their implementations via the Interwebs. I found a LISP-in-BASIC in the March 1983 issue of *80 Microcomputing*, and typed in the entire source from the magazine just like in the old days. Here's how it starts:

Program Listing

```
5 REM      * BASIC LISP VER 1.1 *
10 REM * BY RANDY BEER; AUG., 1981 *
15 CLS:CLEAR325:DEFINTA-E,G-V,X-Z:DEFSTRO:DIMLM(1100),PL(1100),O
B(90),PT(90),ST(350),FP(50),T1(15),X1(15):N=3000
22 PRINTTAB(23)"BASIC LISP VER 1.1":PRINT:PRINT"INITIALIZING . .
 . WAIT":PRINT
24 FORJ=0TO48:READOB(J),PT(J):NEXT:PE=48:FE=1:OB(46)=CHR$(13):FP
(1)=MEM
26 FORJ=1TO1099:PL(J)=J+1:NEXT:PL(1100)=N:AS=1
28 T=3001:LP=3043:RP=3044:CC=33:N1=58:N2=44:LB=3031:QU=3030:NB=3
032
30 A=0:QT=0:J=0:PRINT:PRINT"$ ";:ONERRORGOTO26000:GOSUB50:GOSUB2
65:GOSUB210:GOTO30
50 J1=0:PRINTCHR$(14);:GOSUB90
55 GOSUB100:IFX<>LPRETURN
60 J1=J1+1:X1(J1)=AS:T1(J1)=AS:LM(T1(J1))=0:AS=PL(AS):IFQRETURN
65 GOSUB55:IFX=RPGOTO80
70 IFLM(T1(J1))<>0THENPL(T1(J1))=AS:T1(J1)=AS:AS=PL(AS)
75 LM(T1(J1))=X:IFQRETURNELSE65
80 PL(T1(J1))=N:X=X1(J1):IFLM(X)=0ANDPL(X)=NTHENPL(X)=AS:AS=X:X=
N
85 J1=J1-1:RETURN
90 A$=INKEY$:IFA$=""THEN90ELSEPRINTA$;:KK=ASC(A$):RETURN
100 IFKK=40THENX=LP:GOTO200
105 IFKK=41THENX=RP:IFJ1=1ORJ1=2ANDQTRETURNELSE200
110 IFKK=39THENQ=-1:QT=QT+1:GOSUB60:LM(T1(J1))=QU:Q=0:GOSUB90:GO
SUB55:Q=-1:GOSUB70:Q=0:GOSUB80:QT=QT-1:RETURN
```

Listing continues

By the way, I'd forgotten how careful the editors of *80 Micro* were to ensure that the listing exactly matched the formatting of a 64-column screen. That definitely helped with entering and debugging the source code! And debugging was necessary. Not only did I have to fix the usual typos, like replacing an accidental "-" in place of the proper "=". I also had to repair some of the source, like

```
IF J1=1 OR J1=2 AND QT RETURN ELSE 200
```

(spaces added for readability). The BASIC interpreter is going to understand this as

```
IF J1=1 OR (J1=2 AND QT) RETURN ELSE 200
```

so I corrected it to

```
IF (J1=1 OR J1=2) AND QT RETURN ELSE 200
```

I felt pretty proud of my years of Level II BASIC experience! Correcting this subtly changed the behavior of the LISP interpreter. You see, it "reads" user input *one character at a time* (see line 90 in the source code fragment above). As it was, as soon as you typed in a complete LISP expression like

```
(CAR '(A B C))
```

it would immediately begin processing it *without waiting for the <ENTER> key.* Now at least it waits for that <ENTER>, which is a little less confusing, because even with that correction, this LISP is *slooooooooow.*

That's okay—after all, it's a proof of concept, and it can be used to learn LISP. But there's another problem here. Since this interpreter is using BASIC's INKEY$ to read your input, and because it immediately begins to try to make sense of your typing, you can't backspace over an error like

```
(CARX '(A B C))
```

This LISP will report an error, and let you try again, which is nice, until you mistype in the middle of a lengthy bit of coding. At that point, you might as well type all the closing parentheses and try all over. So how do you correct errors? Say you wanted to type

```
(SET'BROTHERS'(RALPH JOHN))
```

but accidentally entered

```
(SET ' BOTHERX '(RALPH OHN))
```

(which is quite easy to do with the INKEY$ input routine responding to your keystrokes). Can you "delete" BOTHERX from LISP's memory? This interpreter provides a solution: the "(%)" function.



That's tedious, if workable. We had to delete every element (what LISP calls an "atom") from the "object list", so I had to type (%) three times. Oh, well. At least it works.
Sort of.

For example, see the last user entry in the screenshot:

```
(SETQ BROTHERS '(RALPH JOHN))
```

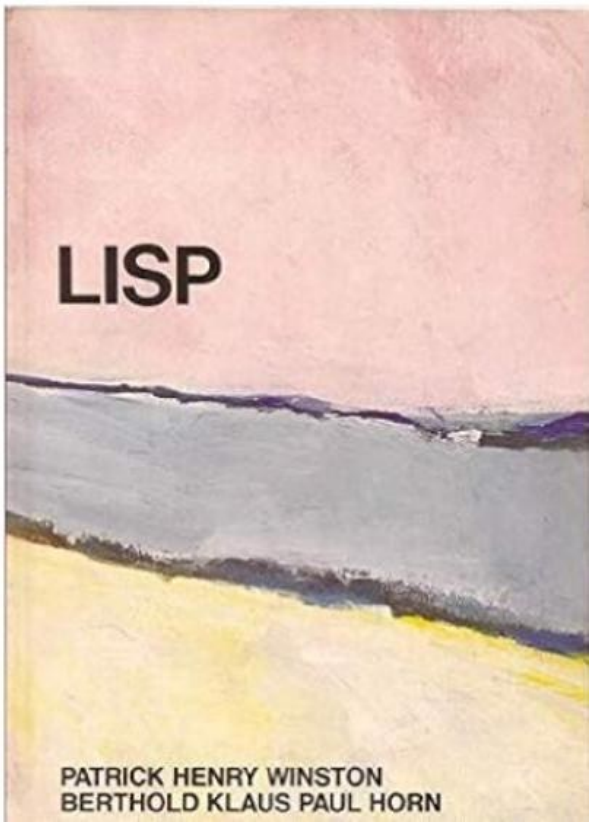
(It removes the necessity of "quoting" the name of the variable (BROTHERS) that you're defining.) "SETQ" works: it does set BROTHERS to have the value (RALPH JOHN). But it also reports ";ERROR" and that makes using it in function definitions impossible (so I use SET instead).
For me, that's still a quibble. If I can get the rest of the original BASIC code to work, I'd love to figure it out, perhaps changing the input method, adding some features, and I can try to create an Assembly Language version.

But

1. It wasn't 1983 yet, so I didn't have access to this fascinating set of articles by Randy Beer; and
2. Meanwhile, an advertisement for a commercial LISP appeared!



"Shut up and take my money!" I yelled, and mailed in my order. I also asked them if there was a LISP textbook they recommended:



Now I was off to the races. I opened the UO-LISP manual and the textbook, and *immediately got bogged down.* The syntax of the manual's code samples didn't match the book at all, so I set it aside, and tried to learn simply from the program's documentation. But it was simply too early in my learning-programming-languages-on-my-own journey, and I soon set LISP aside for other TRS-80 interests.

Recently, I located UO-LISP on Ira Goldklang's trs-80.com website. I also found some more documentation at softwarepreservation.computerhistory.org, including manuals and newsletters. Using these, I tried out UO-LISP once more.

After all, it's got some amazing features:

- It's not only a LISP interpreter, but also a LISP compiler
- Disk access
- Can compile –and assemble—down to machine code
- It's fast
- It comes with "REDUCE", a more modern-looking C-like programming language, *made in LISP*

I didn't know what "kind' of LISP this was—more on this later. So I just followed the instructions in the tutorial (https://softwarepreservation.computerhistory.org/LISP/utah/uolisp/UO-Lisp-Tutorial Version 2.0-1984.pdf).
That worked up until about page 48 or so, where functions are being defined with "local variables". It turns out the TRS-80 version of UO-LISP 'DEFine function' method won't let you do much at all with local variables inside COND or PROG—contrary to the tutorial! That's pretty unhelpful.

I thought I might be able to figure something out if I knew what "kind" of LISP this was, and then follow through a textbook or two. I put out feelers on Reddit and in LISP user groups, and got just a couple of replies. "Yeah, UO-LISP is a version of 'Standard LISP', and the authors knew that local variables weren't working properly," is the gist. *Well, if they didn't know, why did they sell it?* $49.95 in US dollars, and the $20 I probably paid for the textbook, was a lot of money back then.

So, what's "Standard LISP"? It's a pretty official-sounding name for one of dozens of variations of the language back in the day, and I never did find any sort of textbook or tutorial for it. It turns out that just about every kind of LISP was noticeably different than its relatives. No, not like 8-bit BASICs: code for the PET and the Apple and the TRS-80 actually had more in common than these LISPs did.

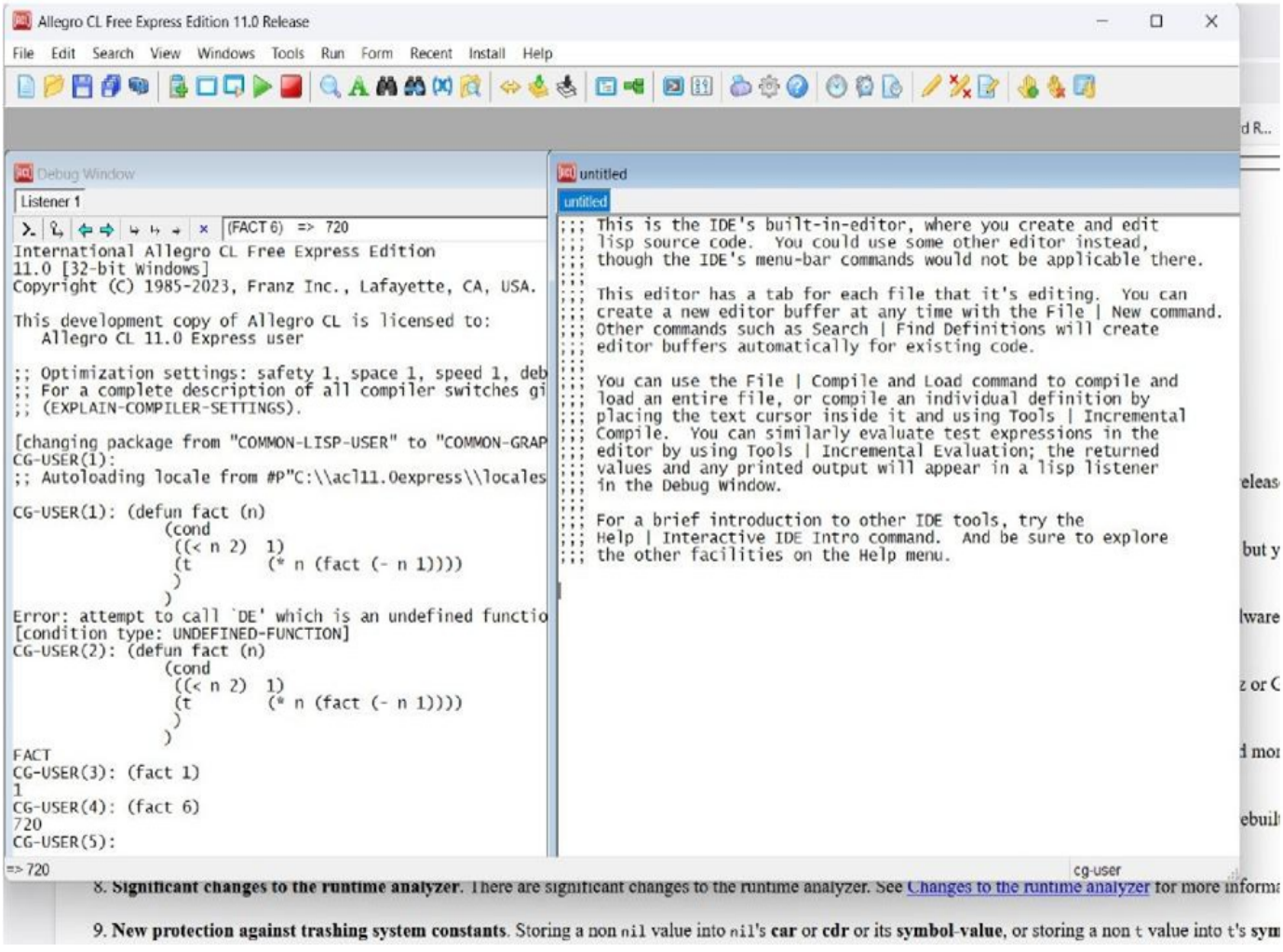
Here's a quick look at some of the early LISPs that the Computer History Museum's "Software Preservation" *LISP History* page details—you'll find UO-LISP somewhere in the middle of the "Standard LISP" section.

- LISP 1.5 family
  - LISP I and LISP 1.5 for IBM 704 709 7090
  - LISP 1.5 for CTSS
  - LISP 1.5 at Stanford
  - SHARE LISP 1.5
- Other LISP 1.5 implementations
  - LISP 1.5 for Univac M-460 (Air Force Cambridge Research Laboratories)
  - LISP 1.5 for AN/FSQ-32N (System Development Corporation)
  - LISP 1.5 for GE 235 (Dartmouth)
  - LISP 1.5 for IBM M44 (Air Force Cambridge Research Laboratories)
  - LISP 1.5 for IBM 7030 (The MITRE Corporation)
  - LISP 1.5 for IBM 360 at SDC
  - LISP 1.5 for Raytheon 704 at SDC
  - MBLISP (Martin Baltimore)
  - YULISP and Hawkinson-Yates LISP (Yale University University of Florida and CENAC Mexico)
  - LISP 1.5 for CDC 6000 series (University of Texas Austin)
  - LISP 3600 for CDC 3600 (Norwegian Defence Research Establishment)
  - LISP/360 for IBM System/360 (University of Waterloo)
  - Stanford Lisp/360
  - NPS LISP 1.5 for IBM System/360 (Naval Postgraduate School)
  - RPI LISP 1.5 for IBM System/360 (Rensselaer Polytechnic Institute)
  - LISP 1.5 for Univac 1108 (University of Wisconsin - Madison)
  - Maryland LISP 1.5 for Univac 1100 (University of Maryland)
  - ULISP for PDP-11 (University of Maryland)
  - ALISP for CDC 6000 (Cyber 72) series (University of Massachusetts Amherst)
  - LISP for Interdata 7116
- LISP 2 family
  - LISP 2
  - GTL for Burroughs B 5500 at Georgia Institute of Technology
  - CRISP for IBM 370 at SDC
- INTERLISP family
  - Basic PDP-1 LISP
  - SDS 930 LISP
  - BBN LISP
  - INTERLISP
    - INTERLISP/FORTRAN: FI F2 F3 and F4
    - INTERLISP/360-370
    - Interlisp-11 (PDP-11)
    - INTERLISP-VAX
  - Interlisp-D
- MACLISP family
  - LISP 1.5/1.6/MACLISP for PDP-6/10
  - Multics Lisp (Multics MacLisp)
  - MDL (MUDDLE)
  - Lisp Machine Lisp
  - NIL (New Implementation of Lisp) S-1 Lisp
  - Franz Lisp
  - Spice Lisp
  - TLC LISP for Intel 8080 and 8086
  - ZIL for IBM MVS/XA at Charles Stark Draper laboratory Stanford LISP 1.6 family
  - Stanford LISP 1.6
  - UCI LISP (University of California at Irvine)
  - Rutgers LISP 1.6
  - Rutgers ELISP
  - Yale TLISP
  - C-MU LISP
- IBM 360/370 LISP family
- Standard LISP family
  - Standard LISP
  - UOLISP for the Zilog Z80 microprocessor
  - PASLSP an implementation of Standard LISP in PASCAL
  - REDUCE 2 and Standard LISP for the Burroughs B6700
  - Portable Standard Lisp (PSL)
  - Cambridge Lisp

Whew! It was only *after* all these different LISPs were created that attempts were made to standardize the language, first with “Scheme” and then with “Common LISP”.

I hunted up and down, back and forth through this list, trying to find enough information to make UO -LISP work for me, but to no avail. I finally realized that the language had worked well enough as a CP/M version, and would be improved for the IBM PC, but the TRS-80 subset was just an afterthought, apparently one not worth repairing or maintaining. Phooie!!

Have I given up on LISP? Not at all. Books like *Practical Common LISP* and free implementations make it possible for me to learn the language, at least on my Windows laptop.



But what about the TRS-80? I’m tempted, like Icarus, to fly close to the sun and try to make an assembly-language version of the BASIC LISP I reviewed above. Will it ever happen? Watch this space!


TANDY LAND
The sweet side of serious computing.
Ian Mauric
MOLASSES SWAMP
LOLLYPOP WOODS
93 MILES
70 MILES
GINGERBREAD PLUM TREE
PEPPERMINT STICK FOREST
START

NEWS FROM

Liberté • Égalité • Fraternité

PASCAL HOLDRY

NEW_JFD-CP

Category: FD,HD TRS-80 CoCo Cards

a card based on the JFD-CP operating principle.

Aim:

- To be able to recover existing software
- Keeping a record of our digital heritage

Provide a diagram for troubleshooting and understanding how the card works.

PERFORMANCE

1. This was the second, and most sophisticated, controller designed by J&M for the CoCo. Externally, it differed from the original model by the presence of a switch for selecting the ROM to use, and a header connector that served as an integrated parallel port. The hard drive to be connected to this controller was initially a 5MB Seagate, housed in a self-contained drive/power supply enclosure with an integrated interface that connected directly to the Centronics-compatible printer port. The software compatible with the hard drive was exclusively OS-9.
2. This card has two PROMs (J & M or TANDY, etc.) to work with different DOS versions. The CP extension indicates that it has a printer output. This output uses special drivers.
3. The 28-pin EPROM sockets accept 64K PROMs and/or EPROMs (the new JDOS 1.2)

Day 1:

Shunted: 6 ms floppy drive head speed
Open: 30 ms from the floppy drive heads

Day 2:

Bypassed: Serial printers
Open: parallel printer

6. **J3:** Pins 1 and 2 closed (I couldn't find any information on J3)

The default jumper settings apply only to JDOS

The parallel port of the JFD-CP card

FUNCTION OF THE SPINS:

No. 1 Writing Strobe Controller
No. 3 D0 Bidirectional
No. 5 D1
No. 7 D2
No. 9 D3 "
No. 11 D4
No. 13 D5
No. 15 D6
No. 17 D7
No. 19 Reading Strobe Controller
No. 21 External device busy
No. 23 N/C
No. 25 External Device Status
No. 26 Reset Controller

All even-numbered pins No. 2 to No. 24 are grounded.
This connector allows most Centronics compatible printers to work with the JFD-CP using an unmodified standard Radio Shack model 100 printer cable.
Pinout taken from page B-3 of the JDOS Disk Basic Reference Manual, J&M Systems

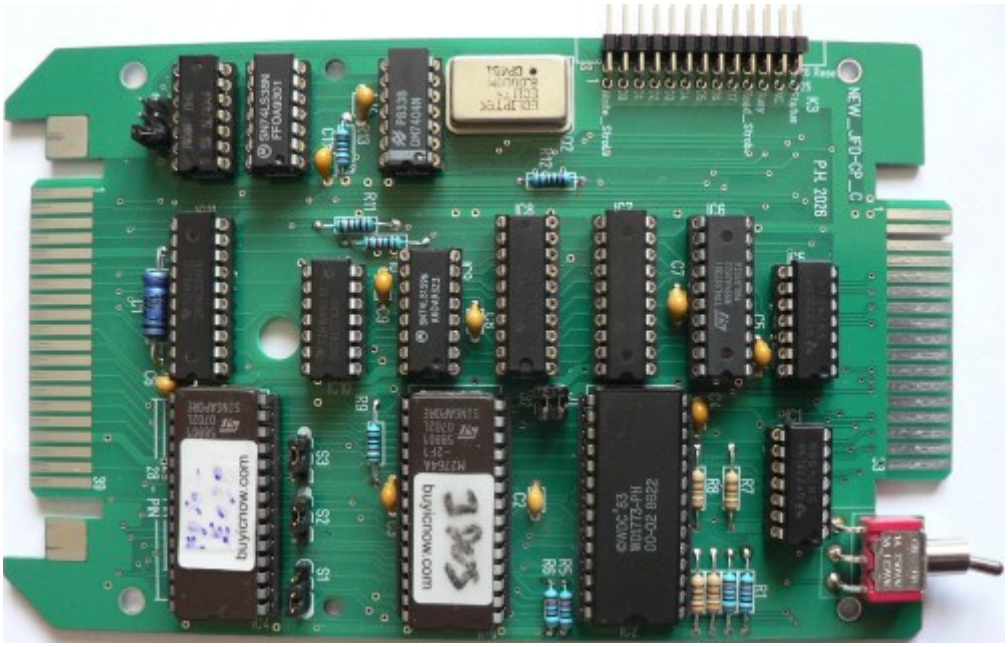
IMPROVEMENTS

1. It has 2 x 28-pin sockets for EPROM and/or PROM.
2. One of the two 28-pin sockets can work with 24-pin and/or 28-pin EPROMs and/or PROMs.
3. The quartz crystal has been replaced by an oscillator block.
4. 3 three-pin connectors (J3, J4, J5) to select the IC4 socket in 24 and/or 28 pins
5. The card is in TADY format

The bindings are compatible with TANDY, DISTO and J&M

This option is only available on the first socket (IC4).

On the side of IC4 are 3 three-pin connectors.
To install a 28-pin PROM, or a 28-pin EPROM in the IC4 socket, insert three jumpers as follows:
The three riders must be inserted on the side with the white screen printing marks.
To install a 24-pin PROM or 24-pin EPROM in IC4 socket, insert three jumpers as follows:
The three riders must be inserted opposite the white screen printing marks.
Insert the PROM and/or EPROM as indicated by the silkscreen on the board.



TOP VIEW

https://prof-80.fr/index.php/new-jfd-cp

POWER SUPPLIES

FOR THE TRS-80 MODEL 3 / MODEL 4/4D/4P



* Uses modern day MEAN WELL power supply units
* 50w for Model 3 version and 65w for the Model 4 version
* The Model 4 version uses a SEPARATE +12V POWER SUPPLY (prevents video "spasm")
* No more RIFA CAP explosions
* No more COLD solder joints
* Units are DIRECT REPLACEMENTS for the original power supplies in all models
* Extra +5V output connectors (1 on Model 3 and 2 on Model 4) for Gotek/freHD boards
* For more information and pricing contact

newSOFT

https://jaynewirth.wixsite.com/newsoft (website)
jaynewirth@verizon.net (email)



80 - NET

ELECTRONIC NETWORK

** FOUNDED NOVEMBER 1980 **

Chairman:- **LEON HELLER**
Secretary:- **BRIAN PAIN**

ADMINISTRATIVE OFFICES:-
40A HIGH STREET,
STONY STRATFORD,
MILTON KEYNES.

Tel: Milton Keynes 0908 - 566660

*** THE NOTICEBOARD FOR ALL MICRO COMPUTER ENTHUSIASTS ***

Special section: Low-cost ideas for your computer

Lowercase and more

Lowercase your Model I and more for $4.98

Model I only

Kenneth Goodwill, Philadelphia, PA

Did you ever have the feeling that your TRS-80 was hiding something from you? If you did, you're right. The Model I has the built-in power to display lowercase letters and thirty-eight other symbols. With a little effort and $4.58 in parts, you can see and use these hidden characters.

I must give credit for the initial idea to Dennis Kitsz and his outstanding book, *The Custom TRS-80*. I built the lowercase modification from his book, but it didn't work. Most likely, this was due to the dead gate on Z25 that he warns you about. Although it didn't work, it did let me see the "special control characters" briefly mentioned by Mr. Kitsz. I decided that I had to have those special characters, so I redesigned the circuit.

Theory

The Model I was built with one bit of video memory missing. This modification fills in that missing bit but lets you determine, via a switch, whether or not the computer will see the restored bit.

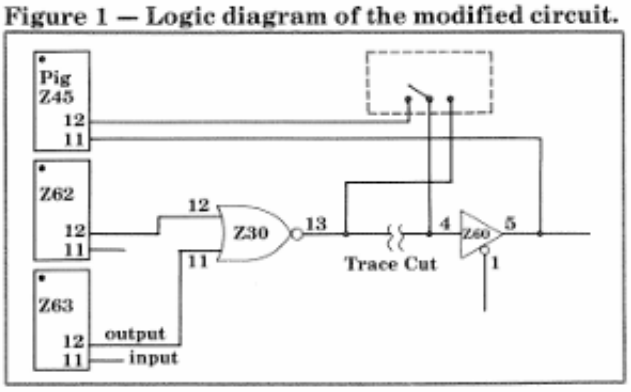
Originally, the missing bit was formed by NORing together the bits adjacent to the absent one. With the added SPDT (single pole, double throw) switch in one position, the missing bit is still formed by combination. In this mode, the added bit can be written to, but not read. When the switch is in the other position, the computer can read the bit you have added. The added bit is now independent of the bits adjacent to it. By using this setup (see Figure 1), the modification can be made invisible to the computer, thus eliminating possible problems with program incompatibility.

Construction

This project requires only two parts in addition to some wire. Any thin, single-strand wire will do. I used 24-gauge. The most important is the 1024×1 bit static RAM (generic code 2102). This can be bought at Radio Shack (cat. no. 276-2501) for $2.79 or less. You will also need a SPDT switch. I used one taken out of an old calculator. You can use an old one of your own, or buy one from Radio Shack (cat. no. 275-613) for $1.79. I tried this Radio Shack switch in my computer and it works fine.

The next step is to open up your computer (an action that voids the warranty). Taking it apart is easy. Just be careful. Lay your computer, keyboard down, on a clean, spacious work area. Remove the black screws, noting

Figure 1 — Logic diagram of the modified circuit.



the position of each. Turn it over and take off the top piece.

Locate the wide cable at the lower left of the keyboard. Be *very careful* with this cable. It tends to fail when stretched or bent sharply. Carefully tilt the keyboard upright and remove the grommets beneath it. Being careful not to bend either board, lift the entire assembly out of the bottom plastic piece. Fold it open so that the keys of the keyboard and the components of the main board are on top. Rotate the assembly to make the keyboard closest to you and put it down on a flat surface.

Piggyback

Find Z45 on the main component board in the corner nearest the power LED. The "Z45" refers to the chip directly to the right. This is where you will piggyback the static RAM.

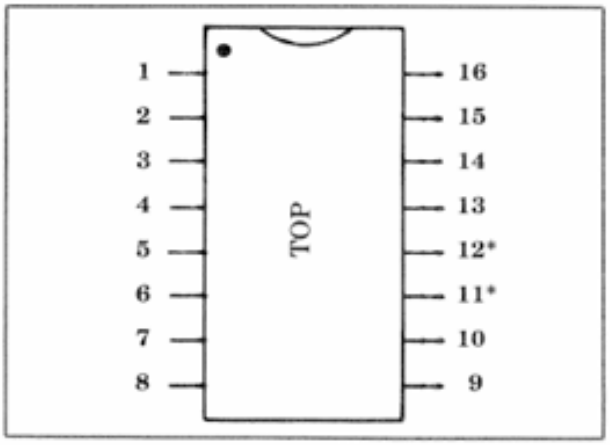
Take your static RAM and locate pins 11 and 12 (see Figure 2). *Carefully* bend these pins at their widest point so that they stick straight out to the side. Place your chip on top of Z45 with pin 1 closest to where "Z45" is printed. Press down enough on your chip to make the two chips touch. Except for pins 11 and 12, the pins on your chip should lightly touch the corresponding pins on Z45. If not, bend the pins on your chip inward slightly until they do.

Warm up your soldering iron of no more than 30 watts. Using thin, rosin-core solder, you are going to connect the matching pins (except 11 and 12) of Z45 and your piggyback chip. When soldering this, you should try to flow solder from the pins on your added chip to the solder at the base of the pins on Z45. Be careful not to overheat the chip with your soldering iron. Use sufficient solder,

but watch for solder bridges between the pins. If any solder bridges develop, use a solder wick to suck up the excess solder.

Find Z27 on the circuit board, just above Z45. This is the best place to make the trace cut shown in Figure 1. Locate pin 13 on Z27. The trace to be cut runs diagonally down and to the right from this pin (Photo 1). Be *sure* that you have the right trace before you start cutting. Using an X-acto knife or single-edged razor blade, cut the trace. Scrape deep enough to be certain that the connection is broken.

Figure 2 — Pin numbering for a 16-pin chip.



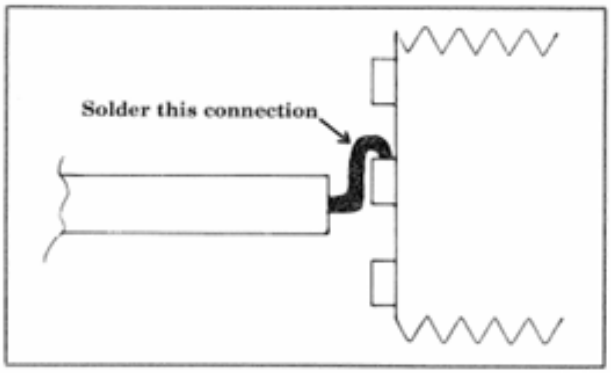


Figure 3 — The finished product of a hook-solder.

Wiring

Cut three pieces of one-conductor wire to 1.5 or 2 feet. Strip one-quarter inch of insulation from both ends of each piece. Using a pair of needle-nose pliers, bend one end of each wire into the shape of a hook. You are going to use this hook to attach the wire to an IC pin. Slip the hook around the pin you want and flow solder from the curved part of the wire to the pin and the solder at the base of the pin. Be sure the solder does not bridge to other pins. Then, bend the wire so that it's perpendicular to the chip (Figure 3).

You are going to hook-solder two of your wires. Connect one to pin 13 of Z30. (Note that this is a 14-pin

THE BI-TECH MICRO MAGNUM

"RELIABILITY" "REMOVABLE AND FIXED HARD DISK SYSTEMS" "DEPENDABILITY"
"MODEL III AND IBM OWNERS" "SUPER DUTY SUB STYSTEM" "SOON MODEL 2,12,16"

SYSTEM TYPE		5 MEG	10 MEG	15 MEG
FIXED SYSTEM	Supplied in a metal case (instrument white) with power supply.	$2150.00	$2260.00	$2370.00
COMBO SYSTEM	Supplied with a 5 Meg MICRO-MAGNUM removable cartridge hard drive and your size choice of fixed media, all in one case with power supply. System gives fast backup capability.	3550.00	3660.00	3770.00
REMOVABLE SYSTEM	5 Meg version only (formatted), front load cartridge, low cost cartridge, system will purge itself, very reliable, supplied in an instrument white case with power supply.	2850.00		

THESE HARD DRIVES ARE BUILT WITH EXTREME DEPENDABILITY YOU WON'T FIND A LINEAR POWER SUPPLY IN THESE UNITS.

All drives are supplied with: DOSPLUS 3.5, ONE YEAR WARRANTY, EASY INSTALLATION — JUST PLUG INTO YOUR ONE DRIVE TRS-80 MODEL III 50 PIN BUS

PRINTER SALE

C-ITOH DAISY F-10 40 CPS $1299.00
TRACTOR ASSEMBLY F-10 . . . . . 249.00
C-ITOH DAISY F-10 55CPS $1599.00
PROWRITER 10" PAR 120 CPS 435.00
FOR THE BUDGET POCKET HERE IS A BUDGET HARD DRIVE . . 5 MEG COMPLETE . . . 1599.00

CERTIFIED CHECK, BANK CHECK OR MONEY ORDER MUST ACCOMPANY ALL ORDERS
Please add $15.00 for freight, ins. per parcel

DOSPLUS TRADEMARK MICRO-SYSTEM SOFTWARE, INC.
IBM P.C. TRADEMARK IBM CORPORATION
TRS-80 TRADEMARK TANDY CORPORATION

Authorized Bi-Tech Dealer

J. CASTORO SOFTWARE
3 WYMAN COURT
CORAM, N.Y. 11727
(516) 473-5349

chip.) Solder the other wire to pin 4 of Z60. This chip is under the ROM connect ribbon of Level II machines. Bend the stripped part of your last wire into a loop and solder it to the outstretched pin 12 on the piggyback chip.

Cut a two-inch piece of wire and strip both ends of it. Hook-solder one end to pin 5 on Z60. Loop the other end and solder it to pin 11 on the piggyback chip.

The Switch

Solder the wire connected to pin 4 of Z60 to the middle terminal of your SPDT switch. Take the two remaining long wires and solder them to either of the side terminals. The operation is now complete!

Now, for a place to put the switch. Although it can be anywhere, a convenient place is next to the break key, in line with the power LED hole. For the Radio Shack switch, simply drill a one-quarter-inch hole and insert the switch. Use the provided nuts to hold it in place.

Testing

Carefully put the computer back together, remembering to be gentle with the keyboard cable. When you turn your computer on, you should see either "MEMORY SIZE?" or a bunch of symbols that you have never seen on your computer before. If you see the symbols, flip the switch to get letters instead. If flipping the switch does not get you "MEMORY SIZE?", reopen your computer. Check the wiring and look for solder bridges. If the lowercase seems to work but the keyboard is giving you problems, replace the cable connecting your computer to the keyboard.

When you flip the switch back and forth, you will probably see reverse apostrophes flash on the screen. This is normal.

Driver 1

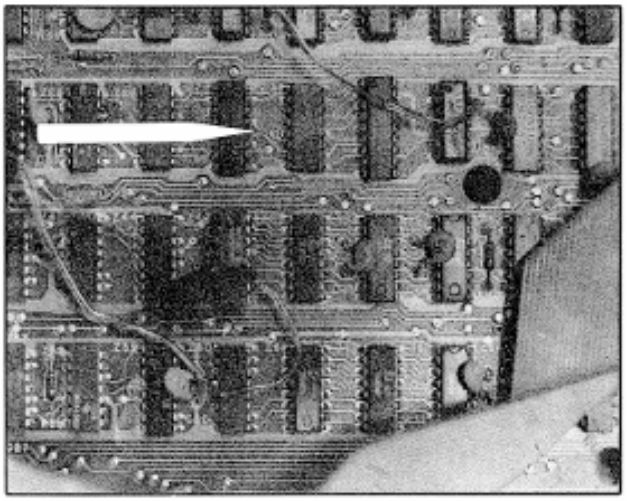
Although you may access lowercase by using the POKEs of the testing program, it's a lot easier to use a special driver program. Listing 1 shows a program that can be assembled for use in computers without disks. As written, it will load into low memory and leave you with your normal amount of space for BASIC programs. For use in disk systems, change line 140 to make the driver routines reside in high memory.

The "Driver Plus" provides many features. It lets you toggle between shift for lowercase and shift for uppercase. This switch can be made by pressing shift, downarrow, and C at the same time. It can also be set by your programs. POKE 16408 with zero for shift for lowercase and with 255 for shift for uppercase.

A second feature this program has is access to the special symbols without using the POKE command. The CHR$ values that are usually used for space compression codes have been altered. The codes from 192 to 223 will give you the special symbols. Run line 20, shown in photo 1, to test this feature. You should get the four display lines labeled 3. If you would rather have the space compression codes, delete line 270 and remove the carry condition in line 280 of Listing 1.

Two new symbols can be accessed from the keyboard. Shift @ will now be displayed as a reverse apostrophe. Hitting the uparrow in the shift for uppercase mode will give you a left brace.

Photo 1—All wiring and the correct trace cut.



Driver 2

Listing 2 shows a BASIC driver for non-disk systems. The routine needs to be executed once by your programs for them to use lowercase. This program will give you shift for lowercase only and no modification of the CHR$ command.

Final Comments

In some computers, the lowercase "a" is raised up. If this is true for you, you'll just have to learn to live with it.

I thought I would have trouble using lowercase in purchased machine language software. In some cases, the driver program is needed, but others (such as Big Five Software programs) will give you lowercase without the driver!

I think you will find this a simple, yet extremely useful, modification. I am sure you will get your money's worth from it.

Listing 1 — Lowercase

```
                00070 FLOWER CASE DRIVER PLUS
                00080 PURITTEN BY NED COMPUTING FOR PUBLIC USE
4016            00100           ORG     4016H           ;KEYBOARD DRIVER ADDRESS
4016 4240       00110           DEFW    KEYSRD          ;CHANGE KEYBOARD DRIVER
4018            00120           DEFS    6               ;SKIP TO VIDEO DRIVER
401E 3E40       00130           DEFW    VID             ;CHANGE VIDEO DRIVER
403E            00140           ORG     4028H           ;THIS CAN BE MOVED UP
403E D96E03     00150 VID       LD      L,(IX+3)        ;HL = LOCATION ON SCREEN
4041 D96604     00160           LD      H,(IX+4)
4044 DA9A04     00170           JP      C,49AH          ;JUMP IF CARRY
4047 DD7E05     00180           LD      A,(IX+5)        ;CHECK CURSOR ON/OFF
404A B7         00190           OR      A               ;SET FLAGS
404B 2801       00200           JR      Z,CUROFF        ;JUMP IF CURSOR IS OFF
404D 77         00210           LD      (HL),A          ;MOVE CHAR. UNDER CURSOR
404E 79         00220 CUROFF    LD      A,C             ;GET NEXT CHARACTER
404F FE20       00230           CP      20H             ;COMPARE TO SPACE
4051 DA0605     00240           JP      C,506H          ;JUMP IF CONTROL CHAR.
4054 FE80       00250           CP      80H             ;COMPARE TO FIRST GRAPHIC
4056 3807       00260           JR      C,IONE          ;JUMP IF ALPHA/NUMERIC
4058 FEC0       00270           CP      192             ;COMPARE TO SPACE COMPR.
405A DA9A04     00280           JP      C,49AH          ;JUMP IF GRAPHICS
405D D6C0       00290           SUB     192             ;SUBTRACT TO GET SYMBOLS
405F C37104     00300 DONE      JP      471H            ;JUMP TO DIRECT DISPLAY
4062 CD6303     00310 KEYSRD    CALL    363H            ;CALL KEYBOARD SCAN
4065 FE03       00320           CP      3               ;COMPARE TO CONTROL C
4067 211840     00330           LD      HL,4018H        ;HL = TOGGLE LOCATION
406A 280E       00340           JR      Z,CONTC         ;JUMP IF CONTROL C
406C 47         00350           LD      B,A             ;SAVE KEYED IN CHARACTER
406D 7E         00360           LD      A,(HL)          ;CHECK SHIFT TOGGLE
406E B7         00370           OR      A               ;SET FLAGS
406F 78         00380           LD      A,B             ;RESTORE KEYED IN CHAR.
4070 C8         00390           RET     Z               ;RETURN IF SHIFT-NORMAL
4071 FE41       00400           CP      65              ;COMPARE TO "A"
4073 D8         00410           RET     C               ;RETURN IF LESS THAN "A"
4074 FE5A       00420           CP      90              ;COMPARE TO SHIFT Z
4076 D8         00430           RET     Z               ;RETURN IF IT IS
4077 EE20       00440           XOR     20H             ;TOGGLE UPPER/LOWER CASE
4079 C9         00450           RET                     ;RETURN TO KBH
407A 3EFF       00460 CONTC     LD      A,255           ;LOAD A WITH TOGGLE VALUE
```

```
407C AE      00470     XOR     (HL)          ;TOGGLE WITH 4018H
407D 77      00480     LD      (HL)+A        ;PUT RESULT IN 4018H
407E AF      00490     XOR     A             ;CLEAR A
407F C9      00500     RET                   ;RETURN TO ROM
6400          00510     END     407CH         ;MAGIC ENTRY
```

Listing 2 — Lowercase

```
5 REM BASIC LOWERCASE DRIVER
6 REM WRITTEN BY KEG SOFTWARE FOR PUBLIC
```

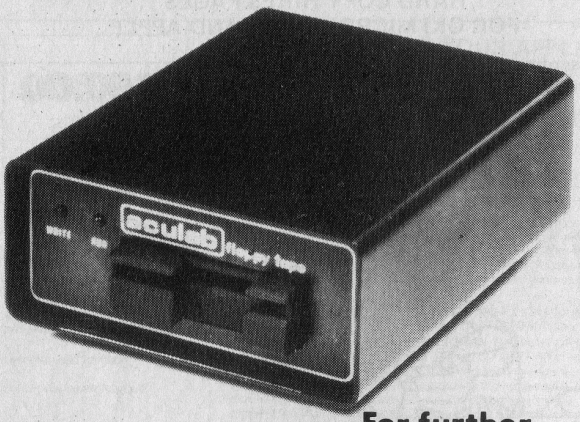
USE

```
10 POKE 16414,62:POKE 16415,64
20 FORQ=16446TO16481:READW:POKEQ,W:NEXTQ
30 DATA 221,110,3,221,102,4,218,154,4,22
1,126,5,183,40,1,119,121,254,32,218,6,5,
254,128,56,7,254,192,218,166,4,214,192,1
95,125,4
```


aculab floppy tape,

The tape that behaves like a disc, For TRS-80 LEVEL II.

Connects directly to **TRS**-80 Level 2 Keyboard. Operating and file handling software in **ROM**. 8 commands add 12 powerful functions to Level 2 **BASIC.** No buttons, switches or volume controls. Full control of all functions from Keyboard or program. Daisy chain multiple drives. Certified digital tape in endless loop cartridges. Reads and writes in **FM** format at 9000 Baud. Soft sectored with parity and checksum error detection for highly reliable operation—just like discs. Maintains directory with up to 32 files on each tape, tapes may be write-protected. Supports Basic and machine-language program files, memory image and random access data files. 12 character filespecs —: **"FILENAME/EXT:d"** (d is drive no. 0—7). Automatic keyboard debounce. Full manual with programming examples and useful file-handling routines.

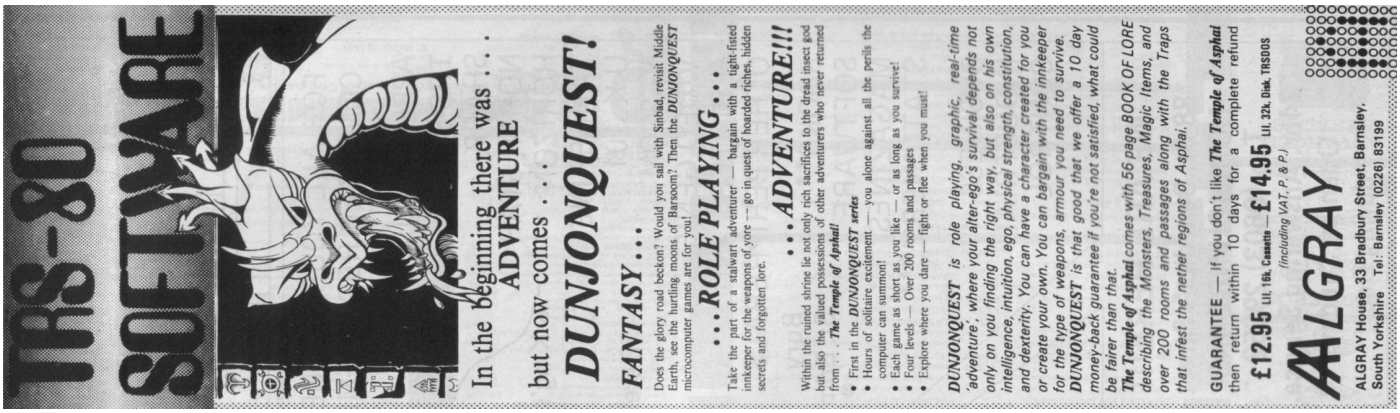


COMMANDS (usually followed with a filespec and possible parameter list).
@SAVE, @LOAD, @RUN —for **BASIC** programs, machine language programs and memory image files. **@GET, @PUT** —moves a 256-byte record between a random access file and **BASIC**'s data buffer. **@KILL** —removes a file from the directory and releases tape sectors for immediate re-use. **@LIST** —displays file directory along with sector allocation and free sectors. **@NEW** —formats tape and creates a blank directory.

Master drive with PSU, Manual and a selection of tapes.
£167-00 +£2-00 pp+vat.
Slave drives with PSU £122-00 +£2-00 pp+vat.
(Export orders pp charged at cost)

For further information, Telephone 0525 371393

aculab Ltd.
24 Heath Road, Leighton Buzzard, Beds. LU7 8AB



CGP-115 LISTS IN

TANDY'S NEW four-pen Colour Graphic Printer could have the same dramatic effect on printing as the change from black and white to colour screen displays has had on computing. It is a direct descendent of the CE-150 2.25in. colour printer produced by Sharp for the PC-1500.

The printer, complete with RS-232 and Centronics interfaces, costs £149 including VAT but not including any connecting cables. It comes with a power supply, operation manual, three black pens, one each of blue, green and red, and a roll of plain paper 4.5in. wide and 180ft. long. The machine is a healthy 8.5in. square by 3in. high and weighs 1.75lb. It is coloured silver-grey, with a black top cover which is raised to gain access to the pens and their holders.

The printer can print either 80 or 40 characters per line. The ASCII character set from code 32 to code 127 is implemented, although unfortunately there is no pound sign. A switch marked special characters provides for Japanese script.

The characters are not produced by a dot matrix or burnt into aluminised paper. In either case, present technology would not permit 80 characters within a space of a little over 3.75in. The characters are formed in the machine's internal ROM and written on to the paper using ink pens. The effect is similar to high-quality type-written text. Except for the lack of proportional spacing of characters, and paper width, the performance is as good as many professional printers.

The printer has two modes of operation, text and graphics. The Graphics mode permits character size to be varied from 80 characters per line to one character per line, in 63 steps retained on return to text mode. Characters can also be rotated in 90° steps. Also available is the facility to draw 15 different types of dashed line. Colour can be changed and a return to text initiated with very simple commands.

Axes are specified in terms of X or Y with a defined step and interval. The step is a multiple of 0.008in. between 1 and 999, positive and negative, and the interval, the number of repetitions of the step, is between 1 and 255.

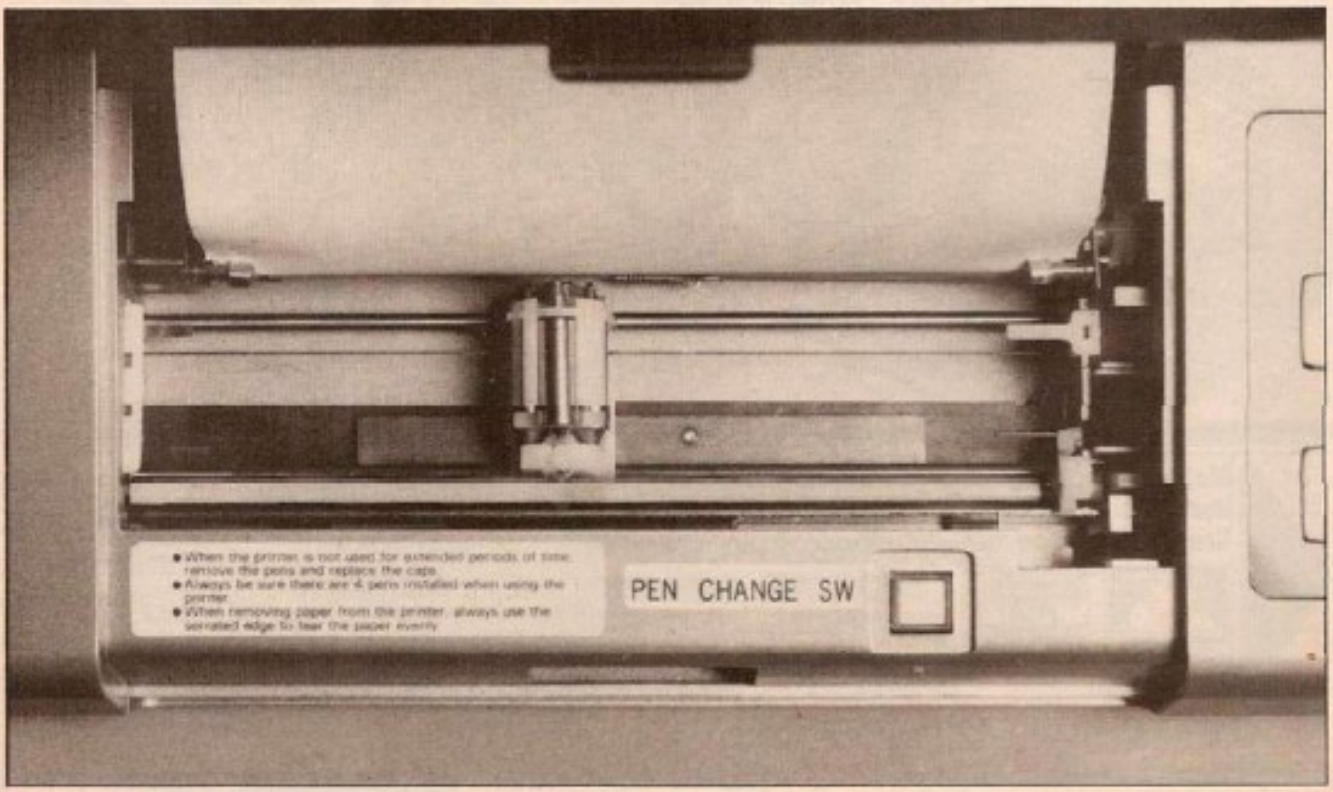
The effective X-axis resolution of 480 steps and equivalent vertical resolution draws convincing circles, without annoying steps on the edge.

The printing speed is a slow 12 characters per second. At present screen dumps of graphics are not possible.

The machine is opened by undoing five screws in the base and releasing four locking tabs to release the top moulding. The base holds the print and feed mechanism at the front. The paper drive is friction-fed by a full-length rubber compound roller and side pressure rollers which also prick the edge of the paper at 0.156in. steps. This gives the effect of tractor drive and has the advantage of providing the Y axis with a quoted accuracy of one percent coupled, with a repetition accuracy of 0.008in.

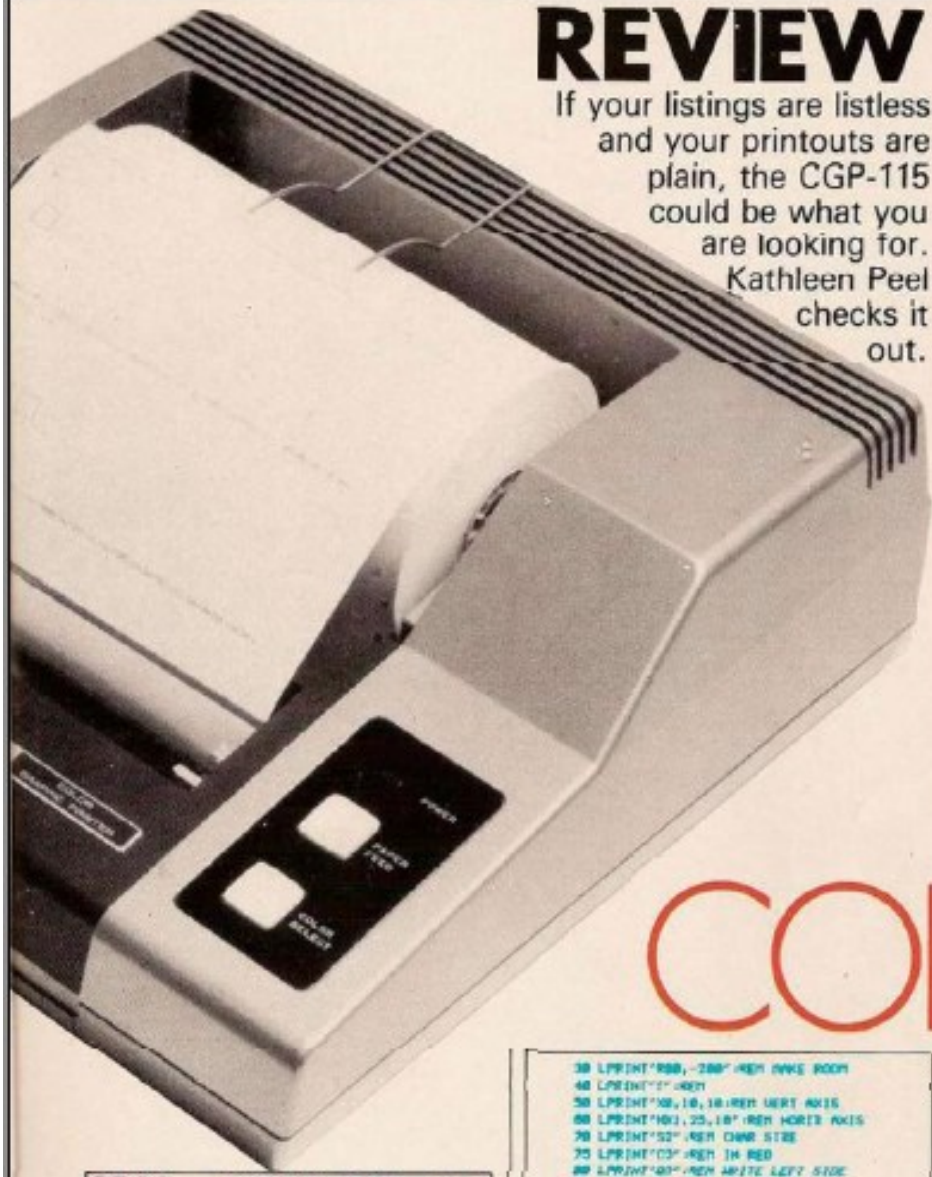
The PCB is screened from interference and the major integrated circuit components are encased in a screened box shaped around the parallel input/output socket — very professional. There is a large heat sink down the right-hand side at the rear.

The top moulding of the unit houses the power switch, serial input/output socket, and power socket at the rear. A small PCB houses the power indicator lamp, the paper-feed and colour-select push buttons. Finally, housed under the mechanism cover is the manual pen-change switch. The top cover lifts to expose instruction labels, one concerning pen



REVIEW

If your listings are listless and your printouts are plain, the CGP-115 could be what you are looking for. Kathleen Peel checks it out.



replacement and the other noting how to look after your printer.

The 50-page manual is well laid out and clear, but there are some omissions. The appendices provide some Basic programs which produce pie charts, sine and cosine curves.

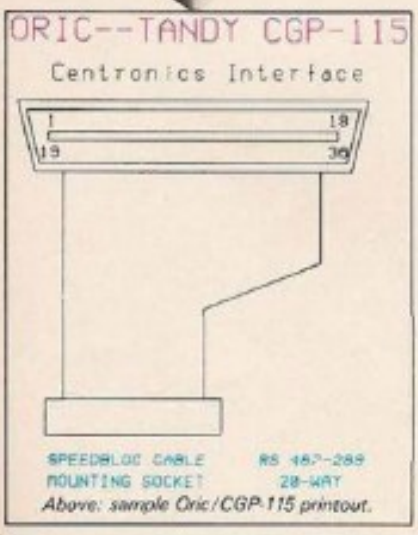
It is surprising that there is no indication of the connections on the four-way DIN socket masquerading as an RS-232 interface. Only pins 2, 3 and 4 are wired. Table 1 gives the connections. The serial interface operates at 600 baud with no parity and two stop bits.

The parallel interface is via a Centronics-type plug but the only connections used are busy — Pin 11, strobe — Pin 1 acknowledge — Pin 10 and the data lines.

Loading the paper was initially a bit of a problem. A protrusion in the centre prevented the paper from being entered into its loading slot completely, and so take up, by pressing the paper-feed button, could not take place. The answer was to cut a strip of paper 6in. long by 1in. wide and load this into the slot close to one edge, such that it could be fed through by hand. The strip was then slid across into a central position and the full paper width loaded over the top of the strip. As soon as take up was established the strip was removed. I only used this procedure twice as, whatever the obstruction was, it disappeared. Loading the pens was straightforward if a little fiddly. Neither operation is likely to be required very often.

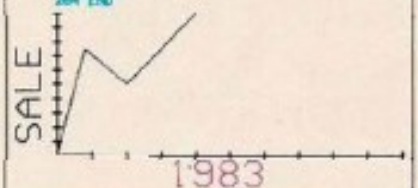
(continued on page 48)

COLOUR



Above: sample Oric/CGP-115 printout.

```
30 LPRINT"R80,-200"REM MAKE ROOM
40 LPRINT"I1"REM
50 LPRINT"V0,10,10"REM VERT AXIS
60 LPRINT"H0,25,10"REM HORIZ AXIS
70 LPRINT"S2"REM CHAR SIZE
75 LPRINT"C3"REM IN RED
80 LPRINT"U0"REM WRITE LEFT SIDE
90 LPRINT"M-10,10"REM POS PEN
92 LPRINT"PSALE"REM WRITE VERT
94 LPRINT"M10,-40"REM POS PEN
96 LPRINT"U0"REM WRITE RIGHT WAY
97 LPRINT"C2"REM IN GREEN
98 LPRINT"P    1983"REM TITLE
100 LPRINT"I1"
170 LPRINT"M0,0"
180 LPRINT"D20,70"
200 LPRINT"D50,50"
210 LPRINT"D100,100"
230 LPRINT"C0"
250 LPRINT"S0"
260 LPRINT10,C0
261 FORX=1TO6:LPRINTL0:NEXTX
262 LLIST
264 END
```



Above: typical 80-character printout.

Pin	Function
1	Not connected
2	Busy
3	Earth
4	Data

Table 1. Four-pin socket terminations.

	Printout	
	Length/£	Sq. ft./£
Sinclair	27.2	9.1
Amber	129.4	24.3
CGP-115	112.8	42.3

Table 3. Paper usage running costs.

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~
```

Auto-test character set.

```
10 REM LINE SAMPLE PROGRAM
20 PRINT,CHR$(18)
30PRINT,"L3"
40PRINT,"J400,0"
50PRINT,"A"
60 END
```

Sample BBC/CGP-115 printout.

(continued from page 45)

When the printer is switched on, the pen carrier traverses to the left-hand edge and rotates the pens to ensure that holder number one is in the top working position. The printer then draws four boxes produced sequentially by the pens in holders 1, 2, 3 and 4. This is to ensure that the user is aware of any dried-up pens.

Pressing the paper-feed button at switch on sets in motion a self-test procedure that also prints the character set in the four loaded pen colours.

CONCLUSIONS

- Tandy's new printer will transform data presentation and allow the computer to interpret results fully, for ease of reading, varying colour, indentation, and print size as necessary.
- The graphics capability provides for considerably better resolution that available to the majority of micros. The lack of a screen-dump facility is likely to be temporary as users will soon develop software to produce screen dumps in colour.
- This high-quality printer can only be faulted on paper width, and slow speed which may make it unsuitable for word processing. ■

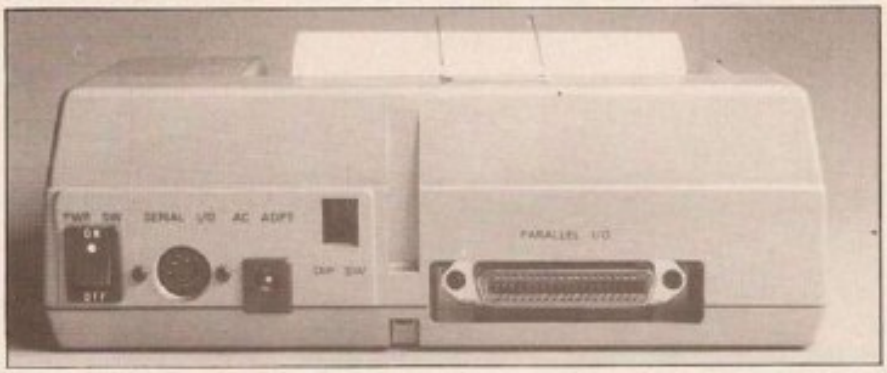


Table 2. Comparison of machines.

	Colours	Characters		Paper			Printer Size in.	Basic Cost £	Accessories		
		line	sec	Type	Width in.	Length ft.			Paper	Rib-bons	life
Sinclair Printer	1	32	50	Metal-ised	4	65	5.5 × 2.9 × 1.9	59.95	11.95 for 5		
Amber 2400	1	24	17	Plain	2.25	88	6.3 × 6.3 × 3.1	89.70	3.40 for 5	2.00 Ribbon	3 × 88ft. rolls
Tandy CGP-115	4	80 to 1in. 63 steps	12	Plain	4.5	150	8.4 × 8.6 × 3	149	3.99 for 3	1.69 for 3 pens	825ft. each pen

NEW SPECTRUM SOFTWARE

TRANSYLVANIAN TOWER
In this spine chilling adventure you explore Count Dracula's dark tower from the dismal dungeons to the terrifying top! Discover his 500 room maze with its incredibly spectacular moving three dimensional graphics. Watch out for the swooping vampire bats as you try to rid the world of this Transylvanian terror! Combines an adventure with a maze with a fast moving graphic action game! A new maze everytime! With a full 'save' routine for use during the daylight hours! 48K SPECTRUM only £6.50

EVEREST ASCENT
Stake your claim to the top of the world in this demanding vertical adventure in which you aim to reach the summit of the world's highest peak. A complex game of strategy and survival.
For 16 or 48K Spectrum (For the 16K Spectrum we have divided Everest Ascent into exciting parts to give you an adventure as complex as the 48K version)
16 & 48K SPECTRUM £6.50

Promotion's the name of the game in 'SHIP OF THE LINE' – An adventurous management game. Fearlessly battle your way up the ranks... encounter enemy fleets... survive mutiny, fever and famine... endure fog, fire and thirst... then... when you think you've done well... rush home to Port for promotion!
16K SPECTRUM £4.95 48K SPECTRUM £6.50

MULTI FUNCTION CASH CONTROLLER
Takes care of your Home Budgeting, Bank Account, Standing Orders, Loan and Mortgage Repayments. Complete security ensured by secret password. A budgeting bargain for only £10. 48K SPECTRUM

SUPER SPY
A complex adventure for the 48K Spectrum incorporating a full 3-D maze. The many sections of this game ensure that you are entertained for hours if not days. Full 'save' facility.
Only £6.50 48K SPECTRUM



Richard Shepherd Software

FREEPOST (No stamp required), Maidenhead, Berks SL6 5BY.

ORDER FORM Please send me

_____	48K Everest Ascent	at	£6.50
_____	16K Everest Ascent	at	£6.50
_____	48K Transylvanian Tower	at	£6.50
_____	16K 'Ship of the Line'	at	£4.95
_____	48K 'Ship of the Line'	at	£6.50
_____	48K Super Spy	at	£6.50
_____	48K Multifunction Cash Controller	at	£10.00

OFFER! Buy 2 get £1 off Buy 3 get £2 off Buy 4 get £3 off

By First Class Post On Quality TDK Cassettes

I enclose my cheque/Postal Order for £ ..... payable to Richard Shepherd

Name .....

Address .....

Machine ..... _____ K Memory



50,25,5,4,3,3

107

IF YOU HARDWARE-HACKERS ARE FINDING THOSE PCB BOARDS TRACKS ARE MERGING INTO ONE OR YOU ASSEMBLER CODERS HAVE LOST A PUSH OR A POP, GIVE YOURSELVES A BREAK.

SIT DOWN WITH A COFFEE, PUT A TIMER ON AND SEE IF YOU CAN CALCULATE, **IN 30 SECONDS**, THE RED NUMBER FROM ANY OR ALL OF THE 6 BLACK NUMBERS, USING EACH NUMBER ONLY ONCE AND USING *,/,+ OR - ONLY.

NO CHEATING, BUT THE ANSWER IS ON PAGE 88

IN MAV'S WORKSHOP

Mav's Keyboard Round-Up for the Model I/III/4



by Ian Mavric

The keyboard is where the human meets the computer. If the keyboard doesn't work, the computer can't understand you. A non-working keyboard can turn a perfectly good computer into useless junk.

The TRS-80 Models I, III and 4 have been blessed with generally good keyboards, especially when you compare them to some other computers sold around the same time. They are nice to type on for hours of programming or word processing, and rugged enough for rigorous game playing.

In this article I will look at the keyboards each TRS-80 came with, and offer suggestions for repair or replacement, to keep your pride-and-joy running and enjoyable.

TRS-80 MODEL I - 2 KEYBOARD TYPES

When the TRS-80 Model I was released the supplier to Radio Shack that provided it's keyboard was Hi-Tek, and it was famous for key-bounce when Level II Basic was introduced.
Radio Shack made a cassette program which fixed the problem, and most DOSes had a keyboard debounce routine incorporated. The keys work by bringing two metal tines together, these tines are soldered to the PCB, and are very difficult to repair of a tine breaks off.

However, this rarely happens. When you get a Hi-Tek keyboard and a key doesn't work, flip it over and check the solder joint for that key with a magnifying glass, it will almost certainly need to be re-soldered with copious amounts of fresh solder.
Any other problems with he Hi-Tek generally cannot be repaired. Suggestion: replace the keyboard with an ALPS or Erz if you cannot fix your Hi-Tek.

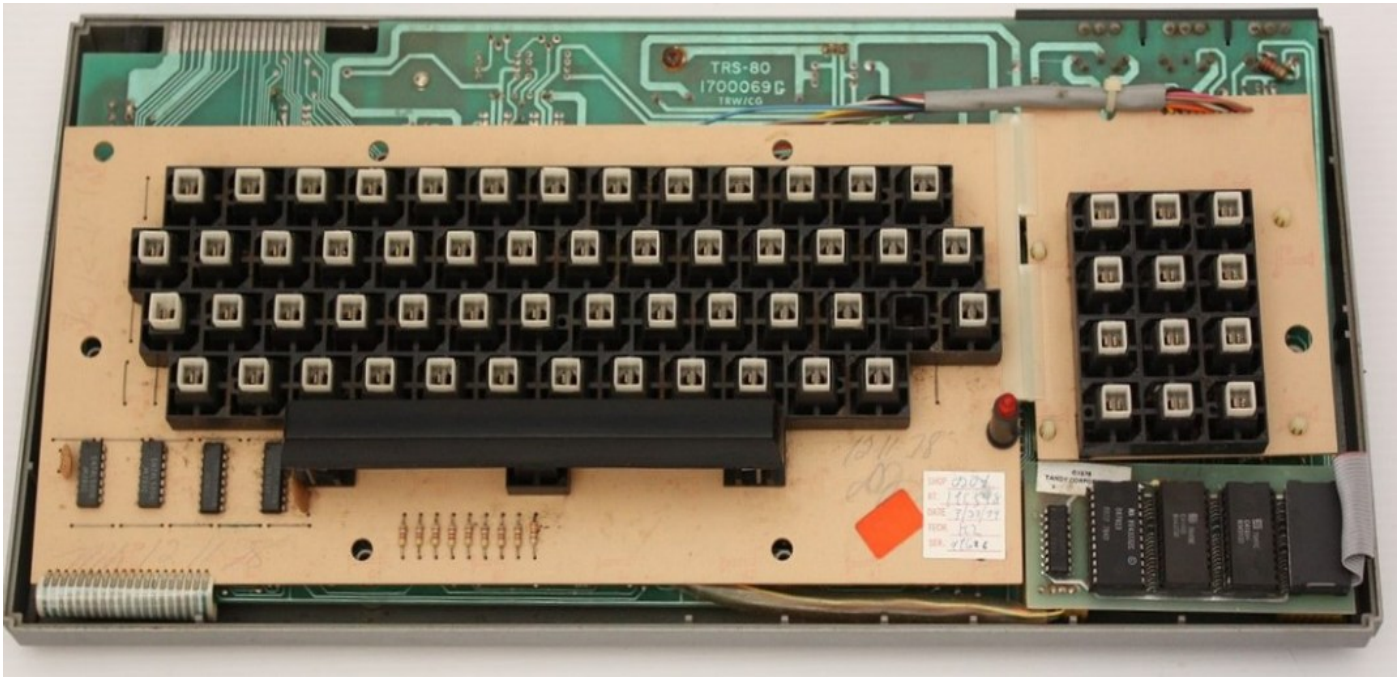


Photo: Hi-Tek keyboard with the key caps removed.

In 1979 Radio Shack changed the supplier of the Model I keyboard to ALPS from Japan. These are very high quality keyboard using the renowned white SKCC keyswitches. These are nicer to type on and don't have the keybounce problem.
Each keyswitch is soldered in individually and are held down by 2 lugs. If a keyswitch fails you can easily unsolder it from the keyboard and solder in a replacement in a couple of minutes.

The SKCC switches can be opened and theoretically cleaned and reset, to make them work again, though I have found this to be rather hit-and-miss. You can easily find used SKCC switches on eBay, or a dead Model III with an early keyboard (see Model III, section, below). If the keyboard is beyond repair I recommend an Erz replacement.



Photo: ALPS keyboard for the Model I uses SKCC keyswitches

An Erz keyboard is designed by Marcel Erz and uses modern MX keyswitches. You can buy them from Russell Smith (he makes them these days) by emailing: russellsmith9950@gmail.com



Photo: Erz replacement Model I keyboard

TRS-80 MODEL III - 2 KEYBOARD TYPES ALSO

The Model III was released in 1980 and again ALPS made the keyboard using the same SKCC keyswitches as the Model I. This continued until about half way into the Model III production (so roughly the end of 1981) and then a new keyboard was supplied by ALPS.

This one looked the same but had completely different keyswitches which were soldered down with 4 lugs each. They used a rubber dome. The dome has a carbon-like pad on it which when pressed makes contact with two metal pads in the base of the keyswitch. When the resistance between the 'carbon' pad and the metal base plates is too high the key no longer works.

These keyswitches can actually be disassembled, the carbon pad cleaned with alcohol and re-assembled. ALPS says to check proper functioning with a multimeter, 80 ohms or less being perfect. In my experience up to 150 ohms resistance still results in a working key. If this does not fix your keys, like the previous keyboard, you can replace the key switches with good used ones from a donor board.

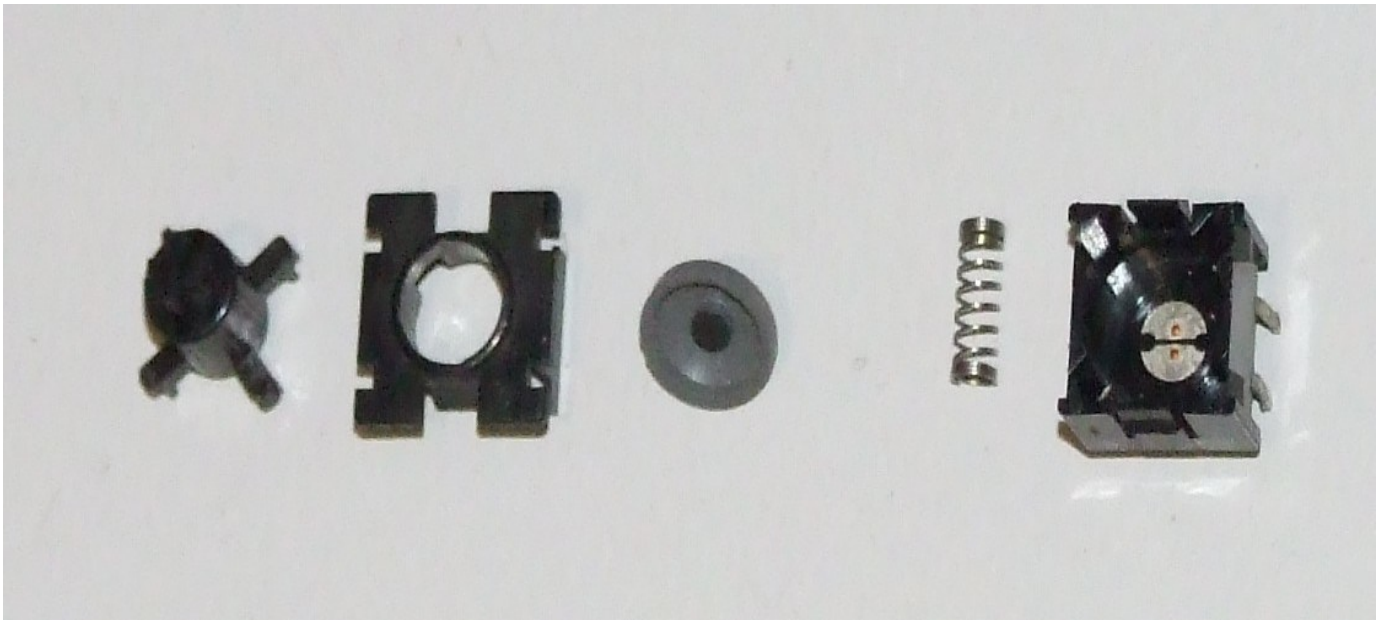


Photo: ALPS rubber dome keyswitch as used in Model III (and 4 and then 4D)

A replacement keyboard can be obtained for the Model III (and 4) from Jay Crutti to manufactures replacement aftermarket keyboards which, again, use the amazing MX keyswitches. To get one you can check Jay's website: https://www.jaycrutti.com/hardware-projects/tandy-trs-80-replacement-keyboards

TRS-80 MODEL 4 - 3 KEYBOARD TYPES

The first few thousand Model 4s came with a keyboard by Maxi-Switch. You don't see them very often because it was quickly replaced with an ALPS keyboard.

The Maxi-Switch uses SMK series 8000 key switches which work on the same principal as the ALPS rubber dome switches, but the parts are not interchangeable with an ALPS. In theory the key switches on the Maxi-Switch can be removed, disassembled, cleaned and put back together however I have never repaired one that way.

I have repaired these keyboards by replacing bad keyswitches with good used ones from a donor keyboard. Unless you buy a lot of Model 4s, or are very lucky, you are unlikely to see a Maxi-Switch keyboard.

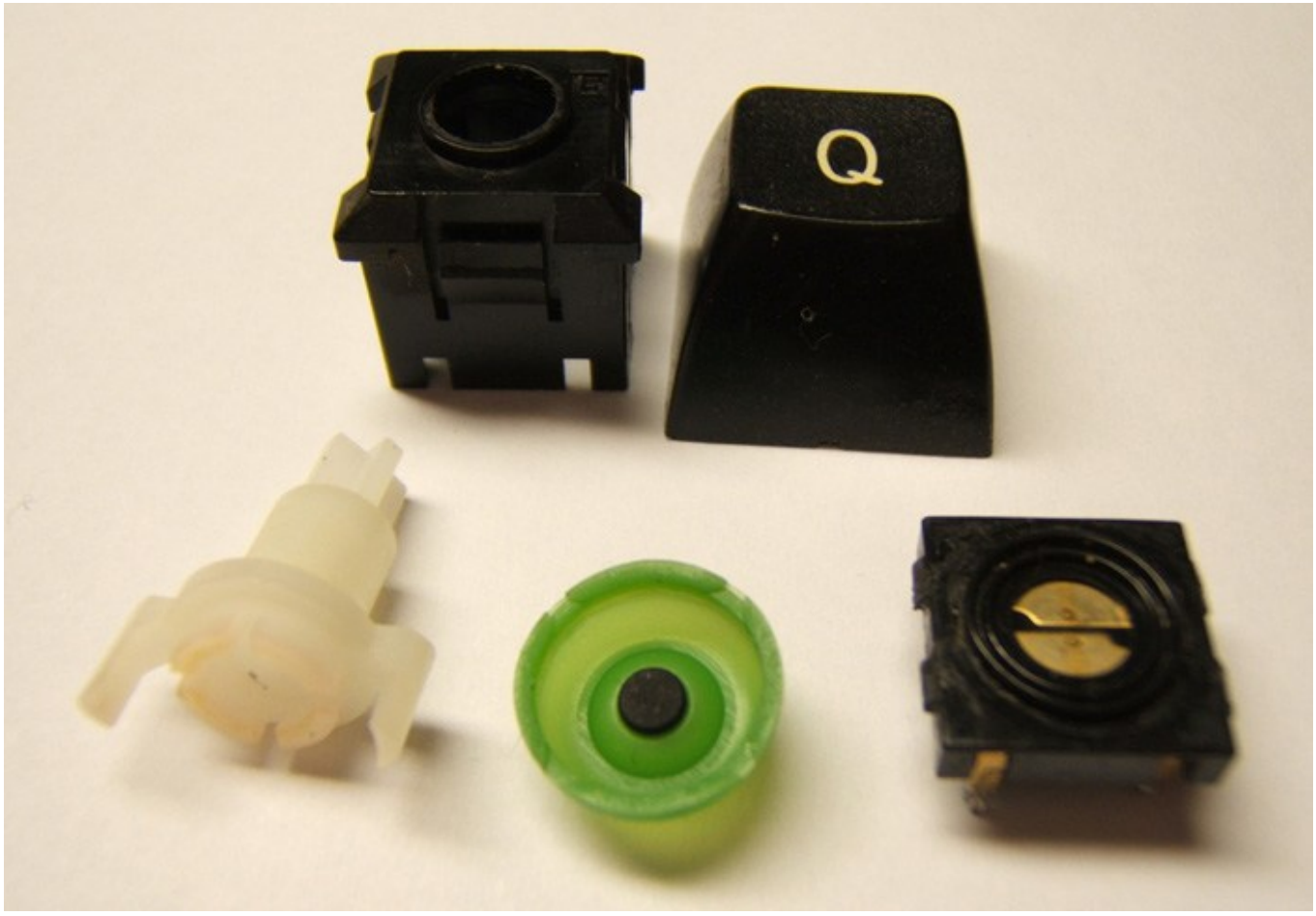


Photo: Maxi-Switch TRS-80 Model 4 disassembled keyswitch (photo: Michael Brutman)

Once the Model 4 production got into full swing Radio Shack changed the keyboard to one very similar to the Model III 4-lug ALPS keyboard. It was also made by ALPS. Interesting factoid: Some of the very last Model IIIs had a Model 4 keyboard with just empty holes where the F1,F2,F3 and Ctrl and Caps keys would have been. They are not wired up to anything, so you can't convert one of these from a Model III to Model 4 keyboard, but you may find one when you open a Model III.

To fix these Model 4 ALPS keyboards refer to the paragraph above on repairing the Model III version.
When the Model 4 was upgraded in 1985 to the Gate Array version, along with the green CRT and new motherboard design, RS also changed the keyboard supplier. This time to an unknown manufacturer (literally so, it isn't marked anywhere on the keyboard) and is of a membrane design, which is cheaper to make than it's predecessors, and feels a little dead to type on. These are sealed keyboards and have no real way to open and service them. They are the very definition of a throw-away item. Over the years people have patiently disassembled the base which is sealed on with melted posts and attempted repairs. I have in fact heard over the years of people who have successfully repaired the membrane keyboard but more often people who tried and failed.
As with all the Model 4 keyboard designs, if it has busted beyond repair just get a Jay Crutti replacement and kiss your Model 4 keyboard woes away.



Photo: Jay Crutti Model 4 replacement keyboard show installed

TANDY TRS-80 MODEL 4D: 2 KEYBOARD TYPES

When the 4D came out in late 1985 (basically 1986) the keyboard was changed and a backspace key added, but besides that no other difference and the keyboard will work on a regular Model 4 and even a Model III.
The first batch of 4Ds came with a membrane keyboard, the same as on the Model 4 Gate-Array version, but with the additional Backspace key added. Like the previous one it doesn't lend itself to repairs by being next to impossible to work on.
If you have one of these you are best getting a replacement keyboard.

The second production run (and subsequent production runs, if there were any, as rumour has it that 1987 was the last year of 4D production, it just took until 1991 for RS to sell them all) of 4Ds had a keyboard change back to ALPS.

Same as the Model 4 2nd version keyboard and also the Model III 2nd version ALPS rubber dome keyswitches returned the nice typing feel of the previous computers to the 4D.
Like the 4 and III you can service the key switches if they fail. As with the previous machines you can replace the keyboard in the 4D with a Jay Crutti unit however you will you will need a new bezel for around the keyboard, and the arrow keys will be back in the traditional TRS-80 layout of up/down on the left of the keyboard and right/left on the right of the keyboard.

Ian Mavric / ianm@trs-80.com

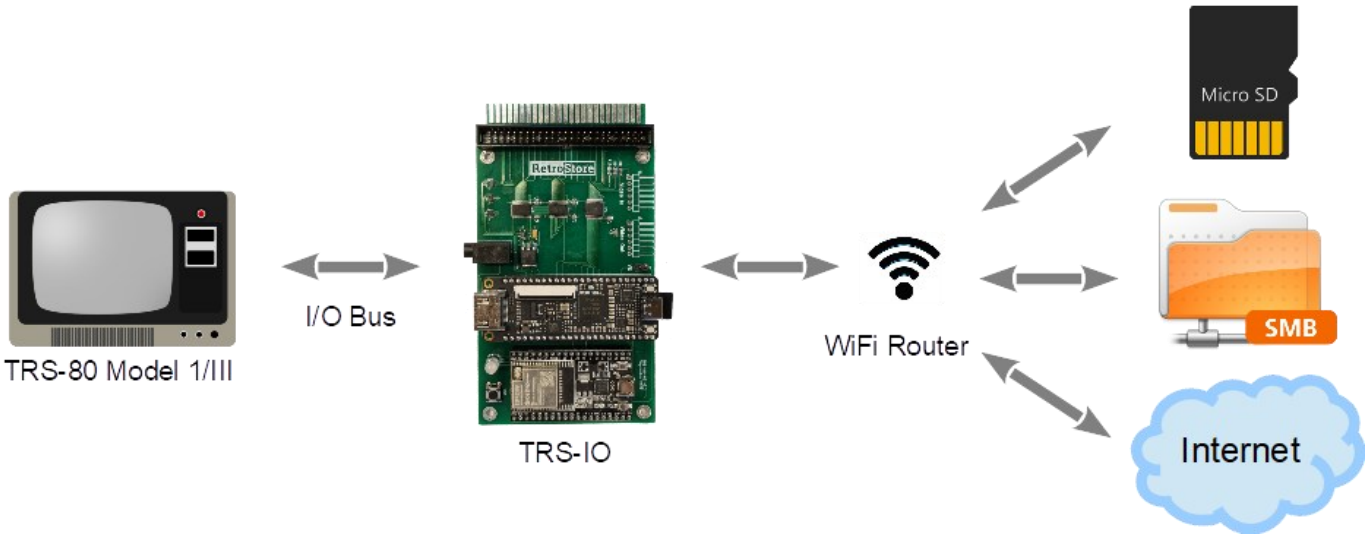


Ian Mavric is an IT Specialist who also restores and collects TRS-80's and classic cars. He lives with his wife and kids in Melbourne, Australia.

Surfin' The Net

TRS-IO OVERVIEW

The figure below depicts the overall architecture. The TRS-80 machine is connected to TRS-IO via the I/O bus. TRS-IO has a WiFi module that connects to the local WiFi router. This allows TRS-IO to connect to the Internet and access various services. Note that TRS-IO can only connect to a 2.4 GHz WiFi network. It is recommended to use an IPEX antenna to improve WiFi reception.



PREREQUISITES

TRS-IO requires a TRS-80 Model III with 48kB of RAM or a Model 1 with 16kB of RAM. Floppy disk drives or a cassette tape are not required.

https://github.com/apuder/TRS-IO?tab=readme-ov-file#overview

XI. Basque Tournament of TRS-80

Egoitz Campo

The last 28th of February, Saturday, the Euskal Retro organized the XI. Basque Tournament of TRS-80. We met 6 players in Bilbao, in Bar Almirante (a bar located in San Ignacio, Bilbao), and we played TRS-80 on a corner of the big bar. It is a big bar and the owner allow us to bring our television and consoles in order to do our tournaments.

It was a great afternoon, funny as always we meet and gather to these old consoles. We started playing TRS-80 after playing Amstrad CPC 464 and ZX Spectrum +3. Although the image is quite bad because my television is PAL, we were able to enjoy the TRS-80 games as always. This time we played these 3 games: Monster Maze, Canyon Climber and Demolition Derbi

Egoitz was the player who achieved more points after playing the 3 games but Maximo, Daniel and the rest played well and passed a good time. For the next tournament, possibly in July of 2026 we hope to have new games like Dragon Fire or Robot Battle, I have to buy new ones in Ebay to change a bit the list of games of our tournaments.

Here you have the results:

Monster Maze
1-Egoitz Campo 5 points (1.770)
2-Daniel Barrientos 4 points (920)
3-Máximo Kulachenok 3 points (800)
4-Joseba Zubiaurre 2 points (420)
5-Carlos Guiracocha 1 point (30)

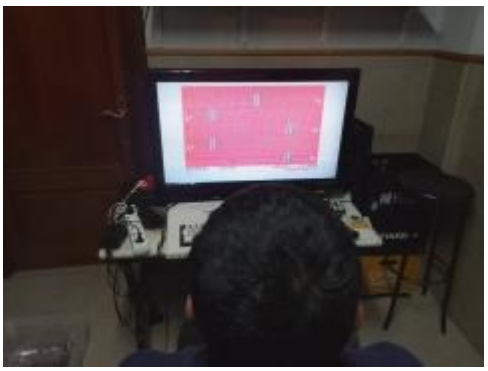
Canyon Climber
1-Egoitz Campo 5 points (23.400)
2-Joseba Zubiaurre 4 points (7.200)
3-Daniel Barrientos 3 points (6.800)
4-Galder Etxebarria 2 points (5.600)
5-Carlos Guiracocha 1 point (3.400)

Demolition Derbi
1-Egoitz Campo 5 points (2.000)
2-Máximo Kulachenok 4 points (1.800)
3-Carlos Guiracocha 3 points (1.400)
4-Joseba Zubiaurre 3 points (1.400)
5-Daniel Barrientos 1 point (900)

Classification of the XI.Basque Tournament of TRS-80

1-Egoitz Campo 15 points (Basque Country)
2-Joseba Zubiaurre 9 points (Basque Country)
3-Daniel Barrientos 8 points (Basque Country)
4-Máximo Kulachenok 7 points (Basque Country)
5-Carlos Guiracocha 5 points (Ecuador)
6-Galder Etxebarria 2 points (Basque Country)

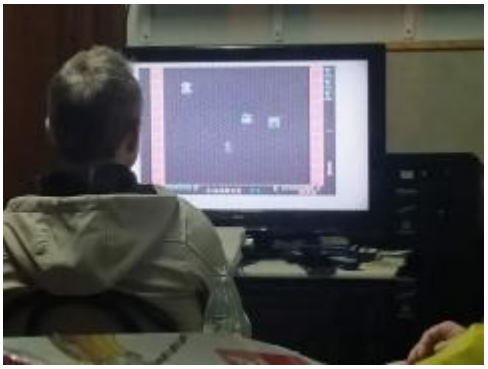






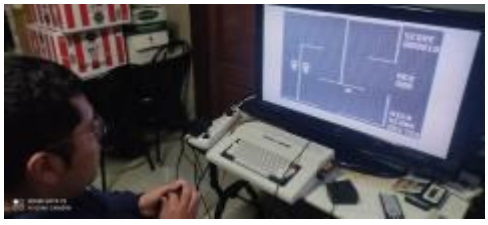


















The *BEST* in TRS-80s

Call The Right Stuff

Ask for Ian
The number is +61 416 184 893

That's The Right Stuff
And he's in Melbourne





COMPETITION ANSWER (5-3)*50+3+4 = 107

TRS8BIT IS SUPPORTED BY YOU AND PRODUCED
AT WWW.TRS-80.ORG.UK, WHERE

WE LOOK FORWARD TO HEARING OF YOUR RECENT TANDY ESCAPADES.