TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

MARCH 2024 - VOL.18 ISSUE 1

PAGE 02 BLURB - (EDITORIAL SOUNDS TOO GRAND!)

PAGE 03 2024 COMPETITION DETAILS.

PAGE 05 IN MAV'S WORKSHOP
IAN MAVRIC
COCO 2 - REPLACEMENT KEYBOARD MEMBRANE

PAGE 12 THE BASIC FAMILY TREE
JIM ADAMS
EVER WONDERED HOW BASIC STARTED? FIND OUT WITH THIS FANTASTICALLY DETAILED ARTICLE

PAGE 35 5X5 MAGIC SQUARE
E.T. FONEHUME
A BASIC ONE-LINER CREATING A 5X5 'MAGIC SQUARE' USING CONSECUTIVE NUMBERS

PAGE 36 CREATING A WAV FILE
JUERGEN BUESSERT

PAGE 39 SOFTWARE TOOLS IN BASIC - EPISODE 3
'OPEN FOR BUSINESS'
VINCE OTTEN

PAGE 57 THE WAY WE WERE
WHAT WERE WE UP TO, BACK IN THE DAY?

PAGE 62 IN MAV'S WORKSHOP
IAN MAVRIC
MAV GIVES A DETAIL ACCOUNT OF THE FB SERIES DISK DRIVES WHICH WERE PROLIFIC BACK IN THE DAY

PAGE 73 U.K. EBAY BARGAINS

PAGE 74 USING TRS-80 ROM CALLS FROM TRSDOS
VINCE OTTEN
MORE USEFUL AND HELPFUL HINTS FROM VINCE

PAGE 79 REAL HARDWARE MADE 'BACK IN THE DAY'
ADDRESS DECODER, MIDI INTERFACE AND BINARY CODED DECIMAL COUNTER.
TONY G

PAGE 85 2024 TANDY ASSEMBLY
IAN MAVRIC

PAGE 90 SURFIN' THE NET
SUPER ITEMS/IDEAS I'VE COME ACROSS - WELL WORTH CHECKING OUT

BLURB - (EDITORIAL SOUNDS A BIT ON THE GRAND SIDE)

HI EVERYONE AND WELCOME TO OUR 73RD ISSUE. AS YOU CAN SEE, I'VE RE-VAMPED THE FRONT PAGE. AFTER 17 YEARS I FELT IT COULD DO WITH A REFRESH! I HOPE IT MEETS WITH YOUR APPROVAL. IF NOT, PLEASE LET ME KNOW.
I'M STILL USING MY FAVOURITE FONT, 'TELETYPE' FOR THIS SORT OF ITEM. IT REMINDS ME SO MUCH OF THE 110 BAUD TELETYPE PRINTER I HAD ATTACHED TO MY M1, THROUGH THE CASSETTE PORT, BACK IN THE DAY.

AS YOU CAN SEE FROM THE FRONT PAGE, ONCE AGAIN, THERE'S A LARGE AND VARIED SELECTION OF ARTICLES THAT I HOPE YOU FIND OF INTEREST.

HAVING JUST ATTENDED TRASH-TALK LIVE, NO.38 AND HAD A VERY ENJOYABLE TIME I DIDN'T REALISE THAT THERE WERE VERY FEW GAMES FOR THE MODEL 2! JUST TO HELP OUT A LITTLE, (USING TRS80GP, OF COURSE), HERE'S ONE OF E.T. FONEHUME'S BASIC ONE-LINER GAMES WHICH I'VE AMENDED TO RUN ON A MODEL 2. IT'S THE SNAIL RACE, WHICH IS ONE OF MY FAVOURITES.

```
10 RANDOM:CLS:S$="  @":FORX=1TO8:PRINT@X*80,X;S$;:PRINT@X*80+79,"*";:SP(X)=X*80+
4:SC(X)=5:NEXT:PRINTCHR$(2):FORZ=0TO1STEP0:A=RND(8):SP(A)=SP(A)+1:SC(A)=SC(A)+1:
PRINT@SP(A),S$;:IFSC(A)=77THENPRINT@880,"WINNER IS NO.";A:PRINTELSENEXTZ
```

```
1                                                    @   *
2                                                @       *
3                                                    @   *
4                                            @           *
5                                            @           *
6                                                        @*
7                                                    @   *
8                                                        @   *
```

WINNER IS NO. 6

ON A DIFFERENT NOTE, I CAME ACROSS A POST THAT MARCEL ERZ HAS PUT UP A GITHUB WITH ASSORTED MODEL I PARTS WHICH CAN BE 3-D PRINTED!
YOU CAN CHECK IT ALL OUT AT:-
HTTPS://GITHUB.COM/RETROSTACK/TRS-80-MODEL-I-3D-OBJECTS.

CURRENTLY PRESENT ARE:

MAIN ENCLOSURE PANEL
PCB STAND-OFF
KEYBOARD STEM (ALPS)
KEYBOARD SPACEBAR CLIP
KEYBOARD CLIP
KEYBOARD BUTTON PLACEHOLDER (ENTER)
KEYBOARD SHIELD
EXTENSION INTERFACE COVER (MEDIUM)



SO, UNTIL JUNE, TAKE CARE EVERYONE - DUSTY

AND NOW, THE NEWS WE'VE ALL BEEN WAITING FOR! ...

TRS8BIT 2024 Competition

Beat “Defense Command”

The objective is to write a program that can play the best game of Big Five Software's “Defense Command”. The program with the highest average score over a series of attempts will win. In the case of ties (e.g., programs can play indefinitely or are not obviously superior to each other) the earlier entrant will be declared the winner.

The programs themselves will be run externally to the emulator running the game. The positions of all game objects and other status information will be sent to an entry program which will respond with updated controller inputs to move the player's ship. The setup will simplify testing and allow competitors to write their program in any language including BASIC.

There are two categories and hence two prizes to be won. The limited category is for programs that run on a TRS-80 itself. It can be any TRS-80 emulated by trs80gp – Model 1, 2, 3, 4, 16, Colour Computer, etc. The unlimited category allows any program you like as long as it can be tested by the competition organizers. Ideally some form of Windows executable or as a service accessible “in the cloud”.

The winners will have their pick of any one of Ian Mavric's fabulous products for the TRS-80 line.

Do not over fit; don't try and predict the game's random number generator as the judging test runs will use different random sequences. Entries must be deterministic – always giving the same response if the input is the same.

Competitors are encouraged to get in touch early in order to receive timely updates on leader board changes, rule clarification and supplementary materials. You are welcome to submit improvements to your entry multiple times.

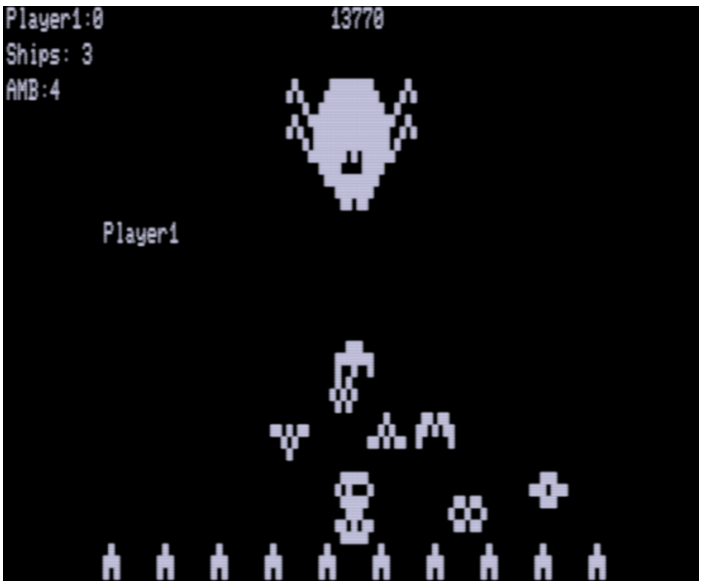
For details on how to talk to the game and a running leader board please consult:

http://48k.ca/comp2024.html

All entries will be judged by George Phillips and
should be sent to george@48k.ca

Competition closing date is 15:00 hrs GMT, 2 November 2023.





LATEST UPGRADE AVAILABLE
VERSION 2.5.3

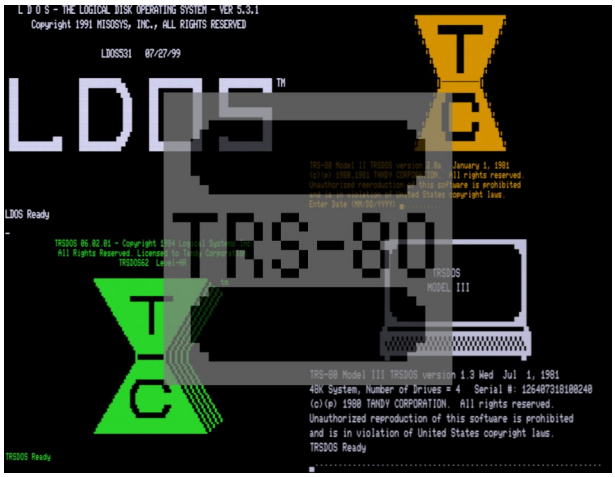
DON'T HAVE A TRS-80?

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 AND VIDEOTEX.

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

NEW VERSION 2.5.3 IS OUT SHORTLY FEATURING:

CASSETTE READING FROM .WAV FILES.
SAVE AND LOAD RAW PRINTER OUTPUT.
GENERATE DISKETTE SUMMARIES ON THE COMMAND LINE.



IN MAV'S WORKSHOP

Review: Coco 2 Replacement Keyboard Membrane

by Ian Mavric



It's been a while since I've reviewed something and so I had the opportunity to try out the TRS-80 Coco 2 Keyboard "Engineered Drop-In Replacement" by Retro Fuzion. The link to buy it will be at the end of the article.

FOR THOSE WHO ARE TOO TIME-POOR TO READ THE ARTICLE

It works well, go buy it, but check your CC2 keyboard version first.

TIME TO REPAIR THE PILE OF COLOR COMPUTER 2s HERE

In my travels I come across computers and a large proportion of generating the means to put food on my table is trading them, usually fixing them as necessary.
It may surprise people that I not only fix and re-sell Tandy computers but generally any computer technology up until the year 2000. Never one to look a gift horse in the mouth over the years I have accumulated a pile of Coco 2s nearly 2ft tall. Interest in the Coco 2 has never been as high as the Coco 3 or even the Coco 1, but I just have a soft spot for them. I set out to repair the broken ones, and upgrade the 16K ones to 64K, and add Extended Color Basic to those which were still original Color Basic models.
The good news was that out of 10 machines, only 2 didn't work, and they were both easily repaired (one had a bad RAM chip and the other a dead CPU). I have always said the CC2 was a very robust machine, so I was glad to see this was still the case... except for the keyboard. Out of the 10, only one had a fully working keyboard.

CAN THE KEYBOARD BE FIXED?

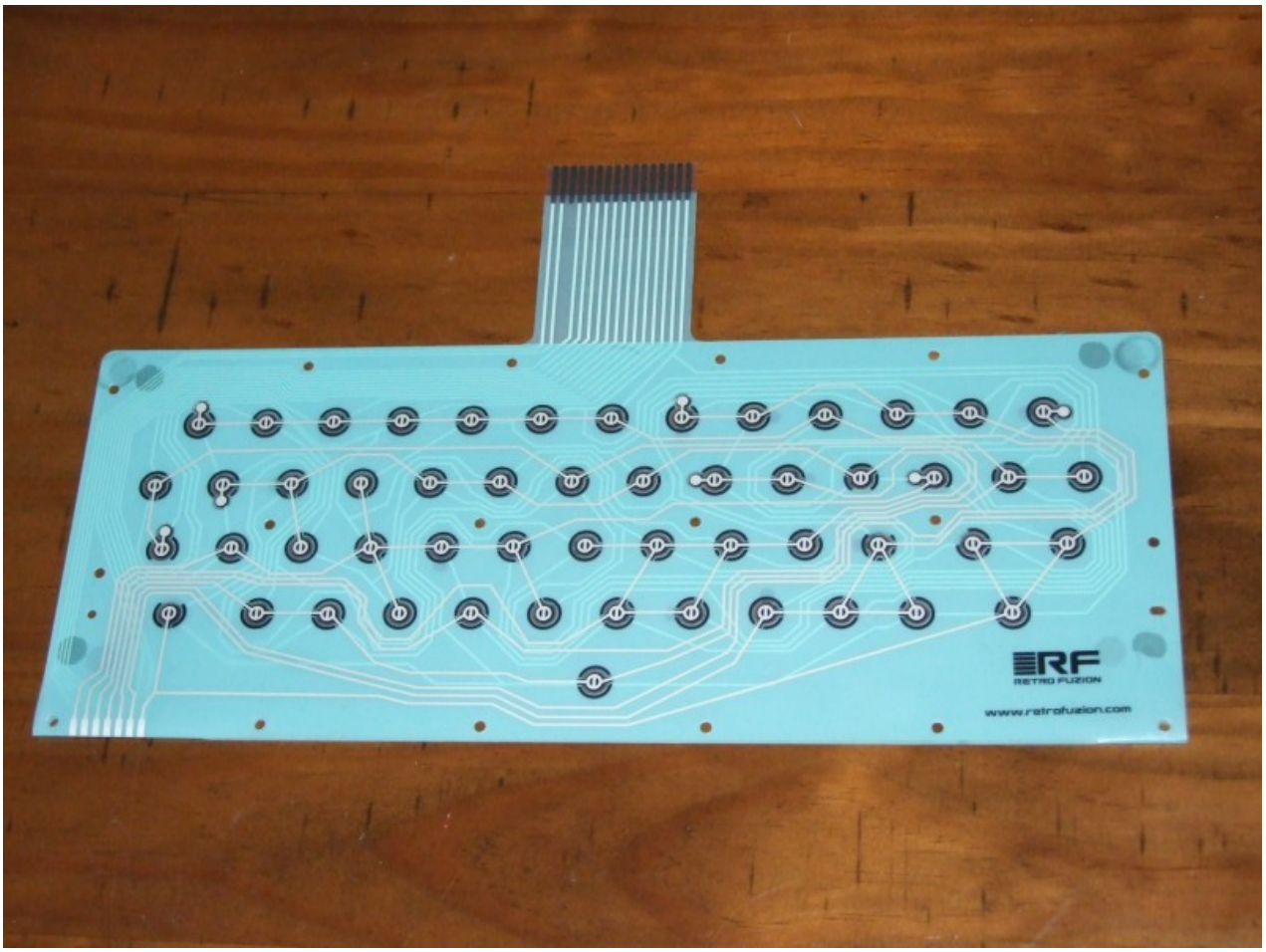
The Coco 2 has a rather simple keyboard, cheaply made and acceptable to type on. It's much better than the Coco 1's chicklet keyboard, and was originally a flatter short-travel design affectionately called the 'melted' keyboard, then a proper typists full(ish) travel keyboard.
Both use a membrane which folds over itself and when a key is pressed an area of the membrane makes contact with itself and registers a key press.

Non working keys generally happen when the black 'pad' area wears out or a trace someone on the membrane develops a fracture. Neither are easy to repair, though some have attempted and succeeded.
The fact is you need to undo 18 tiny screws, make the repair, put the 18 tiny screws in again in order to determine if the repair of successful... then if it wasn't repeat the process again and again until the keyboard is fixed... there has to be a better way...

RETRO FUZION'S DROP-IN REPLACEMENT KEYBOARD MEMBRANE

In a perfect world Radio Shack would still exist, including it's National Parts division and we could just order a new keyboard membrane (or, indeed, whole replacement keyboard), however that has not been possible since the late 1980s.

Retro Fuzion, out of Adelaide, in South Australia, has started producing replacement keyboard membranes for vintage computers. It turns out that not only the Coco 2 is affected by this problem but so is the ZX Spectrum, ZX81, Amiga 500 and 600, Amstrad CPC and others. I paid



my $A34.95 and eagerly awaited it's arrival, which only took a few days.

Photo: Retro Fuzion's Engineered Drop-In Replacement Coco2 kb membrane

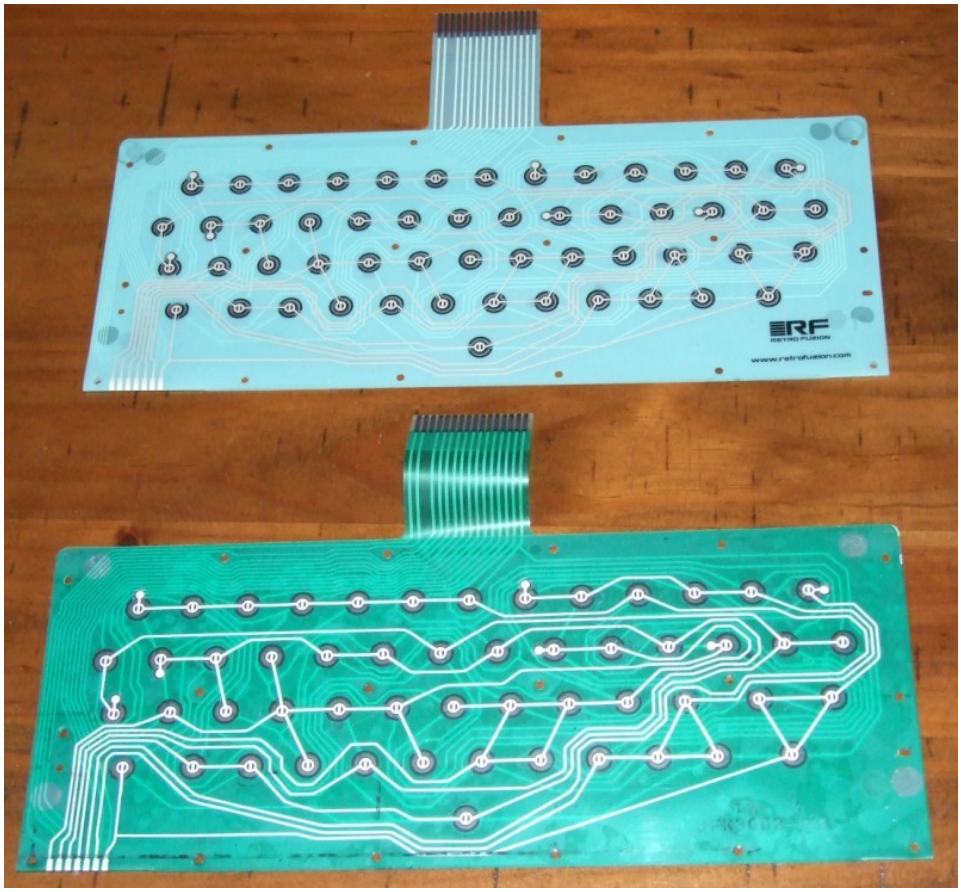


Photo: Replacement membrane and the OEM Tandy membrane

INSTALLATION PROCEDURE

No instructions were included but the installation is dead simple and can be completed in around 15 minutes. Place the keyboard face down but also on both sides chock it up a couple of centimetres so all the keycaps are off the table. This is important, because if there is pressure on any of the keys the rubber domes under each key, which are inside the keyboard, won't sit flush while you place the new membrane.
Undo the 18 screws and put them somewhere safe. Life the metal backing plate off the keyboard and you will see the old membrane. Remove it and place it beside the keyboard.
As mentioned there are rubber domes under each key, some of them will stick to the membrane and others will be dislodged in the process. Put them all back and make sure they are all sitting flush in their recesses. Place the new membrane on the plastic keyboard housing, noting there are two posts which assist in aligning the membrane. Put the metal plate back on and start on the 18 tiny screws. Best practice is put the screw in the hole, turn it backward slowly until you feel it click down, then turn forwards.
Be careful not to over-tighten, remember you are fastening into plastic. Plug it in and try it out. I have done two of them and both worked perfectly first time.



Photo: not a lot to see here but it's returned to my Coco 2 for testing

ANYTHING I NEED TO BE AWARE OF?

Ok so these work on both the early melted keyboard and the later full travel keyboard... most of them. In a perfect world the membrane would work with all Coco 2 keyboards but to complicate things some early computers came with a melted keyboard made by AMP which doesn't lend itself to disassembly and re-assembly. Instead of 18 tiny screws they have some 50 melted plastic posts holding the base on... if you have one of these, sorry, you're out of luck.

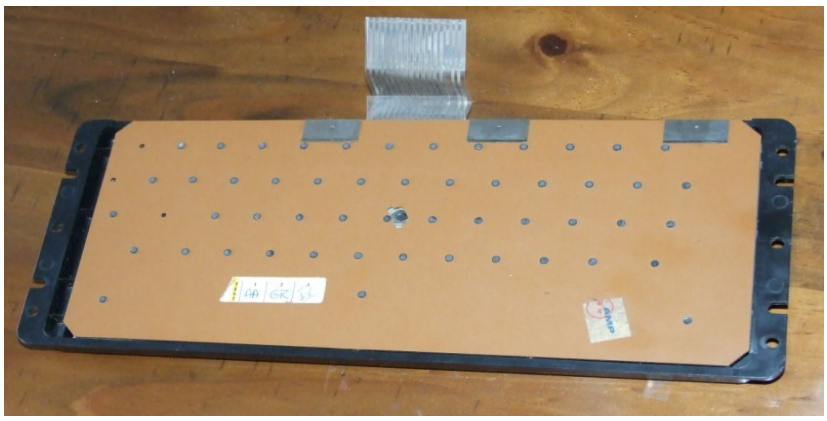


Photo: AMP version Coco 2 keyboard can't use the replacement membrane

Likewise there is a version of the full travel keyboard on later Coco 2s which has a different setup inside and the membrane doesn't line up



inside. The locating posts are in the wrong places.

This one is tricky to identify because it also has 18 tiny screws holding it together. The only way to be sure is to open it up and see if it looks like this one:

Photo: another Coco 2 keyboard design which can't use the membrane

These two keyboards are in the minority. The vast majority of Coco 2 keyboards will be of the design which allows you to replace the membrane.

CONCLUSION

So there you have it, an easy and convenient way to fix the keyboard on your Coco 2. You can order the membrane here:
https://www.ebay.com.au/itm/325851925878

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

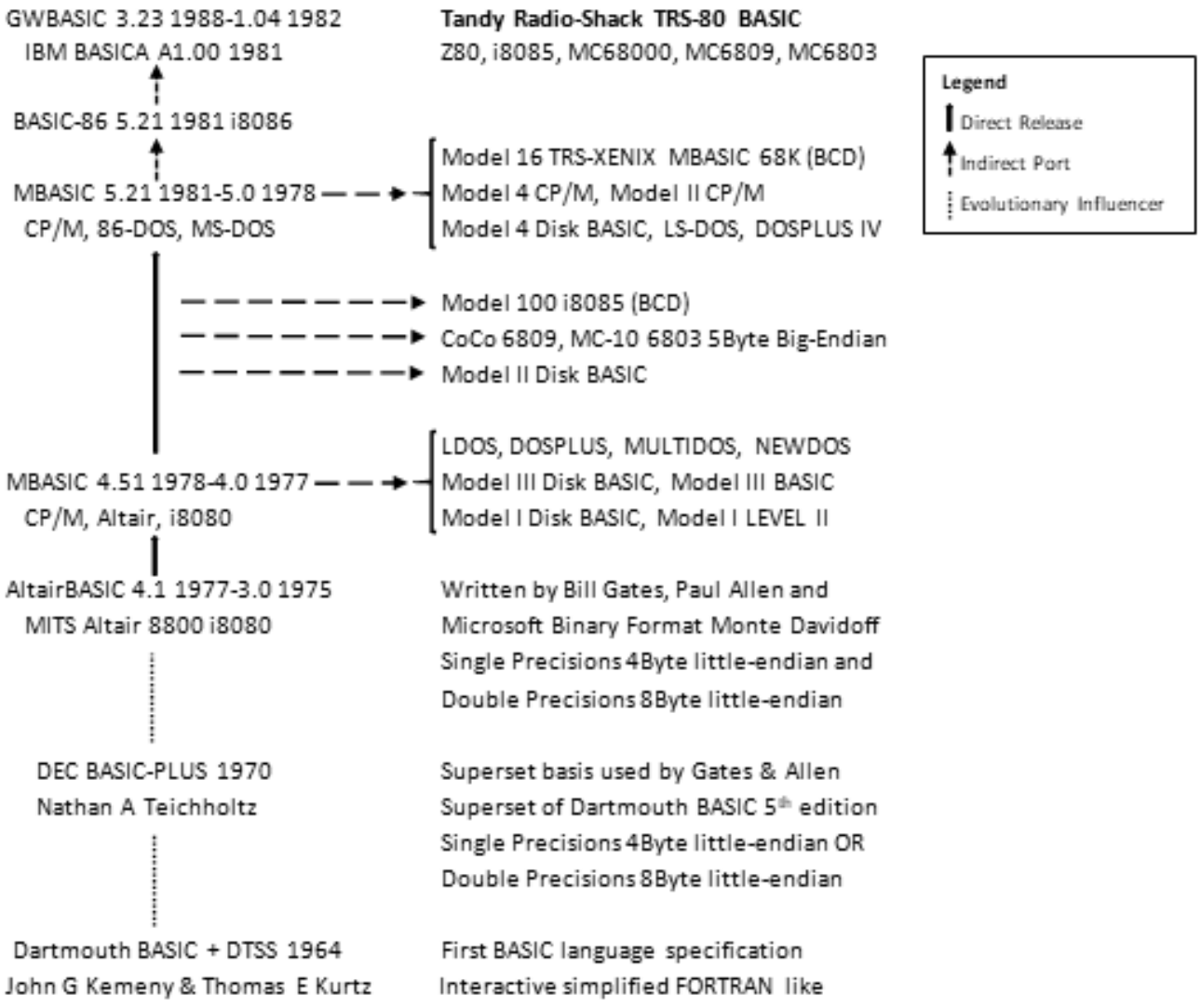
The BASIC Family Pedigree

Jim Adams

Most of us have used BASIC on one or more TRS-80 microcomputer and noticed how each TRS-80 BASIC dialect is very similar yet different from the other. Have you ever wondered about their common origin? Some might have read about *how* the TRS-80 BASIC interpreters came about, while few others already read about its history, and hopefully, after reading this article, few more will become curious to enjoy reading the historical tales. A quick search online yields numerous sites, some with factual events and many with entertaining antidotal stories of how BASIC, TRS-80, TRS-80 BASIC have started.

So I will not rehash the all the history here, instead I will focus on how the BASIC language, in particular the commands, statements, functions and operators, have evolved over time while highlighting few historical events from 1963 to 1990 that greatly influenced the TRS-80 BASIC.

Microsoft BASIC Interpreters Pedigree (Family Ancestry)



To this end, I started searching for firsthand sources, such as biographies, books, articles, and interviews by the original authors or key technologists such as Bill Gates, Paul Allen, Monte Davidoff, Ed Roberts, Nate Teichholtz (since 1984 legally known as Nathan Brookwood), Thomas Kurtz and John Kemeny. I also dived through numerous BASIC reference manuals. Finally, when in doubt about a feature or a topic and the manuals were vague, sometimes intentionally as BASIC documentation was intended for ease of learning not technical completeness, I programmatically validated them using real TRS-80s (I/III/4/4P, II/12/16/16B/6000, 100, CoCo1/2/3), and emulators (trs80gp, Xroar, VirtualT, AltairZ80, Z80Pack, SIMH, pcjs.org, PCE-ibmpc, DOSBox). *[I have done my best to ensure the information is correct, but I am fully aware of my limited knowledge, so please forgive any unintentional errors].*

Now, together we will explore the dawn of the BASIC language and its evolution that led to the most popular family of BASIC interpreters, the TRS-80 BASIC.

Step 0: 1963 BASIC, the Teaching Computer Language

BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code. It was envisioned, created and designed by two Dartmouth professors of Mathematics, Prof. John G Kemeny and Prof. Thomas E Kurtz. Prof Kemeny has sought to create an easy to learn computer language so computer illiterate undergraduate/high school students can quickly grasp it and utilize the computer to pursue their academic endeavors, ie math calculations, statistics problems, humanity correlations, etc. After two prior attempts with DARSIMCO in 1956 and DOPE in 1962, they garnered the lessons learned and applied them in 1963 to create what became the specifications for the BASIC language. BASIC is simple by design, influenced by FORTRAN and ALGOL but without the unnecessary constrains and complexities.

Step 1: 1964 Dartmouth BASIC

Dartmouth Time Sharing System DTSS ***Commands (14)***

BYE, CAT, HELLO, LIST, LIST-xxxx, NEW, OLD, RENAME, RUN, SAVE, SCRATCH, STOP, SYSTEM, UNSAVE
variations +2: GOODBYE, CATALOG
introduced the READY prompt and the 6-character filenames

Statements (15)

DATA, DEF FN*a*, DIM, END, FOR-TO-STEP, GOSUB, GOTO, IF-THEN, LET, NEXT, PRINT, READ, REM, RETURN, STOP
variations +2: GO SUB, GO TO

Functions (11)

ABS(), ATN(), COS(), EXP(), FN*a*(), INT(), LOG(), RND(), SIN(), SQR(), TAN()

Operators (14)

( ), ^, unary + -, * /, + -, < <= = <> >= >
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (1, 1)

variables default to real
arrays default to real()

The first BASIC program ran successfully on 5/1/1964 at 4am at Dartmouth College, hence validating the BASIC language specification, the interactive Dartmouth BASIC language "interpreter" and the Dartmouth Time-Sharing System operating environment. The very first successfully executed two-line program was:

```
10 PRINT 2+2
20 END
```

In February of 1964, Dartmouth installed a General Electric GE-225 20bit 16K word mainframe and connected to a DN-30 18bit 16K word front-end processor; by June of 1964, GE upgraded it to a GE-235 20bit 16K word mainframe that had three times faster core memory.

Dartmouth BASIC first edition, May 1964, was designed for simplicity, intended to be self-taught by students quickly. The DTSS immediate direct commands were entered at the READY prompt without line numbers. The BASIC language statements and functions were entered as program lines each proceeded with a line number from 1 to 99999. The BASIC program was executed via the DTSS RUN command and it will execute from the lowest line number until it encounters a line with the END statement. Variables were numeric only. A variable name referenced either a simple scalar, a single dimension array or a two dimension array. May not use the same name as a scalar and an array, eg either A or A() were valid; but both names may not have appeared in the same program.

In addition to infamous READY prompt and all the commands, statements, functions and operators, Dartmouth BASIC first edition also gave us the following familiar BASIC language features: variables defaulted to real floating-point 6-digit significant, arrays defaulted to 0 to 10 subscripts, and PRINT supported the field separators: comma and semicolon. New editions incrementally added more features.

Step 2: 1970 DEC BASIC-PLUS

DEC PDP-11 RSTS-11 ***Commands (22)***

ASSIGN, BYE, CAT, COMPILE, CONT, DEASSIGN, DELETE *start-end*, HELLO, KEY, LENGTH, LIST, LIST *start-end*, LISTNH *start-end*, NEW, OLD, RENAME, REPLACE, RUN, RUNNW, SAVE, TAPE, UNSAVE
variations +2: GOODBYE, CATALOG

Statements (50)

!, CHAIN, CHANGE, CLOSE, DATA, DEF FN*a*, DIM, END, FIELD, FNEND, FOR-STEP-UNTIL, FOR-STEP-WHILE, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT LINE, INPUT LINE#, INPUT#, KILL, LET, LSET, MAT, MAT INPUT, MAT PRINT, MAT READ, NAME, NEXT, ON ERROR GOTO, ON GOSUB, ON GOTO, OPEN, PRINT, PRINT USING, PRINT#, PRINT# USING, PUT, RANDOM, READ, REM, RESTORE, RESUME, RETURN, RSET, SLEEP, STOP, UNLOCK, WAIT
variations +3: GO SUB, GO TO, RANDOMIZE
modifiers +5: FOR, IF, UNLESS, UNTIL, WHILE

Functions (40)

ABS(), ASCII(), ATN(), CHR$(), COS(), CVT%$(), CVT$%(), CVT$F(), CVTF$(), DATE$(), EXF(), EXP(), FIX(), FN*a*(), INSTR(), INT(), INV(), LEFT(), LEN(), LOG(), LOG10(), MAGTAPE(), MID(), NUM$(), POS(), RAD$(), RIGHT(), RND(), SGN(), SIN(), SPACE$(), SQR(), SWAP(), TAB(), TAN(), TIME(), TIME$(), TRN(), VAL(), XLATE()
system values +8: DET, e, ERL, ERR, NUM, NUM2, PI, RECOUNT

Operators (22)

( ), ^, unary + -, * /, + -, < <= = <> >= > ==, NOT, AND, OR XOR, EQV, IMP
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (3, 3)

variables default to real, integers %, string $
arrays default to real(), integer arrays %(), string arrays $()

During the mid and late 1960's the Dartmouth DTSS and BASIC spread across academia, helped with the fact Dartmouth was giving away the source code for free. Dartmouth allowed high schools remote access to DTSS and BASIC, true to form to Kemeny's and Kurtz's vision to allow students to quickly learn a simple computer language, BASIC, and begin utilizing computers for many applications.

Commercializing on the rapid popularity of BASIC, Hewlett Packard (HP) in 1969 introduced the HP 2000A 16bit mini-computer for 16 concurrent users/students with Time-Shared BASIC (modeled after Dartmouth BASIC fourth edition 1968), targeting the midsize business segment and the educational market.

In response to HP 2000A Time-Shared BASIC success which tokenized BASIC statements, Digital Equipment Corporation (DEC) in 1970, introduced the DEC PDP-11 16bit mini-computer for 16 concurrent users/students with the RSTS-11 operating system and the DEC BASIC-PLUS interpreter designed by Nathan A Teichholtz. To distinguish its product from the competition, DEC BASIC-PLUS specifications added few enhancements to Dartmouth BASIC 5th edition 1969-1970. DEC also ported DEC BASIC-PLUS to its PDP-10 36bit mainframe product line, used by many universities that also offered remote access for high schools.

DEC BASIC-PLUS included more familiar BASIC language features: keywords do not require spaces as delimiters, immediate commands may be entered as a program line with line number (eg 1000 RUN "PGM2"), BASIC statements may be entered as immediate commands without line numbers (eg PRINT 2+2), string variables with $, integer variables with %, variable name plus the data type form a unique variable (eg A, A$, A%, A(), A$(), A$() are all different names and reference different data variables), all variables are initialized to 0 or "", ERR for error number, and ERL for error line number. Later editions incrementally added more features.

A Monumental Step

DEC BASIC-PLUS is a significant stepping stone towards the origin of the TRS-80 BASIC. Already we can see the foundation of the majority of TRS-80 BASIC syntax and features embodied in DEC BASIC-PLUS; and indeed, your patience for reading through the origin of the BASIC language is about to pay off high dividends.

By the fall of 1969, far away from Hanover, New Hampshire (Dartmouth BASIC) and Maynard, Massachusetts (DEC BASIC-PLUS), but at Seattle, Washington (Lakeside private school, grades 5-12), is where the 8th grader William H Gates and the 10th grader Paul G Allen met while they taught themselves programming in DEC BASIC on a Teletype Model 33 ASR remotely connected. By 1970, they had access to a DEC PDP-10 mainframe running TOPS-10 time sharing system with DEC BASIC-PLUS. They mastered DEC BASIC-PLUS so well, that by the time Paul graduated high school in 1971 and Bill graduated in 1973, they both have automated Lakeside's class scheduling using DEC BASIC-PLUS.

Step 3: 1975 MITS Altair 8800 BASIC

ALTAIR EXTENDED BASIC 3.2 12K December 1975

Statements/Commands (43)

? , CLEAR, CLOAD, CLOAD?, CONT, CSAVE, DATA, DEF FN*a*, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERASE, FOR-TO-STEP, GOSUB, GOTO, IF-THEN-ELSE, INPUT, LET, LIST, NEW, NEXT, NULL, ON GOSUB, ON GOTO, POKE, PRINT, PRINT USING, READ, REM, RESTORE, RETURN, RUN, STOP, SWAP, TROFF, TRON, WAIT

variations due to optional space delimiters, eg GOTO, GO TO, DEFDBL, DEF DBL, etc

Functions (30)

ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), EXP(), FN*a*(), FRE(), INP(), INT(), LEFT$(), LEN(), LOG(), MID$(), PEEK(), POS(), RIGHT$(), RND(), SGN(), SIN(), SPC(), SQR(), STR$(), TAB(), TAN(), USR*n*(), VAL()

Operators (20)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR

ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $

arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()

ALTAIR EXTENDED BASIC 4.1 16K July 1977

Statements/Commands (65)

? , ' , AUTO, CLEAR, CLOAD, CLOAR?, CLOAD*, CLOSE, CONT, CSAVE, CSAVE*, DATA, DEF FN*a*, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERASE, ERROR, FOR-TO-STEP, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, LET, LINE INPUT, LINE INPUT#, LIST, LLIST, LPRINT, LPRINT USING, MID$()=, NEW, NEXT, NULL, ON ERROR, ON GOSUB, ON GOTO, OPEN, OUT, POKE, PRINT, PRINT USING, PRINT#, PRINT# USING, READ, REM, RENUM, RESTORE, RESUME, RETURN, RUN, STOP, SWAP, SYSTEM, TROFF, TRON, WAIT, WIDTH

variations due to optional space delimiters, eg GOTO, GO TO, DEFDBL, DEF DBL, etc

Functions (45)

& , &H, &O, ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), EOF(), ERL, ERR, EXP(), FIX(), FN*a*(), FRE(), HEX$(), INKEY$, INP(), INSTR(), INT(), LEFT$(), LEN(), LOG(), LPOS(), MID$(), OCT$(), PEEK(), POS(), RIGHT$(), RND(), SGN(), SIN(), SPACE$(), SPC(), SQR(), STR$(), STRING$(), TAB(), TAN(), USR*n*(), VAL(), VARPTR()

Operators (23)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR, XOR, EQV, IMP

ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $

arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()

[For ALTAIR DISK EXTENDED BASIC language details, please see the Microsoft BASIC-80 4.51 18K 1977]

With the advent of the Intel 8080 8bit microprocessor in 1974, the late Dr. Ed Roberts, MD, quickly designed the Micro Instrumentation and Telemetry Systems (MITS) Altair 8800 microcomputer, originally as a full kit, then offered as a fully assembled units. The watershed moment was featuring the MITS Altair 8800 on the cover of the January 1975 issue of Radio Electronics magazine, causing thousands of back orders by February 1975. The Intel 8080 was Intel's *first* 8bit general purpose MPU after their *initial* i8008 8bit bit-slicing MPU designed for Datapoint 2200 programmable terminal.

William H Gates, while at Harvard as an undergraduate student, discussed the MITS Altair 8800 with Paul G Allen and they embarked on creating a BASIC interpreter for it. Hence, creating the very first BASIC interpreter for the microcomputer industry. Paul developed an Intel 8080 emulator that ran on Harvard's PDP-10, while Bill and Paul proceeded to create Altair BASIC, drawing from their own experiences with DEC BASIC-PLUS on PDP-10 and PDP-8.

They enlisted Monte Davidoff, another Harvard student to create the floating point math package, which later became known as the Microsoft Binary Format used by all the Microsoft language products from 1975 to 1987 when they switched executively to the IEEE 754-1985 standard. *[Enough reminiscing of my other journey researching the MBF origin, DEC PDP floating point influence, accuracy, limits, etc.]*

Interestingly enough, when Paul Allen demonstrated their BASIC interpreter to Ed Roberts for the first time on a real MITS Altair 8800, the first BASIC statement he executed was PRINT 2+2, the very same first test statement of the Dartmouth BASIC.

MITS Altair 8800 BASIC 2.0 4K 1975 paper tape, became Micro-Soft's (as it was originally known) inaugural product and the first microcomputer BASIC language. New versions 3.0, 3.2, 4.0, 4.1... and releases 8K, EXTENDED BASIC 16K, DISK BASIC 18K added more of the familiar TRS80 BASIC features.

Step 4: 1977 Microsoft BASIC

Microsoft non-disk EXTENDED BASIC-80 4.51 16K 1977

[For non-disk EXTENDED BASIC language details, please see the ALTAIR EXTENDED BASIC 4.1 16K 1977]

Microsoft BASIC-80 4.51 18K 1977 for CP/M

Statements/Commands (77)

?, ', AUTO, CLEAR, CLOAD, CLOAR?, CLOAD*, CLOSE, CONT, CSAVE, CSAVE*, DATA, DEF FN*a*, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, DSKO$, EDIT, END, ERASE, ERROR, FIELD, FILES, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, KILL, LET, LINE INPUT, LINE INPUT#, LIST, LLIST, LOAD, LPRINT, LPRINT USING, LSET, MERGE, MID$()=, NAME, NEW, NEXT, NULL, ON ERROR, ON GOSUB, ON GOTO, OPEN, OUT, POKE, PRINT, PRINT USING, PRINT#, PRINT# USING, PUT, READ, REM, RENUM, RESTORE, RESUME, RETURN, RSET, RUN, SAVE, STOP, SWAP, SYSTEM, TROFF, TRON, WAIT, WIDTH
variations due to optional space delimiters, eg GOTO, GO TO, DEFDBL, DEF DBL, etc

Functions (56)

&, &H, &O, ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), CVD(), CVI(), CVS(), DSKF(), DSKI$(), EOF(), ERL, ERR, ERR(), EXP(), FIX(), FN*a*(), FRE(), HEX$(), INKEY$, INP(), INSTR(),

INT(), LEFT$(), LEN(), LOC(), LOF(), LOG(), LPOS(), MID$(), MKD$(), MKI$(), MKS$(), OCT$(), PEEK(), POS(), RIGHT$(), RND(), SGN(), SIN(), SPACE$(), SPC(), SQR(), STR$(), STRING$(), TAB(), TAN(), USR*n*(), VAL(), VARPTR()

Operators (23)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR, XOR, EQV, IMP
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $
arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()

Altair EXTENDED BASIC 4.1 and DISK BASIC 4.1 became the basis for Microsoft BASIC-80 for CP/M, or MBASIC, which became the de facto standard for most 8bit microcomputers. In addition to the original Intel 8080, MBASIC 4.1 1977 to 4.51 1977 were ported to most of the popular 8bit MPUs: 8085, Z80, 6800, 6809, 6502; and were customized for the TRS-80, Apple, PET, ATARI, OSI-500, and many more home computers.

MBASIC 4.x and Altair EXTENDED BASIC 4.x/3.2 all support four data types for scalar variables and four data types for multi-dimension arrays, namely single precision floating points (default), double precision floating points, integers and strings. Variable names are upto 2 character significant. Thus allowing 3,848 unique scalar variables and 3,848 unique arrays. These are theoretical maximum since the memory required for the variables storage (Variable List Table) and the assignment statements (Program Statement Table) will exceed the 64KB maximum memory accessible by 8bit MPUs.

[Here is how I derived the maximum variables, please feel free to correct my misunderstanding]
2char significant (1st char A-Z, 2nd char A-Z or 0-9)
26 A-Z + (26 A-Z * (((26 A-Z) + (10 0-9)) ^1)) = 962 unique names
962 * 4 data types = 3,848 unique scalar variables, and 962 * 4 data types = 3,848 unique arrays

The very first TRS-80 Model I LEVEL II BASIC ROM 12K, announced in 1977 but began shipping in March 1978, was indeed a licensed port of the Microsoft EXTENDED BASIC 16K 4.46 1977 reduced to 12K to meet Tandy's memory map requirements.

Microsoft BASIC-80 5.21 24K 1981 for CP/M

Statements/Commands (84)

?, ', AUTO, CALL, CHAIN, CLEAR, CLOAD, CLOSE, COMMON, CONT, CSAVE, DATA, DEF FN*a*, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERASE, ERROR, FIELD, FILES, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, KILL, LET, LINE INPUT, LINE INPUT#, LIST, LLIST, LOAD, LPRINT, LPRINT USING, LSET, MERGE, MID$()=, NAME, NEW, NEXT, NULL, ON ERROR, ON GOSUB, ON GOTO, OPEN, OPTION BASE, OUT, POKE, PRINT, PRINT USING, PRINT#, PRINT# USING, PUT, RANDOMIZE, READ, REM, RENUM, RESET, RESTORE, RESUME, RETURN, RSET, RUN, SAVE, STOP, SWAP, SYSTEM, TROFF, TRON, WAIT, WEND, WHILE, WIDTH, WIDTH LPRINT, WRITE, WRITE#
requires at least a single space as a keyword delimiter

Functions (54)

&, &H, &O, ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), CVD(), CVI(), CVS(), EOF(), ERL, ERR, EXP(), FIX(), FN*a*(), FRE(), HEX$(), INKEY$, INP(), INPUT$(), INSTR(), INT(), LEFT$(), LEN(), LOC(), LOF(), LOG(), LPOS(), MID$(), MKD$(), MKI$(), MKS$(), OCT$(), PEEK(), POS(), RIGHT$(), RND(), SGN(), SIN(), SPACE$(), SPC(), SQR(), STR$(), STRING$(), TAB(), TAN(), USR*n*(), VAL(), VARPTR()

Operators (23)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR, XOR, EQV, IMP
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $
arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()

MBASIC 5.x, similar to MBASIC 4.x also supports four data types for scalar variables and four data types for multi-dimension arrays, but variables names are upto 40 character significant. Thus allowing 1.5E+63 unique scalar variables and 1.5E+63unique arrays. These are theoretical maximum since the memory required for the variables exceeds the 64KB maximum memory.

[Here is how I derived the maximum variables, please feel free to correct my misunderstanding]
2char significant (1st char A-Z, 2nd to 40th char A-Z, 0-9 or .)
26 A-Z + (26 A-Z * (((26 A-Z) + (10 0-9) + (1 .)) ^39)) = 3.75E+62 unique names
3.75E+62 * 4 data types = 1.5E+63 unique scalar variables, and
3.75E+62 * 4 data types = 1.5E+63 unique arrays

The Microsoft BASIC-80 for CP/M 1977-1981 *(aka MBASIC)* became the de facto BASIC interpreter standard and its ubiquitous availability on most 8bit microcomputers, fueled Microsoft's growth.

Step 5: 1976 Tiny BASIC

Palo Alto Tiny BASIC 1.77K 1976

Commands (3)

LET *L.*, NEW *N.*, RUN *R.*

Statements (11)

FOR-TO-STEP *F. S.*, GOSUB *GOS.*, GOTO *G.*, IF, INPUT *IN.*, LET *implied*, NEXT *N.*, PRINT *P.*, REM, RETURN *R.*, STOP *S.*
variations are due to the optional *abbreviation* of statements, commands and functions

Functions (3)

ABS() *A.()*, RND() *R.()*, SIZE *S.*

Operators (16)

( ), unary + -, * /, + -, < <= = # >= >, * *AND*, + *OR*
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (1, 1)

variables default to integer
a single array defaults to integer ()

Dennis Allison from Stanford SLAC, and encouraged by Bob Alberecht of the Homebrew Computer Club, began a series of articles to describe and revise the minimum BASIC, starting with the *"BUILD YOUR OWN BASIC"* in the *PEOPLE'S COMPUTER COMPANY Newsletter*, volume 3, number 4, May 1975. Dennis' do it yourself kit for BASIC specifications listed the minimum as: < 4K of memory, integer only math, variables are A-Z, RND() and seven statements (GOSUB, GOTO, IF, INPUT, LET, PRINT, RETURN). This eventually grew into *Dr. Dobb's Journal of Tiny BASIC Calisthenics & Orthodontia*, first issue in January 1976. Dr. Dobb's larger journal format began printing source code of Tiny BASIC interpreters for the intel 8008 and intel 8080.

In the May 1976 issue of *Dr. Dobb's Journal*, volume 1, issue 5, it featured the annotated source code listing of the *"Palo Alto Tiny BASIC"* by Dr. Li Chen Wang for the i8080. This public domain version of Tiny BASIC (aka PATB) met Dennis' specifications and more than doubled the number of statements, double the functions, added a single array @(), all while squeezing it into 1.77KB. Li Chen also documented hocks to expand the parser's accepted statements and functions. PATB influence is found in many later BASIC dialects, most notably, the TRS-80 LEVEL I BASIC and the Sharp PC-1211 S'-BASIC (TRS-80 PC-1).

Step 6: 1977 TRS-80 BASIC

TRS-80 LEVEL I BASIC 4K 1977

Statements/Commands (29)

CLOAD, CLS, CONT *C.*, CSAVE, DATA *D.*, END *E.*, FOR-TO-STEP *F. S.*, GOSUB *GOS.*, GOTO *G.*, IF-THEN *T.*, INPUT *IN.*, LET *implied*, LIST *L.*, NEW *N.*, NEXT *N.*, ON GOSUB, ON GOTO, PRINT *P.*, PRINT AT *P.A.*, READ *REA.*, REM, RESET *R.*, RESTORE *REST.*, RETURN *RET.*, RUN *R.*, SET *S.*, STOP *ST.*
Model III LEVEL I 4K 1980 added: LLIST, LPRINT
variations are due to the optional *abbreviation* of statements, commands and functions

Functions (6)

ABS() *A.()*, INT() *I.()*, MEM *M.*, POINT() *P.()*, RND() *R.()*, TAB *T.()*

Operators (16)

( ), unary + -, * /, + -, < <= = <> >= >, * *AND*, + *OR*
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (2, 1)

variables default to single precision real, string $
arrays default to single precision real ()

TRS-80 LEVEL I BASIC 4K for the Model I was Tandy's port of Li-Chen Wang's public domain Palo Alto Tiny BASIC (*PATB*) 1.77K, published in the May 1976 issue of *Dr. Dobb's Journal*. Tandy's Steven W Leininger added floating point math; support for the Model I hardware I/O (keyboard, video, and cassette); and added more BASIC features (15 statements, 3 functions, 1 string variable) while squeezing it all into a 4K ROM as he ported Tiny BASIC in i8080 to the TRS-80. This was the only release of TRS-80 BASIC that was

not developed by Microsoft and does not conform to the Microsoft's BASIC syntax (lacks EDIT, PRINT USING, MBF floating point format).

LEVEL I BASIC supports two data types for scalar variables and one data type single-dimensions array. 1 character significant variable names and the only variables allowed are A-Z, A$, B$, A().

26 real floating point scalar variables A-Z
2 string scalar variables A$, B$
1 real floating point array A()

TRS-80 Model III LEVEL I 4K 1980 was updated by Tandy and adds two statements, the LLIST, LPRINT.

TRS-80 Pocket Computer 1 (PC-1) 1980, was a rebranded Sharp PC-1211 1980, and had a similar BASIC which was also derived from Li-Chen Wang's PATB but independently developed by Sharp.

TRS-80 LEVEL II BASIC 12K March 1978

Statements/Commands (56)

?, ', *comma*, AUTO, CLEAR, CLOAD, CLOAR?, CLS, CONT, CSAVE, DATA, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERROR, FOR-TO-STEP, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, LET, LIST, LLIST, LPRINT, LPRINT USING, NEW, NEXT, ON ERROR, ON GOSUB, ON GOTO, OUT, POKE, PRINT, PRINT USING, PRINT#, PRINT# USING, PRINT@, RANDOM, READ, REM, RESET, RESTORE, RESUME, RETURN, RUN, SET, STOP, SYSTEM, TROFF, TRON
variations due to optional space delimiters, eg GOTO, GO TO, DEFDBL, DEF DBL, etc

Functions (36)

ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), ERL, ERR, EXP(), FIX(), FRE(), INKEY$, INP(), INT(), LEFT$(), LEN(), LOG(), MEM, MID$(), PEEK(), POINT(), POS(), RIGHT$(), RND(), SGN(), SIN(), SQR(), STR$(), STRING$(), TAB(), TAN(), USR(), VAL(), VARPTR()

Operators (18)

( ), ^, unary + -, * /, + -, < <= = <> >= >, NOT, AND, OR
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $
arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()

As Microsoft BASIC-80 spread across the sea of CP/M microcomputers becoming the de facto standard. Consequently, Tandy licensed the Model I LEVEL II BASIC from Microsoft, and proprietary hardware hooks and details furbished by Tandy's own developers. Tandy announced the planned TRS-80 LEVEL II in late 1977 while it shipped the initial orders of the Model I with LEVEL I ROM. LEVEL II BASIC ROM is mostly Microsoft Extended BASIC-80 16K 4.46 reduced to fit into 12K only, per Tandy's memory map specifications.

The TRS-80 LEVEL II BASIC ROM began shipping in March 1978 as an upgrade to existing Model LEVEL I 4K or 16K; upgrade catalog # 26-1120 for $120 including installation. It was also available as new Model I LEVEL II 4K, 26-1054 $689.95 or Model I LEVEL II 16K, 26-1056 $988.95. The final LEVEL II ROM version was 1.3 in 1979.

In 1980, as the Model III was being designed, Tandy internally derived the Model III BASIC ROM 14K 1980 from the Model I LEVEL II BASIC ROM 12K to avoid paying loyalty to Microsoft for the TRS-80 Model III; and to distinguish it as a separate independent product, the Model III BASIC was never officially labeled LEVEL II BASIC but the manuals referred to it as the Model III BASIC. The Model III BASIC ROM was 14K with 2K additional routines to support the lowercase, built in real clock timer and added the TIME$ function.

Microsoft also offered LEVEL III BASIC in 1979, developed by Bill Gates, for the TRS80 Model I LEVEL II. It added functionality removed to reduce Microsoft EXTENDED BASIC-80 16K to LEVEL II BASIC 12K ROM. Similar to the non-disk enhancements included in TRSDOS 2.3 DISK BASIC 2.2 but limited to cassette I/O. It may be loaded from TRSDOS but only supports loading and saving cassette files. It added 14 statements: DEF FN*a*, DEF USR*n*, GET, INPUT#LEN, LINE, LINE INPUT, LINE INPUT#LEN, LOAD, LSET, MID$()=, NAME, PRINT#-3, PUT, SAVE; and 7 functions: &, &H, &O, FN*a*(), INSTR(), TIME$, USR*n*().

George Phillips published TWOHAF in *TRS8BIT* volume 17 issue 4 December 2023, updating Gil Spencer's LEVEL 2.5 that was originally published in the *80 Micro* July 1981 issue 19. George's version adds 3 statements: DEF FN*a*, DEF USR*n*, MID$()=; and 10 functions: &, CVD(), CVI(), CVS(), FN*a*(), INSTR(), MKD(), MKI(), MKS(), USR*n*().

All the LEVEL II BASICs support variable names upto 2 character significant and 4 data types; theoretically allowing 3,848 unique scalar variables and 3,848 unique arrays *[see Microsoft BASIC-80 4.51 1977 for details]*.

TRS-80 DISK BASIC 2.2 16K 1978 for TRSDOS 2.3

Statements/Commands (88)

?, ', *comma*, AUTO, CLEAR, CLOAD, CLOAD?, CLOSE, CLS, CMD"A", CMD"B", CMD"C", CMD"D", CMD"E", CMD"I", CMD"J", CMD"L", CMD"O", CMD"P", CMD"R", CMD"S", CMD"T", CMD"X", CMD"Z", CONT, CSAVE, DATA, DEF FN*a*, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERROR, FIELD, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, KILL, LET, LINE INPUT, LINE INPUT#, LIST, LLIST, LOAD, LPRINT, LPRINT USING, LSET, MERGE, MID$()=, NAME, NEW, NEXT, ON ERROR, ON GOSUB, ON GOTO, OPEN, OUT, POKE, PRINT, PRINT USING, PRINT#, PRINT# USING, PRINT@, PUT, RANDOM, READ, REM, RESET, RESTORE, RESUME, RETURN, RSET, RUN, SAVE, SET, STOP, SYSTEM, TROFF, TRON

variations due to optional space delimiters, eg GOTO, GO TO, DEFDBL, DEF DBL, etc

Functions (51)

&, &H, &O, ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), CVD(), CVI(), CVS(), EOF(), ERL, ERR, EXP(), FIX(), FN*a*(), FRE(), INKEY$, INP(), INSTR(), INT(), LEFT$(), LEN(), LOC(), LOF(), LOG(), MEM, MID$(), MKD$(), MKI$(), MKS$(), PEEK(), POINT(), POS(), RIGHT$(), RND(), SGN(), SIN(), SQR(), STR$(), STRING$(), TAB(), TAN(), TIME$, USR*n*(), VAL(), VARPTR()

Operators (18)

( ), ^, unary + -, * /, + -, < <= = <> >= >, NOT, AND, OR

ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $

arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()

Tandy also licensed the Model I TRSDOS DISK BASIC (4K addon to the Model I LEVEL II BASIC ROM 12K) from Microsoft with hardware hooks and TRSDOS details added or supplied by Tandy's engineers. TRS-80 DISK BASIC is based mostly on the Microsoft BASIC-80 4.46 for CP/M and the 8080 code base.

In 1980, while designing the TRS-80 Model III, Tandy internally derived the Model III Disk BASIC 1980 (5K addon to Model III BASIC ROM 14K) from the Model I DISK BASIC to avoid paying loyalty to Microsoft.

All BASIC for the TRS-80 Model I, the Model III, and the Model 4 in a Model III mode support variable names upto 2 character significant and 4 data types; theoretically allowing 3,848 unique scalar variables and 3,848 unique arrays *[see Microsoft BASIC-80 4.51 1977 for details]*.

All Disk BASIC for the TRS-80 Model I, the Model III, and the Model 4 in a Model III mode, eg {TRSDOS 2.3, TRSDOS 1.3, MODELA/III, LDOS 5.3.1H, MULTIDOS 5.10, DOSPLUS 3.5, NEWDOS/80 2.5, NEWDOS/90 3.2a, VTOS 4.0 ...}, are a modified port of the Microsoft BASIC-80 4.46 code base.

Step 7: 1983 TRS-80 Model 4 Disk BASIC

TRS-80 Model 4 Disk BASIC 1.1.1 24K 1985 for TRSDOS 6.2.1AR

Statements/Commands (83)

? , ' , AUTO, CALL, CHAIN, CLEAR, CLOSE, CLS, COMMON, CONT, DATA, DEF FN*a*, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERASE, ERROR, FIELD, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, KILL, LET, LINE INPUT, LINE INPUT#, LIST, LLIST, LOAD, LPRINT, LPRINT USING, LSET, MERGE, MID$()=, NAME, NEW, NEXT, ON ERROR, ON GOSUB, ON GOTO, OPEN, OPTION BASE, OUT, POKE, PRINT, PRINT USING, PRINT#, PRINT# USING, PRINT@, PRINT@(), PUT, RANDOM, READ, REM, RENUM, RESTORE, RESUME, RETURN, RSET, RUN, SAVE, SOUND, STOP, SWAP, SYSTEM, SYSTEM"*fff*", TROFF, TRON, WAIT, WEND, WHILE, WIDTH, WRITE, WRITE#

requires at least a single space as a keyword delimiter

Functions (59)

&, &H, &O, ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), CVD(), CVI(), CVS(), DATE$, EOF(), ERL, ERR, ERRS$, EXP(), FIX(), FN*a*(), FRE(), HEX$(), INKEY$, INP(), INPUT$(), INSTR(), INT(), LEFT$(), LEN(), LOC(), LOF(), LOG(), LPOS(), MEM, MID$(), MKD$(), MKI$(), MKS$(), OCT$(), PEEK(), POS(), RIGHT$(), RND(), ROW(), SGN(), SIN(), SPACE$(), SPC(), SQR(), STR$(), STRING$(), TAB(), TAN(), TIME$, USR*n*(), VAL(), VARPTR()

Operators (23)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR, XOR, EQV, IMP

ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $

arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()

The Model 4 was introduced on April 26, 1983 and Tandy licensed a new Disk BASIC from Microsoft. It is a port of a subset of the Microsoft BASIC-80 5.21 1981 for CP/M that was modified for the TRS-80

RSDOS 6 and the Model 4 hardware. In addition, the Model 4 could operate in a Model III mode that also included the unmodified Model III BASIC 14K 1980.

All Model 4 Disk BASICs require space delimiters for keywords, have four data types and support variable names upto 40 character significant; theoretically allowing 1.5E+63 unique scalar variables and 1.5E+63 unique arrays *[see Microsoft BASIC-80 5.21 1981 for details]*.

All Disk BASICs for the TRS-80 Model 4 in bank-switching no ROM mode, eg {TRSDOS 6.2.1AR, LS-DOS 6.3.1H, DOSPLUS IV 1.0, DRI CP/M Plus 1.0, MM CP/M 2.2 2.32M ...}, are a modified port of the Microsoft BASIC-80 5.21 code base.

TRS-80 Model 4 Disk BASIC 1.1.2 24K 1990 for LS-DOS 6.3.1H

In June 1988, Misosys Inc acquired Logical Systems Inc, the original author of TRSDOS 6 and in 1990, MSI released the final update to the TRSDOS 6.2.1AR as the LS-DOS 6.3.1H with the final update of the TRS-80 Model 4 Disk BASIC 1.1.2 27K, which was mostly bug fixes.

TRS-80 Model 4 Microsoft BASIC-80 5.21 24K 1981 for CP/M
Digital Research Inc CP/M Plus BIOS 1.0 1984
or Montezuma Micro CP/M 2.2 BIOS 2.32 1984

[see Microsoft BASIC-80 5.21 1981 for details]

Step 8: 1984 Other TRS-80 BASICs

TRS-80 Model II Disk BASIC 1.2 18K 1983 for TRSDOS-II 4.3

Statements/Commands (70)

? , ' , AUTO, CLEAR, CLOSE, CLS, CONT, DATA, DEF FN*a*, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERASE, ERROR, FIELD, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, KILL, LET, LINE INPUT, LINE INPUT#, LIST, LLIST, LOAD, LPRINT, LPRINT USING, LSET, MERGE, MID$()=, NAME, NEW, NEXT, ON ERROR, ON GOSUB, ON GOTO, OPEN, PRINT, PRINT USING, PRINT#, PRINT# USING, PRINT@, PRINT@(), PUT, RANDOM, READ, REM, RENUM, RESTORE, RESUME, RETURN, RSET, RUN, SAVE, STOP, SWAP, SYSTEM, SYSTEM"*fff*", TROFF, TRON
variations due to optional space delimiters, eg GOTO, GO TO, DEFDBL, DEF DBL, etc

Functions (56)

&, &H, &O, ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), CVD(), CVI(), CVS(), DATE$, EOF(), ERL, ERR, ERR$, EXP(), FIX(), FN*a*(), FRE(), HEX$(), INKEY$, INPUT$(), INSTR(), INT(), LEFT$(), LEN(), LOC(), LOF(), LOG(), MEM, MID$(), MKD$(), MKI$(), MKS$(), OCT$(), POS(), RIGHT$(), RND(), ROW(), SGN(), SIN(), SPACE$(), SPC(), SQR(), STR$(), STRING$(), TAB(), TAN(), TIME$, USR*n*(), VAL(), VARPTR()

Operators (23)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR, XOR, EQV, IMP
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $
arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()
[see Microsoft BASIC-80 4.51 1977 for variable names rules, maximum variables and arrays]

Licensed from Microsoft and released with the original Model II TRSDOS in 1979, based on the Microsoft BASIC-80 4.46 1977 and few BASIC-80 5.1 1978. Few interesting facts: lacks POKE, INP(), OUT, PEEK() per Tandy's request to discourage hacking; defaulted to 0 file buffers while most other BASICs defaulted to 3 file buffers; and the Model II had a cassette port built in but Disk BASIC supports disk I/O exclusively.

Disk BASIC 1.2 1981 initially included with TRSDOS 2.0a and continued to be included, unchanged, upto Disk BASIC 1.2 1983 included with TRSDOS-II 4.3 1983.

TRS-80 Model II Microsoft BASIC-80 5.21 24K 1981 for CP/M
Digital Research Inc CP/M Plus BIOS 1.1 1984
or Pickles & Trout CP/M 2.2 BIOS 2.2mH 1985

[see Microsoft BASIC-80 5.21 1981 for details]

TRS-80 Model 16B/16 TRS-XENIX MBASIC 1.2.0 51K 1984

Statements/Commands (89)

?, ', APPEND, AUTO, BREAK, CALL, CHAIN, CLEAR, CLOSE, CLS, COMMON, CONT, DATA, DATE$=, DEF FN*a*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERASE, ERROR, FIELD, FILES, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, KILL, LET, LINE INPUT, LINE INPUT#, LIST, LLIST, LOAD, LOCK, LPRINT, LPRINT USING, LSET, MERGE, MID$()=, NAME, NEW, NEXT, ON BREAK, ON ERROR, ON GOSUB, ON GOTO, OPEN, OPTION BASE, OUTPUT, POKE, PRINT, PRINT USING, PRINT#, PRINT# USING, PUT, RANDOM, RANDOMIZE, READ, REM, RENUM, RESTORE, RESUME, RETURN, RSET, RUN, SAVE, SHELL, STOP, SWAP, SYSTEM, TIME=, TROFF, TRON, UNLOCK, WAIT, WEND, WHILE, WIDTH, WIDTH LPRINT, WRITE, WRITE#
requires at least a single space as a keyword delimiter

Functions (56)

&, &H, &O, ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), CVD(), CVI(), CVS(), DATE$, EOF(), ERL, ERR, EXP(), FIX(), FN*a*(), FRE(), HEX$(), INKEY$, INPUT$(), INSTR(), INT(), LEFT$(), LEN(), LOC(), LOF(), LOG(), LPOS(), MEM, MID$(), MKD$(), MKI$(), MKS$(), OCT$(), PEEK(), POS(), RIGHT$(), RND(), SGN(), SHELL(), SIN(), SPACE$(), SPC(), SQR(), STR$(), STRING$(), TAB(), TAN(), TIME$, VAL(), VARPTR()

Operators (23)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR, XOR, EQV, IMP
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $

arrays default to single precision real () or ![], double precision real arrays #[], integer arrays %[], string arrays $[]
[see Microsoft BASIC-80 5.21 1981 for variable names rules, maximum variables and arrays]

Licensed from Microsoft, a 16bit Motorola 68000 port of the Microsoft BASIC-80 5.21 1981 with added support for the TRS-XENIX and a big-endian adaption of Paul Allen's BCD floating point format *[same as the Model 100 BCD floating points format]*.

TRS-80 Model 100 BASIC 1.1 21K 1983

Statements/Commands (70)

? , ' , AUTO, CLEAR, CLOSE, CLS, CONT, DATA, DEF FN*a*, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, EDIT, END, ERASE, ERROR, FIELD, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, KILL, LET, LINE INPUT, LINE INPUT#, LIST, LLIST, LOAD, LPRINT, LPRINT USING, LSET, MERGE, MID$[]=, NAME, NEW, NEXT, ON ERROR, ON GOSUB, ON GOTO, OPEN, PRINT, PRINT USING, PRINT#, PRINT# USING, PRINT@, PRINT@[], PUT, RANDOM, READ, REM, RENUM, RESTORE, RESUME, RETURN, RSET, RUN, SAVE, STOP, SWAP, SYSTEM, SYSTEM"*fff*", TROFF, TRON
variations due to optional space delimiters, eg GOTO, GO TO, DEFDBL, DEF DBL, etc

Functions (56)

&, &H, &O, ABS[], ASC[], ATN[], CDBL[], CHR$[], CINT[], COS[], CSNG[], CVD[], CVI[], CVS[], DATE$, EOF[], ERL, ERR, ERR$, EXP[], FIX[], FN*a*[], FRE[], HEX$[], INKEY$, INPUT$[], INSTR[], INT[], LEFT$[], LEN[], LOC[], LOF[], LOG[], MEM, MID$[], MKD$[], MKI$[], MKS$[], OCT$[], POS[], RIGHT$[], RND[], ROW[], SGN[], SIN[], SPACE$[], SPC[], SQR[], STR$[], STRING$[], TAB[], TAN[], TIME$, USR*n*[], VAL[], VARPTR[]

Operators (23)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR, XOR, EQV, IMP
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $
arrays default to single precision real () or ![], double precision real arrays #[], integer arrays %[], string arrays $[]
[see Microsoft BASIC-80 4.51 1977 for variable names rules, maximum variables and arrays]

Licensed from Microsoft, a subset of the Microsoft EXTENDED BASIC 4.51 16K 1977 with added support for the Intel 8085. It earned the honor of being the last product primarily developed by Bill Gates, while Paul Allen implemented the Binary Coded Decimal floating point format *[per Tandy's request, to reduce rounding errors, and improve the financial calculations accuracy]*. The Model 100 Disk Video Interface included Disk BASIC 1.1 25K 1984 which added 3 statements: DSKO$, LFILES, WIDTH; and 3 functions: DSKI$[], LOC[], LOF[].

TRS-80 CoCo Disk Extended Color BASIC 1.1 24K 1983

Statements/Commands (94)

? , ' , AUDIO, BACKUP, CIRCLE, CLEAR, CLOAD, CLOADM, CLOSE, CLS, COLOR, CONT, COPY, CSAVE, CSAVEM, DATA, DEF FN*a*, DEFUSR*n*, DEL, DIM, DIR, DLOAD, DOS, DRAW, DRIVE, DSKI$, DSKINI, DSKO$, EDIT, END, EXEC, FIELD, FILES, FOR-TO-STEP, GET, GOSUB, GOTO, IF-THEN-ELSE,

INPUT, INPUT#, KILL, LET, LINE, LINE INPUT, LIST, LLIST, LOAD, LOADM, LSET, MERGE, MID$[]=, MOTOR, NEW, NEXT, ON GOSUB, ON GOTO, OPEN, PAINT, PCLEAR, PCLS, PCOPY, PLAY, PMODE, POKE, PRESET, PRINT, PRINT USING, PRINT#, PRINT# USING, PRINT@, PSET, PUT, READ, REM, RENAME, RENUM, RESET, RESTORE, RETURN, RSET, RUN, SAVE, SAVEM, SCREEN, SET, SKIPF, SOUND, STOP, TIMER=, TROFF, TRON, UNLOAD, VERIFY, WRITE
variations due to optional space delimiters, eg GOTO, GO TO, DEFFN*a*, DEF FN*a*, etc

Functions (44)

&, &H, &O, ABS[], ASC[], ATN[], CHR$[], COS[], CVN[], EOF[], EXP[], FIX[], FN*a*[], FREE[], HEX$[], INKEY$, INSTR[], INT[], JOYSTK[], LEFT$[], LEN[], LOC[], LOF[], LOG[], MEM, MID$[], MKN$[], PEEK[], POINT[], POS[], PPOINT[], RIGHT$[], RND[], SGN[], SIN[], SQR[], STR$[], STRING$[], TAB[], TAN[], TIMER, USR*n*[], VAL[], VARPTR[]

Operators (18)

( ), ^, unary + -, * /, + -, < <= = <> >= >, NOT, AND, OR
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (2, 2)

variables default to single precision real, string $
arrays default to single precision real [], string arrays $[]

The TRS-80 Color Computer was announced on July 31st, 1980 and shipped during October 1980. It featured the Motorola MC6809E and was available with multiple BASICs licensed from Microsoft: the Color BASIC 1.0 8K 1980, the Extended Color BASIC 1.0 1980 and the Disk Extended Color BASIC 1.0 1981. The combined ECB+CB 16K 1980 were an MC6800/MC6809 port of a subset of the Microsoft EXTENDED BASIC-80 4.51 16K 1977 with added support for the Color Computer graphics and sounds. The optional disk controller cartridge included the DISK EXTENDED COLOR BASIC 1.0 8K 1981 (*aka RSDOS*); combined with ECB+CB 16K would be a subset of the Microsoft BASIC-80 4.51 18K 1977.

Supports two data types for scalar variables and two data types for multi-dimension arrays, namely single precision floating points (default) and strings. Variable names are upto 2 character significant. Thus allowing 1,924 unique scalar variables and 1,924 unique arrays. Floating point reals default to a single precision 5byte in big-endian format.

[Here is how I derived the maximum variables, please feel free to correct my misunderstanding]
2char significant (1st char A-Z, 2nd char A-Z or 0-9)
26 A-Z + (26 A-Z * (((26 A-Z) + (10 0-9)) ^1)) = 962 unique names
962 * 2 data types = 1,924 unique scalar variables, and 962 * 2 data types = 1,924 unique arrays

The Color Computer 3 was introduced in 1986 and included the Super Extended Color BASIC 2.0 1986. To avoid licensing a new BASIC from Microsoft, the CoCo3 included the unmodified EXTENDED COLOR BASIC 16K (ECB 1.1 8K + CB 1.2 8K) and a new 16K ROM by Microware Systems Corp. While booting, the Microware 16K ROM copies the original ECB 1.1 8K + CB 1.2 8K to RAM and updates them to SECB 2.0 with an additional 8K in RAM, for a total of 24K in RAM. In other words, the 32K ROM (ECB 1.1 8K+ CB 1.2 8K + SECB 2.0 16K) are loaded, updated and reduced to 24K RAM. Similarly, if the RSDOS 1.1 floppy controller is plugged in, it is loaded into RAM and updated for a combined Super Disk Extended Color BASIC 2.1 32K 1986 in RAM (or SDECB 2.0 32K 1986 for the earlier DECB 1.0 1981).

SDECB 2.1/SDECB2.0/SECB 2.0 *all* add 22 statements: ATTR, HBUFF, HCIRCLE, HCLS, HCOLOR, HDRAW, HGET, HLINE, HPAINT, HPRINT, HPUT, HRESET, HSCREEN, HSET, HSTAT, LOCATE, LPOKE, ON BRK, ON ERR, PALETTE, PALETTE RGB, WIDTH; and 5 functions: BUTTON(), ERLIN, ERNO, HPOINT(), LPEEK().

TRS-80 MC-10 MicroColor BASIC 1.0 8K 1983

Statements/Commands (37)

? , CLEAR, CLOAD, CLOAD*, CLS, CONT, CSAVE, CSAVE*, DATA, DIM, END, FOR-TO-STEP, GOSUB, GOTO, IF-THEN, INPUT, LET, LIST, LLIST, LPRINT, NEW, NEXT, ON GOSUB, ON GOTO, POKE, PRINT, PRINT@, READ, REM, RESET, RESTORE, RETURN, RUN, SET, SKIPF, SOUND, STOP
variations are the same token, eg GOTO or GO TO becomes GOTO; ? or PRINT becomes PRINT

Functions (23)

ABS(), ASC(), CHR$(), COS(), EXP(), INKEY$, INT(), LEFT$(), LEN(), LOG(), MEM, MID$(), PEEK(), POINT(), RIGHT$(), RND(), SGN(), SIN(), SQR(), STR$(), TAB(), TAN(), VAL()

Operators (18)

( ), ^, unary + -, * /, + -, < <= = <> >= >, NOT, AND, OR
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (2, 2)

variables default to single precision real, string $
arrays default to single precision real (), string arrays $()
[see CoCo DECB 1.1 24K 1983 for variable names rules, maximum variables and arrays]

The TRS-80 MicroColor Computer MC-10 was released in 1983 as a response to ~ $100 microcomputer market, eg Sinclair Z81, and it included the MicroColor BASIC 8K 1983. Licensed from Microsoft as a Motorola MC6800/MC6803 port of a subset of the Microsoft EXTENDED BASIC-80 4.51 16K 1977. It is a functional subset of the Color Computer COLOR BASIC 1.1 1981 combined with few features from the EXTENDED COLOR BASIC 1.0 1980.

Included additional undocumented 2 statements: CLOADM, EXEC; and 1 function: VARPTR().

Step 9: 1988 Beyond the TRS-80 BASICs

Microsoft GW-BASIC 3.23 79K 1988

Statements/Commands (128)

? , ', AUTO, BEEP, BLOAD, BSAVE, CALL, CHAIN, CHDIR, CIRCLE, CLEAR, CLOSE, CLS, COLOR, COM, COMMON, CONT, DATA, DATE$=, DEF FN*a*, DEF SEG, DEF USR*n*, DEFDBL, DEFINT, DEFSNG, DEFSTR, DELETE, DIM, DRAW, EDIT, END, ENVIRON, ERASE, ERROR, FIELD, FILES, FOR-TO-STEP, GET, GET#, GOSUB, GOTO, IF-THEN-ELSE, INPUT, INPUT#, IOCTL, KEY, KILL, LET, LINE, LINE INPUT, LINE INPUT#, LIST, LLIST, LOAD, LOAD"CAS1:, LOCATE, LOCK, LPRINT, LPRINT USING, LSET, MERGE, MID$=, MKDIR, MOTOR, NAME, NEW, NEXT, ON COM, ON ERROR, ON GOSUB, ON GOTO, ON KEY, ON PEN, ON PLAY, ON STRIG, ON TIMER, OPEN, OPEN "COM*n*", OPTION BASE, OUT, PAINT, PALETTE, PALLETE USING, PCOPY, PEN, PLAY, POKE, PRESET, PRINT, PRINT USING, PRINT#, PRINT# USING, PSET, PUT, PUT#, RANDOMIZE, READ, REM, RENUM, RESET, RESTORE, RESUME, RETURN, RMDIR, RSET, RUN, SAVE, SAVE"CAS1:, SCREEN, SHELL, SOUND, STOP, STRIG,

SWAP, SYSTEM, TIME$=, TROFF, TRON, UNLOCK, VIEW, VIEW PRINT, WAIT, WEND, WHILE, WIDTH, WINDOW, WRITE, WRITE#
requires at least a single space as a keyword delimiter

Functions (71)

&, &H, &O, ABS(), ASC(), ATN(), CDBL(), CHR$(), CINT(), COS(), CSNG(), CSRLIN, CVD(), CVI(), CVS(), DATE$, ENVIRON$(), EOF(), ERDEV, ERDEV$, ERL, ERR, EXP(), EXTERR(), FIX(), FN*a*(), FRE(), HEX$(), INKEY$, INP(), INPUT$(), INSTR(), INT(), IOCTL$(), LEFT$(), LEN(), LOC(), LOF(), LOG(), LPOS(), MID$(), MKD$(), MKI$(), MKS$(), OCT$(), PEEK(), PEN(), PLAY(), PMAP(), POINT(), POS(), RIGHT$(), RND(), SCREEN(), SGN(), SIN(), SPACE$(), SPC(), SQR(), STICK(), STR$(), STRIG(), STRING$(), TAB(), TAN(), TIME$, TIMER, USR*n*(), VAL(), VARPTR(), VARPTR$()

Operators (23)

( ), ^, unary + -, * /, \, MOD, + -, < <= = <> >= >, NOT, AND, OR, XOR, EQV, IMP
ordered from highest to lowest precedence, commas separate each precedence group

Data Types (4, 4)

variables default to single precision real or !, double precision real #, integers %, string $
arrays default to single precision real () or !(), double precision real arrays #(), integer arrays %(), string arrays $()
[see Microsoft BASIC-80 5.21 1981 for variable names rules, maximum variables and arrays]

On August 12th, 1981 the IBM PC was released with built-in Cassette BASIC ROM, IBM PC-DOS Disk BASIC and Advanced BASIC. All three BASIC interpreters were licensed from Microsoft and they were the ultimate growth of the BASIC-80 5.21 1981 ported to the Intel 8086/8088 MPU with the added support for the IBM PC graphics and sounds. So when the OEMs licensed Microsoft first MS-DOS 1.25 in 1982, they could choose to bundle GW-BASIC 1.04 which was functionally equivalent to IBM PC Advanced BASIC without depending on the IBM PC ROM.

Obviously GW-BASIC does not execute on the 8bit TRS-80, but due to their shared ancestry, they have numerous common features with the exception of those relating to the IBM PC graphics and sounds. So I am including the GW-BASIC details for comparison purpose and for the completeness of the TRS-80 Family Pedigree.

Wrapping it up …

Microsoft Disk Extended BASIC-80 5.21 23.75K 1981

Model 4 TRSDOS 6.x, LS-DOS 6.x, DOSPLUS IV
CP/M 2.2, CP/M Plus 3.0
TRS-XENIX MBASIC (BCD)

Microsoft Disk Extended BASIC-80 4.46 18K 1977

LEVEL II, Model III/I TRSDOS Disk BASIC; Model 4 in Model III Mode *all* DOSs
Model II TRSDOS-II/TRSDOS Disk BASIC; Model 100 BASIC/Disk BASIC (BCD)
CoCo CB, ECB, DECB, SDECB; MC10 MCB

Steve Leininger's LEVEL I 4K 1977

Floating point + *Li Chen Wang's Palo Alto TINY BASIC* published in *Dr Dobbs*, May 1976

Wish all of you, *happy* computing.

BASIC Interpreters Feature Summary Table

BASIC Interpreters	Size	Version	Year	Commands, Statements	Functions	Operators	Comments
Dartmouth	10K	1964	1964	29	11	14	GE225+DN30, GE235+DN30 [a]
BASIC-PLUS	16K	1970	1970	72	48	22	DEC PDP-11, DEC PDP-10 [b,c]
Altair BASIC	12K	3.2	1975	43	30	20	Micro-Soft 1st product, v2.0 [d]
Altair BASIC	16K	4.1	1977	65	45	23	Altair EXTENDED BASIC [e]
Altair DISK	18K	4.1	1977	77	56	23	Altair DISK BASIC
MBASIC	18K	4.51	1977	77	56	23	CP/M BASIC-80 4.1 1977
MBASIC	24K	5.21	1981	84	54	23	CP/M BASIC-80 5.0 1978
Tiny BASIC	2K	1.0	1976	14	3	16	Li-Chen Wang's PATB
TRS80 LEVEL I	4K	1977	1977	27	6	16	Basis for Model III LEVEL I [f,g]
LEVEL II	12K	1.3	1978	56	36	18	Basis for the Model III BASIC [h]
DISK BASIC	16K	2.2	1978	88	51	18	Model I TRSDOS 2.3
M4 Disk BASIC	24K	1.1.1	1985	83	59	23	Model 4 TRSDOS 6.2.1AR
M2 BASIC	18K	1.2	1983	70	56	23	Model II TRSDOS-II 4.3
XENIX MBASIC	51K	1.2.0	1984	89	56	23	Model II TRS-XENIX 1.3.5
M100 BASIC	17K	1.1	1983	87	45	23	Model 100 and 102
CoCo DECB	24K	1.1	1983	94	44	18	Color Computer 1 and 2 [i]
MC10 MCB	8K	1.0	1983	37	23	18	MicroColor MC-10 [j]
GW-BASIC	79K	3.23	1988	128	71	23	1st v1.04 1982, MS-DOS 1.25

Additional Table Comments

The BASIC feature tallies are not absolute and may on the surface appear contradictory to other source. However, the issue is a simple one, how one chooses to enumerate the language features, eg list each command, statement, function and operator. For example: PRINT and ? count them as one statement vs count them as two statements, similarly & &O &H count them as one function vs three functions, I choose the later since it accounts for each symbol accepted by the BASIC parser.

a. Dartmouth BASIC first release (May 4th, 1964) was < 4K 20bit Words, ie 4Kx20bit/8bit < 10K Bytes
b. DEC BASIC-PLUS for PDP-11 running RSTS-11 was < 8K 16bit Words, ie 8Kx16bit/8bit < 16K Bytes
c. DEC BASIC-PLUS for PDP-10 running TOPS-10 was 36bit Words, I don't know its size requirement
d. ALTAIR EXTENDED BASIC 3.2 12K December 1975 Reference Manual
e. ALTAIR EXTENDED BASIC 4.1 16K July 1977 Reference Manual
f. Model I and Model III LEVEL I were derived by Tandy from Li-Chen Wang's Tiny BASIC 2K
g. Model III LEVEL I BASIC ROM 4K 1980 adds printer support, 2 statements: LLIST and LPRINT
h. Model III BASIC ROM was 14K 1980, 2K adds lowercase + real clock support, 1 function: TIME$
i. CoCo3 had SDECB 2.1 1986, derived from DECB 1.1 1983, modified by Microware Systems Corps. Adds 22 statements: ATTR, HBUFF, HCIRCLE, HCLS, HCOLOR, HDRAW, HGET, HLINE, HPAINT, HPRINT, HPUT, HRESET, HSCREEN, HSET, HSTAT, LOCATE, LPOKE, ON BRK, ON ERR, PALETTE, PALETTE RGB, WIDTH; and 5 functions: BUTTON(), ERLIN, ERNO, HPOINT(), LPEEK().
j. Undocumented adds 2 statements: CLOADM, EXEC; and 1 function: VARPTR()

Dedication

To Matthew, our son: A young technological mind that always yearned for innovation, never timid from learning new ideas, whose generous soul fueled his willingness to share all what he had with anyone in need. The loving memory of his short lived yet full life, lives with us every day. Giving me the courage to share what I learned with others so that we all can help one another in our common journey.

Acknowledgments

Vernon B Hester, a true legend of the original TRS-80 era, for sharing his remarkable suite of TRS-80 software (ESOTERIC 3, MULTIDOS 6, MULTIDOS 5, ZEUS, ASTROCAL, VFU, ...), his plethora of BASIC (ESOBASIC, CBASIC, BASICH, CBASICH, SUPERBASIC, BOSS BASIC, ...) and for generously sharing his insight and clarifying MBASIC 5.x basis for Model 4 while MBASIC 4.46 the basis for all Model I/III.

George and *Peter Phillips*, the amazing twins who gifted us their emulator, the *trs80gp*, which I used extensively to validate some of the BASIC dialects of every TRS-80 model (LEVEL I, LEVEL II, DOSs, CPM, XENIX, DECB); George's LEVEL 2.5 BASIC article/program; and for their kindness to actually read and respond to emails with in depth details.

Alan Wallace for generously sharing his BASIC knowledge, especially for his trs80gp Facebook Group posting on October 21, 2023 discussing the "IF cond, GOSUB line number" and his kind emails that followed. He ignited my interest to uncover *what are the various valid forms of IF-THEN-ELSE*, leading to *what is the origin of BASIC*.

Dusty Miller for editing and publishing the *TRS&BIT* newsletter, *the* great medium to quest our thirst for TRS-80 8bit and 16bit microcomputers and the longing for information covering the simpler computing platforms unbounded by layers of virtualization while encapsulated in myriad of abstractions.

Further Readings (ordered by the BASIC timeline steps)

The Kiewit Computation Center and the Dartmouth Time-Sharing System. Dartmouth College. 1971.

BASIC version 2 manual (PDF). Dartmouth College. October 1964.

Time Shared BASIC Manual (PDF). Hewlett Packard. August 1969.

BASIC-PLUS Manual (PDF). DEC. 1972.

Altair BASIC Reference Manual (PDF). MITS. 1975.

BASIC-80 CP/M Reference Manual (PDF). Microsoft. 1979.

ECMA-55 Minimal BASIC Standard (PDF). ECMA. January 1978.

January 1975 issue of *Popular Electronics* was published on November 29, 1974.

Thomas Kurtz Interview with Dartmouth College, July 2, 2002.

Nathan Brookwood (aka Nathan Teichholtz) Interview with DEC Connection, July 12, 2022.

Ed Roberts Interview with *Historically Brewed* magazine, 1996.

Bill Gates Interview with "National Museum of American History", 1993

Allen, Paul. "The Idea Man". Penguin. 2011.

Wallace, James. "Hard Drive: Bill Gates and the Making of the Microsoft Empire". Wiley. 1992.

2022 POWER SUPPLIES
FOR THE TRS-80 MODEL 3 / MODEL 4/4D/4P



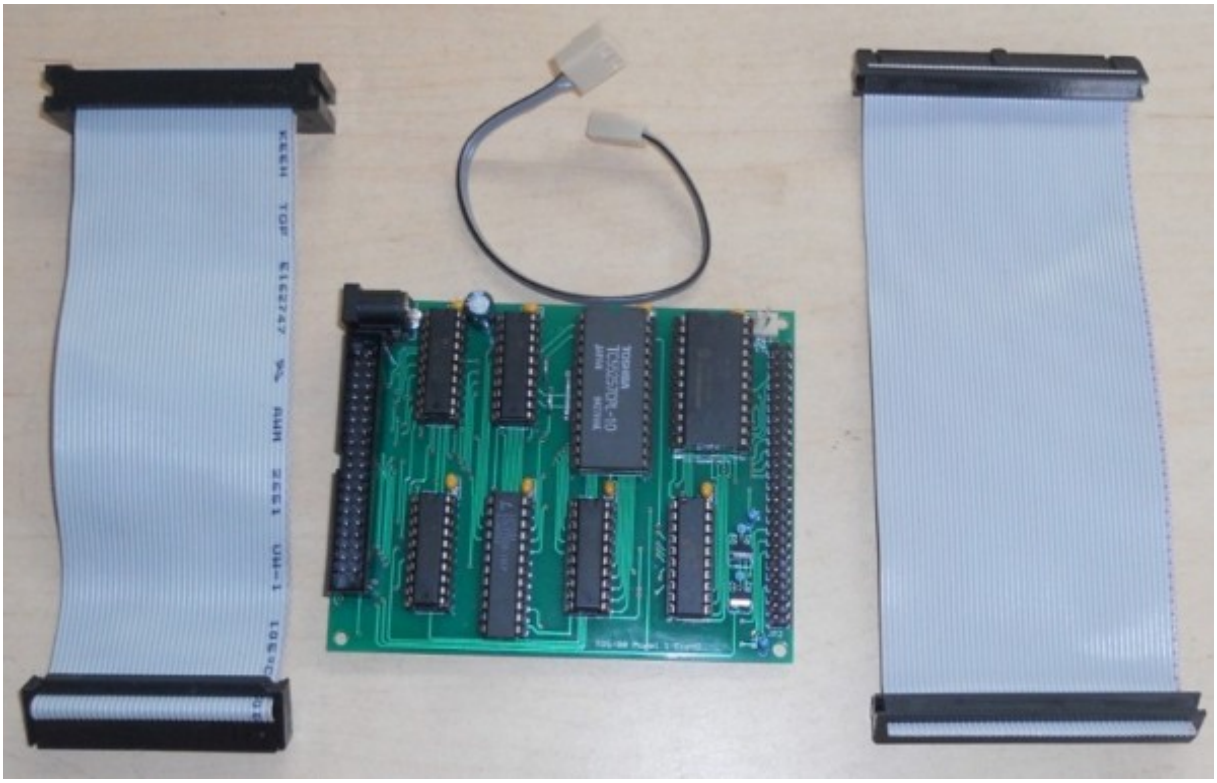
* Uses modern day MEAN WELL power supply units
* 50w for Model 3 version and 65w for the Model 4 version
* The Model 4 version uses a SEPARATE +12V POWER SUPPLY (prevents video "spasm")
* No more RIFA CAP explosions
* No more COLD solder joints
* Units are DIRECT REPLACEMENTS for the original power supplies 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)

“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”

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://members.iinet.net.au/~ianmav/trs80/



E.T. FONEHUME

HI DUSTY. JUST TO KICK OFF THE 18TH YEAR, HERE'S A ONE-LINER TO CREATE A 5X5 MAGIC SQUARE, USING 25 CONSECUTIVE NUMBERS. IT WORKS WITH POSITIVE AND NEGATIVE INTEGERS. THE COLUMNS, ROWS, DIAGONALS AND THE MIDDLE NUMBER, PLUS THE 4 CORNERS ALL ADD UP TO THE 'MAGIC' NUMBER. JUST ENTER YOUR START NUMBER.

```
0 DIMA(25):DATA10,23,6,19,2,3,11,24,7,15,16,4,12,20,8,9,17,0,13,
21,22,5,18,1,14:INPUT"START NO.";S:FORX=1TO25:READN:A(X)=N+S:NEX
T:PRINT:PRINT:FORX=1TO25STEP5:PRINTUSING"#######";A(X);A(X+1);A
(X+2);A(X+3);A(X+4):NEXT:PRINT:FORX=1TO5:T=T+A(X):NEXT:PRINTT
```

```
READY
>RUN
START NO.? 19

     29     42     25     38     21
     22     30     43     26     34
     35     23     31     39     27
     28     36     19     32     40
     41     24     37     20     33

 155
```



How to create a WAV file?

There are people out there, who need this way of creating a disk, as they don't possess a floppy drive at their PC.
I was asked how I get from a DMK image to a WAV file. That I will describe further down ...
(Thanks to Andre in Japan and Andrés in Boston/USA)

A Java program
Analysing a DMK file is no rocket science. At Ira Goldklang's Website there is a outstanding description of the details of the DMK format: https://www.trs-80.com/wordpress/tips/formats/
As a ordinary software developer I am lazy and implemented the analysis in a platform independent Java program:
dmk2cmd.v5.6.zip
To avoid errors during transmission, I created a ZIP archive.
Syntax
DMK2CMD 5.6
Converts a DMK file into header and tracks appropriate to BOOTSTRAP
-v = View only (no files are created)
usage: DMK2CMD <DMK-file> [-v]

What is this for?
It creates a number of CMD files from a DMK file. The header.cmd is combining all relevant info to create a floppy. Further on the content of each track is stored in 00.cmd to xx.cmd.
A batch file
Well there are some steps required, until the WAV file is ready

```
file="MyDisk.dmk"
fullname=`basename "$file"`
name="${fullname%.*}"
WAVType="500.wav"
java -jar ~/HausPCData/Software/Java/dmk2cmd.jar
"$name.dmk"
HighSpeed=$?
if [ $HighSpeed -eq "1" ]; then
  WAVType="1500.wav"
fi
trld header.cmd header.$WAVType
if [ $WAVType == "1500.wav" ]; then
  sox header.$WAVType silence-long3.wav header.wav
else
  sox header.$WAVType silence-long1.wav header.wav
fi
rm header.$WAVType
rm header.cmd
for fn in [0-9][0-9].cmd; do
  number="${fn%.*}"
  trld $number.cmd $number.$WAVType
  trld $number.cmd $number.$CASType
  if [ $WAVType == "1500.wav" ]; then
    sox $number.$WAVType silence3.wav $number.wav
  else
    sox $number.$WAVType silence1.wav $number.wav
  fi
  rm $number.cmd
  rm $number.$WAVType
done
sox header.wav $(ls [0-9][0-9].wav | sort -n)
"$name.$WAVType"
rm header.wav
rm [0-9][0-9].wav
```

Now I am about to explain some things:
Beside my Java program I use some other programs

trld from George Phillips from Canada (http://48k.ca/trld.html)
With this tool I convert the CMD files to WAV files

sox, a Linux tool (https://linux.die.net/man/1/sox)
With this tool I combine the WAV files

basename, a Linux tool (https://www.man7.org/linux/man-pages/man1/basename.1.html)
With this I extract the name of the DMK file

Now the attentive reader will notice, that I use four WAV files
silence1.wav
silence-long1.wav
silence3.wav
silence-long3.wav
They can be created quite easy with sox
sox -n -r 22050 -b 16 -c 1 -L silence1.wav trim 0.0 6.0
sox -n -r 22050 -b 16 -c 1 -L silence-long1.wav trim 0.0 16.0
sox -n -r 22050 -b 16 -c 1 -L silence3.wav trim 0.0 20.0
sox -n -r 22050 -b 16 -c 1 -L silence-long3.wav trim 0.0 55.0

Now we encounter the question:
"What the heck is he doing there?"

The filename is extracted.
The DMK file is splited and the transmission speed is detected at the same time.
500 bit/s for Model I (10 sectors single density)
1500 bit/s for Model III (18 sectors double density)
The header and each track a preceding silence is added.
All parts are combined to one large WAV file.
All temporary files are deleted.

That's it
As I said, no rocket science!
With this way even data disks can be transmitted.

Software Tools in BASIC: Episode 3
OPEN for Business

Vince Otten

If you've been following this series of articles, you know that I'm implementing a version of Kernighan & Plauger's excellent "Software Tools" in BASIC: specifically, for TRS-80 Model I running TRSDOS 2.3. I've written two resources: a "shell" program, SH/BAS, which allows me to type in the name of a program along with details of where its files should come from and go to and its options; and an API, an "application programming interface", which I call SKELETON/BAS: it's the "skeleton" in which you write your "software tool". You don't have to worry about OPENing or CLOSEing standard files, because the "skeleton" automatically sets up all these details before running your program, and it "cleans up" after you're done.

The original "Software Tools" were written in Ratfor, a version of Fortran that looks a bit like C. Here's what a program to copy the standard input to the standard output looks like in that language:

```
# copy – copy input characters to output
    integer getc
    integer c

    while (getc( c) != EOF)
        call putc( c)
    stop
    end
```

The interesting part, the actual code that does work, is

```
    while (getc( c) != EOF)
        call putc( c)
```

Using SKELETON/BAS as its "outline", I wrote the following in BAISC:

```
'
    "WHILE (GETC( C) != EOF)"

40000 GOSUB 2100: IF NOT VR%(0) THEN GOTO 50000
40009 '
          "CALL PUTC( C)"

40100       VA$(0)= VR$(0): GOSUB 2300:
            GOTO 40000

50000 '
```

It's a lot uglier than the Ratfor code. But it does the same thing. There's one GOSUB for "getc()", "GOSUB 2100", and one for "putc()", "GOSUB 2300". The character "c" that's returned by "getc(c)" is "VR$(0)" ("Value of Return 0"), and just like "c" is supplied as an argument to "putc(c)", so VR$(0) is assigned to subroutine 2300 as "VA$(0)" ("Value of Argument 0"). Ugh. But it works.

This ugliness brings up the question: why would you even do it in BASIC? Programming masochism?

Perhaps. But I'm doing it mostly because 1. it's a challenge; and 2. I want to use the BASIC code as proof-of-concept, and as a prototype for Z-80 Assembly development.

So: back to BASICs. In the last article, I presented the code to OPEN not only regular BASIC sequential and random-access files, but also those specifically for "getc()" and "putc()", which read or write 1 character/byte at a time. There was one type of OPEN which I had yet to discuss, and that's "OPEN for Append". In standard TRSDOS 2.3 BASIC, you can't do this: you can't OPEN a previously PRINTed and CLOSEd sequential file and add more to the end. This means that we have to OPEN the file as "R"andom-access, find out where its last byte is, convert that to a record # to GET, determine how far into that record we've already written, and prepare to FIELD the previously PRINTed part, skip over it, and prepare to FIELD the next 1 byte for the newest character to "putc()" there. When we're done, we have to "manually" update the file's DCB ("Disk Control Block") by POKEing in new "last record" and "last byte" values before we CLOSE the file. (I seem to recall that there's at least one DOS that provided this Append feature in BASIC: perhaps NEWDOS/80 or LDOS.)

Here's the option to OPEN a file for "Append" in pseudocode:

```
        case APPEND_MODE:
            /* '7F. OPEN "A"'  */
            AppendOpen( file#);
            /* "GOSUB 60830: IF VR%(0) THEN RETURN"     60660a     */
            if (!Result) {
                fOpenErrorReport( ErrorFname$,
                    "'A'PPEND MODE: CAN'T OPEN", file#);
                }
```

As usual, once I've converted my pseudocode outline into actual BASIC code, that becomes a comment just below the corresponding line. Well, that's easy enough: just call "AppendOpen()" and, if that's successful ["VR%(0)" is set to -1], the file's OPEN; otherwise report the error. We just have to see what "AppendOpen()" looks like...

```
AppendOpen( file#, filename$) {  /* set up default file params for an "A"ppend OPEN */
ErrorType$= "'A'PPEND MODE: CAN'T OPEN";
    if (ERROR( OPEN "R", file#, filename$))
        return FAIL;
    if (tcc= TotalBytes( file#) > 0) {
        /* existing non-empty file*/
        tcc++;  /* bump past current last char/byte in file    */
        OpenType[file#]= APPEND_MODE;
        TotalCharCount[file#]= tcc;
        CurrentCharCount[file#]= tcc;
        ConvertCharCount( tcc);
        RecordCount[file#]= VR!(0);
        CharPos[file#]= VR%(2);
        GET file#, RecordCount[file#];
        }
    else {
        /* empty file: open "O" instead    */
        CLOSE file#;
        if ERROR( OPEN "O", file#, filename$))
            return FAIL;
        OpenType[file#]= OUTPUT_MODE;
        CurrentCharCount[file#]= 0;
        RecordCount[file#]= 1;
        TotalCharCount[file#]= 0;
        ResetDiskBuffer();
        }
    VR%(1)= file#;
    return SUCCESS;
    }
```

Oh. There's a bit more to it.
I've left out the corresponding BASIC code here to make the pseudocode more readable. I'll show that code as I review, line-by-line, below. But first, here's the entry for "AppendOpen()" in the "Skeleton Usage Guide":

```
AppendOpen()     Open VA%(0) for Append Mode
------------------
Usage:          GOSUB 60830
Input:          VA%(0)     File#
                VA$(0)     filename$
Result:         VA%(0)     Success/Fail (-1/0)
Returns:        VA%(1)     Successfully OPENed file #
Temp Vars:      X3%        used in ConvertCharCount()
                X7!        TotalCharCount[file#]
Used in:        fOpen()
Uses:           ConvertCharCount()
Note:           Pseudocode in "Skeleton in PseudoCode"
```

So: on to the first 4 lines of pseudocode, now with the corresponding BASIC:

```
AppendOpen( file#, filename$) {  /* set up default file params for an "A"ppend OPEN v2
            VA%(0) VA$(0)                                                          */
    ErrorType$= "'A'PPEND MODE: CAN'T OPEN";
    /* 'VA$6)= "'A'PPEND MODE: CAN'T OPEN"'                              60830  */
    if (ERROR( OPEN "R", file#, filename$))
    /* 'OPEN "R", X9%, VA$(0): GOTO 60834"                               60832  */
        return FAIL;
        /* "VR%(0)= 0: RETURN' FILE NOT FOUND: ERROR EXIT"               60833  */
```

The arguments “file#” and “filename$” will be found in the BASIC “Value of Argument” array elements VA%(0) and VA$(0), respectively. We first try to OPEN the file in “R”andom mode. This is done so as not to destroy vital End Of File information, since OPENing it in sequential “O”utput mode would reset the EOF stats to 0. If the file can’t be OPENed (if the disk is full, or the filespec is incorrect, perhaps naming an empty drive), we’ll trap that error in line 60833 by setting the Result in VR%(0) to “fail’/0 and simply exiting.

Now, we’re going to see how many bytes are currently in this file. If it’s empty, it must be a new file, so we might as well CLOSE it and reOPEN it in “O”utput mode. Otherwise, we continue to use “R”andom access to get to the final byte in this file.

```
if (tcc= TotalBytes( file#) > 0) {
/* "X7!= FN DX!( X9%): IF X7! = 0 THEN 60836"                    60834     */
    /* existing non-empty file*/
    tcc++;          /* bump past current last char/byte in file    */
    /* "X7!= X7! + 1:"                                           60835a    */
```

"Total Char Count", "tcc" in pseudocode, will be saved in the BASIC temporary variable X7!. How did we get its value? The "TotalBytes()" function, FN DX!(file#), returns the total number of bytes in any open file#. (We detailed how FN DX!() works in the first installment of this series. You can find that article in Trs8bit, June 2023.)

```
    OpenType[file#]= APPEND_MODE;
    /* "SY%(X9%,1)= 6:"                                          60835b    */

    TotalCharCount[file#]= tcc;
    /* "SY!(X9%,2)= X7!"                                         60835c    */

    CurrentCharCount[file#]= tcc;
    /* "SY!(X9%,0)= X7!"                                         60835d    */
```

After setting the Open Mode for "file#" (X9%) to APPEND_MODE or "6", we save the "TotalCharCount", "tcc" into *both* the TotalCharCount slot and the CurrentCharCount slot for this file. They will get updated whenever we append more data.

```
    ConvertCharCount( tcc);
    /* "VA!(0)= X7!: GOSUB 1910:"                  1910 <-       60835e    */
```

"ConvertCharCount()" is a multi-purpose procedure. It converts a file's total character count ("tcc", stored in "TotalCharCount[file#}" / "SY!(X9%,2)") to:

a) A Record # to GET or PUT or store in the file's DCB
b) an offset in the record for the last byte to store in the DCB
c) the size of the spacer field before the character to append to the file

This subroutine, "1910", is a good example of returning multiple values as well as a result value. Here's entry for ConvertCharCount() in the Skeleton Usage Guide:

```
ConvertCharCount()   convert a file's total character count to
-------------------- a) a Record # to    - GET
                                         - PUT
                                         - store in file's DCB
                     b) an offset in record for last byte to store in DCB
                     c) the size of the spacer field before the character to
                          append
Usage:          GOSUB 1910
Input:          VA!(0)      Current Character Count: size of file opened for "A", or
                              total count of characters prior to CLOSE.
                              Stored in SY!(n,0)
Result:         VR%(0)      S/F      F indicates empty (new) file
Returns:        VR!(0)      Record #         [see "a)", above].
                                               Typically, store in SY!(n,0)
                VR%(1)      Offset           [see "b)", above].
                                               Typically, POKE into DCB
                VR%(2)      Spacer size      [see "c)", above].
                                               Typically, store in SY%(n,2)
Temp Vars:      X3%         used to calculate Offset/VR%(1) and Spacer/VR%(2)
Used in:        AppendOpen()
Note:           Pseudocode in "Skeleton in PseudoCode"
```

ConvertCharCount() is designed for two applications: here, when we OPEN a file for Append Mode, and later, when we CLOSE that file. This is what its pseudocode looks like:

```
ConvertCharCount( CharCount) {
    Result= CharCount != 0;
    if (!Result) {
        RecordNo= 0;
        Offset= 0;
        CharPos= 0;
        return;
        }
    RecordNo= Block( CharCount, 256, 1, 1);
    Offset= mod( CharCount, 256);
    CharPos= Offset;
    if (Offset == 0)
        CharPos= 256;
```

```
    Spacer= CharPos - 1;
    }
```

and here it is, broken down with the BASIC code in the comments:

```
ConvertCharCount( CharCount) {      /* convert a file's total character count to
                        VA!(0)              a) a Record # to    - GET
                                                               - PUT
                                                               - store in file's DCB
                                            b) an offset in record for last byte to store in DCB
                                            c) the size of the spacer field before the character to
                                               append                                          */
    Result= CharCount != 0;
    /* "VR%(0)= VA!(0) <> 0"                                                        1910    */
    if (!Result) {
        RecordNo= 0;
        Offset= 0;
        CharPos= 0;
        return;
        }
    /* "IF NOT VR%(0) THEN VR!(0)= 0: VR%(1)=0: VR%(2=0: RETURN"                          */
```

ConvertCharCount will only proceed if the file already contains some data. Otherwise, it will set all its results to 0, and report "failure" (VR%(0)= 0).

```
    RecordNo= Block( CharCount, 256, 1, 1);
    /* "VR!(0)= FN BL!( VA!(0), 256, 1, 1):"                                        1912    */
```

BASIC Random Access files begin with Record #1. A file that contains 257 bytes, for example, will store the first 256 in Record #1, and then the last 257th in Record #2. If we feed the Block() function FN BL!() with a "char count" value of 257 (as well as telling it that "Record numbers start with a 1, and bytes within a record are counted starting from 1", it will return "Record #" 2. This is a defined function which determines a "record number" based on:

a) The number of items in a record (ie, "maximum 255 characters in a BASIC string" or "256 bytes in a TRSDOS sector")
b) The starting count # for records (ie, 0 for BASIC array elements, 1 for TRSDOS Random Access record numbers); and

c) The starting index number within a record (ie, 0 for contents of a byte, 1 for characters in a string).

Here's its entry in the Skeleton Usage Guide:

```
BL!( IC!, BS%, BO%, IO%)      Block( ItemCount!, BlockSize%, BlockOrigin%, ItemOrigin%)
------------------------ --------  Returns Record# for total ItemCount!, based on
                                       Record size              BlockSize%
                                       Record Origin     0/1    BlockOrigin%
                                       Item Origin       0/1    ItemOrigin%
Defined in:          64037
Used in:             ConvertCharCount()
Notes:               1. Block= INT( (ItemCount - ItemOrigin) / BlockSize) + BlockOrigin;
                     2. PseudoCode in "Skeleton in PseudoCode"
Typical Usage:       In TRSDOS 2.3 for the TRS-80 Model I, disk records are numbered from
                     record #1. Also, bytes/characters in a file are counted from byte #1.
                     To determine in which record byte #257 is found, evaluate
                     FN BL!( 257, 256, 1, 1)
```

Here's the pseudocode for Block():

```
Block( ItemCount, BlockSize, BlockOrigin, ItemOrigin) {  /* Returns Record# for
                  total ItemCount, based on
                      Record size
                      Record origin   [0/1]
                      Block origin    [0/1]                */
    return int( (ItemCount - ItemOrigin) / BlockSize) + BlockOrigin;
    /* "DEF FN BL!( IC!, BS%, BO%, IO%)=
        INT( (IC! - IO%) / BS%) + BO%"                                          64037  */
```

Next, we figure out where the file's last character is within the record. In our example, if "char count" is 257, it will be at position 1 of Record #2. In pseudocode, we call that position "Offset" / X5%.

```
    Offset= mod( CharCount, 256);
    /* "X5%= FN MD!( VA!(0), 256): VR%(1)= X5%"                                  */
    CharPos= Offset;
    if (Offset == 0)
        CharPos= 256;
```

```
    /* "IF X5% = 0 THEN X5%= 256"                                          */
    Spacer= CharPos - 1;
    /* "VR%(2)= X5% - 1: RETURN"                                  1916    */
    }
```

Once we know the "Offset", it's easy to determine the positional value "CharPos" / X5% to punch into the file's DCB when we're ready to CLOSE this file. Just a little juggling: if "CharCount" turns out to be a multiple of 256, "Offset" will have been calculated as 0, but the actual position is item 256, not 0.
We can also calculate how much "space" we need before this character. For example, if the last character in a record is at byte #10, we need a "spacer" of 9 bytes before it, ie in this BASIC code example:

```
100     FIELD #1, 9 AS SP$, 1 AS C$
```

Since we've now calculated "CharPos" to be some value of 1..256, "Spacer" will correspondingly be from 0..255.

Back to AppendOpen(). We've fed the total count of characters in this file to ConvertCharCount() , and it's supplied us with the last Record # in VR!(0), and the position of the last character within that record in VR%(2). Now we just store those values in the slots for this file, and GET the last record, ready to Append more characters as needed.

```
    RecordCount[file#]= VR!(0);
    /* "SY!(X9%,1)= VR!(0):"                                  60835f    */
    CharPos[file#]= VR%(2);
    /* "SY%(X9%,2)= VR%(2):"                                  60835g    */
    GET file#, RecordCount[file#];
    /* "GET X9%, SY!(X9%,1):"                                 60835h    */
    }
    /* "GOTO 60839"                                 60839 <-  60835i    */
```

Now: what if the file we OPENed for "R"andom access actually was empty (contained 0 bytes), either because it was new or had nothing written to it yet? Why not just CLOSE it and reopen it as a sequential file?

```
else {
    /* empty file: open "O" instead   */
```

```
CLOSE file#;
/* "CLOSE X9%:"                                          60836a    */
if ERROR( OPEN "O", file#, filename$))
/* 'OPEN "O", X9%, VA$(0): GOTO 60838"                   60836b    */
    return FAIL;
    /* "GOTO 60833' ERROR EXIT"                          60837     */
OpenType[file#]= OUTPUT_MODE;
/* "SY%(X9%,1)= 2:"                                      60838a    */
CurrentCharCount[file#]= 0;
/* "SY!(X9%,0)= 0:"                                      60838b    */
RecordCount[file#]= 1;
/* "SY!(X9%,1)=1:"                                       60838c    */
TotalCharCount[file#]= 0;
/* "SY!(X9%,2)= 0:"                                      60838d    */
ResetDiskBuffer();
/* "GOSUB 60810"                              60810<-    60838e    */
}
```

After we've CLOSEd it and reOPENed it in "O"utput sequential-file mode, we update the file's OpenType[], CurrentCharCount[], RecordCount[], and TotalCharCount[] values to their default settings. Then we ResetDiskBuffer():

```
ResetDiskBuffer()      Resets Disk Input Buffer for "G" and "A" modes
-----------------------
Usage:          GOSUB 60810
Used in:        AppendOpen()
                DefaultFileParams()
Note:           Pseudocode in "Skeleton in PseudoCode"

ResetDiskBuffer( file#) {   /* resets Disk Input Buffer for "G" & "A" modes    60810 */
/*                  VA%(0)                                                          */
    Buffer$[file#]="";
    /* 'SY$(X9%,1)= ""'                                                           */
    CharPos[file#]= 256;
    /* "SY%(x9%,2)= 256"                                                          */
    }
```

Now, instead of using a "R"andom access file with GET and PUT and FIELD, we're using a Sequential access file mode.

Finally, in either case (empty file OPENed in "O" mode, or existing file OPENed in "R" mode), we're going to return the file # in VR%(1), and report Success in VR%(0):

```
    VR%(1)= file#;
    /* "VR%(1)= X9%:"                                          60839a    */
    return SUCCESS;
    /* "VR%(0)= -1:"                                           60839b    */
    }
    /* "RETURN"                                                60839c    */
```

Finally, back to fOpen(), which we began discussing in the previous article. Remember that big switch statement which OPENed a file based on its mode (Input / Output / Random / Getc / Putc / Append)? Here's the relevant section of pseudocode again:

```
    switch( mode#) {
    /* "ON X1% GOTO 60610, 60620, 60630, 60640, 60650, 60660:"    60540a    */

        case INPUT_MODE:
            /* '7A. OPEN "I"'   */
            OPEN "I", file#, filename$;
            /* 'OPEN "I", VA%(0), VA$(0)'                              60610 */
            break;
            /* "GOTO 60700"  */

        case OUTPUT_MODE:
            /* '7B. OPEN "O"   */
            OutputOpen( file#);
            /* GOSUB 60840: IF VR%(0) THEN 60700"              60840 <-  60620 */
            break;
            /* "GOTO 60700"  */

        case RANDOM_MODE:
            /* '7C. OPEN "R"'   */
            OPEN "R", file#, filename$;
            /* 'OPEN "R", VA%(0), VA$(0):'                             60630 */
```

```
    break;
    /* "GOTO 60700"       */

case GETC_MODE:
    /* '7D. OPEN "G"   */
    if GetcIsByteStream
    /* "IF SY%(4,0) THEN"                                    60640a    */
        OPEN "R", file#, filename$;
        /* "60630" (see case 3: above)                       60640b    */
    else
        OPEN "I", file#, filename$;
        /* "ELSE 60610" (see case 1: above)                  60640c    */
    break;

case PUTC_MODE:
    /* '7E. OPEN "P"    */
    OPEN "O", file#, filename$;
    /* "GOTO 60620"                                          60650     */
    break;

case APPEND_MODE:
    /* '7F. OPEN "A"'   */
    AppendOpen( file#);
    /* "GOSUB 60830: IF VR%(0) THEN RETURN"                  60660a    */
    if (!Result) {
        fOpenErrorReport( ErrorFname$,
            "'A'PPEND MODE: CAN'T OPEN", file#);
        /* "ELSE 60690"                                      60660b    */
        }
    else
        return;

default:
    ErrorType$= "CAN'T OPEN IN MODE '" +
        MID$( "IORGPA", mode#, 1) + "'";
    fOpenErrorReport( ErrorFname$, ErrorType$, file#);
    return;
    /* "VA$(6)= "CAN'T OPEN IN MODE '" +
```

```
                MID$( "IORGPA", X1%, 1) + "'":
                GOSUB 60950: RETURN"          60950<-    60690  */
        }
```

The pseudocode leaves just a little bit out. Take INPUT_MODE, for example. Here's what the uncommented BASIC code actually looks like:

```
60610 OPEN "I", X9%, VA$(0): GOTO 60700
60615 GOTO60690
```

What's that line 60615 directly after 60610? When would we ever actually execute it? After all, line 60610 ends with "GOTO 60700".
Line 60615 is the ON..ERROR RESUME NEXT line. If there was an File Open Error, specifically #54, "File Not Found", the error manager in lines 65500..65504 would trap that and RESUME execution on line 60615, which sends us to the error reporting code at lines 60700..60790. Here's the error manager:

```
65499 '

     * FILE NOT FOUND ERROR HANDLER *

65500 VR%(0)= FN E4%( ERR) = 54: IF VR%(0) THEN RESUME NEXT
65502 ON ERROR GOTO 0
65504 RESUME
```

and here's the "File Not Found" error reporting code:

```
60690 VA$(6)= "CAN'T OPEN IN MODE '" + MID$( "IORGPA", VA%(1), 1) + "'": GOSUB 60950:
RETURN
```

It calls fOpenErrorReport() at line 60950, which does two things: decrements the number of OPEN files (which we'd incremented earlier while attempting to OPEN the file); and it formats the error message, calling PutErr() with this message to print the it on "stderr", the standard error file (or, more likely, the standard error device, the screen). Here's the pseudocode for both fOpenErrorReport() and PutErr():

```
fOpenErrorReport( filename$,  ErrorFunction$, ErrorType$) {
/*                    VA$(0),        VA$(5),        VA$(6)                    */
/* Reports "ERROR", filename, "OPEN", file#, and returns FAIL                 */
```

```
    e$= "ERROR: '" + filename$ + "'-" + ErrorFunction$ + " " + ErrorType$ + "IN" + STR$( file#);
    /* 'VA$(10)= "ERROR: '" + VA$(0) + "'-" + VA$(5) + " " + VA$(6) + "IN" + STR$( X9%)'
                                                                              60950      */
    Result= FAIL;
    /* "VR%(0)= 0"                                                           60952      */
    PutErr( e$);
    /* "GOSUB 65520"                                          65520 <-  60954      */
    }

PutErr( e$) {                       /* append newline to string e$ and write on stderr
    VA$(10)                                                                           */
    file#= StreamTranslate( STDERR);
    /* "VA%(0)= FNST%( SY%(3,0)):"                                           65520a     */
    if (file# == 0)
        file#= SCREEN;
        /* "IF VA%(0) = 0 THEN VA%(0)= FN SC%(0)"                            65520b     */
    putc( e$ + newline, file#);
    /* "VA$(0)= VA$(10) + CHR$( 13): GOSUB 60100"                            65522      */
    }
```

"StreamTranslate()" interprets the "file handle" stored in the "stderr slot" at SY%(3,0). If it's a positive number, it converts it to an actual file handle, and error messages will go to that file. If it's a negative number, it's a device, and FN ST%() will return the appropriate device handle (-1: keyboard; -2: screen; -3: printer). If it's 0, no stderr file or device was specified by SH/BAS in the CMDLINE/SH file, so it defaults to the screen. At this point, the error message (with a newline appended) is sent to putc() along with the device or file that putc() will write it to. (We'll discuss how putc() is implemented in a later article.)

Back yet again to fOpen(). Notice that the switch statements for "I"nput mode, "O"utpot mode, and "R"andom Access mode all share the same ERROR recovery line following the OPEN statement.

```
60599 '
          7A. OPEN "I"

60610 OPEN "I", X9%, VA$(0): GOTO 60700
60615 GOTO 60690'
```

7B. OPEN "O"

```
60620 GOSUB60840: IF VR%(0) THEN 60700
60625 GOTO 60690'
```

7C. OPEN "R"

```
60630 OPEN "R", X9%, VA$(0) : GOTO 60700
60635 GOTO 60690
```

(OPENing in “G”etc or “P”utc mode actually jumps to the “R” , “I”, or “O” code here, so they make use of the same ERROR code.) If there was no error in OPENing the file, we GOTO 60700, which sets up default parameters for the file in question. OPENing in “A”ppend mode does not go there, however: AppendOpen() did its own initialization, so we just RETURN from fOpen() in that case. Here’s the final initialization code from lines 60700..60790, first in BASIC, and then in pseudocode:

```
'
        SET UP DEFAULT FILE PARAMS

60700 SY%(X9%,1)= X1%: SY!(X9%,0)= 0: SY!(X9%,1)= 1: SY!(X9%,2)= FN DX!( X9%): GOSUB
60810'
 SET UP OPEN MODE, CHAR POS IN RECORD, CHAR COUNT, CHAR POS, TOTAL CHAR COUNT

60790 VR%(0)= -1: RETURN
```

Now in pseudocode:

```
    OpenType[file#]= mode#;
    /* "X9%= VA%(0): SY%(X9%,1)= X9%:"                    60700a    */
    CurrentCharCount[file#]= 0;
    /* "SY!(X9%,0)= 0:"                                   60700b    */
    RecordCount[file#]= 1;
    /* "SY!(X9%,1)= 1:"                                   60700c    */
    TotalCharCount[file#]= TotalBytes( file#);
    /* "SY!(X9%,2)= FN DX!( X9%):"                        60700d    */
    ResetDiskBuffer();
    /* "GOSUB 60810"                                      60700e    */
```

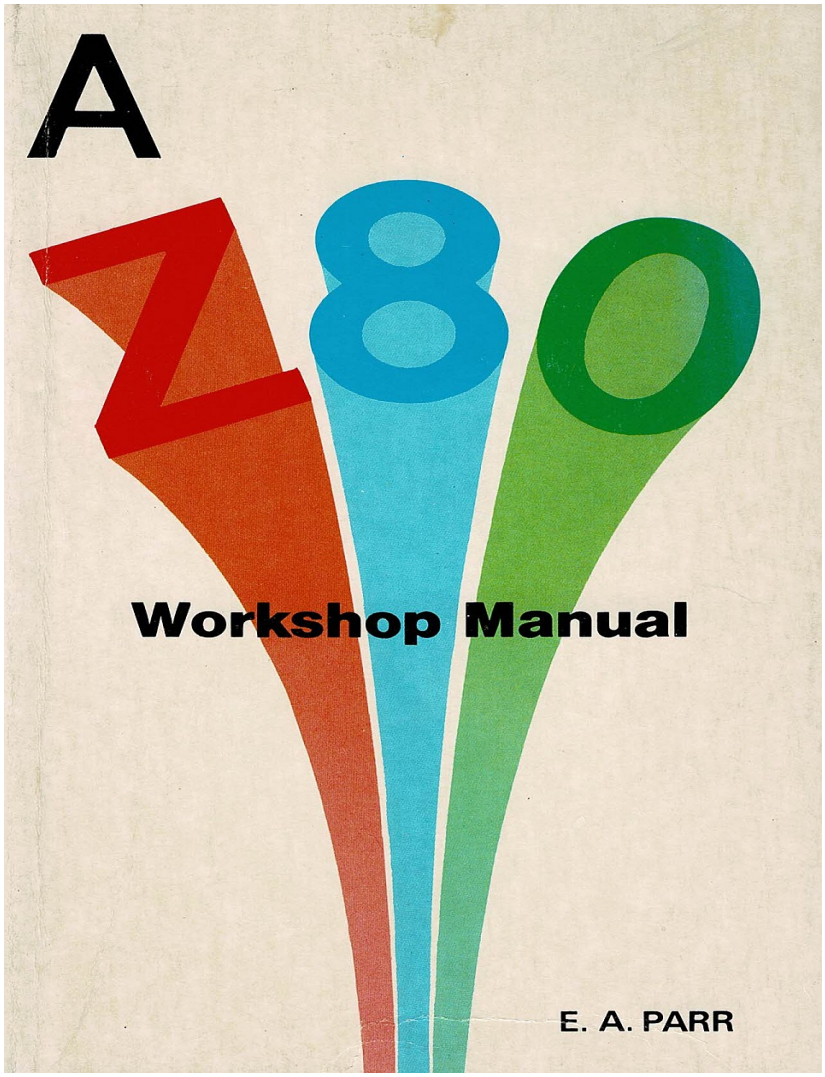
```
    return SUCCEED;
    /* "VR%(0)= -1: RETURN"                                          60790     */
```

"ResetDiskBuffer()" / GOSUB 60810 looks like this:

```
ResetDiskBuffer( file#) {       /* resets Disk Input Buffer for "G" & "A" modes
                    VA%(0)                                                      */
    Buffer$[file#]="";
    /* 'SY$(X9%,1)= ""'                                              60810a    */
    CharPos[file#]= 256;
    /* "SY%(x9%,2)= 256"                                             60810b    */
    }
```

That's it! At last, the file is OPEN.

Next Episode: now that the door's OPEN, how do we find out what's inside? getc() and putc() explained.



Thirsting for adventure?
Brass lantern getting dusty?



Take your Model 1, 3 or 4 on exciting new adventures never available until now....

M4ZVM enables your TRS-80 Model 1,3 or 4 to play Infocom and post-Infocom adventures, taking full avdantage of your hardware. Supports old favourites like Zork and Seastalker (including Sonarscope), as well as new-to-Model-4 games including Trinity, Border Zone and Sherlock, plus hundreds more available in the Interactive Fiction archive.



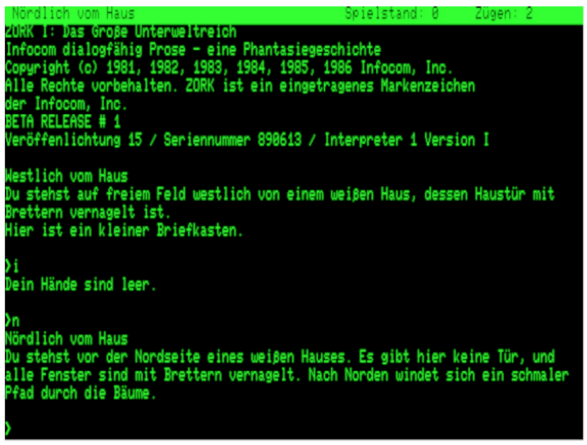
M4ZVM is available for download from https://gitlab.com/sijnstral/m4zvm (includes feature details and source)
Enhanced versions ready for download!

*128K recommended

Actual Screenshots of games new to the model 4, running M4ZVM (on trs80gp)



Beyond Zork



Zork 1 (German)

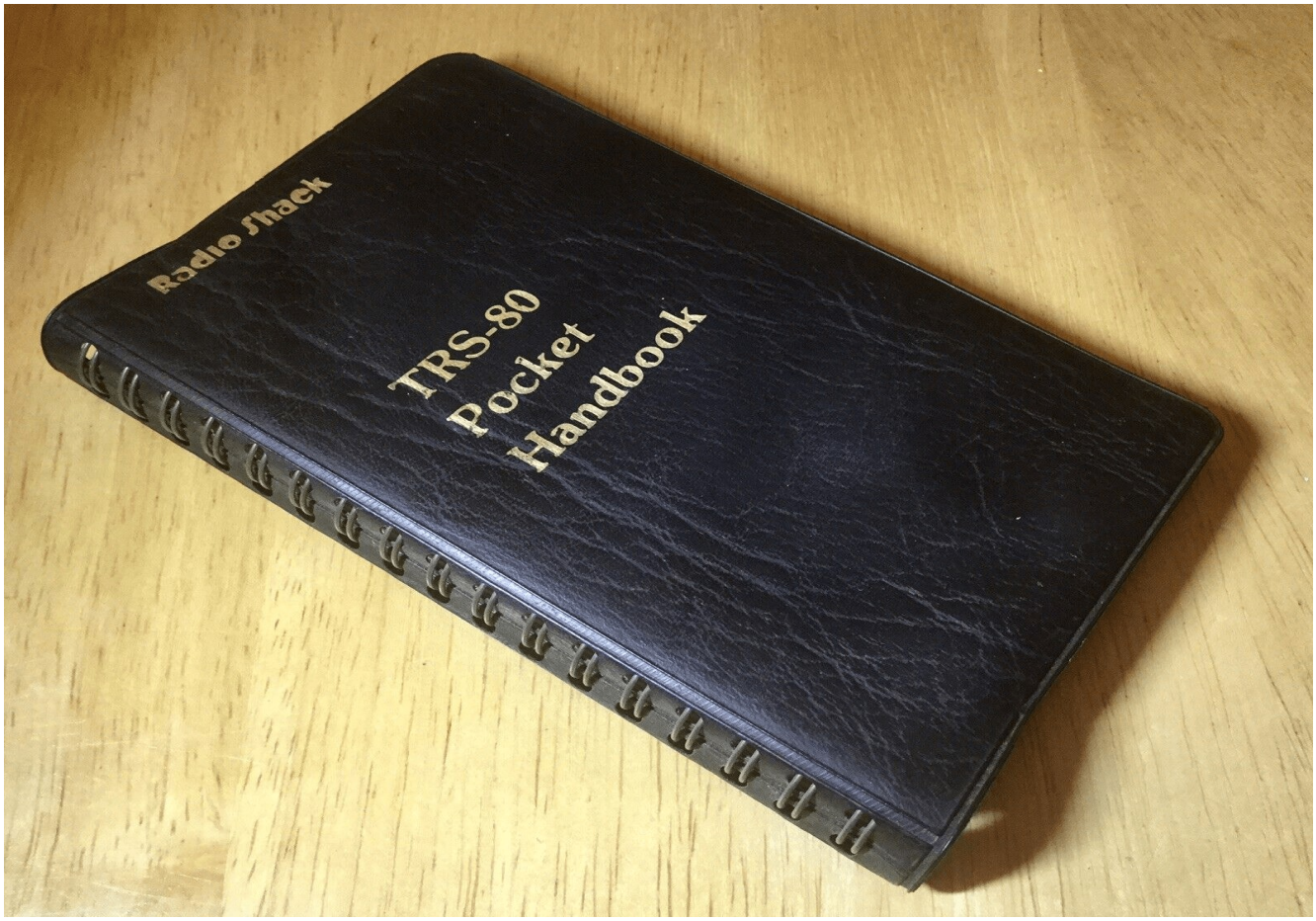
M4ZVM features include:
* z1-z8 game execution (z6 support is experimental)
* named game save/load
* accented characters
* 128K recommended - NEW Minimised 64K version
* Optionaly set the infamous "Tandy bit"
* NEW Model 1 and Model 3 versions



HTTPS://WWW.YOUTUBE.COM/WATCH?V=ZE3NMXTLGJK

THE WAY WE WERE

1985 1976 1979 1983 1978 1980 1981 1977 1982 1984



Bill Harvey
204 Vienna Drive
Milpitas CA 95035

The Exatron Stringy Floppy

The ESF is a device for those who don't want a cassette system and can't afford a disk.

At some point, your system will outgrow its memory to store all those "neat" programs. In the meantime, you have to suffer through waiting for your cassette (or, heaven forbid, your Teletype!) to load and store your creations. At present you have a Super 13.978K Ultra-Extended BASIC (FORTRAN, COBOL, Trek game, etc.) that loads in under two (or maybe 20) minutes . . . if you have set the tape counter correctly and remembered to switch off the pause control . . . if the volume and tone controls have not shifted since last time you used them . . . and if your little brother has not used the tape to record secret messages! That is, after you rewind the cassette (45 seconds), because the program you want *is always* on the other side. It may be time to look for something a little bit faster.

Have you got $700 to $1100, 16 hours for assembly time and the patience of Job to interface that "great" software—all to invest in a mini-floppy? No? Well, have I got just the thing for you.

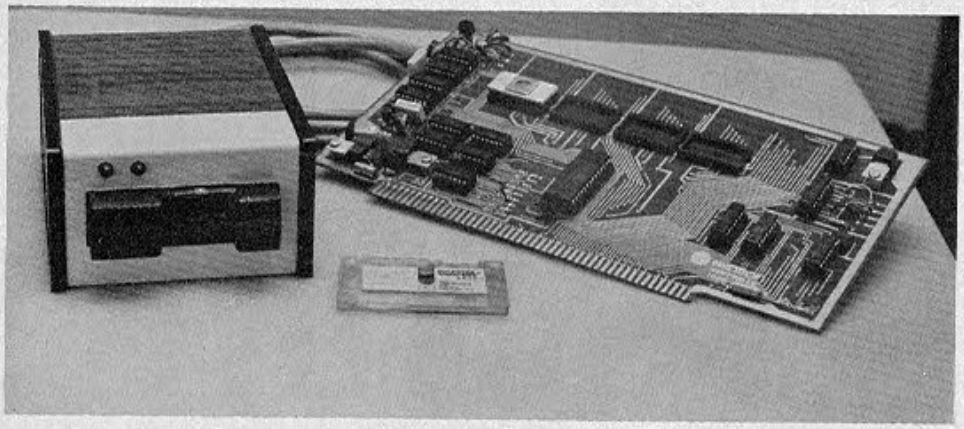
How does 7.2K bits/sec transfer rate sound . . . full software control of transport in EPROM (2708) . . . 45,000 bytes of storage per wafer . . . three extra EPROM sockets for your own use . . . completely assembled, tested and ready to run (plug it in, turn it on and GO!) . . . and it really works . . . all for less than $300? Sound like a possible winner? Read on, friend.

The ESF Package

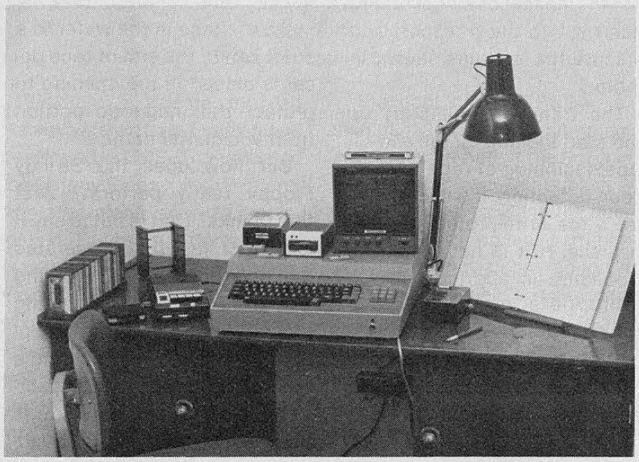
The item under discussion is the Exatron Stringy Floppy. Exatron is the maker of a line of industrial automated IC testers and three computer products: the Exatron Stringy Floppy (ESF), an unusual 36K RAM/EROM Memory Card and a soon-to-be-released EPROM Programmer, all S-100 bus oriented. In plain English, the ESF is a completely assembled and tested endless-loop-tape mass-storage system complete with transport, controller/EPROM software printed circuit board (presently available as an S-100 bus card suitable for 8080 or Z-80 operation. Exatron is now advertising one for the SWTP 6800), 6K assembler-editor monitor and several blank tape wafers.

With the ESF is included a complete head cleaning kit to ensure long lead life. The owner's manual, sold separately so that potential purchasers can determine the suitability of the ESF Mass Storage Subsystem for their proposed use, contains one of the most complete descriptions of a computer product I have ever seen. The manual contains a description of the product, installation and checkout, operation (general), utility program descriptions with flowcharts (24 pages of flowcharts alone!) and a complete source-code listing of the software, troubleshooting procedures for transport and controller, plus manufacturer data sheets for the LSI chips used in the ESF—an excellent value in itself.

Both the manual and ESF are supplied with a 30-day money-



Complete Stringy Floppy system as received with transport, controller, PC board and test wafer. The white IC is the 2708 software EPROM with room for three more 2708s on board. Addressing must be in the same 4K block, typically B000 or D000.



My home-brew system with (left to right) cassette recorder, console/CRT, Exatron Stringy Floppy and paper-tape reader. The mainframe is mounted in the homemade desk behind the door in the lower right.



Close-up of peripherals.

back guarantee. The ESF comes with a one-year normal-service warranty. And with a $10 second-year membership in the Exatron Stringy Floppy Owner's Association (ESFOA)—the first year's membership is included with the owner's manual—you get a second year's warranty for the ESF. Unheard of, you say? Too good to be true? My initial thoughts exactly.

My first introduction to Stringy Floppy was at a local computer repair/design shop where a Stringy Floppy was being custom-fitted into a demo computer. Not having the aforementioned $700 to $1100, and as I was looking for an alternative to my cassette system, I investigated further.

I went to Exatron to purchase an owner's manual and to check out the company behind this unusual product. I was welcomed in by Bob and Mike Howell, the owners/operators of Exatron. After a long discussion and a short tour, I was on my way home to study my owner's manual in preparation for the Saturday-morning users group meeting, of which I was now a member. I figured this was a chance to seek out disgruntled owners and find the product's weaknesses! No such luck, though.

Everyone there had had good luck getting the ESF running quickly with almost no problems, save one experienced by a user with a Z-80 System. He later discovered a defective dynamic memory card that, when removed, improved his success rate significantly.

The ESFOA also issues a professionally produced monthly newsletter that is distributed to all members. Topics include interfacing techniques for popular BASICs, save-load routine descriptions for the assembler, utility-software routine summaries and topics of discussion at the ESFOA Saturday meetings.

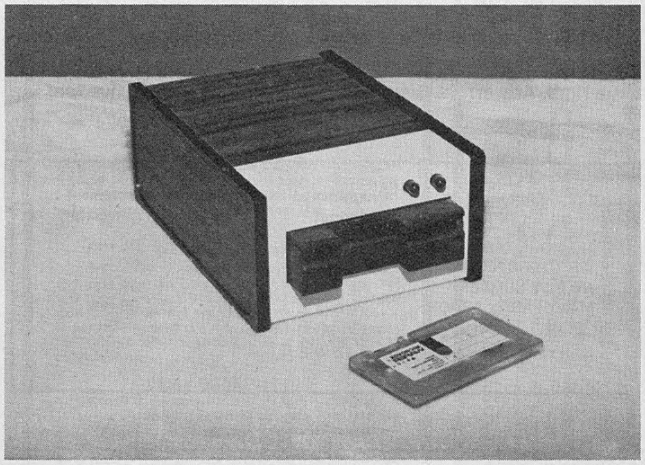
With the product seemingly standing head and shoulders above the competition in relation to value for money, I plunked my money down. In only four days I had my Stringy Floppy, and off I went to ponder, "Why doesn't it work?" You see, I have a Turn-key, home-built, S-100 bus system, currently running a TDL Z-80 CPU with 25K of static RAM, 1K of PROM, 32-character-per-line CRT with keyboard, cassette interface and paper-tape reader. Nothing, especially software, seems to be readily compatible with my system. So a hardware/software combination could never work without modification.

Surprise! It worked the first time! The only modification required was a change in software to move the error display from the front panel to my CRT. Even this is in RAM, so no EPROM change was required.

Installation and Testing

Let me describe a typical installation and checkout of an ESF. The box that the ESF arrives in has a cut-to-fit foam liner that offers excellent protection during transport. The package contains the Stringy Floppy Transport, hard-wired to the controller printed circuit board and a head-cleaning kit. Several wafers, one with the assembler/editor to start you off, are also included.

Following the instructions in the owner's manual, insert the controller into an empty slot in your S-100 bus. The three-foot interconnect cable allows convenient placement of the tape transport. Check to see that you have no memory or other

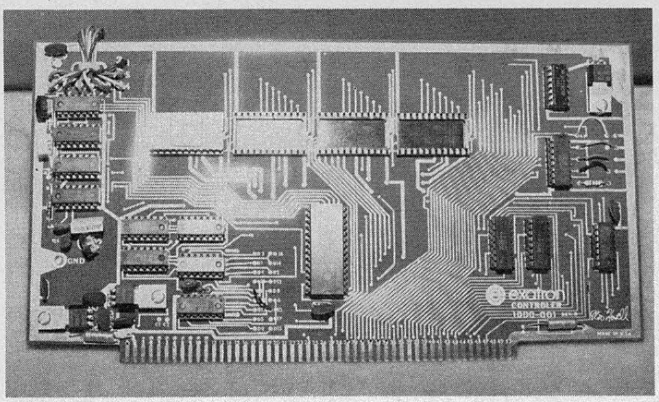


Stringy Floppy tape transport with wafer ready for insertion.

```
0000   31   00   04   BEG LXI SP, 0400H
0003   21   0E   02     LX1 H, SHOW
0006   E5              PUSH H
0007   21   9A   10     LX1 H, 4250D
000A   E5              PUSH H
000B   C3   D2   DD     JMP 0DD02H
000E   2F              SHOW CMA
000F   D3   FF          OUT OFFH
0011   C3   00   00     JMP BEG
```

To certify we set the stack, push the Return if Error address, push the byte count required to nearly fill a 5 foot wafer, jump to the Certify routine in EPROM, display results on the programmed output port, and return to start to run the test, again.

Table 1.



Controller card for Stringy Floppy.

device addressed to the D000H block as the controller is assigned to this block and may conflict with other devices if similarly addressed.

After turning your computer on, you must enter a short "Certify" program, as shown in Table 1. This will allow certification of your wafers, a process that is necessary with all new wafers. Because the wafers are the endless-loop type, several complete passes may be necessary to distribute the tape lubricant before accurate results can be obtained.

Certifying runs the tape in and checks the reliability at the same time. Any errors detected will normally be displayed on the front panel according to the chart in Table 2. (In my system the error is taken from the "A" register and output to a port to be displayed on the CRT. No display = no errors.)

Successfully certifying a wafer means you have simultaneously tested the transport and controller printed circuit board. Writing a wafer with data is the next test.

The software contains three "Quick Store" routines designed to write 1, 4 and 8K blocks that start storing data from 0000H upward. The procedure is to select a wafer of the appropriate length, five feet for 1 or 4K, ten feet or longer for 8K, insert it into the transport and execute the address shown in Table 3.

The transport will start up and start looking for the end-of-tape/beginning-of-tape marker. Upon detection, it will check to see if the write-protect decal is in place. If it is not there, the procedure is aborted with an appropriate error code displayed. Otherwise, the Write LED will come on indicating the storage of data. Upon completion, control will return to 0000H.

Similar "Quick Load" programs are available to complement the "Quick Store" routines. It is only necessary to insert a wafer with a "Quick Stored" program in the transport and execute the appropriate address in Table 4 according to the recorded block length.

With a successfull write/record test the unit has completed its checkout. It is now ready for use.

Software

The software contains many other routines too numerous to explain in detail at this time, but they are listed in Table 5. In general, the ESF software allows the reading and writing of data in a length to 40K with user defined "User's Return with Error Address," "Control Transfer Address" (possible Auto-Start or Load'N Go), Block Length with Begin and End Addresses and File Number routines to aid in location of a particular file among many on a wafer.

There are also routines to "Verify" a wafer against memory, "Add" a file to a wafer, "Write and Verify," "Add and Verify" and "Park," that is, move the tape in the wafer to a point where the end-of-tape decal is placed in the opening to protect the magnetic portion from accidental harm.

But how does the Stringy Floppy really perform? Just fine thanks! That is not to say it is perfect, just that it operated the first time, is still working and gives me a certain feeling of confidence about further operation. It is certainly not of the type of quality normally expected of the hobbyist market. It is of the best "commercial" quality in terms of material, assembly and software.

The minor problems include not being able to consistently get 50-foot tapes to run on my unit and documentation that, although may be a cut above the competition, is still somewhat incomplete. This latter problem is being corrected by the ESF Newsletter. Exatron has been most helpful in attempting to solve my tape problem, and I am sure we will resolve it soon. Other users have not made mention of any tape problems, so this may be a one of a kind.

Conclusion

For those who need an alternative to a relatively slow, manually operated cassette system and who cannot afford the time and money for a floppy disk the Stringy Floppy is the only way to go. The price is right, the quality is high, and the system works with little or no effort, which, I think, is the key. Hard to believe from a hobbyist computer company, but true. Very true. ■

EXATRON
3555 Ryder Street
Santa Clara CA 95051

Error Code	Programmed Output	ASCII Character	Complement In A-Register
NO MORE TAPE	21 Hex	!	DE Hex
WRITE-PROTECTED	23 Hex	#	DC Hex
NO FILE NUMBER	25 Hex	%	DA Hex
MANY PARITY ERRORS	26 Hex	&	D9 Hex

Table 2.

LABEL	ADDRESS	FROM	TO	FILE#
STOR1	DF2D	0000H	3F0H	FFF1
STOR4	DF39	0000H	0FF0H	FFF4
STOR8	DF45	0000H	1FF0H	FFF8

Table 3. Quick Store routines.

LABEL	ADDRESS	FROM	TO	EXECUTE STARTS
LOAD1	DF07	0000	03F0H	0000H
LOAD4	DF10	0000	0FF0H	0000H
LOAD8	DF19	0000	1FF0H	0000H

Table 4. Quick Load routines.

CERTIFY	Certify a wafer
WRITE	Write a file
READ	Read a file
VERIFY	Verify a file against memory
ADD	Add a file to a wafer in addition to the current file already recorded
WRITE & VERIFY	Write a file and verify
ADD & VERIFY	Add a file to the wafer and verify
PARK	Park the wafer in a protected position

Table 5. General-purpose utility procedures.

Hybrid programming

For Models I & III, tape only, with EDTASM

R B Nottingham, Deerfield Beach, Florida

Those of us who have had computers for a couple of years and spent a few thousand hours programming tend to forget what a confusing world microcomputers and their programming can be. After graduating from Level I to Level II and acquiring 16K of memory, I wrote and copied many BASIC programs. Copying programs from other, better programmers was a good way to learn. Even programs in which one has no real interest can teach useful programming techniques.

Early in my progress, I bought TBUG and the Editor/Assembler. After spending a few hours with them, I put them aside and went back to writing BASIC. My IBM friends kept telling me I should learn assembly language, but the motivation was lacking. Machine language programs work 100 times faster than BASIC, but for me they would also take 100 times as long to write.

I then discovered assembly language programs which would do things I couldn't do in BASIC, or which required prohibitive time to run. I mastered the technique of converting assembly language into BASIC POKE programs and setting memory size from the program. Now I have a library of BASIC machine language programs which do things like:

1. Activate the MERGE command, so one can quickly splice two programs together.
2. Provide single step execution and scrolled listings.
3. Print a list of all line number cross-references and a list of all variable names and the line numbers on which they occur.
4. Make it possible to save and reload an entire array, very rapidly, rather than handling it very slowly by means of PRINT#-1 and READ#-1.

Here is the step-by-step procedure used to convert assembly language programs to BASIC POKE programs. First, remember that when you want to clean house in computer memory you do not need to turn the power off and on again. Simply enter SYSTEM, and when the prompt (*?) appears, type "/0". There is our old friend, MEMORY SIZE, and you are starting fresh.

Listing 1 contains a trivial program to illustrate the technique. Assemble this program and put it out onto a cassette. Note the RET at line 200 is what brings us back to BASIC after the program has executed. If we take the address 7000H and multiply by 4096, we get 28672, the decimal starting address.

Return to BASIC and set the memory size to 28671 (one less than the starting address). Load your machine language program by entering SYSTEM, and the name you have given it. When the system prompt (*?) comes back on, your program is loaded. Hit BREAK - don't execute it yet. (We assume you have executed it a time or two as a machine language program to make sure it works.) Now start at line 100, and enter this short BASIC program:

```
100 FOR X = 28672 TO 28687
110 PRINT PEEK(X);
120 NEXT X
```

Type RUN. You should see the data shown in line 50 of listing 2 appear on the screen. Start with line 50, and copy this data directly into a DATA statement. For our example we will need only one line of data, but some programs will need more, of course.

You got the starting memory address by decoding the Hex address from assembly language. How do you get the ending address? When you write your FOR...NEXT loop, guess the length of the program in bytes, add a safety factor and add that to the starting address to get the upper limit. Now, run it and watch for 201. This is the RET command in decimal. Following it should be a series of 0's and 255's. This will enable you to determine the highest memory address.

HYBRID

When a program has several data lines, it's a chore to try to find a typographical error; therefore, verify each data line as it is entered. To do this, write another FOR...NEXT program at a higher line number, say 1000, using the same values for X as above, where READ D, PEEK (X) and IF D <> PEEK (X) STOP. After entering each DATA line, type RUN 1000, and if the message is OD (out of data) or READY, fine - but if it is BREAK, then there is an error in the last line of data. So find it and correct it.

Now that we have data statements which match our machine language, we delete the PEEK statements and the verification loop at 1000, and make our program look like lines 10-40 of listing 2. These lines will POKE into the proper memory locations the same instructions which our machine language program contained.

Line 1 is the line which sets memory size. Note the CLEAR statements at the beginning and end, and note that it is line 1. The clear statements are important, and since we can run into trouble if we go too far in the program before setting memory size, do it first.

The memory addresses 16561 and 16562 are the location for memory size. How do we arrive at the addresses to poke? Remember that the starting address of our machine language is 28672. With our computer in the command mode, we enter: ?INT(28672/256) and get 112. We want to set the memory size at one less than that. The least significant byte of the address comes first. One from 112 equals 111 as the most significant byte and 255 as the least significant. Therefore, we poke 255 and 111. In line 2, the poke addresses are those where the computer looks when it reads USR, the call for a machine language subroutine. Since 28672 is evenly divisible by 256, we poke 0 in the first address and 112 in the second. If our address had not been evenly divisible, we would have obtained N2, by INT(*address*/256) and N1 by *address*-256*N2.

Lines 100-130 are a short BASIC program which calls the subroutine in 100 with the USR call, holds the screen white for a short time with line 110, then clears the screen and prints a message in line 130. If you load the program and type RUN, there will be a brief delay, followed by READY. This means you have loaded the program and reached line 60. Then type RUN 100 and watch the screen flash.

Suppose your main program has DATA statements. Your computer when it is told to READ, always starts with the lowest numbered data statement. No problem. Change line 60 to DELETE 1-60. Yes, delete is a valid command, and it will delete itself. When you first type RUN, the data pokeing program will execute and then be deleted, leaving your routine in machine language, patiently waiting for you to execute a USR command.

More than one machine language routine can reside in memory at the same time, if they are given memory locations which do not overlap. To call any routine, you must execute the equivalent of line 2, before calling it so that the USR instruction will jump to the proper location.

If you have not used the techniques in this article before, try the procedure as described with the program shown, then look through your back issues of magazines, find an assembly language program that you really want to use, and use it!

Listing 1

```
100             ORG     7000H
110 CLEAR       LD      HL,3C00H
120             LD      BC,1024
130 LOOP        LD      A,0BFH
140             LD      (HL),A
150             INC     HL
160             DEC     BC
170             LD      A,B
180             OR      C
190             JR      NZ,LOOP
200             RET
210             END
```

Listing 2

```
  1 CLEAR 1000:POKE 16561,255:
    POKE 16562,111:CLEAR
  2 POKE 16526,0:POKE 16527,112
 10 FOR X = 28672 TO 28678
 20   READ D
 30   POKE X,D
 40 NEXT X
 50 DATA 33,0,60,1,0,4,62,191,119,35,
    11,120,177,32,247,201
 60 END
100 A = USR(0)
110 FOR T = 1 TO 1000: NEXT T
120 CLS
130 PRINT "WHITE, WASN'T IT?"
```

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

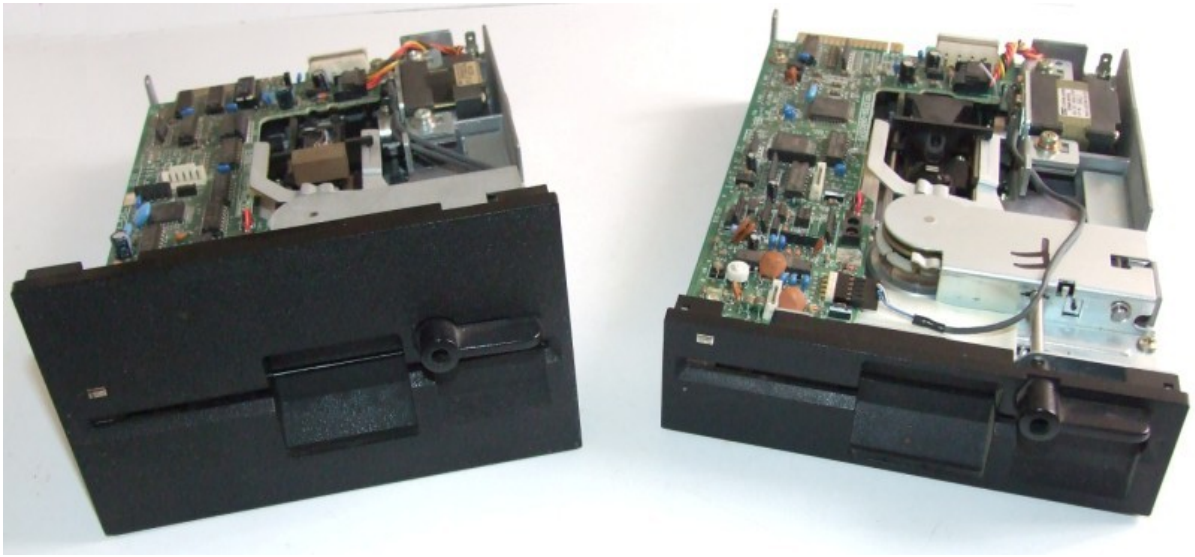
RETRO RESTORATION SPECIALISTS

Call Corfu 26610 26358 Mobile 698 755 8427

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

IN MAV'S WORKSHOP
Let's Talk About TEC
FB Series Disk Drives

Ian Mavric



In 1986 Radio Shack stores quietly introduced an update to the Model 4 called the 4D. Very little advertising or publicity accompanied the new computer, it just... appeared. The big difference, Radio Shack explained, was the D stood for DeskMate, a free suite of productivity apps included with the 4D. To most of us however, was the interesting new disk drives which were now double-sided holding around 360K of information each. They also looked weird, as they had a twist latch, like a PC disk drive, but they filled the space in the case set aside for full height disk drives. Tandy had finally moved away from TP (Texas Peripheral) belt-drive single sided klunkers to modern disk drives straight out of Japan. In this article I'll explain the TEC FB series disk drives.

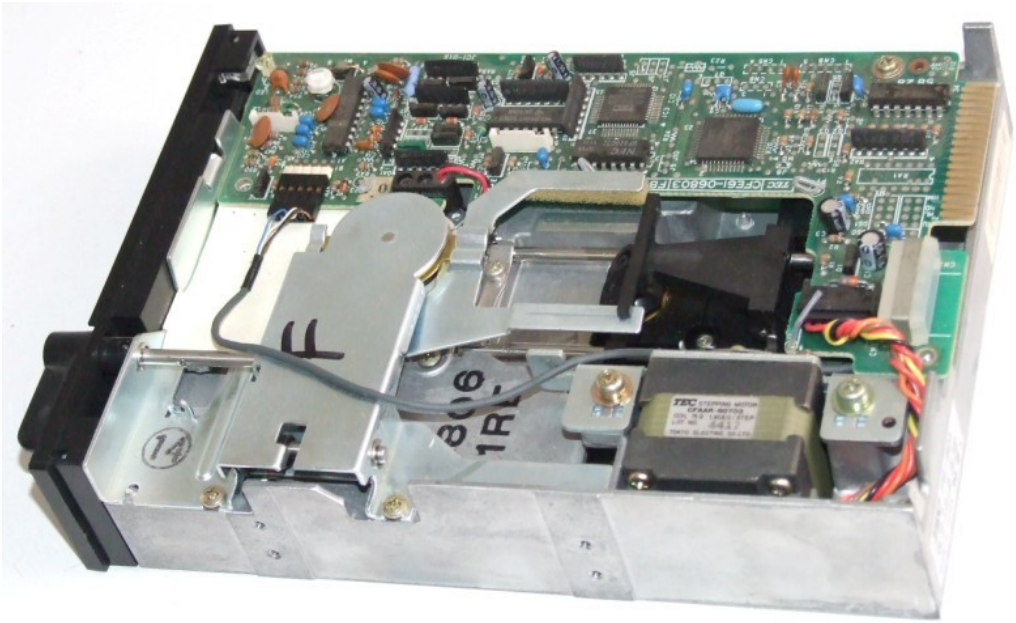


Photo: TEC FB-501 is a single-sided version of the 4D disk drive

IT WASN'T JUST THE MODEL 4D WHICH GOT THESE

The 4D got the FB-503 version of the disk drive, which means a double-sided 40 track double density drive. But other Radio Shack products also got an FB series drive at the same time. The Coco FD-501 disk drive got a single-sided version called the FB-501 and the Model 100 DVI unit also got one of these.

MODELS IN THE FB SERIES:

FB-501: single-sided double-density 40 tracks 184K
FB-502: single-sided double-density 80 tracks 360K
FB-503: double-sided double-density 40 tracks 360K
FB-504: double-sided double-density 80 tracks 720K

Logic board models:
CFE61-06803/FB-500: Single sided only, has a connection for only one read-write head



CFE61-05004/FB-500C-2: double-sided , has connections for two read-write heads - these are ones found on 4Ds



TEC, which stands for Tokyo Electric Co., is a company you may have heard me mention before. They were some sort of contractor for Radio Shack, among other companies, and some of TEC's previous work included the last "Made in Japan" Model I and the Coco 1 26-3022 original silver disk drive. In the case of the FB series twist-latch disk drive TEC also sold them to other computer companies and dealers were able to buy them for custom PC builds. While they weren't a worldwide run-away success like the Teac FD-55 series of 5.25in disk drives, they weren't only limited to use in Tandy computers, and this is handy to know when looking for used ones. Most of them will have a normal sized front bezel not the big one the 4D had, and there was even an 80-track double-sided version which stored some 720K of information called the FB-504. More on it later.



Photo: FB-504 can be converted to a FB-503, explanation later.

THE VERY SMALL ELEPHANT IN THE ROOM

Just over a decade ago I bought my first 4D. Remember they were only sold in the USA and Canada, so I obtained mine on eBay and imported it. Once I had the machine I noticed both disk drives didn't work, which turned out to be a common theme among 4D owners. A Google search at the time didn't turn up any leads so it was time to take a closer look at the drive itself. The design itself looks good with robust mechanicals and a well designed logic board. The first thing I check is electrolytic capacitors and saw it straight away, two were leaking corrosive ooze on to the PCB.

People with good a good memory may remember the capacitor plague of the 1990s, and while the e-caps used on the TEC FB series drives pre-date the capacitor plague, there are some similarities in that the 47uF e-caps used are of an unknown brand and the electrolyte in them is corrosive and leaks out, even if the computer goes for years or decades unused. The only positive out of this is that the e-caps are tiny so the volume of electrolyte that can leak out is very small. However, when it does it corrodes the PCB underneath it but you need to look closely with a magnifying glass to see it. On the FB-500C-2 logic board the caps are C17 and C26 (both are 47uF). While I am at it I also replace C28 which is a 100uF, though I have yet to see one of those that has gone bad.

RESTORATION OF A TEC FB SERIES DISK DRIVE

The robust nature of the mechanicals of the disk drive mean that I rarely find these disk drives with bad alignment, broken stepper motor or broken direct drive motor. This guide will mainly concern with the electronics on the logic board and direct drive motor control board.

Top logic board:

99% of the problems with the drive come from those three capacitors I mentioned before. Two are 47uf and while the logic board is out it's a good idea to replace the 100uF cap as well. NOTE: if you are repairing the single-sided logic board (CFE61-06803/FB-500) then the caps are C11 (1uF) and C1, C3 (both 47uF). I clean any left over electrolyte accumulated from the leaking capacitors with IPA. If the traces look badly damaged I test them with a multi-meter for continuity. I have never seen them badly enough damaged to require repair with bodge-wires, but you may need to resort to that if the traces are severely corroded away.

Three screws hold the logic board down, and there are various wires plugged into it. Under it is some cardboard, don't lose it - it's there for a reason. While the logic board is out I spray some WD-40 Silicon spray onto the rails the read/write head runs along, and check for smooth movement. If the rails show pitting or rust you may get away with the WD-40 but bear in mind the useable life of the drive might be quite short.

While the logic board is out I give a clean to the gold plated edge connector with 1200-grit paper, the idea here is to bring out the shine in the gold plating again, not rub it away altogether.
NOTE: VR1 is an variable resistor - don't move it. ALSO NOTE: CN7 connects to the sensor which reads the index pulse from the diskette. Make sure it's clean.

Direct-drive motor board:
When you flip the drive over you can see the bottom of the direct drive motor, and under it is another circuit board. Certain versions of the board have capacitors which leak like the ones on the logic board.

Occasionally you can see them on the solder-side of the direct-drive motor board as fuzziness of the solder and surrounding area under a leaking capacitor.

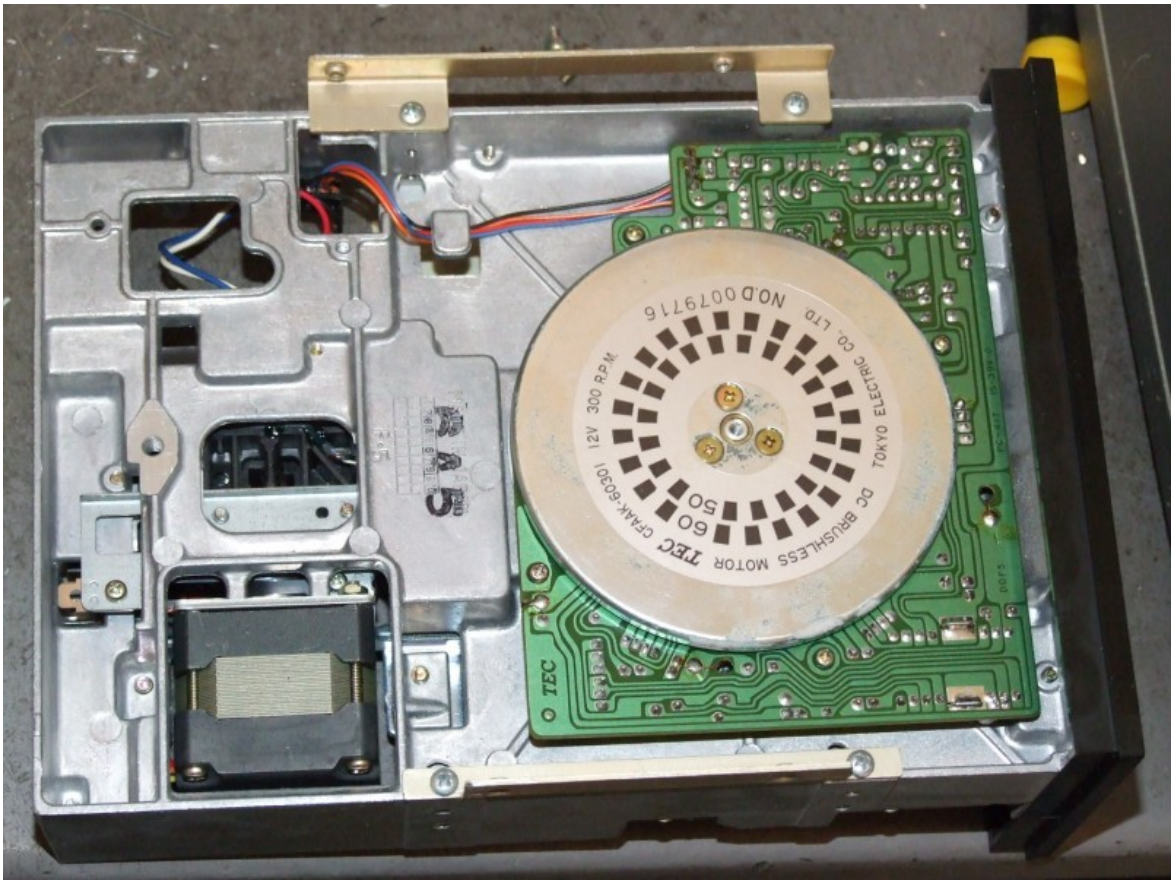


Photo: Underside of a 4D FB-503 disk drive

The circuit board is actually easy to get to. Near the centre of the big direct-drive flywheel are three Phillips-head screws. Remove them and the flywheel lifts up, as well as a number of brass spacers... don't lose them and put them back in the order you remove them.
Once they have been set aside the motor board can be undone and most of the cables unplug from it.



Photo: direct-drive flywheel, spacer and 3 screws

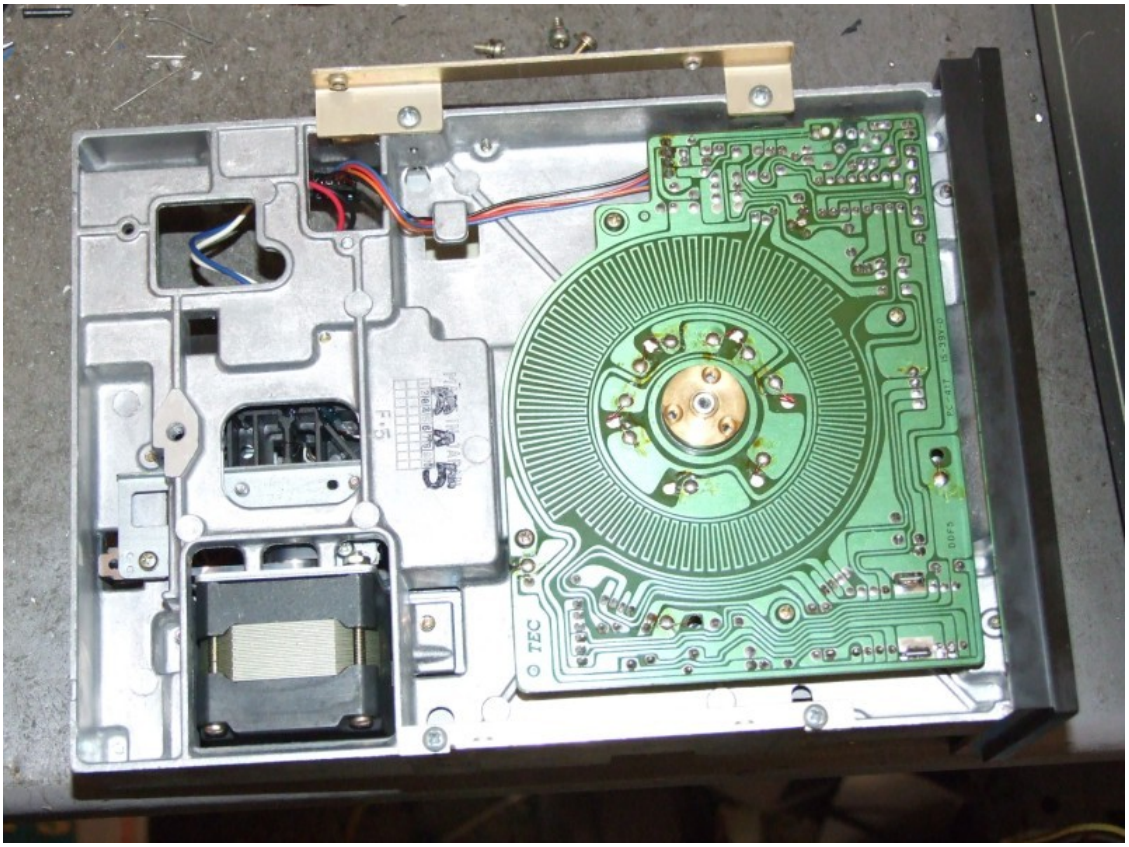


Photo: Now you have access to the direct-drive motor board

There is one connection which doesn't lend itself to disconnection, so I just handle the board carefully while I replace the bad e-caps. Unlike the top logic board which has only two versions, the motor board has many versions, so I can't say which are more problematic than others. If you suspect an e-cap it's better to be safe than sorry and just replace it on the motor board.
Re-assembly is just the reverse of disassembly. Put back the direct-drive motor board and the flywheel. Bear in mind there are some strong magnets in it so it can take a while to get it (the brass spacers and large flywheel) all back into place where you can put the three small screws back in. Re-fit the top logic board and you should be set. Test the drive in the usual manner. I usually run a head-cleaning disk on the drive around this point in time to make sure it has a nice clean read/write head.

FINDING AND USING REPLACEMENT TEC FB SERIES DISK DRIVES

Occasionally TEC FB series disk drives come up for sale on the usual places like eBay and vintage computer forums. Are they worth buying? If they are well priced and seem to be in good shape I think they are worth a shot. Regardless of what the seller says the FB-50x number is the important factor. I have seen eBay listings where they have misidentified a FB-503 as a 1.2Mb high-density drive, and a FB-501 as a 360K drive. Both are wrong, just bear in mind if you need a single-sided drive you need an FB-501 or 502, and for a double-sided drive FB-503 and 504, and there are no high-density versions.

Amazingly, at the time of writing (25/2/2024) this article there are literally no TEC FB series drives for sale on eBay and none in the completed listings (sold or not sold) either, meaning they are becoming very hard to find. What can you do if a FB-504 comes up?

CHANGING AN 80 TRACK TEC FB-504 TO 40 TRACKS

So by some miracle you've found a FB-504 drive for sale somewhere and are wondering if there is a way to make it double-step (or something) to read you disks because all your disks are 40-track? This is an interesting little hack I discovered and again could find nowhere on the internet but it turns out the FB-504 is really an FB-503 with an extra link in place which makes the stepper motor half-step so you get 80 tracks. Break the link at J7 on the CFE61-05004 top logic board and it's back to a 40 track drive. You lose the 80-track 720K capability but you do get a good reliable 360K replacement drive for your 4D.

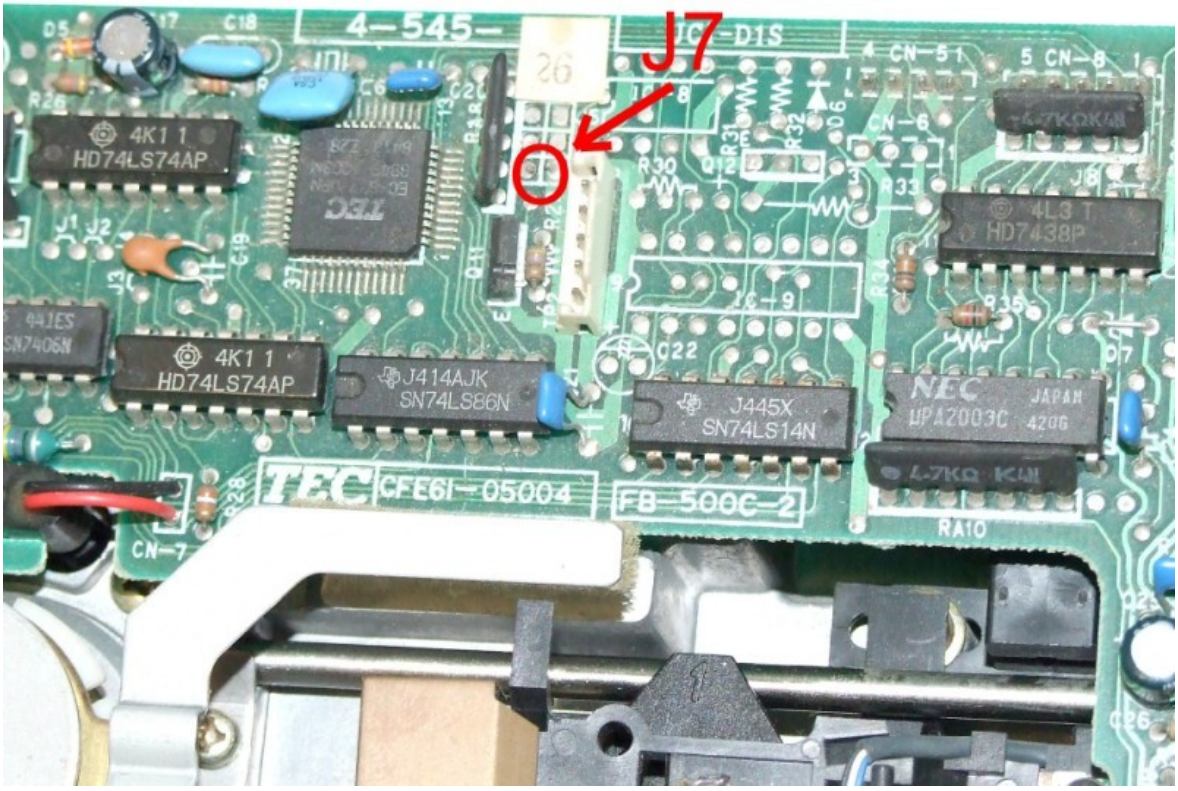


Photo: Where to find J7, the 40/80 track decider link

FINISHING OFF YOUR 4D DISK DRIVE

It probably goes without saying but the front bezel on the half-height FB series drive can be swapped with the oversized one from the 4D very easily.
A small Phillips-head screw holds the latch on, and then it just pulls off. Then under the front bezel are two more larger Phillips-head screws and the bezel comes off. So there you have it, a disk drive replaced in your Model 4D.

RESOURCES:

The TEC OEM manual with lots of technical information can be found here:
https://www.minuszerodegrees.net/manuals/TEC/TEC%20FB-50x%20series%20-%20Technical%20manual.pdf

and the Tandy official 4D Service Manual Addendum can be found here:
https://archive.org/details/m-4-daddendum

Interestingly the vast majority of the 4D Service Manual Addendum is devoted to the disk drives, otherwise there is very little different between the 4D and it's predecessor the 4 Gate-Array version.

CONCLUSION

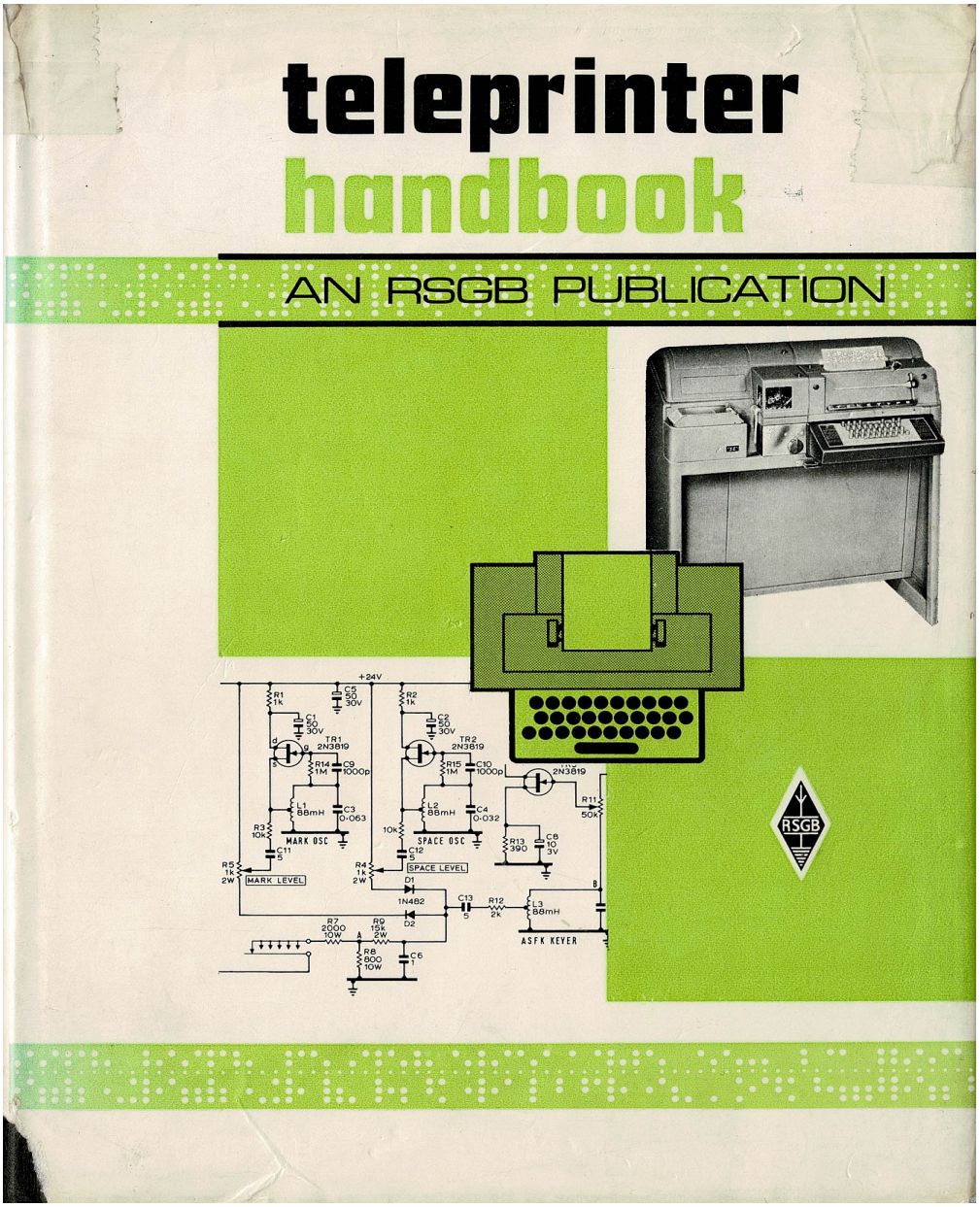
I had fun researching and writing this article and I hope many people get good use out of it. The 4D disk drives are unique in that very few computers used a half-height drive in a full height bay and made it look just right. Anyone can replace a disk drive in a Model 4 or 4D with basically any 40-track single or double-sided disk drives and while it does work, it never looks 100% right when you remove the 4D original TEC drives. With this article you should be able to make those OEM drives last a few more decades.

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.



EBAY.CO.UK 'BARGAINS'

HERE ARE JUST SOME OF THE MORE INTERESTING TANDY EBAY LISTINGS,
SINCE OUR LAST ISSUE. WERE YOU ONE OF LUCKY ONES?
HOW ABOUT LETTING US ALL KNOW HOW YOU'RE GETTING ON WITH YOUR BUY?



£
£13



£98



£135



£1



£90



£28



£36



£230

Using TRS-80 ROM calls from TRSDOS

Vince Otten

Finally! It works!!

For months, I've been trying to test Model I Level II BASIC ROM calls from assembly-language programs written to run from TRSDOS 2.3, but I couldn't get any math code in ROM to work. ROM calls which write to the screen, or accept user input, were okay. For example, this code from James Farvour's "Microsoft BASIC Decoded" worked fine:

CALL 01C9 Clear Screen

Clears the screen, selects 64 characters and homes the cursor. All registers are used.

```
;
; CLEAR SCREEN, HOME CURSOR, SELECT 32 CHAR/LINE
; SKIP 4 LINES
;
      CALL    01C9H       ; CLEAR SCREEN
      LD      A,17H       ; SELECT 32 CHAR/LINE
      CALL    0033H       ; SEND CHAR SIZE TO VIDEO
      LD      B,4         ; NO. OF LINES TO SKIP
      LD      A,1AH       ; CODE TO SKIP ONE LINE
LOOP  PUSH    BC          ; SAVE BC
      CALL    33H         ; SKIP I LINE
      POP     BC          ; GET COUNT
      DJNZ    LOOP        ; LOOP TILL FOUR LINES DONE
```

as did this:

CALL 05D9 Wait For Next Line

Accepts keyboard input and stores each character in a buffer supplied by caller. Input continues until either a carriage return or a BREAK is typed, or until the buffer is full. All edit control codes are recognized, e.g. TAB, BACKSPACE, etc. The calling sequence is: On exit the registers contain:

```
;
; GET NEXT LINE FROM KEYBOARD. EXIT IF BREAK STRUCK.
; LINE CANNOT EXCEED 25 CHARACTERS
;
SIZE  EQU     25          ; MAX LINE SIZE ALLOWED
      LD      HL,BUFF     ; BUFFER ADDRESS
      LD      B,SIZE      ; BUFFER SIZE
      CALL    5D9H        ; READ NEXT LINE FROM KEYBOARD
      JR      C,BREAK     ; JMP IF BREAK TYPED
      .
      .
BUFF  DEFS    SIZE        ; LINE BUFFER
      .
      .
```

However, this useful call would send my TRS-80 off to never-never land:

CALL 0E6C — ASCII To Binary

Converts the ASCII string pointed to by HL to binary. If the value is less than 2**16 and does not contain a decimal point or an E or D descriptor (exponent), the string will be converted to its integer equivalent. If the string contains a decimal point or an E, or D descriptor or if it exceeds 2**16 it will be converted to single or double precision. The binary value will be left in WRA1 and the mode flag will be to the proper value.

Why? I soon realized that ROM routines like the one at 0E6CH somehow jumped out to RAM in the "Communications Region" at 4000H..41EBH (specifically, what Radio Shack called "RESERVED" in their Memory Map in Appendix D of their Level II BASIC Reference Manual). But the math code portions are only set up if we run Cassette (Level II) BASIC or Disk BASIC. How do we get useful access to these subroutines *without* having to load any BASIC?

For those of you who hate the rambling life story that seems to preface any recipe on the Internet, here's the "TL;DR" ("Too Long; Didn't Read") listing of my setup code to do just that. When I want to use math ROM calls from an assembly-language DOS program, I just

```
CALL I00CRM      ; "Initialize..Communication Region Math"
```

```
;       TRS80ROM/INC--FOR MODEL I ROM CALLS
;               FEB 26, 2024
;               TESTED: 2/26/24
;

        TITLE   'TRS80ROM/INC'

;
;       * DEFINES *
;
VECSIZ  DEFL    3*7

INITRM  EQU     I00CRM

;
;       * SUPPORT ROUTINES *

;
;       V4000S -- SAVE VECTOR(S) AT 4000H+
;
V4000S  LD      HL,4000H
        LD      DE,V4000D
        LD      BC,VECSIZ
        LDIR
        RET
V4000D  DEFS    VECSIZ          ; STORAGE FOR VECTOR(S)
```

```
;       V40000R -- RESTORE SAVED VECTORS TO 4000H+
;
V4000R  LD      HL,V4000D
        LD      DE,4000H
        LD      BC,VECSIZ
        LDIR
        RET

;       V4000M -- DISK BASIC DOS EXITS AT
;                       4000H..4012H
;
V4000M  JP      1C96H           ; RST 08 / 4000H
        JP      1D78H           ; RST 10 / 4003H
        JP      1C90H           ; RST 18 / 4006H
        JP      25D9H           ; RST 20 / 4009H
        JP      4BA2H           ; RST 28 / 400CH
        JP      44B4H           ; RST 30 / 400FH
        JP      4518H           ; RST 38 / 4012H

;
;       ** INITIALIZE COMMUNICATION REGION
;               TO ACCESS MATH ROM ROUTINES **
;
;       1. PATCH "RET" FOR DISK BASIC ROUTINES
;
I00CRM  LD      HL,41A6H        ; PATCH 41A6H..41E2H
        LD      B,15H           ; 15X WITH
I0LOOP  LD      (HL),0C9H       ; "RET"
        INC     HL               ; ORIGINAL CODE @
        INC     HL               ; 00A1H
        INC     HL
        DJNZ    I0LOOP
;
;       2. "DIVISION SUPPORT ROUTINE"
;
        LD      HL,18F7H        ; COPY OF CODE @ 0075H:
        LD      DE,4080H        ; PATCH 4080H..40A7H
        LD      BC,27H
        LDIR

;
;       3. PATCH "LD A,(2C00H)" @ 41E5H..41E8H
;
        LD      HL,R00080       ; COPY CODE @ 0080H..
        LD      DE,41E5H        ; PATCH 41E5H..41E8H
        LD      BC,3
        LDIR

;
;       4. PATCH 28 COPIES OF "JP 012DH" INTO
;               4152H..41A5H (DISK BASIC VECTORS)
;               TO PRODUCE "SN ERROR L3" IF
;               THEY ARE CALLED (FROM L2 BASIC)
;
        LD      DE,012DH        ; "012D"
        LD      B,1CH           ; 28 COPIES
        LD      HL,4152H        ; START @ 4152H
L00R96  LD      (HL),0C3H       ; "JP"
        INC     HL
        LD      (HL),E          ; "2D"
        INC     HL
        LD      (HL),D          ; "01"
        INC     HL
        DJNZ    L00R96
```

```
;
;       5. SET "DOS EXITS" AT 4000H..4015H
;               TO THOSE USED BY LEVEL II BASIC
;
        CALL    V4000S          ; A. SAVE EXISTING
                                ; DOS EXITS
                                ; B. INSERT L2 EXITS
        LD      HL,V4000M       ; LIST OF "JP"S FOR MATH
        LD      DE,4000H
        LD      BC,VECSIZ
        LDIR
        RET

;
;       * PATCH CODE *
;
R00080  LD      A,(2C00H)       ; SEE ROM @ 080H
```

Here's a run-through of the initialization process, starting at I00CRM. Steps 1 to 4 duplicate code found in the ROM from 0075H onward. This is code which is not executed if a floppy disk was located and booted.

Step 1: This may not be entirely necessary, but this step patches all Disk BASIC extensions (like "GET" or "DEF FN") with simple RETurns back to the ROM.

Step 2: If any of my ROM disassembly references (like "Microsoft BASIC Decoded" or "TRS-80 ROM Routines Documented" by Jack Decker) refer to the need to initialize math routines in RAM, they'll mention the "Division Support Routine". I'm not sure why Microsoft chose to do this, but it's necessary to copy the 39 bytes of code at 18F7H out to 4080H for the math routines, particularly division, to work.

Step 3: Again, I'm not completely sure this small code insertion is necessary. Level II BASIC initializes 41E5H with a "LD A,(2C00H)" instruction, whereas Disk BASIC does not.

Step 4: See Step 1, above. This initialization ensures that Cassette (Level II) BASIC will report a "SN ERROR" if any Disk BASIC extensions (like "GET" or "DEF FN") are encountered within a BASIC program.

Step 5 is quite different from the first 4 steps. It deals with what many call "DOS Exits". These are links or jumping-off points for the various RSTs, other than RST 0H. A "RST" is a quick jump to one of 8 lower memory locations: RST 0H jumps to 0000H, essentially a reboot of the computer. RST 08H jumps to 0008H. What's there? A "JP 4000H", a direct jump to address 4000H. And what's at 4000H? Why, a direct jump *back into* ROM, "JP 1C96H". The same is true for RST 10H (jumps to 4003H, which jumps back to 1D78H), RST 18H (jumps to 4006H, which jumps back to 1C90H), and RST 20H (jumps to 4009H, which jumps back to 25D9H).

These JPs at 4000H..4009H are set up whenever you boot up a TRS-80 Model I. The JPs at 4000H and 4003H are particularly important, since many math routines use RST 08H and RST 10H.

The remaining "DOS Exits" for RSTs 28H..38H at 400CH..4012H are filled with simple RETurn instructions (0C9H). But they don't have to stay that way. For instance, when Disk BASIC is booted, the first 4 "DOS Exits" at 4000H..4009H remain the same. But now the RETs at 400CH..4012H are replaced with JPs into DOS code (at 4BA2H, 44B4H, and 4518H).

What really threw me for a loop, though, was the fact that the assembly-language development system I was using to test out those nifty Math ROM calls (like 0E6CH, described above) was Microsoft's Editor / Assembler Plus, which I'll refer to as EA+.

I first encountered EA+ around 1980, in a cassette version. This was the first "integrated development environment" or "IDE" that I'd ever encountered. It combined a line-based editor, similar to the one I'd become familiar with in Level II BASIC; a macro assembler which could produce working code right in memory; a symbolic debugger which allowed me to examine code and data with the labels that the assembler had used; and it was all resident in memory at the same time, so it was a cinch to write some code, assemble it, try to run it in memory, and if it failed, to debug it and return to the editor to fix and re-assemble. I fell in love.

I still enjoy that retro atmosphere. These days I'm using a disk-based version of EA+, which allows me to INCLUDE (insert) files like TRS80ROM/INC, listed above) in a master assembly document.

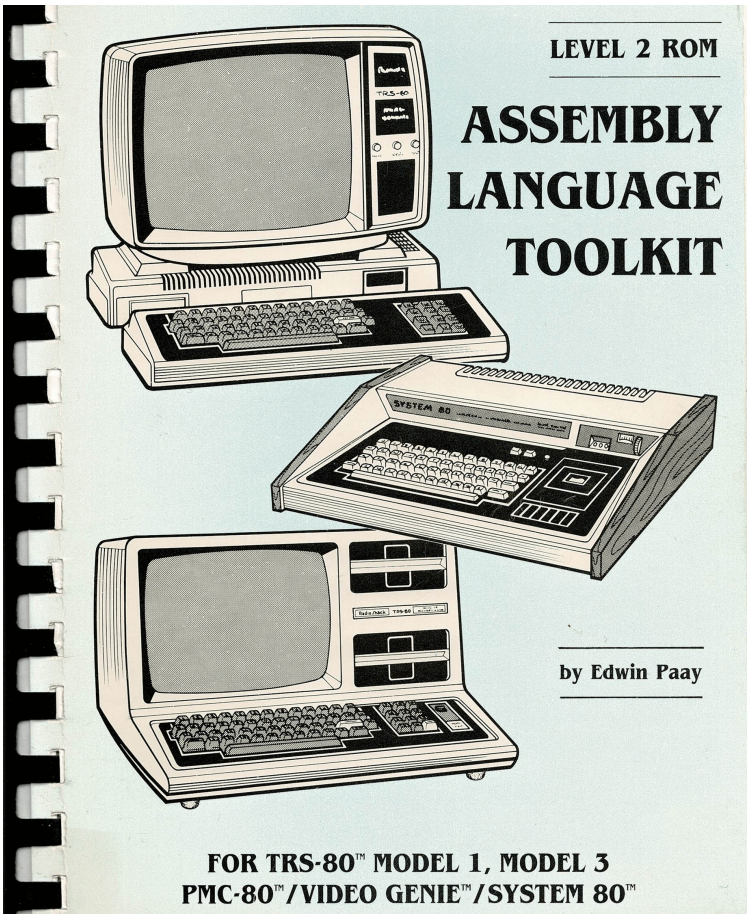
There's only one problem. The Math ROM calls didn't work. Or else, like CALL 0E6CH, they only partly worked. Why?

EA+ re-assigns the JPs at 4003H and 4009H to 4012H!

Step 5 is my solution. Before I CALL any Math ROM code, I

1. Save the current "DOS Exits" from 4000H..4012H to "V4000D" ("Vector at 4000H Data") using CALL V4000S ("Vector at 4000H Save"); and
2. Loading the "DOS Exits" with a complete set of JPs that Disk BASIC uses, "V4000M" ("Vector at 4000H Math").

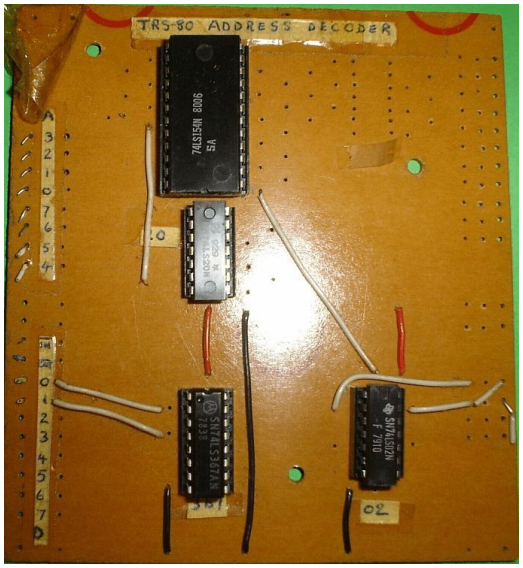
At this point, I'm ready to CALL any Math ROM routines I need. When I'm done with them, or when I'm about to exit my program, I should restore all those DOS Exits to whatever their original JPs were before, using CALL V4000R ("Vector at 4000H Restore").

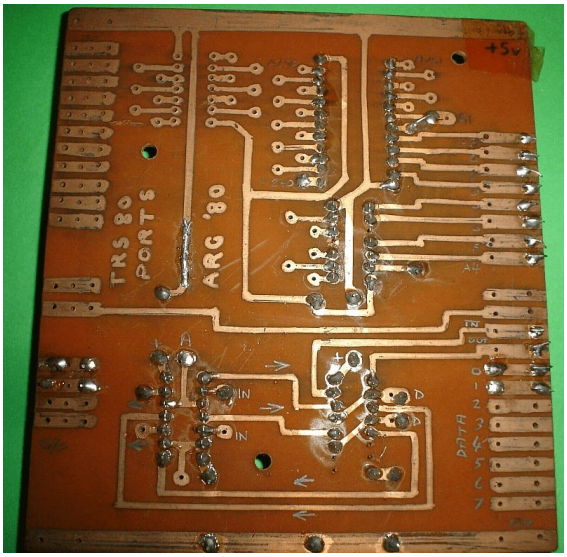


REAL HARDWARE MADE 'BACK IN THE DAY'

TONY G.

HERE'S DETAILS OF A NICE SURPRISE I FOUND ON EBAY. NORMALLY, MOST TANDY RELATED ITEMS ARE SO EXPENSIVE I DON'T GET A LOOK IN. HERE WAS A RATHER CLEVER DECODER BOARD THAT ONLY FETCHED 99P. HERE'S THE POSTED PICTURES AND DESCRIPTION.





Tandy TRS-80 computer address decoder board built in the 1980s
The board contains the following ICs (chips) in IC holders
74LS154N 24 pins
74LS20N 14 pins
74LS367AN 16 pins
74LS02N 14 pins

Etched copper circuit board Size 10 x 11 cm approx.
The 4 IC holders are soldered to the board

The board has been in storage for many years so I cannot guarantee any of the ICs still work.
I have listed it as NOT WORKING because I don't have facilities to test it.

I CONTACTED THE SELLER, TONY, TO EXPLAIN ALL ABOUT TRS8BIT AND HOW I WAS VERY INTERESTED IN ANY TANDY RELATED ITEMS. AS YOU CAN SEE, HE KINDLY REPLIED WITH ALL SORT OF OTHER INTERESTING BITS OF PROJECTS/INFORMATION. HE ALSO INCLUDED INFORMATION OF HIS PUBLISHED ARTICLE FOR A MIDI INTERFACE BOARD AND A BINARY CODED DECIMAL COUNTER WHICH HE HAD PRODUCED.

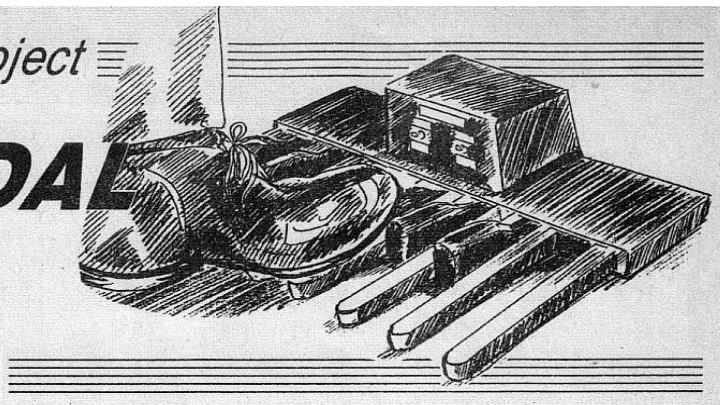
"HELLO DUSTY,
YES, I ETCHED THE PCB MYSELF IT WILL BE THE DATE 1980 AND ARG MY INITIALS. THE TRS80 WAS MY FIRST COMPUTER I BOUGHT IT IN 1980. I WAS A TELECOM ENGINEER BY TRADE SO I UNDERSTAND ELECTRONIC DIAGRAMS ETC. I CANNOT FIND ANY DOCUMENTS FOR THE INTERFACE. I WAS JUST DABBLING IN TRYING TO MAKE THE COMPUTER DO SIMPLE THINGS LIKE LIGHT UP SEQUENCES OF LEDS ETC. IT HAD A 16 BIT ADDRESS BUS AND 8 DATA BITS I SEEM TO REMEMBER. I WAS NOT A MEMBER OF ANY COMPUTER GROUP, I USED

TO BUY MAGAZINES LIKE PC WORLD AND PRACTICAL COMPUTING. I WOULD BE INTERESTED IF YOU GET IT WORKING AND WOULD BE PLEASED TO SEE AN ARTICLE ON IT. I HAD A M1 BACK IN THE DAY WHICH I USED FOR BASIC PROGRAMING.

REGARDS, TONY G"

Constructional Project

MIDI PEDAL BOARD



Take centre stage and make great sounds with this MIDI output pedal – Let your feet do the "talking".

★ *Fully Polyphonic* ★ *Touch Sensitive* ★ *One Octave Range* ★
★ *Standard MIDI Interface Output* ★ *Switchable MIDI Channel* ★

AFTER many fruitless hours searching for an inexpensive add-on pedal board for use with the popular keyboards and synthesisers it was decided a "build-your-own" approach was the only answer. The result is a project that will, it is hoped, appeal to two groups of electronics constructors: those who play the organ and wish to convert a synthesiser into one; and keyboard players who wish to experiment with foot control

This article describes the construction of a one octave pedal board for use with any musical device having a standard MIDI input. For example it can turn a keyboard or synthesiser into a home organ. Also, the pedal unit has been used successfully with a Clavinola, which has some organ sounds.

MIDI PEDAL

The MIDI Pedal Board comprises two sections. One is the mechanical pedal subassembly which is available complete. It has 13 individual pedals, one per semitone.

The other section is the electronic circuit with MIDI output. The circuit is built around the keyboard scanner i.c. type E510. This i.c. is fully Polyphonic, Touch Sensitive and has a selectable MIDI output channel – Channel 1 or Channel 2.

The E510 will scan up to 128 notes by use of a fan-out arrangement of scanning i.c.s 74HC138, but has been reduced to 13 notes here. (See Fig. 2)

The circuit operates at five volts, so a stabilised power supply is included. (See Fig. 3). This allows a general purpose 9V to 12V unregulated power supply to be used.

The unit draws only a small current when in use (20mA), so one of the popular battery-eliminator type power supplies giving +9V to +12V output is adequate. Alternatively, the unit could be powered from a small +9V battery inside the box.

CIRCUIT DESCRIPTION

The full circuit diagram for the MIDI Pedal Board is shown in Fig. 2. The E510 keyboard scanner IC1 can scan up to 128 keys by means of the 7-bit address bus A0 to A6 (pins 7 to 1), with a resolution of 256µS. Details of the pinout information for the E510 are given in Fig. 1.

A large address decoder would be required to decode all 128 addresses. The arrangement of IC2, IC3 and IC4 decodes 13 adjacent addresses for scanning the 13 pedal contacts of the one octave pedal sub-assembly. A two stage decoder i.c. type 74HC138, using 3-bit binary to 1-out-of-8 decoder function, is used for IC2, IC3 and IC4.

ADDRESS DECODER

The four most significant bits (MSBs) of the address from IC1 (A3-A6) are fed to the data inputs of IC2, pins 1 to 4. See Fig. 1. There are three data leads, plus Enable ($\overline{E1}$) lead which acts as a fourth data lead.

The output of IC2 is an active-Low signal on one of the eight output leads, pins 7, 9 to 15. These signals are used to select each of the secondary decoder i.c.s in turn (IC3 and IC4). The three least significant bits (LSBs) of the address from IC1 (A0 to A2) are fed directly to the secondary decoders IC3 and IC4.

Links L1 and L2 transfer the Enable signal to IC3 and IC4, see Note Range Table 1. This allows selection of the musical range of the single octave.

IC3 and IC4 decode the three LSBs of the address to allow scanning of eight pedal contacts per i.c. IC3 can therefore scan the first eight contacts of the pedal unit, corresponding to the first eight semi-tones C to G. IC4 scans the next eight semi-tones, but only five are required to complete the octave, namely G# to C.

KEYBOARD OUTPUT

Every time a key is pressed or released, the velocity is calculated by the E510 keyboard scanner (IC1) using a 7-bit reverse counting timer. At the same time every press and release is tested for validity. The key is only recognised if it goes from pin 10 to pin 11 or vice-versa, and thus key bounce is eliminated.

An internal FIFO register (First-In-First Out type of storage device) allows fully polyphonic playing and thus several keys can be pressed simultaneously. The MIDI data output on pin 9 transmits data according to the MIDI specification; key number, key velocity, key ON or OFF, channel number.

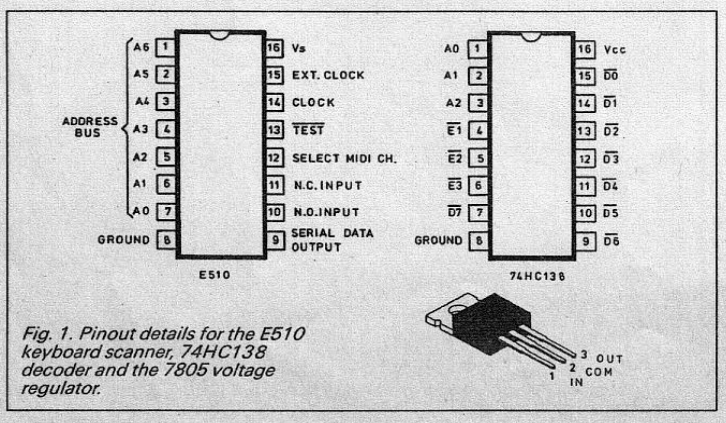


Fig. 1. Pinout details for the E510 keyboard scanner, 74HC138 decoder and the 7805 voltage regulator.

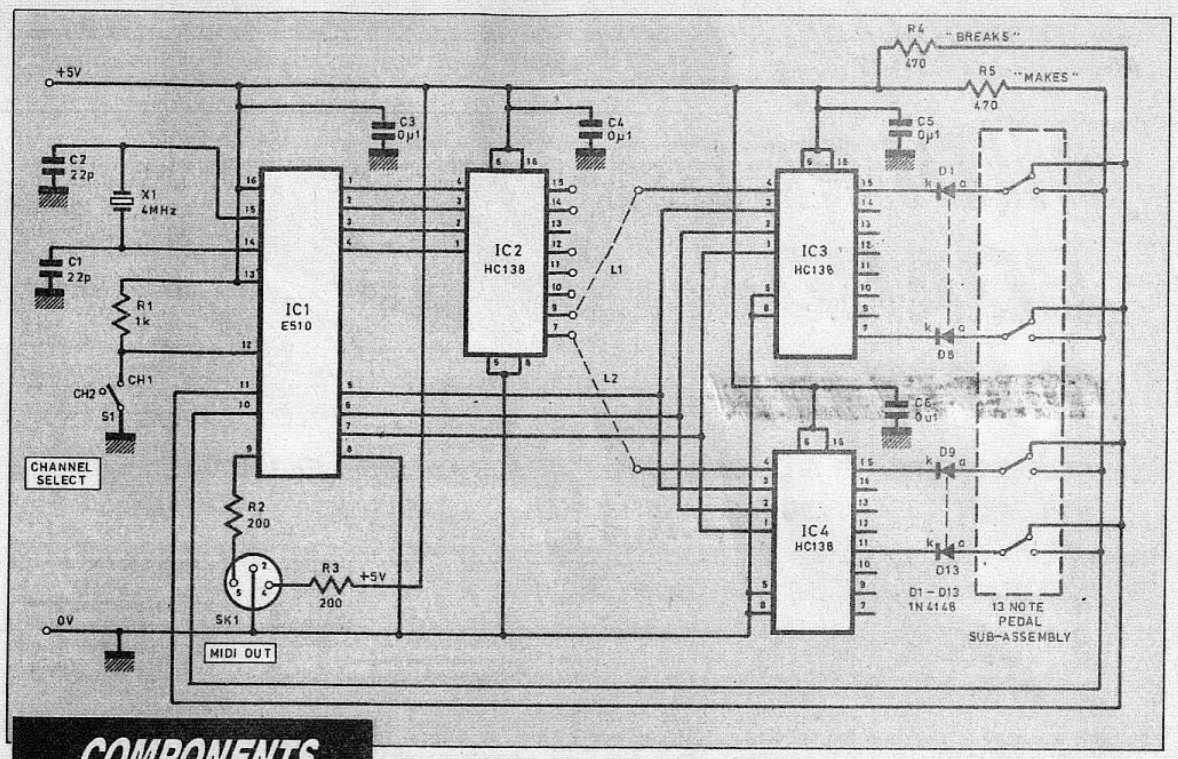


Fig. 2. Full circuit diagram, except voltage stabliser, for the MIDI Pedal Board. MIDI: Musical Instrument Digital Interface.

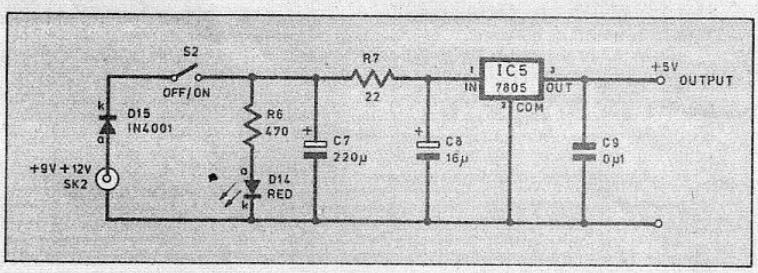
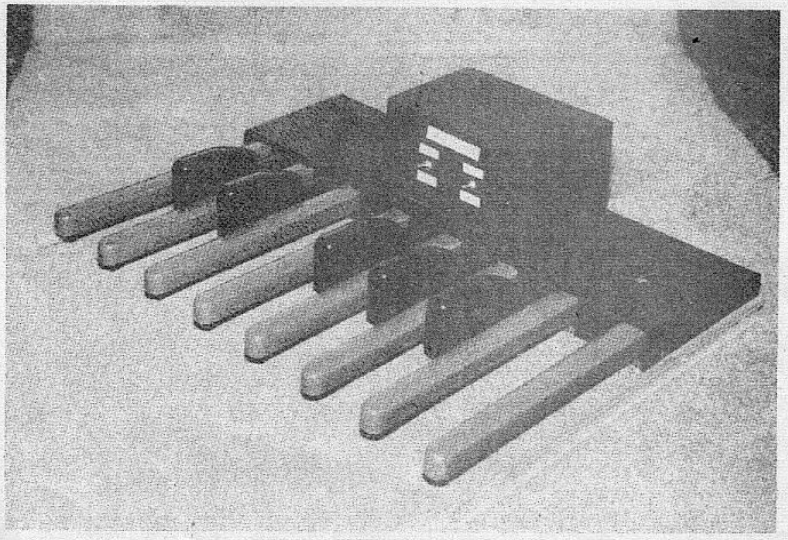


Fig. 3. Circuit diagram for the power supply voltage stabiliser. The unit drains only a small current, about 20mA, so a 9V to 12V battery eliminator or a 9V battery can be used as the power source.



The completed Pedal Board with the "Scanning" unit mounted on top of the made-up pedal housing.

COMPONENTS

See SHOP TALK Page

Resistors

R1	1k	
R2, R3	200	(2 off)
R4, R5, R6	470	(3 off)
R7	22	

All 0·6W 1% metal film

Capacitors

C1, C2	22p ceramic (2 off)
C3 to C6, C9	0µ1 min. disc ceramic (5 off)
C7	220µ radial elect. 16V
C8	16µ bead tantalum, 16V

Semiconductors

D1 to D13	1N4148 signal diode (13 off)
D14	3mm or 5mm red l.e.d.
D15	1N4001 rec. diode
IC1	E510 MIDI keyboard scanner
IC2 to IC4	74HC138 3-bit binary to 1-in-8 decoder (3 off)
IC5	7805 +5V 1A voltage regulator

Miscellaneous

X1	4MHz crystal
S1, S2	s.p.s.t. min. toggle switch (2 off)
SK1	5-pin 180 degree DIN socket
SK2	d.c. power socket, 2·1mm

Plastic case, size 176mm x 120mm x 84mm approx; 0·1in. stripboard, 39 strips x 62 holes; pedal sub-assembly, with changeover contacts – see text; 16-pin d.i.l. socket (4 off); coloured multistrand connecting wire; aluminium or wood for legs and pedal cover, cut to fit.

Approx cost guidance only **£50** *excluding pedal board*

TRS8BIT - VOLUME 18 ISSUE 01 - MARCH 2024 - PAGE 81 - TRS8BIT

ELECTROSTATIC PRECAUTIONS

All semiconductor devices can be degraded or even destroyed by the action of electrostatic discharge (ZAP!) from the human body. Much has already been written on this subject, so only brief details of recommended precautions are given:

Use full ESP kit – conductive bench mat, wrist band, cord. or

Try to prevent high electrostatic charge in the work area. Work on an Earthed metal surface. Do not wear any synthetic materials which cause a build-up of static like rayon and polyester.

CONSTRUCTION

Construction should commence with the circuit board which is a piece of 0·1in. matrix stripboard, size 39 strips by 62 holes. The topside component layout and details of breaks required in the trackside copper strips is shown in Fig 4.

First place and solder all the passive

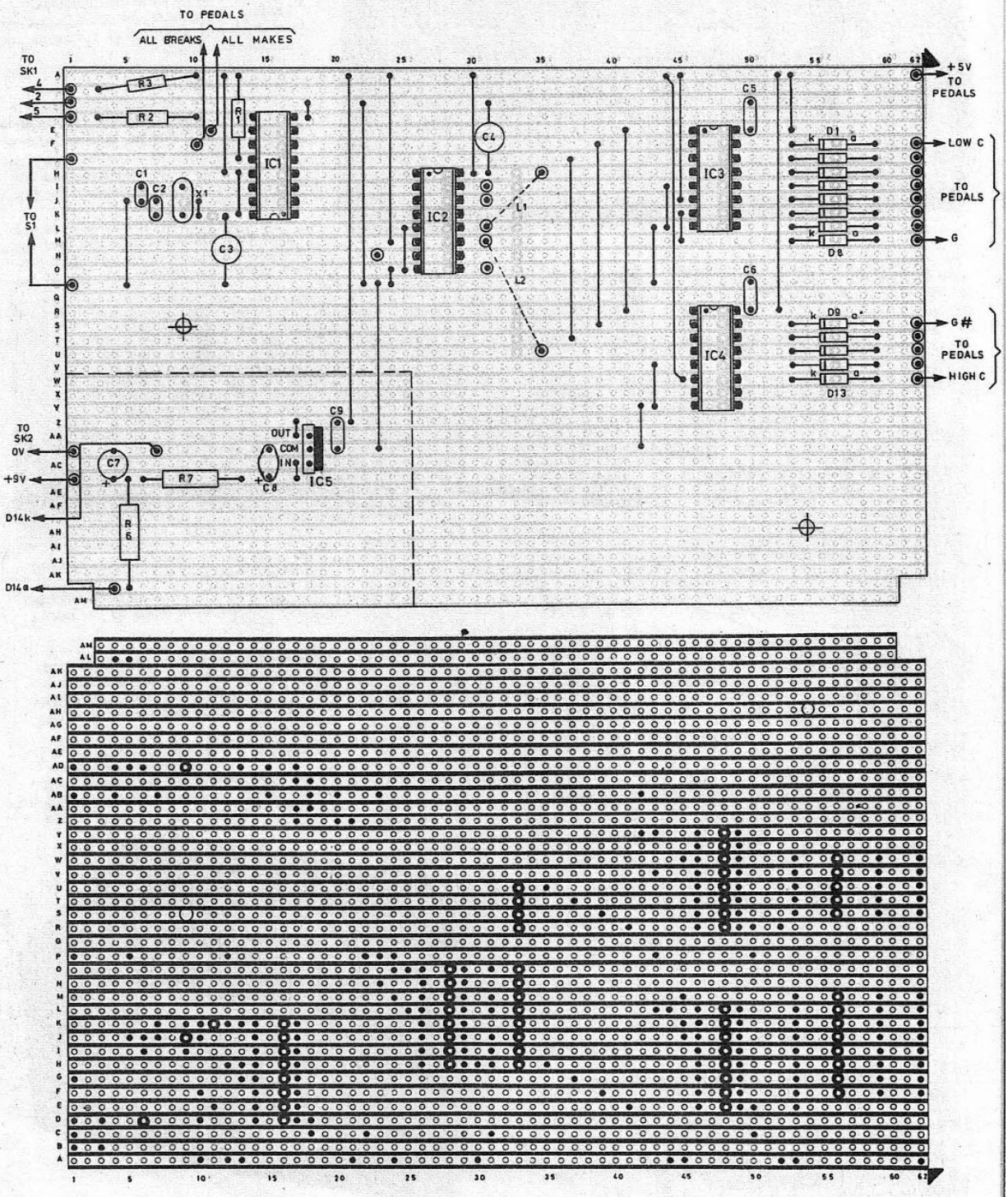
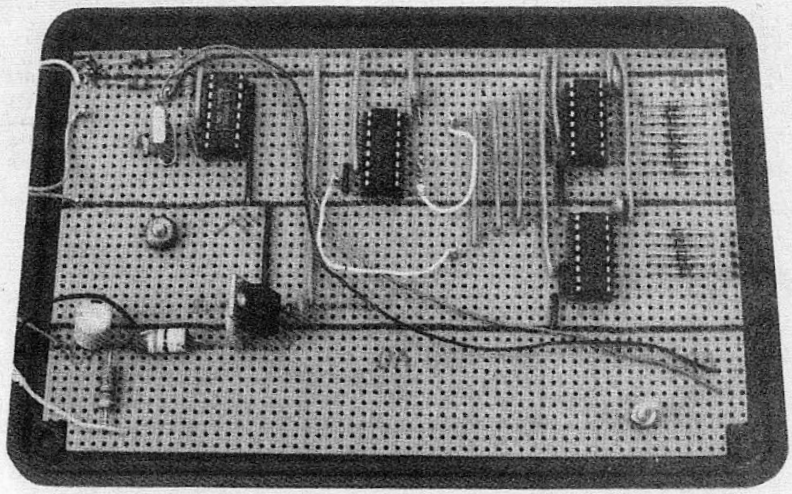


Fig. 4. Stripboard topside component layout and details of breaks required in the underside copper strips. The circled connections on the topside layout represent solder pins.

TRS8BIT - VOLUME 18 ISSUE 01 - MARCH 2024 - PAGE 82 - TRS8BIT



The complete prototype circuit board mounted on the underside of the case lid. The two leads that go from the top left to the right-hand edge are so that all connections to the keyboard come from the same side of the board.

components – i.c. sockets, resistors, link wires etc. Also, set the two jumper "note" links L1 and L2 for the required note range using Table 1. Then cut the underside tracks as indicated.

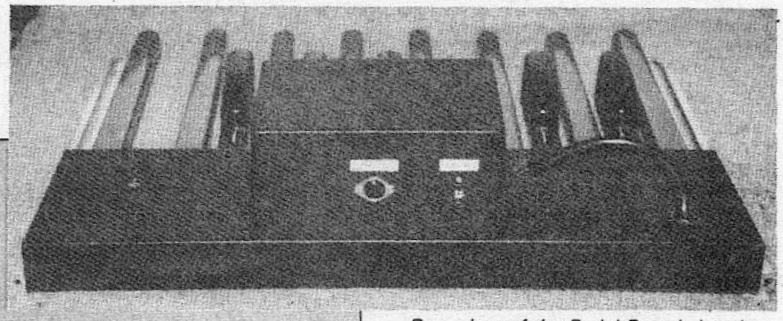
Next, fit the trackside decoupling capacitors C3 and C4 to IC1 and IC2 pins is as shown. Fit diodes and crystal, then lastly insert the i.c.s into their holders.

Before fitting i.c.s you could power-up and check with a meter that +V is only on the following pins:

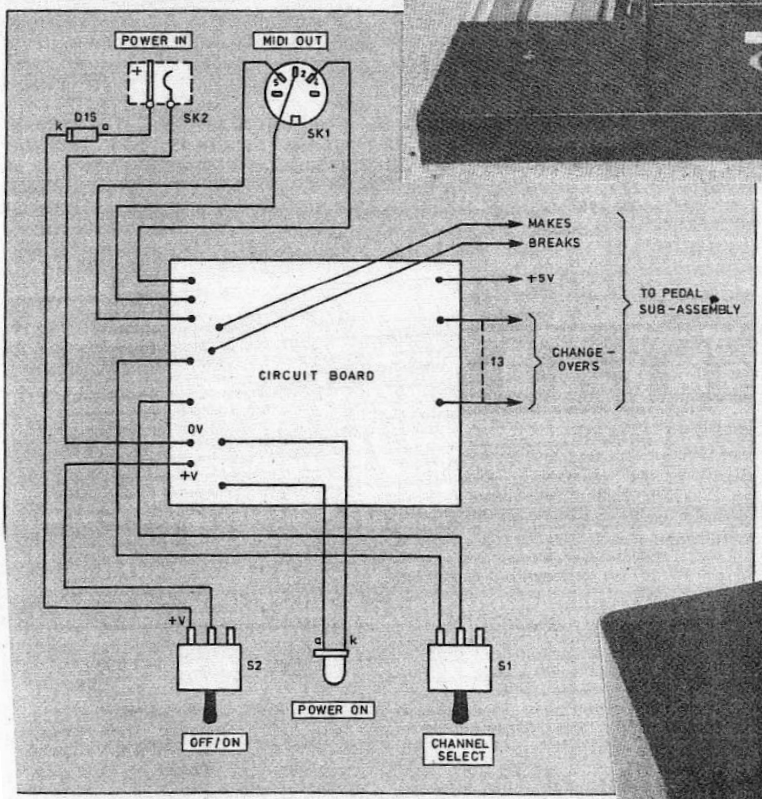
IC1 pins 13 and 16; IC2, IC3 and IC4 pins 6 and 16.

Visually check the board for faults. The board should then be fitted onto the underside of the plastic box lid, the sockets and switches being fitted in the sides of the box. A 16-way cable was brought out of the box to wire to the pedal sub-assembly. See Fig. 5 and photograph.

On the front of the plastic box are the On/Off and CH1/CH2 switches, and the Power On l.e.d. The user may wish to consider placing the switches on one of the other faces of the box, to prevent accidentally catching them with the foot when in use. On the back of the box is the MIDI output DIN socket, and the +9V power input socket.



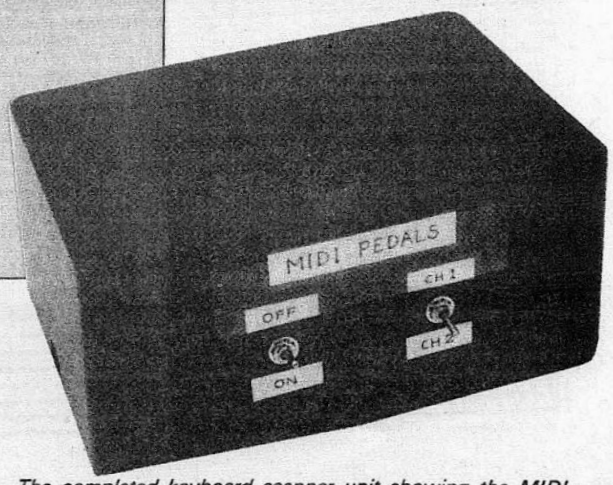
Rear view of the Pedal Board showing the MIDI input and the Power-In sockets. Note also the 16-core cable sleeving, taking the note leads to the pedal sub-assembly.



g. 5. Interwiring from off-board components to the circuit board

PEDAL ASSEMBLY

The pedal sub-assembly requires two stabilising legs, one on either side to enable the unit to be free-standing. These should



The completed keyboard scanner unit showing the MIDI Channel select switch and On/Off switch. These switches could be mounted on one of the side panels, away from any "flying feet."

Table 1: Jumper connections for Note Ranges

Link L1	Link L2	Note Range
IC2/15 – IC3/4	IC2/14 – IC4/4	C-2 to C-1
IC2/12 – IC3/4	IC2/11 – IC4/4	C0 to C1
IC2/9 – IC3/4	IC2/7 – IC4/4	C2 to C3

lote: Some keyboards do not accept the lowest note range, but should accept the highest range which is the octave just below dle C. (Middle C = C3).

TRS8BIT - VOLUME 18 ISSUE 01 - MARCH 2024 - PAGE 83 - TRS8BIT

be fashioned from aluminium or wood approximately 356mm × 25mm × 12mm – see Fig. 6. Likewise a cover should be made from sheet aluminium or plywood to cover the contact body.

On the pedal sub-assembly, all the "make" contacts are strapped together, and all the "break" contacts are also strapped together as shown in the circuit diagram Fig. 2. The +5V supply is connected, via resistors R4 and R5, to the strapped-up contacts. The wire for each pedal from diodes D1 to D13, are connected to the "changeover" contacts. (D1=Low C, D13=High C). Once the wiring has been completed, the box containing the electronics should be securely fastened to the body of the sub-assembly.

SETTING UP

Connect the MIDI output to the MIDI input of your keyboard using a MIDI cord. Power-up the Pedal Unit and the Keyboard. Press each pedal and confirm that the correct note is heard. Pressing quickly should give a louder note due to the touch sensitive processing in the E510.

The E510 IC1 has an internal scanner which scans each pedal in turn, via IC2, IC3 and IC4, and diodes D1 to D13. The contact transit time, and hence the pedal speed, is measured by IC1 pins 10 and 11.

Thus the harder you press the pedal, the faster it moves and so the louder will be the note velocity signal generated by the E510. It is important that the contact gaps be all the same so that the volume of each pedal is the same, see Fig 7.

When a pedal is pressed, the E510 sends MIDI NOTE-ON and VELOCITY data for that note on the selected MIDI CHANNEL. The data is then converted into a musical note by the synthesiser in the remote keyboard.

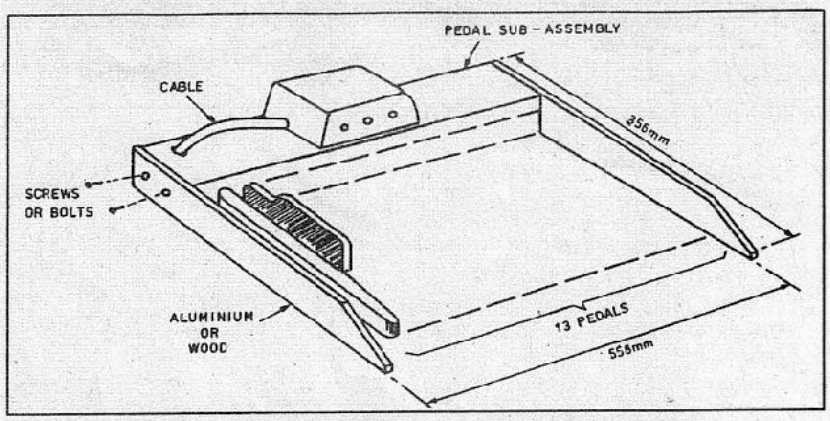


Fig. 6. Suggested method of housing the pedal sub-assembly.

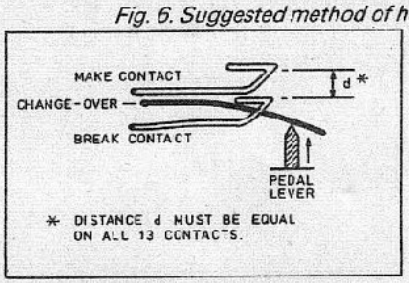


Fig. 7. Pedal switch contacts.

MIDI COMPATABILITY

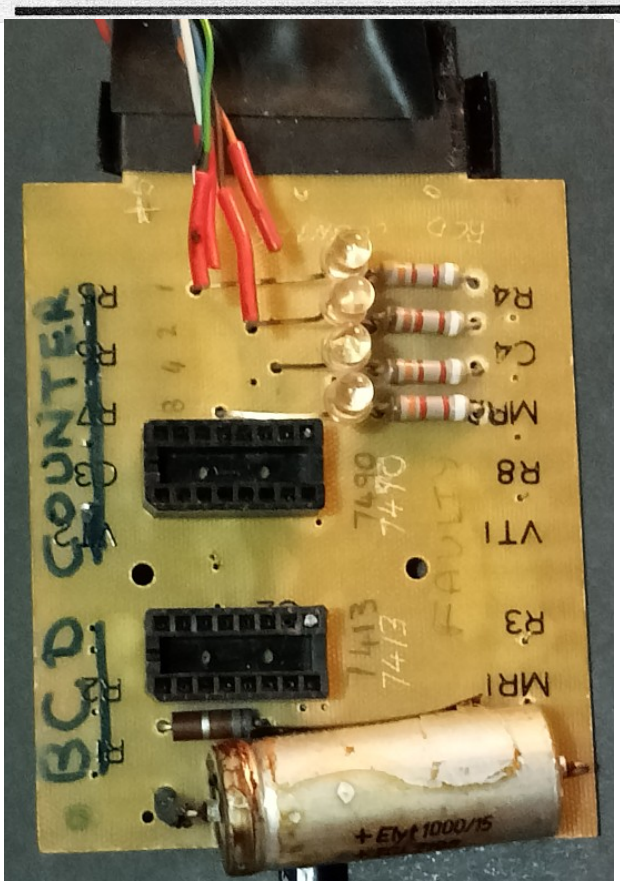
The MIDI Pedal Unit sends data on MIDI Channel 1 or Channel 2, depending upon the setting of Channel Select S1. Many keyboards default to OMNI when first switched on, this means that they will accept MIDI data on any channel. You can usually select which channel you want to receive MIDI data on, consult your keyboard manual for how to do this.

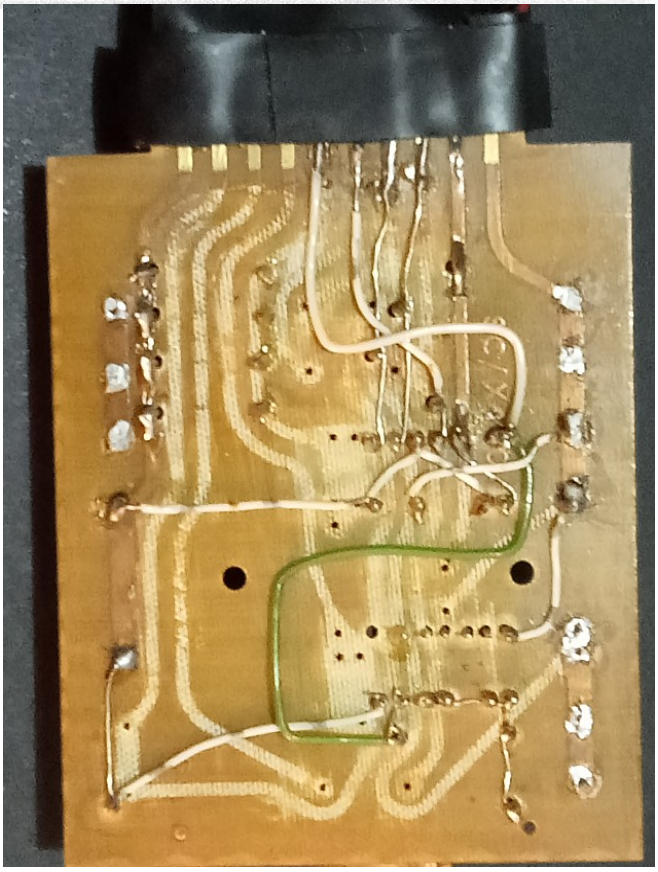
Depending on the type of keyboard or synthesiser, you may be able to control two individual voices: for example the keyboard on Channel 1 playing a solo voice, and the pedal-board on Channel 2 playing a bass voice. Your keyboard manual should explain how to do this.

TROUBLE SHOOTING

If you do not hear any sounds on your keyboard when you press the pedals, switch off the power and re-check the circuit board for errors, touching tracks, solder splashes, dry-joints etc. Check pedal wiring.

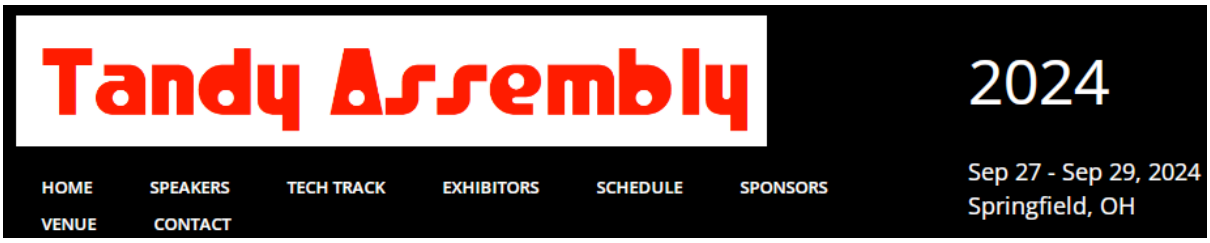
Check that the jumper links L1 and L2 are appropriate for your keyboard. Check also that your keyboard MIDI receive is set to OMNI or Channel 1 or Channel 2. Check that any other parameters are set correctly on your keyboard to enable data receive, e.g. MIDI REC=ON – refer to your keyboard manual for further information. □





MAV'S TANDY ASSEMBLY QUARTERLY COLUMN
TA24 IS 6 MONTHS AWAY

by Ian Mavric



In this new column which will appear in each issue of TRS8Bit I will be talking about the yearly Tandy Assembly event, typically held in late September/early October each year in Ohio, USA. I'll update with any information or advice I think any potential attendees would like to know. For those handful of people who don't know yet, I am on the TA committee, so part of the tasks I have given myself is to spread the word of the show including in social media. Check in on this column quarterly to stay up to date.

TA24 - THE DATE AND VENUE HAVE BEEN SET

As you can see from the picture from the website at the top of this article, the date has been set to September 27-29 this year, and it's being held at the usual place, which is the Courtyard by Marriott in Springfield, Ohio. Just like previous years we have a block of rooms set aside for us at the Marriott at a cheaper rate than normal. These rooms do sell out, and this year the word on the street is that they are selling faster than in previous years. I would call them sooner rather than later. In previous years it was possible to wait until late May or early June and still get a room, but I would not wait until then. You can call them on +1 937-322-3600 and tell them you need a room for the Tandy Computer Convention.

The format of the show will be the same as previous years so Friday will be Tandy Tech Track classes, Saturday the show with keynote speakers, the auction and trivia game, and Sunday more speakers and finishing of the show.

The sponsors are all starting to get their donations in and are appearing on the Sponsors web page. If you would like your Tandy/TRS-80-related company on that page and help out the show it's only $US50.00 and info is on the website.

2024 SPONSORS



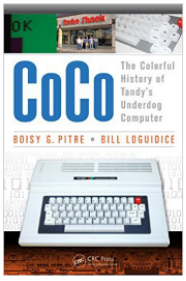
THE RIGHT STUFF (TRS)

FreHD, Hi-Res graphics, Quinnterface, adapters, cables, memory upgrades, kits for the Model I/III/4/Coco

TandyShowcase.com

EDDIE MALPHRUS

Tandy Collector



BOISY PITRE

CoCo historian and author



BLAIR ROBINS

Enthusiastic supporter of Tandy Assembly and proud member of the thriving TRS-80 community.



JAY NEWIRTH

newSOFT



TRS8BIT

The World's Best TRS-80 Newsletter



VAL SEE & KEN BROOKNER

Upgrading the Past



JOHN ELLIOTT IV

Former Tandy R&D Engineer



TANDY EMERITUS

Developers of New Hardware and Software for Tandy 8" Systems



TRS-80 TRASH TALK PODCAST

TRS-80 Trash Talk is the show about all things TRS-80.



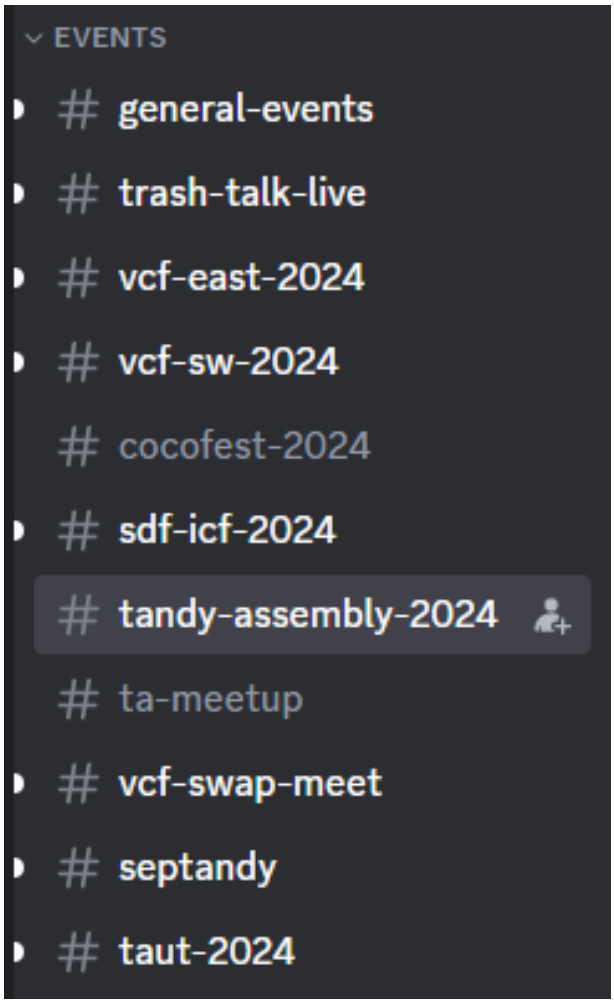
COCO CREW PODCAST

A delicious adventure into the world of retro-computing news and information, featuring the Tandy Color Computer.

THE FACEBOOK AND DISCORD TANDY ASSEMBLY GROUPS

Did you know Facebook has a Tandy Assembly group? Yep we're that up to date we even have a presence on FB. Joking aside it has been there for a while, but since joining the team I have taken it upon myself to post a weekly update, usually each Sunday, of anything I think newsworthy about the show, or a look back at some aspect of a previous TA event. They have all been recorded so I invite you to a walk down memory lane and re-live some of the highlights of previous TA. You can find the FB page here:
https://www.facebook.com/tandyassembly

Like most other vintage computer platforms the Tandy TRS-80 is supported on the great 'Tandy' Discord group, and it has its very own section on Tandy Assembly 2024, among other events. Discussion and questions/answers in real time is one of it's features.



The Taut-2024 section you see at the bottom is for those who arrive a day early to Tandy Assembly, so is about what activity we want to do the day before the show. This is on an official part of Tandy Assembly but is a *traditional* part. Many people who travel across multiple time zones like to have something to do to kick the jet-lag before the show starts.

FOR THOSE WHO CAN'T WAIT 6 MONTHS

VCF-SW - the SW stands for South West - is being held in Dallas, TX on June 14-16 and Tandy Assembly will have a gathering called TA Meetup, a smaller sub-event different from the normal Ohio Tandy Assembly. A group of tables in an area has been set aside for exhibits of TRS-80 computers. It should appeal to those who live closer to Dallas than Springfield OH and find coming to OH limiting.



You can get more information from their website: https://www.vcfsw.org/

Speakers planned for VCF-SW already includes Tandy Assembly regular Boisy Pitre who talks about porting OS/9 to a new platform Foenix F256 and Marcel Erz who will talk about Digital Electronics and KiCAD. The name is familiar because he was the genius who did the re-implementation of the TRS-80 Model I main board and keyboard recently, however this is only one of many vintage computer re-implementations he has completed.



BOISY PITRE

Porting OS-9 to the Foenix F256 - Friday @ 3:15pm

The Foenix F256 is a new retro computing platform for 6502 and 6809 enthusiasts. In this presentation, I talk about how I approached porting the OS-9 operating system to this exciting system.

About the presenter:
Boisy's retro-computing interests lie primarily in the home computers of the 1980's, notably the Tandy Color Computer (aka CoCo). Along with Bill Loguidice, Boisy co-authored the book ''CoCo: The Colorful History of Tandy's Underdog Computer'' (2013), published by Taylor & Francis.



MARCEL ERZ

Introduction to Digital Electronics: From Schematics to Circuits - Saturday @ 10:15am

Join me for an immersive crash course in digital electronics, where you'll learn to read simple schematics and assemble digital circuits on a breadboard. This workshop is geared toward beginners who want to extend their vintage computers. It offers hands-on experience with standard IC components and breadboards, covering the fundamentals of digital circuit design. Materials will be provided.

Mastering PCB Design with KiCAD: From Concept to Creation - Saturday @ 2pm

Dive into the world of PCB design with KiCAD in this workshop, which is geared towards beginners. From crafting schematics to generating PCB layouts and ordering prototypes, you'll gain practical skills applicable to vintage computing and beyond. Templates for schematics for a few vintage computer extensions (e.g. memory extensions) will be provided.

de

About the presenter:

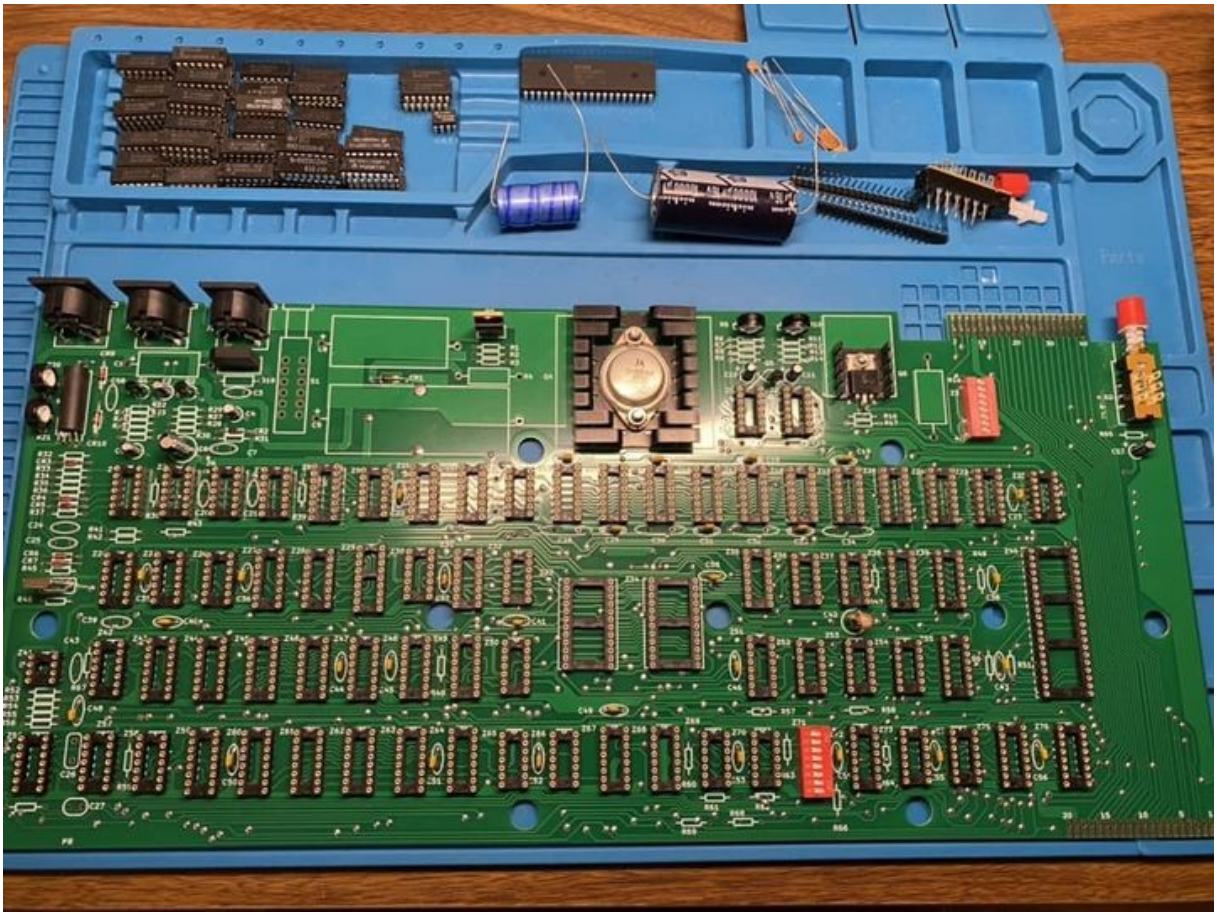
While a software engineer by profession, Marcel has always been deeply interested in electronics, specifically digital electronics. In recent years, he has devoted time to explore the low-level implementation of computer systems. Over the last few years, he began analyzing schematics of vintage computers to understand their workings and developed extensions for them, including for the Apple 1 and Ti99/4A. Most recently, he began open-sourcing some of his latest creations on GitHub, which includes a reimplementation of the TRS-80 Model 1 and a disk controller for the Exidy Sorcerer.

Keep an eye out for my next update in the June issue of TRS8Bit.

Ian Mavric
ianm@trs-80.com



Surfin' The Net



Here's an interesting major project, Marcel Erz has done the complete repro of the Model I motherboard and keyboard PCBs. A massive amount of work that I am super impressed with. If you always wanted to build your own Model I this drops into the original plastic enclosure.

For more information you can read up on it here:
https://github.com/RetroStack/TRS-80-Model-I-G-E1

Question? What do you think would be a great piece of TRS-80 hardware to repro next and why? - Mav.



"RETRO REPAIR GUY" PERFORMS A COMPLETE RESTORATION OF A TRS-80 MODEL 100 VINTAGE LAPTOP FROM RADIO SHACK WHICH WAS ONE OF THE FIRST COMPUTER LAPTOPS FROM 1983. FOR THIS RESTORATION HE REPLACES ALL THE CAPACITORS AND REPLACED THE BATTERY WITH A SUPER-CAPACITOR.

IF YOU'VE A SPARE 30 MINUTES, IT'S WELL WORTH THE TIME! ED



HTTPS://WWW.YOUTUBE.COM/WATCH?V=10C5PIS5HJW&T=1057S



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

http://members.iinet.net.au/~ianmav/trs80/





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

WHERE 8 BITS AND 16K STILL RULE OK

TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

JUNE 2024 - VOL.18 ISSUE 2

PAGE 02 MAV'S T.A. COLUMN - LATEST NEWS
CHECK-OUT THE 'SPECIAL DEAL' FOR CO-CO FANS

PAGE 06 **2024 COMPETITION DETAILS.**

PAGE 058 IN MAV'S WORKSHOP
IAN MAVRIC
SORT CO-CO 3 KEYBOARDS

PAGE 14 BASIC DIALECTS
JIM ADAMS
JIM HAS COMPILED A DETAILED LIST OF 18 VERSIONS.

PAGE 33 FULL CARD SHUFFLE - BASIC ONE-LINER
E.T. FONEHUME

PAGE 40 TRS-80 M4GA
JOHN MINEUR

PAGE 48 IN MAV'S WORKSHOP
IAN MAVIC
PRINTER PROBLEMS WITH A MODEL 1

PAGE 56 APRIL BASQUE TRS-80 TOURNAMENT
EGOITZ CAMPO

PAGE 59 SOFTWARE TOOLS - EPISODE 4
VINCE OTTEN
'GET CRACKING'

PAGE 83 THE WAY WE WERE
THE LATEST IDEAS FROM 'IN THE DAY'

PAGE 92 U.K. EBAY BARGAINS
WERE YOU ONE OF THE LUCKY ONES?

PAGE 95 SURFIN' THE NET
MORE NEW AND LEADING TECH IDEAS FROM YOUTUBE

PAGE 99 WHERE ARE THEY NOW?
REMEMBERING LEADING FIGURES FROM 'IN THE DAY'

PAGE 102 'AMAZING' FUN-TIME PUZZLE - JUST FOR A BREAK!

PAGE 103 EDITOR ASSEMBLER PLUS
VINCE OTTEN

PAGE 116 IN MAV'S WORKSHOP
IAN MAVRIC
MODEL II MOSTEK MADNESS!

MAV'S TANDY ASSEMBLY QUARTERLY COLUMN

TA24 NEWS

by Ian Mavric



Brining you the latest news you need to know about the greatest Tandy TRS-80 show on the planet. This time I talk about planned improvements to the trivia game, and a special offer for Coco enthusiasts who went to CocoFEST on May 4-5, 2024.

CALL FOR SPEAKERS AND TRIVIA GAME CONTESTANTS

As the business end of the year since TA23 fast approaches we extend the invitation to any and all who would like to do a talk at TA24 on any Tandy/TRS-80 topic. Doesn't matter what it is, we will find a place for it. The audience are very receptive too and look forward to what you have to say. Technical tutorials, history lessons, debuts of new software or hardware, everything is welcome here at Tandy Assembly, and it's the best place to introduce your new idea, and get helpful feedback from the masses to turn it into a reality.

Join the ranks of famous people who have spoken at previous Tandy Assembly events like Don French, Scott Adams, Paul T. Schreiber, Vernon Hester and myself.

CHANGES TO THE TRIVIA GAME

Evolving each year since I have taken over hosting the Saturday night trivia game I have always been interested in ideas to improve the game so I will be outlining the two which will debut this year.

1. "Lifeline": Is an idea to allow each of the three contestants to effectively "Phone a friend" in the audience. If a contestant answers a question and get's it wrong they may elect to use their Lifeline, to ask someone in the audience who might know the answer, to answer for them. Each contestant get's one Lifeline, and if the person they choose also gets the answer wrong... well that's embarrassing.

2. "Expert Rounds": Is an idea I got from the Australian TV trivia show "Hard Quiz", where each contestant has an Expert Round. Previously I have been advised about each contestant's field of expertise, and I come up with a set of questions based around that field. If the contestant gets it right they get 1 Point, but if one of the other two contestants buzzes in and gets it right the winner gets 2 Points. This may seem a little unfair however each of the three contestants has their own Expert Round so plenty of opportunity to steal points from the other two players. TIP: choose a very niche area of expertise in the Tandy/TRS-80 world to try to stump your opponents and limit their ability to steal points from you ;-)



So if you want to be on the stage for the Trivia Game this year reach out to me and let me know your area of expertise for your Expert Round questions, and you will be forever remembered with admiration, or infamy...

WE TAKE A MOMENT TO THANK OUR SPONSORS

The sponsors are all starting to get their donations in and are appearing on the Sponsors web page. If you would like your Tandy/TRS-80-related company on that page and help out the show it's only $US50.00 and info is on the website.

2024 SPONSORS



THE RIGHT STUFF (TRS)

FreHD, Hi-Res graphics, Quinnterface, adapters, cables, memory upgrades, kits for the Model I/III/4/Coco



EDDIE MALPHRUS

Tandy Collector



BOISY PITRE

CoCo historian and author



BLAIR ROBINS

Enthusiastic supporter of Tandy Assembly and proud member of the thriving TRS-80 community.



JAY NEWIRTH

newSOFT



TRS8BIT

The World's Best TRS-80 Newsletter



VAL SEE & KEN BROOKNER

Upgrading the Past



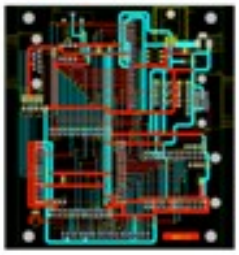
JOHN ELLIOTT IV

Former Tandy R&D Engineer



TANDY EMERITUS

Developers of New Hardware and Software for Tandy 8" Systems



BARTLETT LABS

Unique and powerful expansion devices for TRS-80 microcomputers.



PRETTY IN PINK

Vintage computers, but pink!



SEBHC

Society of Eight-Bit Heathkit Computerists



OCF RETRO COMPUTING

The OCF Network. Rocking the Web like it's 1995.



TRS-80.COM

Ira Goldklang's TRS-80 Revived Site



TRS-80 TRASH TALK PODCAST

TRS-80 Trash Talk is the show about all things TRS-80.



COCO CREW PODCAST

A delicious adventure into the world of retro-computing news and information, featuring the Tandy Color Computer.

A SPECIAL OFFER FOR CocoFEST 2024 ATTENDEES

I'm always looking for ways to grow the show. Did you know that every year since it's inception in 2017 the attendance to the show has increased, except for 2021 (which was due to the pandemic lockdowns excluding many people travelling long distances, and from Canada and other countries). Tandy Assembly prides itself on being the biggest and best Tandy/TRS-80 related show on the planet and I have deduced many people from the Color Computer world go to only one fest a year (CocoFEST), often citing budgetary limitations.

The Tandy Assembly Team discussed a way to grow the show and get more people along, and a novel way was to invite free entry for those who took the trouble to attend the 32nd "Last" CocoFEST which was held on May 4-5, 2024. All people need to do is bring along their door pass from CCF and show it as the entry to Tandy Assembly and you are in, free of charge. They look like this:



Photo: one of at least two CocoFEST 2024 door pass designs

I like to think of it as getting two great festivals for the price of one. Coco inclusion has been increasing at Tandy Assembly and the size of the event is set to also increase in the upcoming years. Being in the presence of other Tandy/TRS-80 platforms such as the monochrome machines and now the MS-DOS boxes. With so many different platforms even Coco people will be able to see what others are doing and apply some creative endeavours to the Coco world.

Keep an eye out for my next update in the September issue of TRS8Bit. It will be a big one since the show will be less than 30 days away from the release of September TRS8Bit.

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.

TRS8BIT 2024 Competition

Beat “Defense Command”

The objective is to write a program that can play the best game of Big Five Software's “Defense Command”. The program with the highest average score over a series of attempts will win. In the case of ties (e.g., programs can play indefinitely or are not obviously superior to each other) the earlier entrant will be declared the winner.

The programs themselves will be run externally to the emulator running the game. The positions of all game objects and other status information will be sent to an entry program which will respond with updated controller inputs to move the player's ship. The setup will simplify testing and allow competitors to write their program in any language including BASIC.

There are two categories and hence two prizes to be won. The limited category is for programs that run on a TRS-80 itself. It can be any TRS-80 emulated by trs80gp – Model 1, 2, 3, 4, 16, Colour Computer, etc. The unlimited category allows any program you like as long as it can be tested by the competition organizers. Ideally some form of Windows executable or as a service accessible “in the cloud”.

The winners will have their pick of any one of Ian Mavric's fabulous products for the TRS-80 line.

Do not over fit; don't try and predict the game's random number generator as the judging test runs will use different random sequences. Entries must be deterministic – always giving the same response if the input is the same.

Competitors are encouraged to get in touch early in order to receive timely updates on leader board changes, rule clarification and supplementary materials. You are welcome to submit improvements to your entry multiple times.

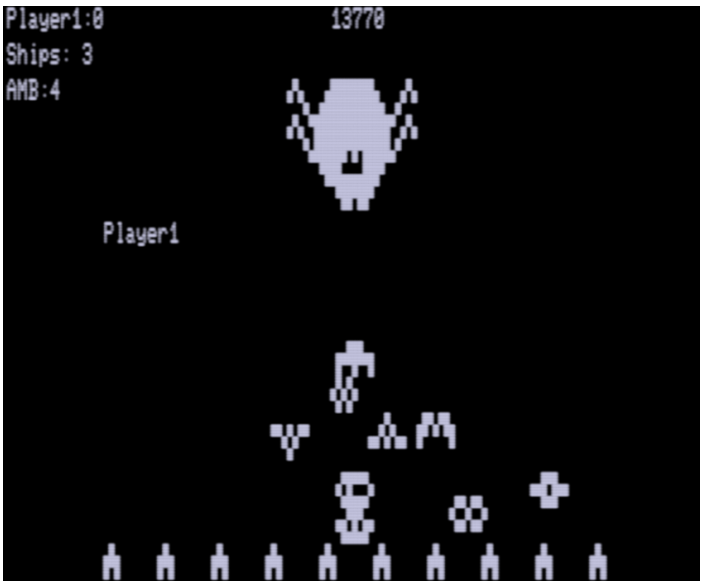
For details on how to talk to the game and a running leader board please consult:

http://48k.ca/comp2024.html

All entries will be judged by George Phillips and
should be sent to george@48k.ca

Competition closing date is 15:00 hrs GMT, 2 November 2024.





LATEST UPGRADE AVAILABLE
VERSION 2.5.3

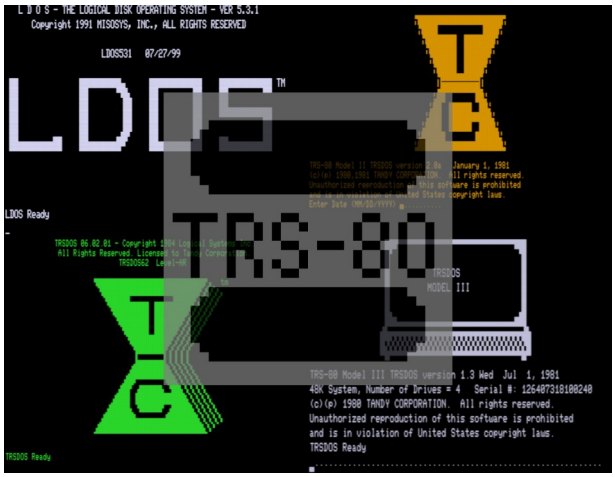
DON'T HAVE A TRS-80?

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 AND VIDEOTEX.

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

THE LATEST VERSION 2.5.3 IS NOW AVAILABLE, FEATURING:

CASSETTE READING FROM .WAV FILES.
SAVE AND LOAD RAW PRINTER OUTPUT.
GENERATE DISKETTE SUMMARIES ON THE COMMAND LINE.



IN MAV'S WORKSHOP

Coco 3 Keyboard Replacement

Follow up to: "Review: Coco 2 Replacement Keyboard Membrane"



by Ian Mavric

Last time[1] I talked about a replacement keyboard membrane for the Coco 2 keyboard. Long story short, it works great and is easy to install. At $A34.95 it's a bargain. But what about the Coco 3?

KEYBOARD REPAIR OPTIONS FOR THE CC3 ARE LIMITED

Considering the versatility and collectability of the Coco 3 it seems strange that no one has come up with a Coco 3 replacement keyboard membrane. Indeed people reached out to me after reading my article last issue to ask me if I knew of or when a drop-in replacement membrane for the CC3 would be available. Well I don't really know. What I do know is that it would be a sellout. As we know many more Coco 2s were sold than Coco 3s, but in 2024 the model most Coco enthusiasts spend their time working on is the CC3.

Like the Coco 2 the CC3 is a robust unit that hardly anything goes wrong with, until that keyboard membrane fails, which it will eventually do. I suggest sending a polite message to the producers of the Coco 2 membrane, Retro Fuzion, at their eBay page to encourage them to make a CC3 version. Once they see there is considerable demand for it they may take the steps to put it into production.

https://www.ebay.com.au/itm/325851925878

I CAN'T WAIT AND MY CC3 KEYBOARD HAS DEAD KEYS NOW

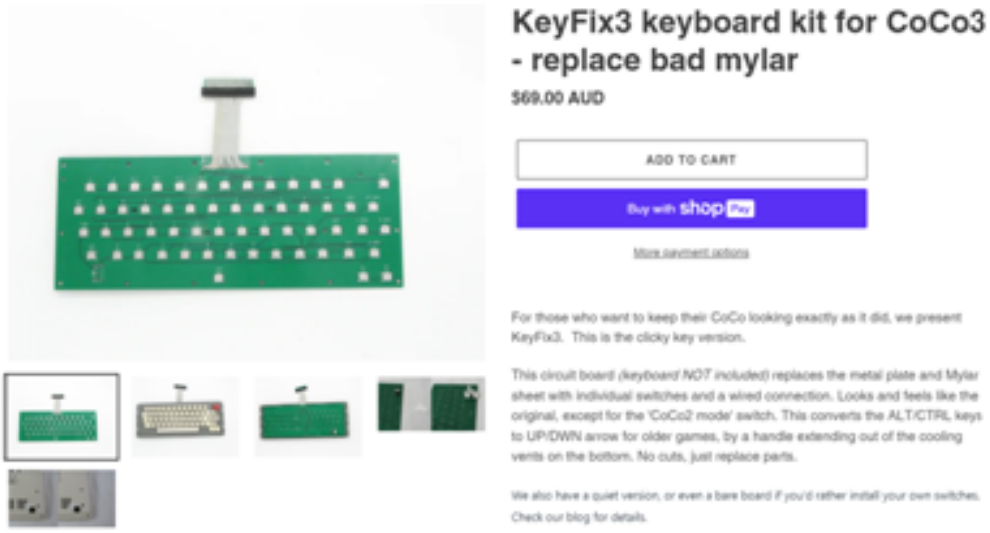
Did you know that you can use a Coco 2 keyboard on the CC3? The matrix is the same, so the CC2 keyboard plugs in. Obviously you lose the F1, F2, ALT and CTRL keys but depending on which software you mainly use this may not be much of a problem. CC1 and 2 software doesn't require the keys, nor does the CocoSDC. I can't comment on

which of the popular CC3 games utilise these four keys, and that would make them unusable.. Think of putting a Coco 2 keyboard in a Coco 3 a way of getting out of a pickle until another Coco 3 keyboard solution comes along.

THAT'S NOT REALLY A FIX, THOUGH: KeyFix3 IS A CC3 FIX

Granted using a Coco 2 keyboard is a less than satisfactory way to continue using your Coco 3, and at the time of writing there is no drop in membrane replacement, there are two more ways I have found to get the keyboard working. The first is the KeyFix² by ComputerConect, and it's made by Rick Ulland.

Rick's approach replaces the complete membrane (referred to as a mylar in ComputerConect nomenclature) and the metal sheet from under a CC3 keyboard with a thick circuit board which has individual clicky key switches. This mounts under the original keyboard for a factory original appearance. A CC2 mode switch changes the ALT and CTRL keys to Up and Down arrows making it easier to play classic games popular during the CC1/2 era.



Picture: KeyFix3 Coco 3 keyboard repair kit by Rick Ulland

The other good thing about this one is that it is infinitely repairable. Say a key switch goes bad years or decades later? Just unsolder it and replace it. These keyboards are also available with a choice of clicky or quiet key switches. It's $US45.00 which works out to about $A69.00 For more information: http://computerconect.com/

A COMPLETE KEYBOARD BECAUSE OF THE SIMPLE THINGS IN LIFE

The second product which does away with the membrane is the CocoMECH keyboard from Ed Snider, which can be ordered through his website, the Zippster Zone. These are complete drop-in keyboard assemblies, similar to what Coco 1 enthusiasts would remember back in the day like the HJL-57 and Mark Data keyboards. You open the computer, remove the old keyboard and plug in the new one, and close the computer. Instructions are included but it's really that simple.

Ed's keyboards are made using high-quality Gateron green key switches and the keycaps are laser etched. They Gaterons are considered up there with the Cherry MX as some of the finest key switches available and found in some of the premium PC keyboards. You will be very happy with the resulting feel this gives to you Coco 3. There is also a few options to keycap colours, other than the traditional CC3 key colours of white, beige and red.

CoCoMECH keyboards

These mechanical keyboard are in the standard CoCo3 key layout and feature Gateron green keyswitches and custom laser markings. View the project page here for the story on how and why I developed these. There's an installation video at the bottom of the page. Make sure your system has a full set of the proper case screws for best results.

At present, only CoCo3 installation kits are available. A set of CoCo3 supports will be shipped with each keyboard. Installation kits for other machines will be available in the future.

Find a keyboard with a keycap scheme you like and select your shipping option. Then click 'Buy now' to be directed to Paypal for payment.



Color Computer 3 keyboard - 01

Key colors as pictured. Click the image for a larger picture.

Price: $125.00

Sold Out



Color Computer 3 keyboard - 02

Key colors as pictured. Click the image for a larger picture.

Price: $125.00

Sold Out

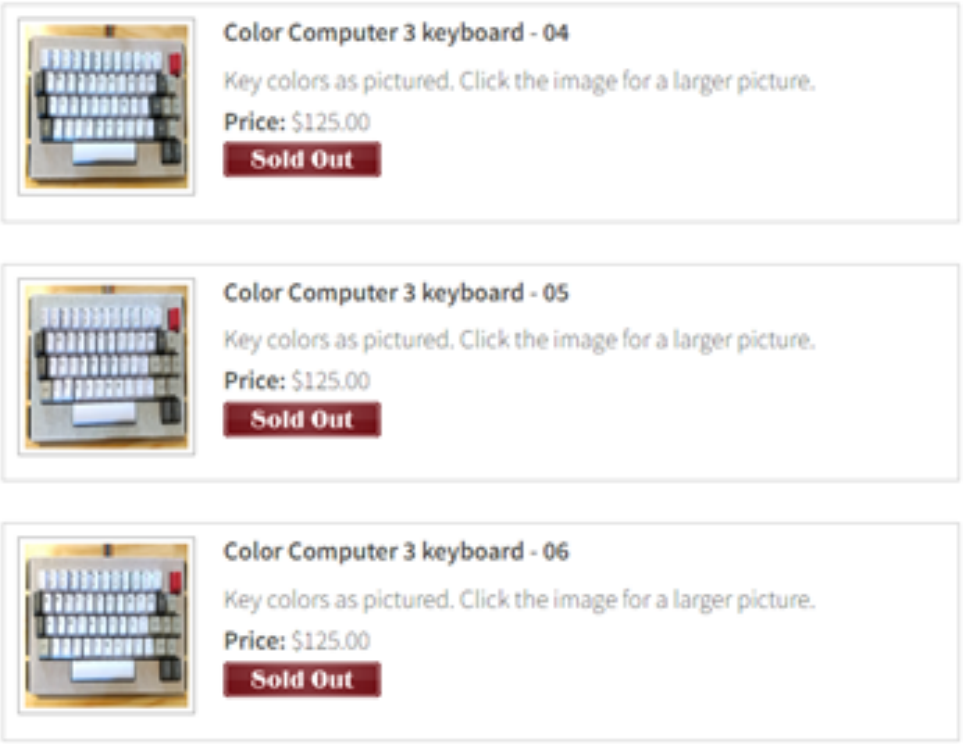


Color Computer 3 keyboard - 03

Key colors as pictured. Click the image for a larger picture.

Price: $125.00

Sold Out



Obviously due to the time and materials required to construct each CocoMECH they are not cheap. Currently $US125.00 each. However their reputation proceeds them and whenever a run of them is produced they sell out quickly, so keep a regular eye on Ed's website if you are after one, or contact Ed[3] to find out when more will be manufactured.

https://thezippsterzone.com/keyboards/

CONCLUSION

The suggestions above should be able to keep most people's Coco 3s up and running. Remember with the expensive keyboard options they will often be the last keyboard you ever need to buy for your computer and should last the rest of it's lifetime.

Ian Mavric ianm@trs-80.com

References:

1. http://www.fabsitesuk.com/tandy/trs8bit1801.pdf Pp.5-10

2. Rick Ulland/ComputerConect also makes version of the KeyFix for the Coco 1 and 2, appropriately called KeyFix1 and KeyFix2.

3. https://thezippsterzone.com/contact/

TRS-80 BASIC Cross Dialects Tables

Jim Adams v1.0 May 2024

In the micro computing world, BASIC was almost ubiquitously present but with many unique twists. It is like "English is spoken here" but each locality has its own idiomatic expressions. Moving between dialects or even between different TRS-80 models always requires finding out what BASIC functions are missing, or how few BASIC statement are spelled differently. I have found a simple table, which lists which BASIC elements are available in a given dialect, to be very helpful. At a quick glance, I can tell if the same feature appears in which dialect. Of course, there is no substitute to reading the manuals to lookup the exact *syntax* (mechanics) and *semantics* (meaning) of a given BASIC feature.

Please allow me to share with you a collection of cross referenced tables of the TRS-80 BASIC dialects along selected few evolutionary BASIC interpreters that are greatly similar to the TRS-80 BASIC.

Cross Reference Tables of BASIC Dialects

- 226 Commands and Statements Table (alphabetical order)
- 115 Functions Table (alphabetical order)
- 33 Operators Table (precedence order)
- 17 Data Types Table (bit size order)
- 26 Special Characters Table (ASCII order)
- 18 BASIC Dialects Summary Table (family tree order)

Legend:

Y	All versions
3	Model III only
1	Model I only
D	Model 100 with Disk Video Interface only [DSKO$, LFILES, WIDTH, DSKI$(), LOC(), LOF()]
U	MC-10 Undocumented but available [CLOADM, EXEC, VARPTR()]
P	IBM PC 5150 1981 and PCjr 4860 1984 only; both supported cassette I/O
C	Dartmouth CARDBASIC batch compiler with: MAT, MAT PRINT, MAT READ
+	DEC BASIC-PLUS had one real format, either 32bit or 64bit configured by Sys Admin
$	Dartmouth, PATB supported string constants only, no string data variables

18 BASIC Dialects Cross-Reference Table

Commands and Statements (226)

Model 100									DECB 1.1 CoCo2/1									
TRSDOS 2.0b M2										MCB 1.0 MC10								
TRS-XENIX MBASIC M16											GW-BASIC 3.23 PC							
Palo Alto Tiny BASIC												MBASIC 5.21 CP/M						
LEVEL I M3/M1													MBASIC 4.51 CP/M					
LEVEL II M3/M1														Altair EXTENDED BASIC 3.2				
TRSDOS 2.3 M1															DEC BASIC-PLUS			
TRSDOS 1.3 M3																Dartmouth BASIC DTSS		
TRSDOS 6.2.1AR M4																	ANSI X3.60-1978 Minimum	
STATEMENTS part 1 of 7																		Comments
! *exclamation*															Y			Same as REM in DEC BASIC-PLUS
' *apostrophe*	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y				Same as REM
, *comma*		Y	Y	Y														Same as THEN
? *question*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y			Same as PRINT
APPEND							Y											
ASSIGN															Y			
AUDIO										Y								
AUTO	Y	Y	Y	Y			Y	Y				Y	Y	Y				
BACKUP										Y								
BEEP									Y			Y						
BLOAD												Y						
BREAK							Y											
BSAVE												Y						
BYE															Y	Y		
CALL	Y						Y		Y			Y	Y					
CAT															Y	Y		
CHAIN	Y						Y					Y	Y		Y			
CHANGE															Y			
CHDIR												Y						
CIRCLE										Y		Y						
CLEAR	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y			
CLOAD		Y	Y	Y	Y				Y	Y	Y		Y	Y	Y			
CLOAD?		Y	Y	Y					Y					Y	Y			
CLOAD*											Y			Y				
CLOADM									Y	Y	U							Undocumented MC-10
CLOSE	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y		Y		
CLS	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y						
CMD"A"		Y																
CMD"B"		Y																
CMD"C"		Y																
CMD"D"		Y	Y															
CMD"E"		Y																
CMD"I"		Y	Y															

18 BASIC Dialects Cross-Reference Table

Commands and Statements (226)

Model 100									DECB 1.1 CoCo2/1									
TRSDOS 2.0b M2									MCB 1.0 MC10									
TRS-XENIX MBASIC M16									GW-BASIC 3.23 PC									
Palo Alto Tiny BASIC									MBASIC 5.21 CP/M									
LEVEL I M3/M1									MBASIC 4.51 CP/M									
LEVEL II M3/M1									Altair EXTENDED BASIC 3.2									
TRSDOS 2.3 M1									DEC BASIC-PLUS									
TRSDOS 1.3 M3									Dartmouth BASIC DTSS									
TRSDOS 6.2.1AR M4									ANSI X3.60-1978 Minimum									
STATEMENTS part 2 of 7																		Comments
CMD"J"		Y																
CMD"L"		Y																
CMD"O"		Y																
CMD"P"		Y																
CMD"R"		Y	Y															
CMD"S"		Y	Y															
CMD"T"		Y	Y															
CMD"X"		Y																
CMD"Z"		Y																
COLOR										Y		Y						
COM									Y			Y						
COMMON	Y						Y					Y	Y					
COMPILE															Y			
CONT	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		LEVEL I abbr: C.
COPY										Y								
CSAVE		Y	Y	Y	Y				Y	Y	Y		Y	Y	Y			
CSAVE*											Y			Y				
CSAVEM									Y	Y								
DATA	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I abbr: D.
DATE$=							Y		Y			Y						
DAY$=									Y									
DEASSIGN															Y			
DEF FN*a*	Y	Y	Y				Y	Y		Y		Y	Y	Y	Y	Y	Y	
DEF SEG												Y						
DEF USR*n*	Y	Y	Y					Y		Y		Y	Y	Y	Y			
DEFDBL	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y			
DEFINT	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y			
DEFSNG	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y			
DEFSTR	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y			
DEL										Y								
DELETE	Y	Y	Y	Y			Y	Y				Y	Y	Y	Y	Y		
DIM	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
DIR										Y								

18 BASIC Dialects Cross-Reference Table

Commands and Statements (226)

	Model 100	TRSDOS 2.0b M2	TRS-XENIX MBASIC M16	Palo Alto Tiny BASIC	LEVEL I M3/M1	LEVEL II M3/M1	TRSDOS 2.3 M1	TRSDOS 1.3 M3	TRSDOS 6.2.1AR M4	DECB 1.1 CoCo2/1	MCB 1.0 MC10	GW-BASIC 3.23 PC	MBASIC 5.21 CP/M	MBASIC 4.51 CP/M	Altair EXTENDED BASIC 3.2	DEC BASIC-PLUS	Dartmouth BASIC DTSS	ANSI X3.60-1978 Minimum	
STATEMENTS part 3 of 7																			Comments
DLOAD										Y									
DOS										Y									
DRAW										Y		Y							
DRIVE										Y									
DSKI$										Y									
DSKINI										Y									
DSKO$								D		Y				Y					Added with Disk Video Interface
EDIT	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y	Y				
END	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I abbr: E.
ENVIRON												Y							
ERASE	Y						Y	Y				Y	Y	Y	Y				
ERROR	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y					
EXEC										Y	U								Undocumented MC-10
FIELD	Y	Y	Y				Y	Y		Y		Y	Y	Y		Y			
FILES							Y		Y	Y		Y	Y	Y					
FNEND																Y			
FOR-STEP-UNTIL																Y			
FOR-STEP-WHILE																Y			
FOR-TO-STEP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: F. TO S.
GET	Y	Y	Y					Y		Y		Y	Y	Y		Y			
GET#												Y							
GOSUB	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: GOS.
GOTO	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: G.
HELLO																Y	Y		
IF						Y													
IF-THEN					Y						Y						Y	Y	LEVEL I abbr: IF T.
IF-THEN-ELSE	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y	Y	Y			
INPUT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	LEVEL I, PATB abbr: IN.
INPUT LINE																Y			
INPUT LINE#																Y			
INPUT#	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y		Y			
INPUT$									Y										
IOCTL												Y							

18 BASIC Dialects Cross-Reference Table

Commands and Statements (226)

Model 100									DECB 1.1 CoCo2/1									
TRSDOS 2.0b M2									MCB 1.0 MC10									
TRS-XENIX MBASIC M16									GW-BASIC 3.23 PC									
Palo Alto Tiny BASIC									MBASIC 5.21 CP/M									
LEVEL I M3/M1									MBASIC 4.51 CP/M									
LEVEL II M3/M1									Altair EXTENDED BASIC 3.2									
TRSDOS 2.3 M1									DEC BASIC-PLUS									
TRSDOS 1.3 M3									Dartmouth BASIC DTSS									
TRSDOS 6.2.1AR M4									ANSI X3.60-1978 Minimum									
STATEMENTS part 4 of 7																		Comments
IPL									Y									
KEY									Y			Y			Y			
KILL	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y		Y		
LCOPY									Y									
LENGTH																Y		
LET	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LET keyword maybe *omitted*
LFILES									D									Added with Disk Video Interface
LINE									Y	Y		Y						
LINE INPUT	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y				
LINE INPUT#	Y	Y	Y				Y	Y	Y			Y	Y	Y				
LIST	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: L.
LISTNH																Y		
LLIST	Y	Y	Y	Y	3		Y	Y	Y	Y	Y	Y	Y	Y				Model III only, not Model I
LOAD	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y				
LOAD"CAS1:												P						IBM PC 5150 and PCjr 4860 only
LOADM									Y	Y								
LOCATE												Y						
LOCK							Y					Y						
LPRINT	Y	Y	Y	Y	3		Y	Y	Y		Y	Y	Y	Y				Model III only, not Model I
LPRINT USING	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y				
LSET	Y	Y	Y				Y	Y		Y		Y	Y	Y		Y		
MAT																Y	C	CARDBASIC batch compiler
MAT INPUT																Y		
MAT PRINT																Y	C	CARDBASIC batch compiler
MAT READ																Y	C	CARDBASIC batch compiler
MAXFILES=									Y									
MDM									Y									
MENU									Y									
MERGE	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y				
MID$()=	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y				
MKDIR												Y						
MOTOR									Y	Y		P						IBM PC 5150 and PCjr 4860 only
NAME	Y	Y	Y				Y	Y	Y			Y	Y	Y		Y		

18 BASIC Dialects Cross-Reference Table
Commands and Statements (226)

	Model 100	TRSDOS 2.0b M2	TRS-XENIX MBASIC M16	Palo Alto Tiny BASIC	LEVEL I M3/M1	LEVEL II M3/M1	TRSDOS 2.3 M1	TRSDOS 1.3 M3	TRSDOS 6.2.1AR M4	DECB 1.1 CoCo2/1	MCB 1.0 MC10	GW-BASIC 3.23 PC	MBASIC 5.21 CP/M	MBASIC 4.51 CP/M	Altair EXTENDED BASIC 3.2	DEC BASIC-PLUS	Dartmouth BASIC DTSS	ANSI X3.60-1978 Minimum	
STATEMENTS part 5 of 7																			Comments
NEW	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		LEVEL I, PATB abbr: N.
NEXT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: N.
NULL													Y	Y	Y				
OLD																Y	Y		
ON BREAK							Y												
ON COM									Y			Y							
ON ERROR	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y		Y			
ON GOSUB	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			
ON GOTO	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	
ON KEY									Y			Y							
ON MDM									Y										
ON PEN												Y							
ON PLAY												Y							
ON STRIG												Y							
ON TIME$									Y										
ON TIMER												Y							
OPEN	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y		Y			
OPEN"COM*n*"												Y							
OPTION BASE	Y						Y					Y	Y					Y	
OUT	Y	Y	Y	Y					Y			Y	Y	Y					
OUTPUT							Y												
PAINT										Y		Y							
PALETTE												Y							
PALETTE USING												Y							
PCLEAR										Y									
PCLS										Y									
PCOPY										Y		Y							
PEN												Y							
PLAY										Y		Y							
PMODE										Y									
POKE	Y	Y	Y	Y			Y		Y	Y	Y	Y	Y	Y	Y				
POWER									Y										
PRESET									Y	Y		Y							

18 BASIC Dialects Cross-Reference Table
Commands and Statements (226)

Model 100	DECB 1.1 CoCo2/1
TRSDOS 2.0b M2	MCB 1.0 MC10
TRS-XENIX MBASIC M16	GW-BASIC 3.23 PC
Palo Alto Tiny BASIC	MBASIC 5.21 CP/M
LEVEL I M3/M1	MBASIC 4.51 CP/M
LEVEL II M3/M1	Altair EXTENDED BASIC 3.2
TRSDOS 2.3 M1	DEC BASIC-PLUS
TRSDOS 1.3 M3	Dartmouth BASIC DTSS
TRSDOS 6.2.1AR M4	ANSI X3.60-1978 Minimum

STATEMENTS part 6 of 7																		Comments
PRINT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: P.
PRINT AT					Y													LEVEL I abbr: P.A.
PRINT USING	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y	Y	Y		
PRINT#	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y		Y		
PRINT# USING	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y		Y		
PRINT@	Y	Y	Y	Y				Y	Y	Y	Y							
PRINT@()	Y							Y										
PSET									Y	Y		Y						
PUT	Y	Y	Y				Y	Y		Y		Y	Y	Y		Y		
PUT#												Y						
RANDOM	Y	Y	Y	Y			Y	Y								Y		
RANDOMIZE							Y					Y	Y				Y	
READ	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I abbr: REA.
REM	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	PATB stats it accepts REM*ARK*
RENAME										Y						Y	Y	
RENUM	Y						Y	Y		Y		Y	Y	Y				
REPLACE																Y		
RESET		Y	Y	Y	Y					Y	Y	Y	Y					LEVEL I abbr: R.
RESTORE	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y LEVEL I abbr: REST.
RESUME	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y		Y		
RETURN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y LEVEL I abbr: RET. PATB: R.
RMDIR												Y						
RSET	Y	Y	Y				Y	Y		Y		Y	Y	Y		Y		
RUN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: R.
RUNM									Y									
RUNNW																Y		
SAVE	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y		Y	Y	
SAVE"CAS1:												P						IBM PC 5150 and PCjr 4860 only
SAVEM									Y	Y								
SCRATCH																	Y	
SCREEN									Y	Y		Y						
SET		Y	Y	Y	Y					Y	Y							LEVEL I abbr: S.
SHELL							Y					Y						

18 BASIC Dialects Cross-Reference Table
Commands and Statements (226)

	Model 100	TRSDOS 2.0b M2	TRS-XENIX MBASIC M16	Palo Alto Tiny BASIC	LEVEL I M3/M1	LEVEL II M3/M1	TRSDOS 2.3 M1	TRSDOS 1.3 M3	TRSDOS 6.2.1AR M4	DECB 1.1 CoCo2/1	MCB 1.0 MC10	GW-BASIC 3.23 PC	MBASIC 5.21 CP/M	MBASIC 4.51 CP/M	Altair EXTENDED BASIC 3.2	DEC BASIC-PLUS	Dartmouth BASIC DTSS	ANSI X3.60-1978 Minimum	Comments
STATEMENTS part 7 of 7																			
SKIPF										Y	Y								
SLEEP																Y			
SOUND	Y							Y		Y	Y	Y							
STOP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I abbr: ST. PATB: S.
STRIG												Y							
SWAP	Y						Y	Y				Y	Y	Y	Y				
SYSTEM	Y	Y	Y	Y			Y	Y				Y	Y	Y			Y		
SYSTEM"*fff*"	Y							Y											*fff* is a TRSDOS 6.x command
TAPE																Y			
TIME$=									Y			Y							
TIME=							Y												
TIMER=										Y									
TROFF	Y	Y	Y	Y			Y	Y		Y		Y	Y	Y	Y				
TRON	Y	Y	Y	Y			Y	Y		Y		Y	Y	Y	Y				
UNLOAD										Y									
UNLOCK							Y					Y				Y			
UNSAVE																Y	Y		
VERIFY										Y									
VIEW												Y							
VIEW PRINT												Y							
WAIT	Y						Y					Y	Y	Y	Y	Y			
WEND	Y						Y					Y	Y						
WHILE	Y						Y					Y	Y						
WIDTH	Y						Y		D			Y	Y	Y					Added with Disk Video Interface
WIDTH LPRINT							Y						Y						
WINDOW												Y							
WRITE	Y						Y			Y		Y	Y						
WRITE#	Y						Y					Y	Y						

18 BASIC Dialects Cross-Reference Table

Functions (115)

	Model 100								DECB 1.1 CoCo2/1								
	TRSDOS 2.0b M2								MCB 1.0 MC10								
	TRS-XENIX MBASIC M16								GW-BASIC 3.23 PC								
	Palo Alto Tiny BASIC								MBASIC 5.21 CP/M								
	LEVEL I M3/M1								MBASIC 4.51 CP/M								
	LEVEL II M3/M1								Altair EXTENDED BASIC 3.2								
	TRSDOS 2.3 M1								DEC BASIC-PLUS								
	TRSDOS 1.3 M3								Dartmouth BASIC DTSS								
	TRSDOS 6.2.1AR M4								ANSI X3.60-1978 Minimum								
FUNCTIONS part 1 of 4																	Comments
&	Y	Y	Y				Y	Y		Y		Y	Y	Y					&H or &O depending on the dialect
&H	Y	Y	Y				Y	Y		Y		Y	Y	Y					
&O	Y	Y	Y				Y	Y		Y		Y	Y	Y					
ABS()	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: A.()
ASC()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y				
ASCII()															Y				
ATN()	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	
CDBL()	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y				
CHR$()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			
CINT()	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y				
COS()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
CSNG()	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y				
CSRLIN									Y			Y							
CVD()	Y	Y	Y				Y	Y				Y	Y	Y					
CVI()	Y	Y	Y				Y	Y				Y	Y	Y					
CVN()										Y									
CVS()	Y	Y	Y				Y	Y				Y	Y	Y					
CVT%$()															Y				
CVT$%()															Y				
CVT$F()															Y				
CVTF$()															Y				
DATE$	Y						Y	Y	Y			Y							
DATE$()															Y				
DAY$									Y										
DET															Y				
DSKF()														Y					
DSKI$()									D					Y					Added with Disk Video Interface
e															Y				
ENVIRON$()												Y							
EOF()	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y					
ERDEV												Y							
ERDEV$												Y							
ERL	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y		Y			

18 BASIC Dialects Cross-Reference Table

Functions (115)

Model 100									DECB 1.1 CoCo2/1									
TRSDOS 2.0b M2									MCB 1.0 MC10									
TRS-XENIX MBASIC M16									GW-BASIC 3.23 PC									
Palo Alto Tiny BASIC									MBASIC 5.21 CP/M									
LEVEL I M3/M1									MBASIC 4.51 CP/M									
LEVEL II M3/M1									Altair EXTENDED BASIC 3.2									
TRSDOS 2.3 M1									DEC BASIC-PLUS									
TRSDOS 1.3 M3									Dartmouth BASIC DTSS									
TRSDOS 6.2.1AR M4									ANSI X3.60-1978 Minimum									
FUNCTIONS part 2 of 4																		Comments
ERR	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y		Y			
ERR()														Y					
ERRS$	Y							Y											
EXF()																Y			
EXP()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
EXTERR()												Y							
FIX()	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y		Y			
FNa()	Y	Y	Y				Y	Y		Y		Y	Y	Y	Y	Y	Y	Y	
FRE()	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y				
FREE()										Y									
HEX$()	Y							Y		Y		Y	Y	Y					
HIMEM									Y										
INKEY$	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y					
INP()	Y	Y	Y	Y					Y			Y	Y	Y	Y				
INPUT$()	Y						Y	Y	Y			Y	Y						
INSTR()	Y	Y	Y				Y	Y	Y	Y		Y	Y	Y		Y			
INT()	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I abbr: I.()
INV()																Y			
IOCTL$()												Y							
JOYSTK()										Y									
LEFT()																Y			
LEFT$()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y				
LEN()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			
LOC()	Y	Y					Y	Y	D	Y		Y	Y	Y					Added with Disk Video Interface
LOF()	Y	Y	Y				Y	Y	D	Y		Y	Y	Y					Added with Disk Video Interface
LOG()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
LOG10()																Y			
LPOS()	Y						Y		Y			Y	Y	Y					
MAGTAPE()																Y			
MAXFILES									Y										
MAXRAM									Y										
MEM	Y	Y	Y	Y	Y		Y	Y		Y	Y								LEVEL I abbr: M.
MID()																Y			

18 BASIC Dialects Cross-Reference Table

Functions (115)

	Model 100								DECB 1.1 CoCo2/1								
		TRSDOS 2.0b M2								MCB 1.0 MC10							
			TRS-XENIX MBASIC M16								GW-BASIC 3.23 PC						
				Palo Alto Tiny BASIC								MBASIC 5.21 CP/M					
					LEVEL I M3/M1								MBASIC 4.51 CP/M				
						LEVEL II M3/M1								Altair EXTENDED BASIC 3.2			
							TRSDOS 2.3 M1								DEC BASIC-PLUS		
								TRSDOS 1.3 M3								Dartmouth BASIC DTSS	
									TRSDOS 6.2.1AR M4								ANSI X3.60-1978 Minimum
FUNCTIONS part 3 of 4																	Comments
MID$()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y		
MKD$()	Y	Y	Y				Y	Y				Y	Y	Y			
MKI$()	Y	Y	Y				Y	Y				Y	Y	Y			
MKN$()										Y							
MKS$()	Y	Y	Y				Y	Y				Y	Y	Y			
NUM															Y		
NUM2															Y		
NUM$()															Y		
OCT$()	Y						Y	Y				Y	Y	Y			
PEEK()	Y	Y	Y	Y			Y		Y	Y	Y	Y	Y	Y	Y		
PEN()												Y					
PI															Y		
PLAY()												Y					
PMAP												Y					
POINT()		Y	Y	Y	Y					Y	Y	Y					LEVEL I abbr: P.()
POS()	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y	Y	Y	
PPOINT()										Y							
RAD$()															Y		
RECOUNT															Y		
RIGHT()															Y		
RIGHT$()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y			
RND()	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	LEVEL I, PATB abbr: R.()
ROW()	Y						Y	Y									
SCREEN()												Y					
SGN()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
SHELL()							Y										
SIN()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SIZE()						Y											
SPACE$()	Y						Y	Y	Y			Y	Y	Y		Y	
SPC()	Y						Y	Y				Y	Y	Y	Y		
SQR()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
STICK()												Y					
STR$()	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y			

18 BASIC Dialects Cross-Reference Table

Functions (115)

	Model 100	DECB 1.1 CoCo2/1	TRSDOS 2.0b M2	MCB 1.0 MC10	TRS-XENIX MBASIC M16	GW-BASIC 3.23 PC	Palo Alto Tiny BASIC	MBASIC 5.21 CP/M	LEVEL I M3/M1	MBASIC 4.51 CP/M	LEVEL II M3/M1	Altair EXTENDED BASIC 3.2	TRSDOS 2.3 M1	DEC BASIC-PLUS	TRSDOS 1.3 M3	Dartmouth BASIC DTSS	TRSDOS 6.2.1AR M4	ANSI X3.60-1978 Minimum	FUNCTIONS part 4 of 4	Comments
STRIG()						Y														
STRING$()	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y								
SWAP%()														Y						
TAB()	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y						LEVEL I abbr: T.()
TAN()	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			
TIME()														Y						
TIME$	Y	Y	Y	3		Y	Y	Y			Y									Model III only, not Model I
TIME$()														Y						
TIMER									Y		Y									
TRN()														Y						
USR()				Y																
USR*n*()	Y	Y	Y				Y		Y		Y	Y	Y	Y						
VAL()	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y					
VARPTR()	Y	Y	Y	Y		Y	Y	Y	Y	U	Y	Y	Y							Undocumented MC-10
VALPTR$()											Y									
XLATE()														Y						

The table entries list the intrinsic functions that are builtin the BASIC interpreters.
FN*a*() allows up to 26 user defined BASIC functions, FN*A*() to FN*Z*(), written in the BASIC language.
USR*n*() allows up to 10 user defined functions, USR*0*() to USR*9*(), written in machine language and preloaded into memory (preferably in reserved space so they are not overwritten by BASIC).

18 BASIC Dialects Cross-Reference Table

Operators (33)

	Model 100	TRSDOS 2.0b M2	TRS-XENIX MBASIC M16	Palo Alto Tiny BASIC	LEVEL I M3/M1	LEVEL II M3/M1	TRSDOS 2.3 M1	TRSDOS 1.3 M3	TRSDOS 6.2.1AR M4	DECB 1.1 CoCo2/1	MCB 1.0 MC10	GW-BASIC 3.23 PC	MBASIC 5.21 CP/M	MBASIC 4.51 CP/M	Altair EXTENDED BASIC 3.2	DEC BASIC-PLUS	Dartmouth BASIC DTSS	ANSI X3.60-1978 Minimum	
OPERATORS																			Microsoft Precedence Groups
(*parenthesis*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	1 highest precedence group
) *parenthesis*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	1
functions	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	2 intrinsic functions(), FN*a*(), &
^ *exponential*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	3
** *exponential*																Y			3 same as ^
- *unary negative*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	4
+ *unary positive*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	4
* *multiplication*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	5
/ *division*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	5
\ *integer division*	Y						Y	Y	Y			Y	Y	Y	Y				6 same as (INT(*x*/*y*))
MOD *modulus*	Y						Y	Y	Y			Y	Y	Y	Y				7 same as (*x* - *y* * INT(*x*/*y*))
+ *addition*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	8
- *subtraction*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	8
+ *string concatenation*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			9
= *equality*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	10
== *equality rounded 6digit*																Y			10
<> *inequality*	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	10
>< *inequality*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y				10
# *inequality*						Y													10
< *relational less than*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	10
<= *less than or equal*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	10
=< *less than or equal*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y				10
>= *greater than or equal*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	10
=> *greater than or equal*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y				10
> *relational greater than*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	10
NOT *logical complement*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			11
AND *logical conjunction*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			12
* *logical conjunction*					Y	Y													12
OR *logical disjunction*	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			13
+ *logical disjunction*					Y	Y													13
XOR *logical exclusive OR*	Y						Y	Y	Y			Y	Y	Y		Y			14 ((*x* AND NOT *y*) OR (NOT *x* AND *y*))
EQV *logical equivalence*	Y						Y	Y	Y			Y	Y	Y		Y			15 same as (NOT (*x* XOR *y*))
IMP *logical implication*	Y						Y	Y	Y			Y	Y	Y		Y			16 same as (NOT *x* OR *y*)

18 BASIC Dialects Cross-Reference Table
Operators (33)

Precedence and Groups
Operators fall into four categories, in high-to-low precedence order:

Operators Category high-to-low order	Precedence Group in the Operators Table
Numerical	Groups 1-8
String	Group 9
Relational	Group 10
Logical	Groups 11-16

Operators are evaluated according to their *relative* precedence; starting with the highest precedence first then to the relatively lower precedence in the BASIC expression.

Operators within the same precedence group are evaluated from the left to the right as they appear in the BASIC expression.

Precedence group numbers *illustrate* the relative order of evaluation; however, there are no such numbers are assigned or utilized by BASIC.

Microsoft's precedence order and groups are listed in the table as they are the most ubiquitously implemented family of BASIC interpreters. Other BASICs use differing precedence order and groups; please read the documentations for the details of their expression evaluation precedence.

Derived Operators
Many earlier BASIC interpreters did not implement these five operators: \, MOD, XOR, EQV, IMP; however they may be derived using other intrinsic operators.

Operator	Description	Derived Expression with the Same Results
*x**y*	Integer Division	(INT(*x*/*y*))
x MOD *y*	Modulus *remainder*	(*x* - *y* * INT(*x*/*y*)) same as (*x* - (*y* * INT(*x*/*y*))), for positive *x*, *y*
x XOR *y*	Exclusive Disjunction	((*x* AND NOT *y*) OR (NOT *x* AND *y*))
x EQV *y*	Equivalence	(NOT ((*x* AND NOT *y*) OR (NOT *x* AND *y*))) same as (NOT (*x* XOR *y*))
x IMP *y*	Implication	(NOT *x* OR *y*) same as ((NOT *x*) OR *y*)

18 BASIC Dialects Cross-Reference Table

Data Types and Defaults (17)

	Model 100	TRSDOS 2.0b M2	TRS-XENIX MBASIC M16	Palo Alto Tiny BASIC	LEVEL I M3/M1	LEVEL II M3/M1	TRSDOS 2.3 M1	TRSDOS 1.3 M3	TRSDOS 6.2.1AR M4	DECB 1.1 CoCo2/1	MCB 1.0 MC10	GW-BASIC 3.23 PC	MBASIC 5.21 CP/M	MBASIC 4.51 CP/M	Altair EXTENDED BASIC 3.2	DEC BASIC-PLUS	Dartmouth BASIC DTSS	ANSI X3.60-1978 Minimum	
DATA TYPES																			Comments
Integer 16bit	Y	Y	Y	Y		Y	Y	Y	Y			Y	Y	Y	Y	Y			% 2bytes two's complement
Single Precision 32bit	Y	Y	Y	Y	Y		Y	Y	Y			Y	Y	Y	Y	+		Y	! 4bytes floating point real
Single Precision 40bit										Y	Y						Y		5bytes floating point real
Double Precision 64bit	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y	+			# 8bytes floating point real
String	Y	Y	Y	Y	Y	$	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	$	Y	$ length 0bytes up to 255bytes
Integer Array	Y	Y	Y	Y		Y	Y	Y	Y			Y	Y	Y	Y	Y			% 2bytes two's complement
Single Precision Array	Y	Y	Y	Y	Y		Y	Y	Y			Y	Y	Y	Y	+		Y	! 4bytes floating point real
Single Precision Array										Y	Y						Y		5bytes floating point real
Double Precision Array	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y	+			# 8bytes floating point real
String Array	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			$ length 0bytes up to 255bytes
1 dimension Arrays	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Single-dimensional arrays, vectors
2 dimensions Arrays	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Two-dimensional arrays, matrices
Up to 255 dimensions	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y				Multi-dimensional arrays
Default Integer 16bit						Y													PATB only variables data type
Default Single 32bit	Y	Y	Y	Y	Y			Y				Y	Y	Y	Y	+		Y	Assumed without any specifiers
Default Single 40bit										Y	Y						Y		The only numeric data type
Default Double 64bit							Y		Y							+			Assumed without any specifiers

ANSI X3.60-1978 did not specify the 32bit single precision, but implied it by specifying: 6 significant digits, with a minimum valid range from 1E-38 to 1E+38.

+ DEC BASIC-PLUS 1970, for the PDP-11 RSTS-11, supported *one* floating point-format only, either the 32bit single precision *or* the 64bit double precision, selectable at *sysgen* to build the MONITOR.

$ Dartmouth BASIC 1964, Palo Alto Tiny BASIC 1976 had string constants only, used by PRINT and INPUT for display messages. No string data variables were supported in these initial releases.

18 BASIC Dialects Cross-Reference Table

Special Characters (26)

	Model 100	TRSDOS 2.0b M2	TRS-XENIX MBASIC M16	Palo Alto Tiny BASIC	LEVEL I M3/M1	LEVEL II M3/M1	TRSDOS 2.3 M1	TRSDOS 1.3 M3	TRSDOS 6.2.1AR M4	DECB 1.1 CoCo2/1	MCB 1.0 MC10	GW-BASIC 3.23 PC	MBASIC 5.21 CP/M	MBASIC 4.51 CP/M	Altair EXTENDED BASIC 3.2	DEC BASIC-PLUS	Dartmouth BASIC DTSS	ANSI X3.60-1978 Minimum	
SPECIAL CHARACTERS																			Comments
Space optional		Y	Y	Y	Y	Y		Y	Y	Y	Y			Y	Y	Y	Y		Space(s) and/or Tab(s) delimiters
Space required	Y						Y					Y	Y					Y	Space(s) and/or Tab(s) delimiters
!	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y	Y	Y			Single Precision, DEC Comments
"	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	String constants
#	Y	Y	Y	Y		Y	Y	Y	Y	Y		Y	Y	Y	Y	Y			Double, PRINT USING, device#
$	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	String variables, string functions
%	Y	Y	Y	Y			Y	Y	Y	Y		Y	Y	Y	Y	Y			Integer, PRINT USING
&	Y	Y	Y				Y	Y		Y		Y	Y	Y					16bit constants: Hex, Octal
'	Y	Y	Y	Y		Y	Y	Y	Y	Y		Y	Y	Y		Y			Comments; PATB, DEC String
(	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Precedence, function()
)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Precedence, function()
*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Multiplication, PATB AND
+	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Positive, addition, concatenation
,	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Delimiter, PRINT field
-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Negative, subtraction
.	Y	Y			Y	Y	Y	Y		Y		Y	Y	Y					EDIT, LIST; LEVEL I, PATB abbrev
/	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Division
:	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			Statements separator
;	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	PRINT field, INPUT
<	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Relational less than
=	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Assignment, relational equal
>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Relational greater than
?	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y				PRINT
@	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y								PRINT @, PATB integer array @()
\	Y						Y	Y	Y			Y	Y	Y	Y				Integer division
^	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Exponential

Symbols are listed in ASCII order

space(s) or tab(s) are optional in most BASIC interpreters {as with FORTRAN66 and FORTRAN77}; but they are required delimiters in MBASIC 5.x, TRSDOS 6.x, XENIX, GW-BASIC, ANSI X3.60-1978

. All floating point BASICs accept . as a decimal point; but the table entry refers to either the line number option for AUTO, DELETE, EDIT, LIST, RENUM; or LEVEL I and PATB . for abbreviations

The 18 BASICs in this article, used 24 of the 32 total 7bit ASCII printable special symbols; 8 unused are [ ] _ ` { | } ~

18 BASIC Dialects Cross-Reference Table

Features Summary

Family Tree Step	BASIC Interpreter	Size	Version	Year	Maximum Bytes Free	Commands Statements	Functions	Operators
1	Dartmouth BASIC+DTSS [a]	10K	1964	1964	*TBD*	29	11	14
2	DEC BASIC-PLUS+RSTS-11 [b]	16K	1970	1970	*TBD*	72	48	23
3	Altair EXTENDED BASIC	12K	3.2	1975	55306	44	30	21
4	MBASIC (MS BASIC-80 CP/M)	18K	4.51	1977	37369	77	56	24
4	MBASIC (MS BASIC-80 CP/M)	24K	5.21	1981	30776	84	54	24
5	Palo Alto Tiny BASIC (PATB)	2K	1.0	1976	*TBD*	14	3	16
6	TRS-80 LEVEL I [c]	4K	1977	1977	15871	27	6	16
6	TRS-80 LEVEL II [d]	12K	1.3	1978	48338	56	36	19
6	TRSDOS 2.3 DISK BASIC	16K	2.2	1978	38288	78	50	19
6	TRSDOS 1.3 DISK BASIC	19K	1.3	1980	38202	88	51	19
7	Model 4 Disk BASIC	24K	1.1.1	1985	29924	83	59	24
8	Model II Disk BASIC	18K	1.2	1983	33608	70	56	24
8	TRS-XENIX MBASIC [e]	51K	1.2.0	1984	31341	89	56	24
8	Model 100 BASIC	17K	1.1	1983	29382	87	45	24
8	CoCo Disk ECB [f]	24K	1.1	1983	22823	94	44	19
8	MC-10 MCB [g]	8K	1.0	1983	19526	37	23	19
9	GW-BASIC (MS-DOS)	79K	3.23	1988	60300	128	71	24
n/a	ANSI X3.60-1978 [h]	*n/a*	*n/a*	1978	*n/a*	20	12	14

a. Dartmouth BASIC first release (May 4th, 1964) was < 4K 20bit Words, ie 4Kx20bit/8bit < 10K Bytes
b. DEC BASIC-PLUS for PDP-11 running RSTS-11 was < 8K 16bit Words, ie 8Kx16bit/8bit < 16K Bytes, DEC BASIC-PLUS for PDP-10 running TOPS-10 was 36bit Words, I don't know its size requirement
c. Model I LEVEL I were derived by Tandy from Li-Chen Wang's Palo Alto Tiny BASIC 1.77K, Model III LEVEL I BASIC ROM 4K 1980 added printer support, 2 statements: LLIST and LPRINT
d. Model III BASIC ROM was 14K 1980, 2K added lowercase + real clock support, 1 function: TIME$
e. XENIX MBASIC-68K for the Model 16 TRS-XENIX 1.3.5 also ran on Tandy 6000 XENIX 3.2.0 and the unreleased 3.3.0. A Tandy 6000 with the maximum theoretical RAM of 7168KB, mbasic –m6000000 yields 199437 Bytes free.
f. CoCo3 had SDECB 2.1 1986, derived from DECB 1.1 1983, modified by Microware Systems Corps. Adds 22 statements: ATTR, HBUFF, HCIRCLE, HCLS, HCOLOR, HDRAW, HGET, HLINE, HPAINT, HPRINT, HPUT, HRESET, HSCREEN, HSET, HSTAT, LOCATE, LPOKE, ON BRK, ON ERR, PALETTE, PALETTE RGB, WIDTH; and 5 functions: BUTTON(), ERLIN, ERNO, HPOINT(), LPEEK().
g. MC-10 undocumented added 2 statements: CLOADM, EXEC; and 1 function: VARPTR()
h. ANSI X3.60-1978 (ECMA-55-1978) Standard specified the Minimum BASIC language as a common denominator; but it was mostly ignored. In all practicality, MBASIC was the *de Facto* standard.
i. Family Tree Step numbers correspond to the relatively significant evolutionary BASIC's mile stones as detailed in the "BASIC Family History" article in the March 2024 issue of *TRS8BIT Newsletter*.

18 BASIC Dialects Cross-Reference Table
Wrapping Up

Numbers, more or less…

The BASIC features and various table tallies are not absolute and on the surface may appear contradictory to other sources. However, the issue is a simple one, how one chooses to enumerate the language features, that is, how to list each command, statement, function and operator. For example: PRINT and ? how should they be counted, as one statement *or* counted as two statements? Similarly, & &O &H count them as one function *or* three functions; I choose the later approach since it accounts for each symbol accepted by the BASIC parser.

Astute readers will notice the blending of some of the BASIC interpreters with their operating environment when counting the "Commands & Statements" category; this is because in the early days of computing, to the average user, *BASIC was the environment*. That is, a university operated the Dartmouth Time-Sharing System with Dartmouth BASIC; a school district purchased an HP 2000A with the Time-Shared BASIC; or a hobbyist purchased a TRS-80 with LEVEL II BASIC; primarily to program in BASIC. So the apparent blending of all the commands and statements was accepted as part of the BASIC programming. Obviously, that quickly changed to support multiple languages, then the separation between the environment and the languages became clearer to the average user.

Syntax and Semantics Warning

These tables provide a relative overview of the 18 BASIC interpreters, but a statement, function, operator or special symbol code that appears across multiple BASICs, is more often than not, will have different meaning (semantics) and different format (syntax) in each BASIC. So, use the tables for the initial lookup, but may I recommend following through with reading the manuals for the correct meaning and usage, that is, *the devil is in the details*.

For a detailed history, motivation and advantage of many of the dialects referenced here, please see "The BASIC Family Tree" published in the March 2024 issue of *TRS8BIT Newsletter*.

This has been a fun and rewarding exercise and I hope you find the tables useful. *Happy* computing.

Dedication

To Mary, my lifelong soulmate who teaches me something new every day; a beautiful, generous, loving wife, an amazing MasterChef and a creative Master Baker. I will forever be indebted to your love.
To James, our son, for his accomplishments, career, motivation fueled by the willingness to take on any challenge, while always thinking outside the box. Proud fully, I thank you for all of your achievements.
To Andrew, our son, for applying his academic astuteness to his career, while retaining a delightful sense of humor that is enjoyed by all of his friends. Proud fully, I thank you for all of your achievements.

Acknowledgments

Dusty Miller for editing and publishing the *TRS8BIT* newsletter, the great medium to quest our thirst for TRS-80 8bit and 16bit microcomputers with especial gratitude for his encouragement and support with which the articles would not have been possible.

Further Readings

Cross Dialects

TRS-80 BASIC Quick Reference Handbook 2nd Edition 62-2314. William Barden Jr. 1985.
Legendary author, masterful TRS80 BASICs cross reference, has +20 negligible misinformation.

BASIC Conversion Handbook 62-2088. David A Brain. 1981.
Microsoft Disk BASIC conversion across TRS-80, PET, Apple II.

BASIC Handbook, 2nd Edition. David Alvin Lien. 1984.
Great breadth of numerous BASIC dialects; syntactical examples limited to most common cases.

BASIC Book (PDF). Harry Helms. 1983.

TRS-80 Color Computer Quick Reference Guide 26-3194. Tandy. 1982.

The BASIC Family Tree (PDF). *TRS8BIT Newsletter*. March 2024.

Tandy Radio Shack

LEVEL 1 BASIC User's Manual 26-2101. David Alvin Lien. 1978.
Tandy's Don French instructor at San Diego State University; CompuSoft Publishing

LEVEL II BASIC Reference Manual 26-2102. Tandy. 1979.

TRSDOS 2.3 Disk BASIC Reference Manual 26-2104. Tandy. 1979.
Randy V Cook TRSDOS; Microsoft Disk BASIC

Getting Started with TRS-80 BASIC 26-2107. George Stewart. 1983.

Operation and BASIC Language Reference Manual 2e 26-2112. Tandy. 1981.

TRSDOS 1.3 Disk System Owner's Manual 26-2111. Tandy. 1980.
Tandy TRSDOS; Tandy Disk BASIC

TRSDOS 6.2 Disk System Owner's Manual 26-2117. Tandy. 1983.
Logical Systems Inc TRSDOS; Microsoft Disk BASIC

BASIC Evolutionary Steps

BASIC version 2 manual (PDF). Dartmouth College. October 1964.

Time Shared BASIC Manual (PDF). Hewlett Packard. August 1969.

BASIC-PLUS Manual (PDF). DEC. 1972.

Altair BASIC Reference Manual (PDF). MITS. 1975.

BASIC-80 CP/M Reference Manual (PDF). Microsoft. 1979.

ECMA-55 Minimal BASIC Standard (PDF). ECMA. January 1978.



E.T. FONEHUME

HI DUSTY. IN YOU LAST EMAIL, I HADN'T REALISED THE IMPLICATION OF YOUR COMMENT ON RANDOMISING AN ARRAY OF NUMBERS. 'IT MEANING NOTHING MORE THAT JUST THAT. IT'S AN ARRAY OF NUMBERS, RANDOMISED!'
IF THEY ARE TO BE USED AS, SAY, PLAYING CARDS WITHIN A PACK, EACH NUMBER IS TOTALLY IRRELEVANT UNTIL YOU CAN MATCH THAT NUMBER TO AN ACTUAL UNIQUE PLAYING CARD. SO OBVIOUS, I KNOW, BUT I JUST HADN'T THOUGHT OF IT LIKE THAT!

SO I SET TO ON THE ONE-LINER TRAIL, JUST TO SEE IF IT WAS POSSIBLE, NOT ONLY TO HAVE JUST AN ARRAY OF 52 ELEMENT WITH UNIQUE RANDOM NUMBERS, RANGING FROM 1 TO 52, BUT TO MATCH THEM UP TO 52 INDIVIDUAL PLAYING CARDS.

THIS SENT ME ON QUITE A BIT OF A TRAIL, HAVING TO TURN TO KNUTH AND THE INTERNET.
RANDOMNESS PLAYS A CRUCIAL ROLE IN VARIOUS APPLICATIONS, RANGING FROM GAMING TO CRYPTOGRAPHY. ONE POWERFUL ALGORITHM THAT BRINGS THE MAGIC OF RANDOMNESS TO THE FOREFRONT IS THE FISHER-YATES SHUFFLE, ALSO KNOWN AS THE KNUTH SHUFFLE. DEVELOPED BY RONALD A. FISHER AND FRANK YATES IN 1938 AND LATER POPULARIZED BY DONALD KNUTH IN HIS SEMINAL WORK "THE ART OF COMPUTER PROGRAMMING," THIS ALGORITHM PROVIDES A SIMPLE YET EFFICIENT METHOD FOR SHUFFLING ELEMENTS IN AN ARRAY.

THE FISHER-YATES SHUFFLE OPERATES ON AN ARRAY OF ELEMENTS, TYPICALLY REPRESENTING, AS WE TALKED ABOUT, A DECK OF CARDS OR ANY ORDERED SEQUENCE. THE ALGORITHM TRAVERSES THE ARRAY FROM THE LAST ELEMENT TO THE FIRST, SWAPPING EACH ELEMENT

WITH A RANDOMLY CHOSEN ELEMENT THAT PRECEDES IT. THIS PROCESS CONTINUES UNTIL THE ENTIRE ARRAY IS SHUFFLED, GUARANTEEING AN UNBIASED AND UNIFORMLY RANDOM PERMUTATION.

I ASKED 'CHATGPT' TO CODE THE 'F-Y' ALGORITHM WHICH IT DID, BUT IT WAS IN 'PYTHON' I HAD NO IDEA ABOUT PYTHON, HAVING NEVER COME ACROSS IT, SO ASKED FOR A BASIC VERSION, WHICH, OF COURSE, IT PROMPTLY PRODUCED.

PYTHON

```
COPY CODE
IMPORT RANDOM

DEF FISHER_YATES_SHUFFLE(ARRAY):
    FOR I IN RANGE(LEN(ARRAY) - 1, 0, -1):
        J = RANDOM.RANDINT(0, I)
        ARRAY[I], ARRAY[J] = ARRAY[J], ARRAY[I]
```

BASIC

```
10 DIM DECK(52)
30 FOR I = 1 TO 52
40   DECK(I) = I
50 NEXT I

60   FOR I = 52 TO 2 STEP -1
70     J = INT(RND * I) + 1
80     TEMP = DECK(I)
90     DECK(I) = DECK(J)
100    DECK(J) = TEMP
110  NEXT I

120 FOR I = 1 TO 52
130  PRINT DECK(I);
140 NEXT I
```

NOTE THAT THE RND FUNCTION, HERE, GENERATES A RANDOM NUMBER BETWEEN 0 AND 1. THE PROGRAM USES THIS FUNCTION TO IMPLEMENT THE FISHER-YATES SHUFFLE ALGORITHM.

KEEP IN MIND THAT MICROSOFT BASIC MAY HAVE VARIATIONS ACROSS DIFFERENT VERSIONS, SO YOU MAY NEED TO MAKE MINOR ADJUSTMENTS DEPENDING ON THE SPECIFIC VERSION YOU ARE USING.

THE ALGORITHM IMPLEMENTATION ENSURES THAT EACH ELEMENT HAS AN EQUAL PROBABILITY OF ENDING UP AT ANY POSITION IN THE ARRAY, CREATING A GENUINELY RANDOMIZED ORDER.

THE FISHER-YATES SHUFFLE FINDS APPLICATIONS IN VARIOUS DOMAINS, INCLUDING GAMING, STATISTICAL SIMULATIONS, AND CRYPTOGRAPHIC PROTOCOLS. ITS UNBIASED AND EFFICIENT NATURE MAKES IT A PREFERRED CHOICE WHEN TRUE RANDOMNESS IS REQUIRED, AND ITS SIMPLICITY MAKES IT ACCESSIBLE FOR BOTH BEGINNERS AND SEASONED DEVELOPERS ALIKE.

I DID HAPPEN TO READ THAT IN THE EARLY EDITION OF KNUTH'S WORK, HE DOESN'T EVEN MENTION FISHER OR YATES! A BIT ON THE MEAN SIDE I THOUGHT!

ANYWAY, AFTER HAVING A QUICK CHAT WITH YOU, I USED THE FISHER-YATES ALGORITHM, (WITH JUST A SMALL 'FIX' FOR LEVEL 2 BASIC), TO RANDOMISED THE 52 ELEMENT ARRAY AND THEN SET ABOUT 'TYING' THE NUMBERS TO THEIR UNIQUE PLAYING CARD, TREATING THE FIRST 13 NUMBERS AS BEING DIAMONDS, THE NEXT 13 BEING HEART ETC. ETC.

STATING THE OBVIOUS, C$ REPRESENTS THE 4 PLAYING CARD SUITES, DIAMONDS, HEARTS, CLUBS AND SPADES. INFACT, IF THERE WAS JUST A LITTLE MORE ROOM, YOU COULD ALWAYS RANDOMISE THE ORDER OF 'C$'

I JUST MANAGED TO GET IT INTO A ONE-LINER, BUT THE 10 OF EACH SUITE IS REPRESENTED BY A 'T', AND NOT A '10'. THERE WASN'T ENOUGH ROOM TO ALLOW FOR 2 CHARACTERS EACH, IN THE 'S$' VARIABLE.

IT IS NECESSARY TO GO INTO THE LEVEL 2 EDITOR AND USE THE 'X' COMMAND, TO COMPLETE THE WHOLE LINE.

BY THE WAY DUSTY, THIS IS THE FIRST TIME I'VE EVER BEEN INVOLVED WITH 'PYTHON'. IT LOOKS QUITE INTERESTING. IS/WAS IT EVER AVAILABLE FOR THE MODEL 1?

```
0 RANDOM:DIMD(52):CLS:C$="DHCS":S$="A23456789TJQK":FORI=1TO52:D(
I)=I:NEXT:FORI=52TO2STEP-1:J=RND(((2)-1)*I):T=D(I):D(I)=D(J):D(J
)=T:NEXT:FORI=1TO52:B=-(D(I)>=14)*-((D(I)>=14)+(D(I)>=27)+(D(I))
=40)):PRINTMID$(S$,D(I)-(B*13),1);:PRINTMID$(C$,B+1,1),:NEXT
```

AH	8S	KS	4C
KH	3C	6C	JD
3D	TC	JS	KD
3H	QC	TS	3S
8D	9C	4D	6D
5S	2S	6S	4S
2D	AS	JC	6H
9H	7S	8H	7D
8C	QD	5H	KC
9S	7H	9D	5D
QH	4H	TD	2C
2H	JH	AD	7C
5C	AC	TH	QS



WELL, YOUR QUESTION, RE PYTHON, HAD ME STUMPED E.T.! AS I SAID IN MY EMAIL, I CAN'T EVER REMEMBER IT FROM THE DAY, I THINK IT'S JUST TOO NEW! HOWEVER, I ASKED ON THE FORUM IF ANYONE WAS AWARE OF A VERSION FOR THE 8 -BIT RANGE OF TANDYS AND HERE'S THE RESULT...

Python was developed in the late 80's at the CWI in The Netherlands, its first release was Python 0.9.0 in 1991. This was well after the heights of the Z80 so no contemporary Python implementation for the TRS-80 has ever existed. There are a few lightweight implementations as PyMite and Python-on-a-chip but none for the Z80.
The other way around is easier to find: there are quite a few Z80-emulators written in Python! - **Bart**

There is nothing 80s in Python. Its development was throughout the 90s. The language that made an impact was Python 2, a 21st century language.
Python 2 does not run on 16 bit platforms. - Zare

The other day I was going thought a shoebox of old CD's and found a folded cardboard like guide for "Shanghai", a Mahjongg type game. I did a deep search and could come up the matching CD. I then went online and found the game on some site and downloaded it to my XP gamer (32-bit). As the game was installing, there was a pop-up for a version of Python, which needed an okay, and evidently was part of the game's setup. So the install process continued and it ran as I remembered from long ago. First time I've ever encountered anything visible concerning Python. Hope your coffee didn't get cold reading this. - **Agent Orange**

Micropython does run on at least one 16-bit microcontroller. https://github.com/micropython/micropython/tree/master/ports/pic16bit I think even Micropython is too large for the Model I. - **krebizfan**

More the reason to upgrade to TRS-OS and its guest operating systems TRSDOS 6 and soon to come a newer 24 bit version.
Built to take advantage of eZ80 24 bit addressing and can tear into micropython with power to spare.
Join the millions that have upgraded today! - **danielboonemerica**

Texas Instruments makes a "Python Edition" of the TI-84 Plus CE calculator, which is based on the eZ80... and it runs a weird fork of CircuitPython on an ARM co-processor instead of running it on the eZ80. Early versions of their "python app" did this externally with a silly dongle.
Supposedly the root cause of this situation is limitations with the available C compilers for the eZ80, but, *shrug*, not my circus, not my monkeys. - **Eudimorphodon**

The ABC language was a precursor to Python, it was worked on by the same author as Python, Guido van Rossum.
Early Python info, the early days: https://reinout.vanrees.org/weblog/2018/04/25/origin-of-python-abc.html
The original inspiration for Python was as a systems administration language, that is, it would be used the same as bash or perl. - **MrImprovement**

I can't even begin to imagine what degree of "glacial" a Python for a 2 MHz 8080 would be. Heck, it's glacially slow on a 184MHz F4 ARM in comparison, to say, C. (see my results for MicroPi on the "Pi day" thread.) - **Chuck(G)**

I think personally I mostly find the idea of using a language that depends on massively wasteful quantities of white space on a limited memory microcontroller mildly bemusing. Say what you will about BASIC, but if it's a tokenizing interpreter you can at least pack it pretty tightly. - **Eudimorphodon**

That's true--and quite often, on 8 bit systems, a good compile-to-p-code could give a compile-to-machine code version a run for its money. - **Chuck(G)**

I think in some ways that's fair, but in other ways it's not fair: if you were to compare Python and C source from the same person, I think you'd find the Python code had less "wasted" characters because it doesn't have the braces; the C code would still have the indentation because otherwise the code wouldn't be readable. It'd still have the indentation, right??
Okay, I get that the difference is you don't need to install the C source onto your microcontroller, but with Python you most likely would because I don't think there's a standard way to do compilation (not that I've spent much time learning about that). - **doshea**

Python internally compiles to a bytecode that is then run on a bytecode virtual machine. You can precompile python code to a '.pyc' file that is quite a bit smaller than the '.py' source. See https://docs.python.org/3/library/py_compile.html for more information. - **lowen**

Darn you guys, you revealed my secret project! I wanted to spring it on the world and amaze everybody, but you let it slip!
Actually, I don't have any Z80 Python. It's only in my head. But I have been thinking deeply about what it would take to make a least a limited interpreter. Yes it would be very slow and crammed: probably needing lots and lots of overlays and all 4 disk drives online. Forget any bytecode: due to it's object-oriented nature Python's bytecode doesn't really gain you anything since the "instructions" are very elaborate for anything bigger than integer math. It would have to be more like threaded functions like Forth.

I do see a path to attack the problem. Ignoring the whitespace issue, Python needs plenty of lists, tuples, and dictionaries. There actually is a Z80 language that has a good chunk of the pieces you need: Lisp. Specifically, MuLisp and it's MuMath derivative. I've taken apart both the CP/M versions, and the TRS-80 MuMath, and I'm still amazed at how much that system really contains. The only difference between MuLisp and MuSimp/MuMath are the text of keywords (FIRST vs CAR), and having the MUSIMP.LSP & MUSMORE alternate language files already loaded. I've patched a version of TRS-80 MuMath back into MuLisp. It has many more list processing functions than the other Z80 Lisps. Parsing is configurable on the fly, with adjustable operator precedence. It even has BigInts built-in! And all of it was already available, buried inside the obscure 32K & 48K TRS-80 MuMath. In 1980! For only $75!
(Sorry, I'm raving again.)
Anyway, I've imagined (as a first cut) writing a Python-ish parser language in MuLisp, and see how many features can be crammed in. It can't handle meaningful whitespace, but we could do the slicing and tuple enforcement, and objects & methods as MuSimp partly does that anyway in associative arrays. Like Basic everything is global, so you'd have to hide intermediate objects as they're created. And MuLisp is very slow: could be 50 times slower than Microsoft Basic! (Rule of thumb on the Model I is a empty integer FOR loop is 500 per second. The equivalent MuLisp (LOOP __) is only about 10 per second! The disadvantage of BigInts.)
Only after experimentation could a real Python-80 be written in assembly from scratch. I have ideas, but would take me years. Maybe someday after I retire. - **jrd**

Thanks, I forgot about that. I was aware that they're used as a cache for imported modules, but this stackoverflow question explains how you can compile the main executable scripts too, and then run them via the interpreter. Unfortunately that has the small drawback that you now have to run python3 foo.pyc instead of (if your file had a #!/usr/bin/env python3 or similar line) just ./foo.py. - **doshea**

I don't really know anything about Z80s and TRS-80s, but when I wanted an open-source interpreter I could extend that would run natively on Windows 3.1 (not just a DOS one) recently, I found that the Python interpreters available for that platform (all of which were Python 1.x) seemed to all be a bit too big and have dependencies I didn't want. I thought to myself: what small interpreted languages were popular back then? I figured Lisp and Forth (perhaps people here might have had some other/better answers, I'm not an expert there), and from hunting around I found XLISP-PLUS, and although its Windows 3.x support had been inadvertently broken in its most recent release, I fixed that (and one day I'll get around to posting that on GitHub hopefully!). I guess I had the benefit of having some previous knowledge of Lisp from using Emacs, but I think it's a nice, powerful language in its own right, and I'd be tempted to just try to add some modern niceties to it rather than try to change its syntax. Perhaps though it's a terrible language to use if you don't have an editor that helps you match up the parens? - **doshea**

THANKS TO EVERYONE FOR THEIR FASCINATING REPLIES. PERHAPS IT MIGHT MAKE AN INTERESTING COMPETITION PROBLEM FOR NEXT YEAR?


BRITAIN'S BEST-SELLING DIGITAL CULTURE MAGAZINE
PIXEL ADDICT
£6.50 ■ 2024 ■ ISSUE 19
Tandy
The Story of an Electronics Giant
PLUS
NEIL WEST OF SEGA POWER
40 YEARS OF KING'S QUEST
MULTITECH'S APPLE II CLONE
20 GOTO 10 BOOK REVIEW
20 GOTO 10

TRS-80 M4GA with a modern twist

John Mineur



With our machines not getting any younger and often limited spare parts at our disposal, I decided to turn a Model 4 Gate Array motherboard (26-1080A) into a functional machine with a modern twist.

Step 1 – Choosing a case

The case I chose to house the motherboard was the Thermaltake P3 T6 Pro. This case, in my opinion, offers the space I needed and the flexibility to be customised. However, like so many other modern cases, there is no direct provision for 5.25 drive bays, of which I needed seven. Trigger warning - when the time comes to mount the motherboard and bays, be prepared to reach for the drill (after measuring three times).

Step 2 – Creating a drive bay tower

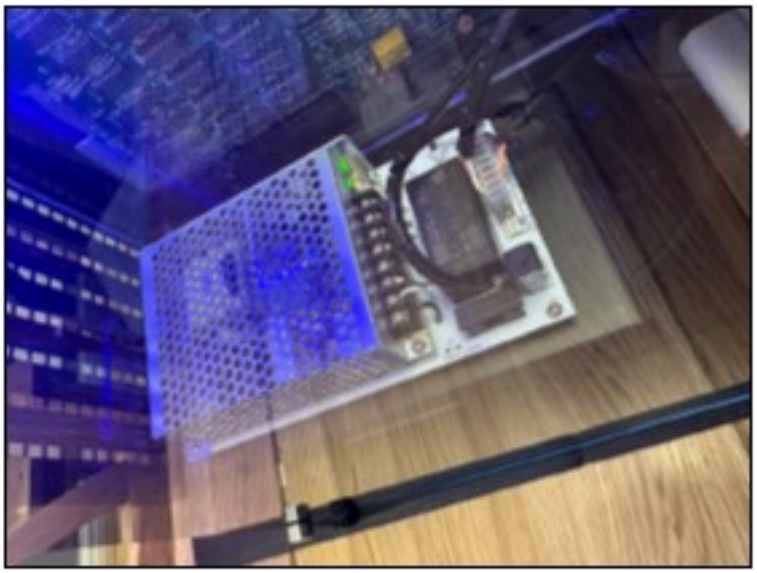
This stage involved me browsing eBay for any potential donors. I struck gold, or aluminium in this case, when I stumbled upon two tower plates from a Silverstone TJ107. Although they don't have any internal framework, I was able to use them to bolt to the sides of my 5.25 half-height sleds to create the main structure.





Step 3 – Sort out the power requirements

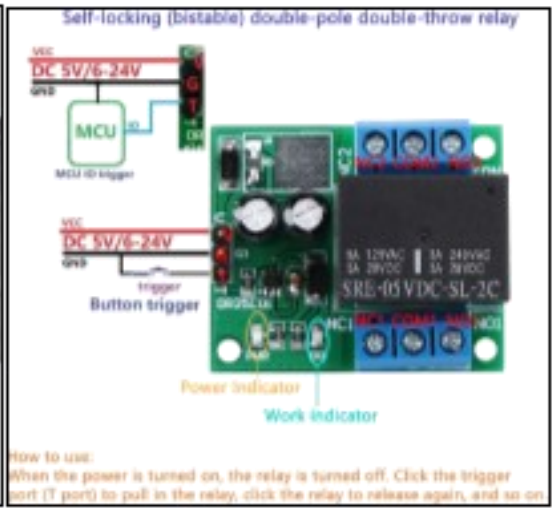
The main power supply was supplied by Jay Newirth of Newsoft. I have a few old supplies lying around but Jay's work is so good, I knew if I wanted reliability, build quality and for some like myself, good looks (sadly the supply not me), this was the obvious choice. Jay is brilliant to deal with and comes highly recommended. (https://jaynewirth.wixsite.com/newsoft).



Step 3A – Momentary power switching

One of the problems with a modern PC case is that it comes with a momentary power and reset switches. My solution here was to use two Flip-Flop DPDT relay modules, one for main power switching and the other to provide a switchable 5V circuit for accessories like Talker/80 which like to be powered up first. I converted the reset button into a switch and added resistors to the case LEDs to avoid a bright but very short lifespan.





Step 3B – 5V supplies

I say supplies, because although Jay's power supply comes with two 5V outputs for devices like the FreHD or a Gotek, the supplies are not live until the main power switch is pressed. Here I used a cheap 5V LED power supply from Amazon and housed it along with the relays in an old power supply case so I could bolt it into the position provided by the case.





Step 4 – Data storage

My drive tower contains two LotharekHXC Rev F drives, which can be switched to emulate four in total and a FreHD SD hard disk. This step was straight forward, only needing to make new drive cables which are folded underneath the motherboard to avoid obscuring the view. I also opted to add a transparent acrylic cover to the motherboard, to enhance the view when illuminated.

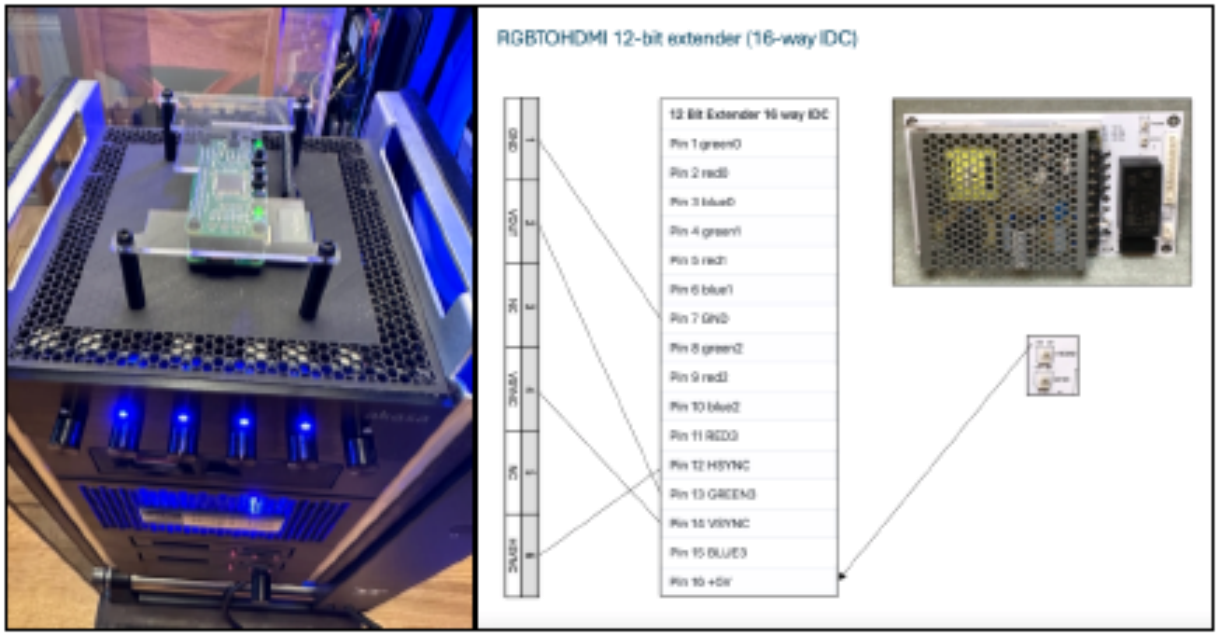


Step 5 – HDMI out for modern monitors

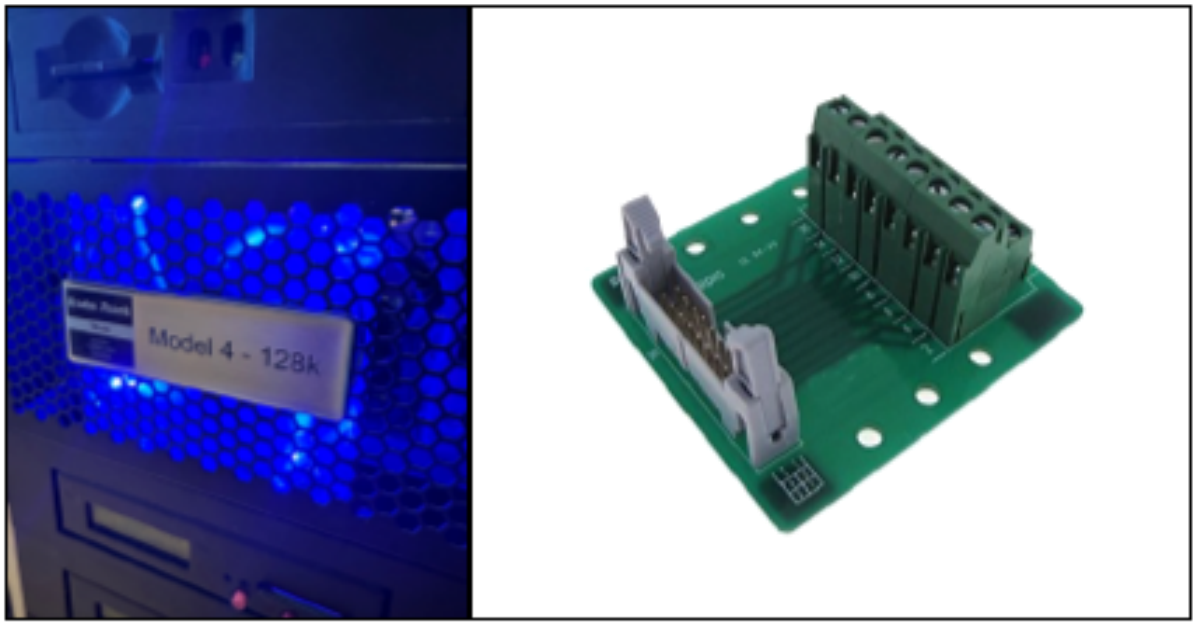
Bas Gialopsos of BetaGamma Computing kindly pointed me in the direction the RGBtoHDMI project. (https://github.com/hoglet67/RGBtoHDMI/wiki)

The hardware operates using a Raspberry Pi Zero, various options here, but I already owned a Pi Zero 2 W which had the name of this project written all over it. To this you add a 12-bit buffer with 16 pin out and the RGBtoHDMI board itself.

I wanted to use the video out header (J8) on the motherboard and after chatting with Ian and Steve of retro_tinkering on eBay, who sells the hardware, a solution was formed. The only mod you need to do to the M4GA motherboard is to remove the jumper from J13/J14 and ground U20 (centre pin of J13/14) using a short wire to the board ground plane.



To make life easy for myself, I used a 16-pin breakout for the ribbon cable and housed it in a hard drive enclosure on a 3.5 sled. I also 3D printed a top panel to which I mounted the RGBtoHDMI for convenient access.

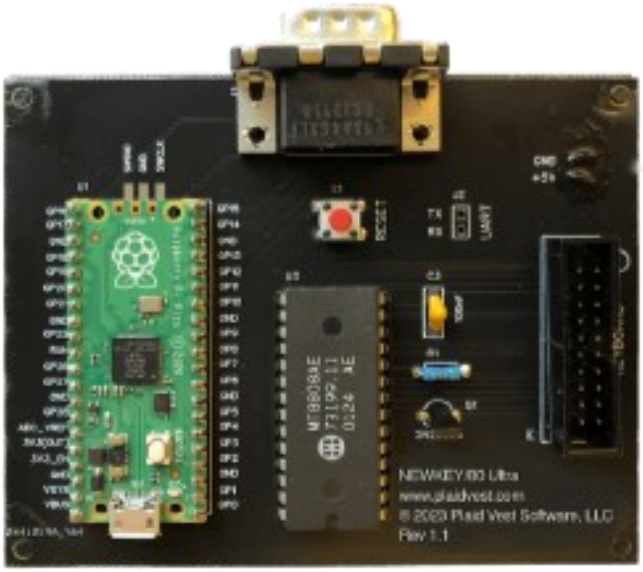


Profiles for the TRS-80 have already been created within the RGBtoHDMI project and many thanks to those that took the time to develop them. You have the option here to customise the profiles as much as you want to create the exact the look you require.

There are separate profiles when running in Model III and Model IV graphics mode and this can be selected by a press of a button.

Step 6 – USB keyboard and joystick connections

There are various options available here, but I chose a highly polished product, the Newkey/80 Ultra. Plaid Vest Software (https://plaidvest.com/) supply this little gem and Kyle is a pleasure to deal with.



Step 7 – Add the sound

I'm sure many have wondered why the TRS-80 did not arrive with an amplifier and speaker built in. To this end, I added a cheap amplifier module and speaker from eBay, connected it to the cassette output pins and housed this in the bottom 5.25 enclosure of the drive stack (see Step 4). This costs only a few pounds, assuming you have the enclosure already.





Step 8 – Lights, (camera), action

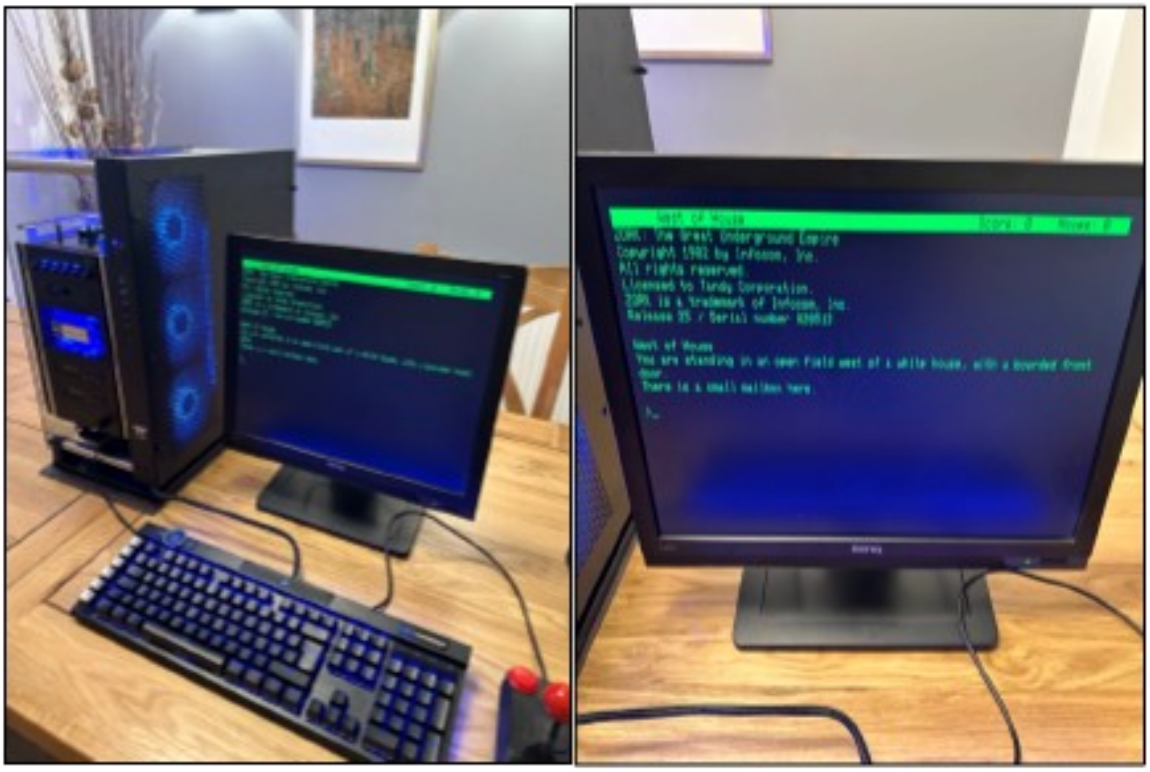
My initial goal was to create a modern looking TRS-80 and give it the features that would hold my attention. For lights, I added some led strips and a few LED fans to give it extra kerb appeal. The fans I control using an Akasa FC.6, again mounted in the drive stack. If the truth be known, most of the time I run with just the front fan on its lowest setting and the LED lights, but the case fans will provide a stream of air (on me) if we ever get a hot summer in Wales, or I achieve an exceptionally high score on Attack Force.

For the action I have added:

- Corsair LED keyboard
- RAM upgrade to 128K
- High resolution graphics
- Auto boot for the FreHD

The last three were supplied by Ian Mavric through his eBay store TRS-80 Universe. Again, I'm sure we all greatly appreciate the products and support that Ian continues to provide to the TRS-80 community.

I know this project will not appeal to everyone, especially the purists, but if you want a little fun and have some parts lying around, then why not give it a try?



Anyway... got to go, my wife wants her table back!


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


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

IN MAV'S WORKSHOP

Fixing the Non-Printing Model I

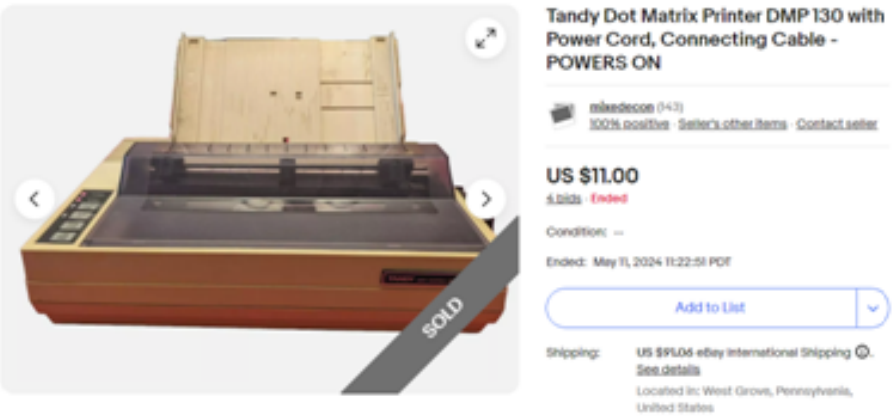


by Ian Mavric

That serious looking Clark Kent-esque businessman would have his day ruined if that Line Printer I stopped printing. Back in the 1970s when the TRS-80 Model I was new, printing was one of the important abilities of the machine... programmers need hard copies of their listings, accountants need financials printed, everyone needed to print all manner of things from Word Processing applications. What would happen if one day you couldn't print?

DO PEOPLE STILL PRINT FROM THEIR TRS-80?

These days in 2024 printing from a TRS-80 is a rare occurrence indeed. Even in almost all platforms of retrocomputing people rarely have a need for hard copy, and this is reflected in the price for used printers from back in the 1980s. If you need a printer for your TRS-80, or even your Commodore 64 or Atari 800, you can easily find them very inexpensively. Printers have just never caught on in retrocomputing circles like other accessories have.



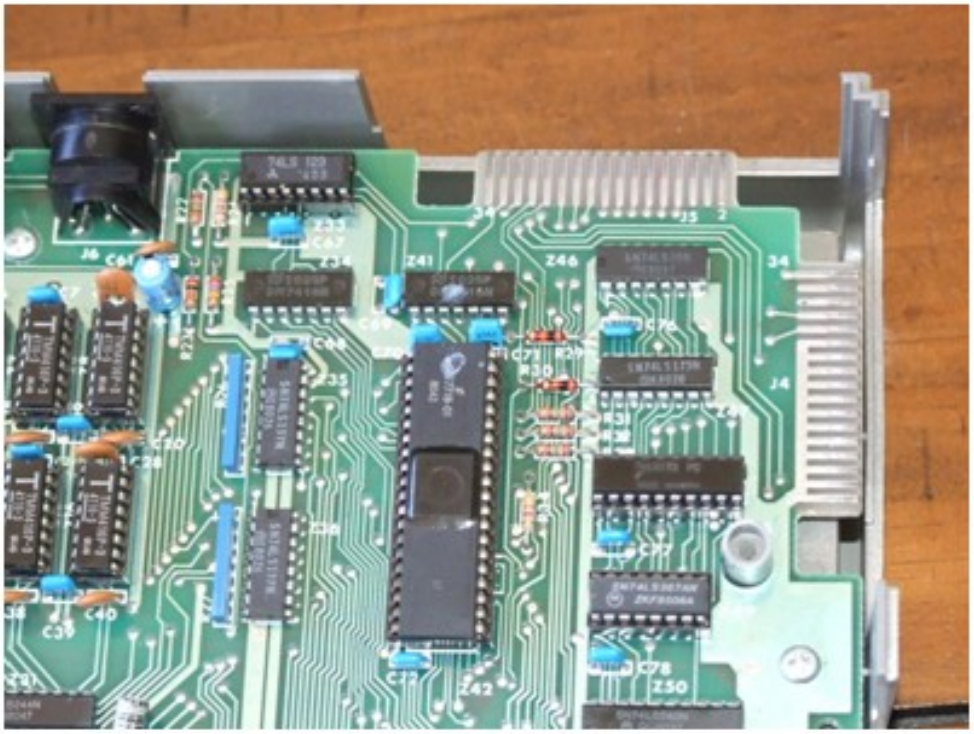
Picture: Typical cheap printer for a TRS-80 doesn't cost much at all

TURNS OUT SOME PEOPLE STILL WANT TO PRINT...

Interestingly two decades has just flown by with no one seemingly needing to print from their Model Is, then in the space of 6 months,

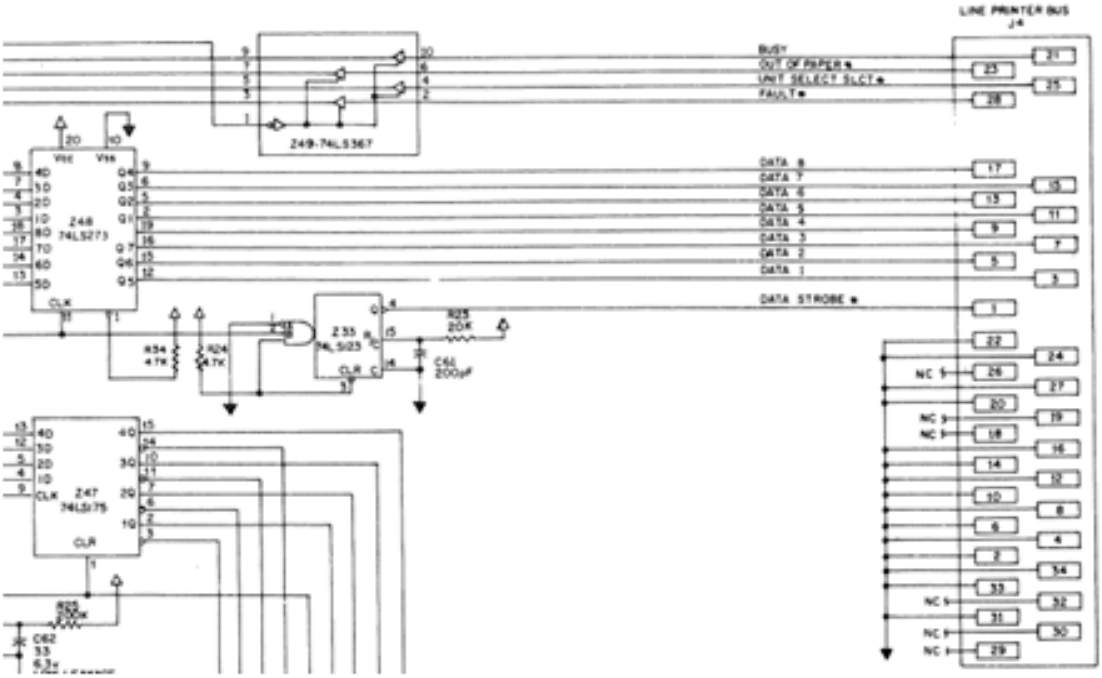
two people approached me about their systems inability to print. After the usual questions I ask to make sure a simple misunderstanding has occurred... "Is the printer set to Online?", "Is paper present?", "The the cable plugged into the correct place?" (ie. typical RFTM questions) I deduced there was something wrong with the computer, and that would need to be put right. A non-functional parallel printer port, like a non-functional serial port, will not prevent the computer from running, and could have failed any time between when the computer last printed something (the early 1980s in a lot of cases) and present day. Let's look at the Expansion Interface.

THE LATER (2nd DESIGN) EXPANSION INTERFACE



Picture: Printer port (J4) on the EI.

A quick look at the Expansion Interface board may make you think that components Z46-50, which are closest to the J4 (the printer port) are related to printing. However a quick look at the schematic diagram for the EI shows only Z48, Z49 and Z33 (which lives above the 7416 line driver in the FDC zone of the board) are required for the printing to function.



Picture: Z33, 48, 49 are all the printer port consists of.

QUICK AND DIRTY WAY TO FIX IT

Just replace those three chips if you have de-soldering station remove those three chips, replace them with sockets and put in new 74LS123, 74LS273, and 74LS367. 99.% of the time that will fix the

problem, and these chips are readily still available for around $2 each.

WHAT ACTUALLY GOES WRONG?

As stated before the parallel printer port consists of 3 chips:

Z48, the **74LS273** is an 8-bit parallel register. It spits out the parallel data to the printer. If any of the 8 bits are incorrect you will get spelling errors on the printer. More commonly it fails altogether, so no printing will occur and the system just hangs.

Z49, the **74LS367** is a tri-state hex buffer. In this application it monitors the paper-out signal, the Fault and Busy lines, and the unit select. Any of these signal malfunctions will result in nothing happening and a hung system.

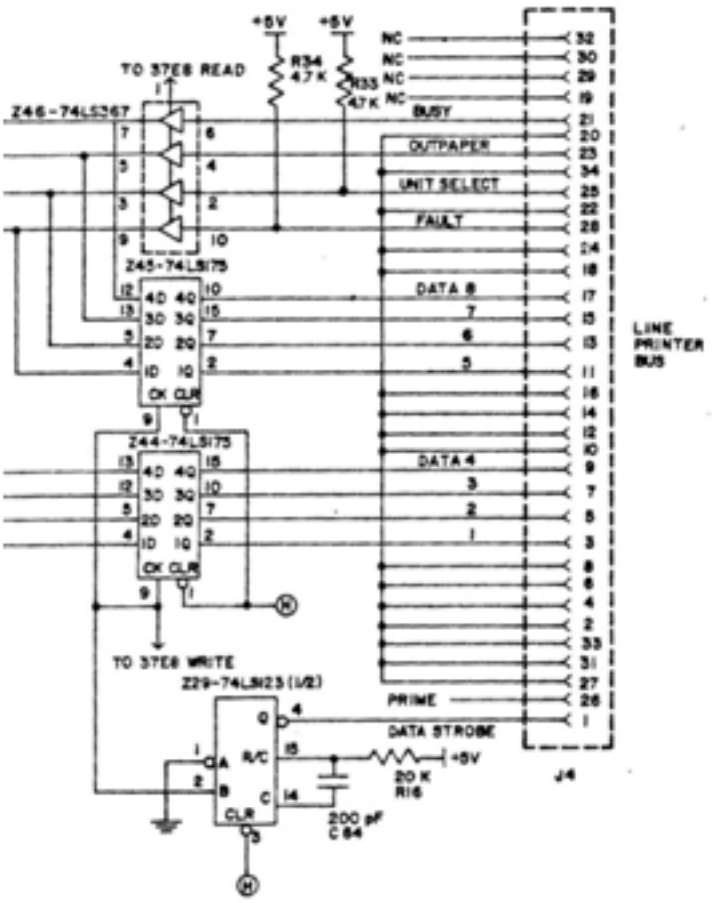
Z33, the **74LS123** is a monostable multivibrator. The stobe signal between the printer and the computer is monitored here, so that the parallel data flows at a rate acceptable to the printer. When this chip fails the handshaking with the printer is not able to be established so it will not print, and the system hangs.

So you can probably see here that not a lot of useful information can be deduced from the system itself; for the most part it just hangs. As mentioned earlier I saw two systems which didn't print, and repaired them. On the first system Z49 and Z33 had failed, and on the 2nd system Z48 and Z49 had failed. So like I said before, if you have the equipment, changing the three chips, perhaps even one at a time, you will most likely find that two of them will be preventing the

system from printing. You *could* scope around with an oscilloscope to ascertain which chip(s) is/are bad before proceeding, however in my experience this is one where you could easily spend more time messing around with interpreting signals than it actually takes to replace the chips mentioned.

DOES THIS APPLY TO THE 1st DESIGN EXPANSION INTERFACE?

On the early Expansion Interface the design and function of the parallel printer port is the same, however the job accomplished by the 74LS273 is done with a pair of 74LS175 Quad D Flip-Flop chips.



Picture: Early EI printer port schematic showing 4 chips do the job

CONCLUSION

Maybe printing from a TRS-80 is never going to be a requirement for 99.9% of the retro computing enthusiasts that collect and use the machines. For those of you who ever wondered what the printer port consists of and how to fix it, I hope this information comes to be of value one day.

People still ask me from time to time what is the best possible printer for their machine, if they need one just in case... I recommend the DMP-200 for 80-column work, and the DMP-420 for 132-column data output. And the best of all, both these printers work with any TRS-80 with a parallel port, and have connections for the Coco and MC-10 serial port.

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.

2022 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)

VII. Basque Tournament of TRS-80

The last 2nd of April, Tuesday, the Euskal Retro organized the VII. Basque Tournament of TRS-80. We met 6 players in Bilbao, in Café Berria (a bar located in Santutxu)

It was a fun afternoon, although before starting player I had problems with parking my car and the parking meters I had problems with parking my car and the parking meters, but I solved it and after getting the TRS-80 we started playing. We had a good time playing the TRS-80 as usual.

We played these games:
Monster Mazer, Canyon Climber, Demolition Derby and Clowns & Ballons.

I have bought recently a new TV and with this new TV the games look better, the image has improved and we can even enjoy the sound and music of the games. It has been an advance.

Egoitz won the tournament after playing to these 4 games. But the other players played well and the differences were minimal. For the next tournament, possibly in July 2024, we hope to have new games like Dragon Fire or Robot Battle.

Here you have the results:

Canyon Climber

1-Egoitz Campo 3 points (14.600)
2-Joseba Zubiaurre 2 points (6.200)
3-Andoni Mariezkurrena 1 point (5.600)

Demolition Derby

1-Egoitz Campo 4 points (1.800)
2-Andoni Mariezkurrena 3 points (1.400)
3-Joseba Zubiaurre 2 points (1.200)
4-Andi Erromo 1 point (600)

Clowns & Ballons

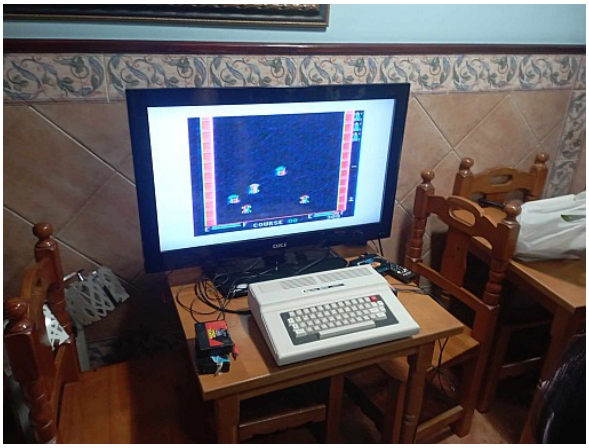
1-Egoitz Campo 4 points (3.800)
2-Gorka Perez 3 points (900)
3-Joseba Zubiaurre 2 points (660)
4-Andoni Mariezkurrena 1 point (330)

Monster Mazer

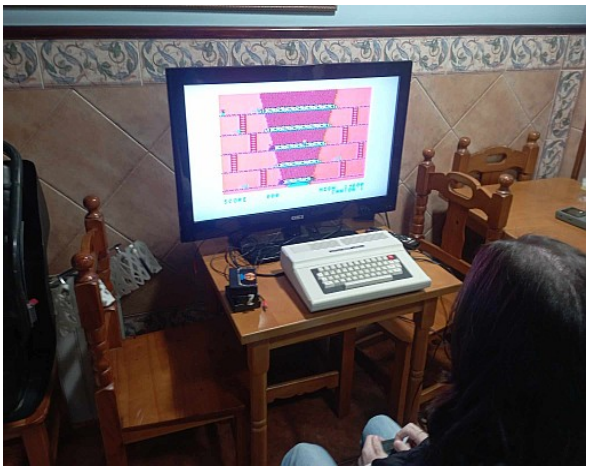
1-Gorka Perez 4 points (3.080)
2-Egoitz Campo 3 points (1.700)
3-Joseba Zubiaurre 2 points (550)
4-Andoni Mariezkurrena 1 point (400)

Classification

1-Egoitz Campo 14 points (Basque Country)
2-Joseba Zubiaurre 8 points (Basque Country)
3-Gorka Perez 7 points (Basque Country)
4-Andoni Mariezkurerna 6 points (Basque Country)
5-Andi Erromo 1 point (Basque Country)





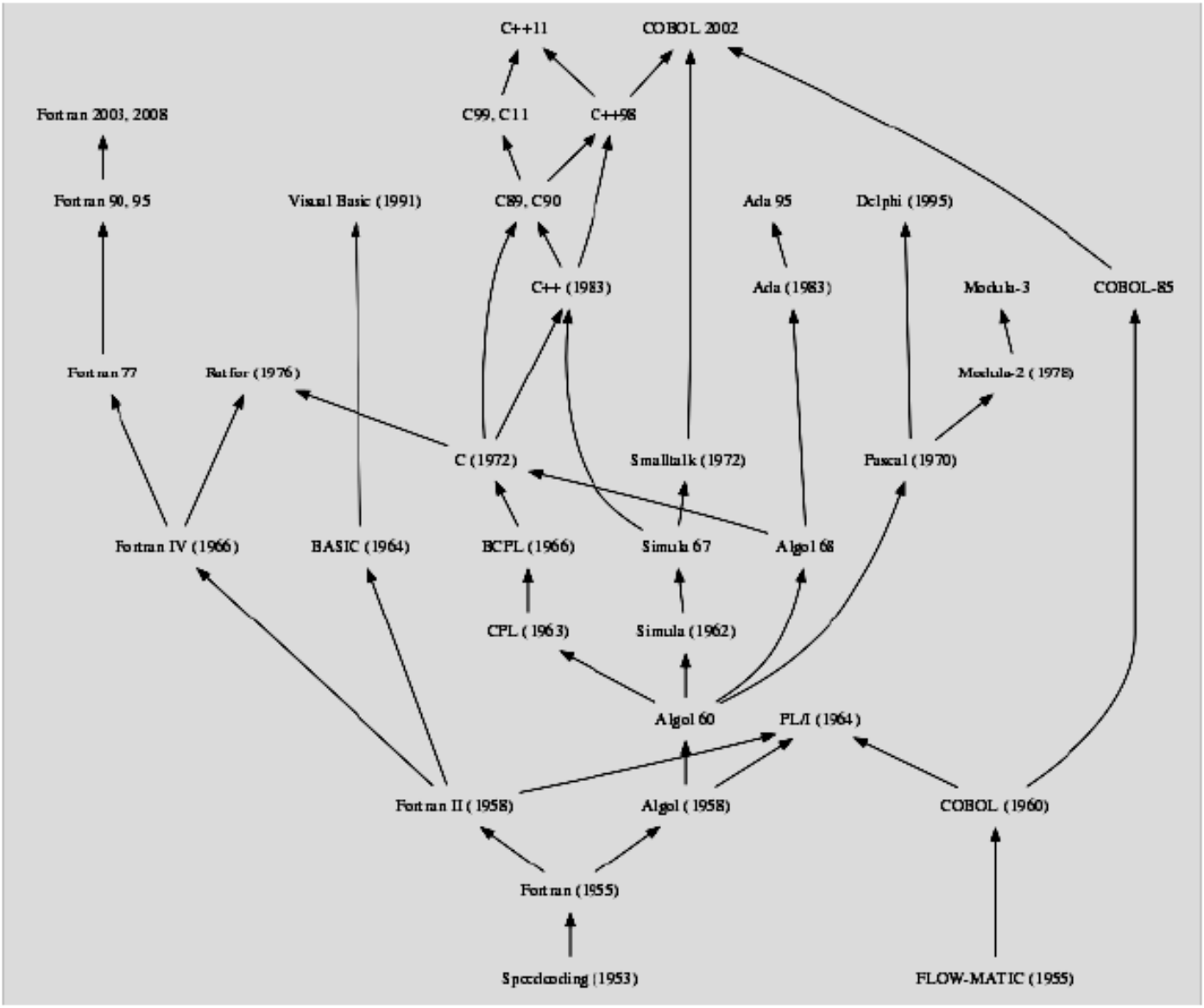



COURSE 00

Software Tools Episode 4: "Get Cracking!"

Vince Otten

In the early days of computers, programming languages had evolved from machine code through assembly language to Fortran and COBOL. By 1958, an influential number of computer scientists had joined to propose a big step forward in programming languages: "Algol". You may not have heard of Algol, but many of its breakthough concepts, like code blocks, structured programming, and extended data structures like "records" and "sets", have set the tone for most modern languages like Pascal, C, Java, and Python.



(Source: Shazz, Borkowsk, image in Wikipedia article on Algol)

But here's what's truly *weird* about the official definition of Algol: *it had no Input/Ouput (I/O) facilities.*

What's a program without input or output? Algol left it up to implementers of the language to define their own I/O facilities. Interestingly, unlike BASIC or Pascal, the C programming

language also has no built-in I/O: it's supposed to be shipped with a standard I/O library that gets linked in with your C program when it's compiled. That standard library is called "stdio.h". Within "stdio.h" is a group of functions for output, like printf() and putc(), and input, like getc(). In BASIC, on the other hand, what we use for output and input, like PRINT and INPUT, are *statements* built into the language, rather than functions which live in a separate "library". What's a function? Think of something like SQR in Level II BASIC: when we want to print the square root of 9, we could enter PRINT SQR(9). Here, SQR() is a function that takes one argument, "9", and returns one value, the square root of 9, which is "3". And in this example, PRINT is a *statement* which takes 0 or more arguments to display on the screen, like

```
PRINT
PRINT SQR( 9)
PRINT A$; B!, C%; D#, "That's all folks!"
```

It's easy to see why BASIC makes PRINT a statement rather than a function: there's no "value" we need returned from a PRINT. It just does its job. On the other hand, it makes sense that C's getc() "get a character" is a function: it does return a value, the character that it "gets". BASIC's INPUT isn't a function, though: it's still a statement that basically tells the interpreter, "Get whatever the user types at the keyboard and put it in a variable." For example, BASIC's syntax is

```
INPUT A$
```

whereas C's would be

```
a= getchar();    /* "getchar()" is a version of getc() that reads from the standard input */
```

All this to say: languages like Fortran, BASIC, and Pascal have I/O *built in* to the language, but languages like Algol, C, Ratfor, and the original "Software Tools" do not.
Whatever. If we're going to write Software Tools in BASIC, can't we just use BASIC's INPUT and PRINT statements in place of "getc()" and "putc()"? There's three reasons why not:

1. getc() is supposed to return just 1 character or byte. INPUT (or LINE INPUT) does not.
2. getc() should be able to return *any* character or byte with a value of 0..255; INPUT does not. PRINT should be able to output any of those characters.
3. getc() and putc() should be able to read from or write to any appropriate device or file. To PRINT to a printer, for example, BASIC needs LPRINT; to a file, PRINT#; to a cassette, PRINT#- *n*.

So, SKELETON/BAS, the "Software Tools in BASIC" framework, simply allows you to fOpen() any appropriate device or file with a specific mode ("Output" / "Random" / "Put" / "Append"), and

then uses the same putc() subroutine call with the same list of arguments to print a character at a time in a consistent way. Similarly, you can fOpen() any appropriate device or file with a specific mode ("Input" / "Random" / "Get") and then use the same getc() subroutine call in the same way.

First, let's look at putc(), because that's a lot simpler to implement than getc(). Here's the pseudocode, without the BASIC code in comments:

```
putc(   c$, file#) {  /* write char or string c$ on device or disk file#  2200    */
    if (file# > 0) {
        /* disk file       */
        /* DiskPutc( c$, file#)  */
        if (OpenType[file#] == APPEND_MODE {
            for (p= 1; p <= LEN( c$); p++)
                Append( c$[p], file#);
            }
        else {
        /* "ELSE" -- "P" mode                                  */
            PRINT# file#, c$;
            }
        return;
        }
    if (file# == SCREEN) {
        PRINT c$;
        return;
        }
    if (file# == PRINTER) {
        LPRINT c$;
        return;
        }
    /* "' FOR NOW: INSERT TESTS FOR CAS1..2, RS-232 LATER"          (2260..)/*
    }
```

This is relatively straightforward. There are three basic destinations for the character/byte that we're going to putc(): a disk file, the screen, or the printer. (Since I developed SKELETON/BAS on the excellent "trs80gp" emulator, I haven't been able to get cassette DATA files to work yet. The emulator wants to eject the cassette after the first PRINT#-*n* statement and asks you for a file name, rather than allowing as many PRINT#- s as you want. Hopefully someday this will be addressed.)

Devices are all negative integers, and are “hardwired” to the constants provided by defined functions DEF FN SC%() and DEF FN PR%(). The character C$ is either PRINTed to the screen or LPRINTed to the line printer, depending on the value of file#: simple.
Disk file numbers are all positive integers. There are two Open Modes for disk files that we can putc() to: “P”utc mode, or “A”ppend mode. The first mode just OPENs a file for “O”utput like a typical BASIC

```
OPEN “O”, “FILENAME/EXT”, 1
```

for example. If FILENAME/EXT exists, any content will be lost, and we’ll begin PRINTing characters to it from its beginning. If it doesn’t exist, a new file will be created.
Pretty simple so far: the same basic operation for PRINTing to a device or a disk file. Here’s the pseudocode for all of putc() again, this time with the BASIC code interspersed:

```
putc(   c$, file#) {  /* write char or string c$ on device or disk file#  2200   */
/*  VA$(0),
      VA%(0)  */
    if (file# > 0) {
    /* "X5%= VA%(0): IF X5% < 1 THEN 2250"                              2200   */
        /* disk file       */
        /* DiskPutc( c$, file#)  */
        if (OpenType[file#] == APPEND_MODE {
        /* "IF SY%(X5%,1) <> 6 THEN"                                    2202a  */
            /* "X9$= VA$(0):"                                           2210a  */
            for (p= 1; p <= LEN( c$); p++)
            /* "FOR X4%= 1 TO LEN( X9$):"                               2210b  */
                Append( c$[p], file#);
                /* VA$(0)= MID$( X9$, X4%, 1): GOSUB 2500:
                    NEXT: RETURN"                                       2210b  */
            }
        else {
            /* "ELSE" -- "P" mode */
            PRINT# file#, c$;
            /* "PRINT# VA%(0), VA$(0): RETURN"                          2202b  */
            }
        return;
        }
    if (file# == SCREEN) {
```

```
    /* "IF X5% = FN SC%( 0)"                                    2250a  */
        PRINT c$;
        /* "PRINT VA$(0);: RETURN"                              2250b  */
        return;
        }
    if (file# == PRINTER) {
    /* "IF X5% = FN PR%( 0)"                                    2255a  */
        LPRINT c$;
        /* "LPRINT VA$(0);": RETURN"                            2255b  */
        return;
        }
    /* "' FOR NOW: INSERT TESTS FOR CAS1..2, RS-232 LATER" (2260..)     */
    }
```

Now comes the trickier code, where we APPEND to an existing disk file:

```
        if (OpenType[file#] == APPEND_MODE {
            for (p= 1; p <= LEN( c$); p++)
                Append( c$[p], file#);
            }
```

Notice that we've provided for the possibility that we're going to putc() a string of characters rather than just one character, appending them to the end of the disk file. This allows us to make a "puts()" ("put string") routine which is just a synonym for "putc()" later on. But that's not what makes the "A"ppend Mode section of this code more complex. The complexity is hidden withing the "Append(c$[p], file#)" call.

If you've been following this series, you may recall that in the discussion of OPENing files we determined that OPENing a file for "A"ppend Mode will set the "next character" pointer to the end of the file, ready for appending. Now every call to Append() will just add a character there. What does Append() look like? Here it is in pseudocode, first without any BASIC code comments:

```
Append( c$,    file#) {  /* append 1 char/byte to file#; used by putc()        */
    FIELD file#, CharPos[file#] AS Spacer$, 1 AS Char$;
    LSET Char$= c$;
    CharPos[file#}++;
    TotalCharCount[file#]++;
    if CharPos[file#] == RECORD_END {
```

```
        PUT file#, RecordCount[file#];
        RecordCount[file#]++;
        CharPos[file#]= 0;
        }
    }
```

All these variables, CharPos[], TotalCharCount[], and RecordCount[], were initialized when file# was OPENed in "A"ppend mode. In BASIC, the file is actually OPEN in "R"andom mode, and so we FIELD the current record in order to place the next character. Spacer$ just "spaces" us past the existing data in this record; then we LSET the next byte to be Char$, a single character. Both CharPos[] and TotalCharCount[] are incremented: CharPos[] to tell us where to place the next character, if any; and TotalCharCount[], which will be converted into values that can be "punched in" to the file's Disk Control Block prior to CLOSEing it.

RECORD_END is a constant. For TRSDOS 2.3, its value is 256, the size of a sector. If CharPos[] has been incremented up to this value, it's time to flush the buffer out to disk, in other words to PUT this RecordCount[]. Once that's done, we increment RecordCount[] and reset CharPos[] to 0.

Here's the pseudocode for Append(), this time with the BASIC code interspersed:

```
Append( c$,    file#) {  /* append 1 char/byte to file#; used by putc()        */
/*  VA$(0), VA%(0)                                                             */
    /* "X8%= VA%(0): X7%= SY%(X8%,2):"                          2500a  */
    FIELD file#, CharPos[file#] AS Spacer$, 1 AS Char$;
    /* "FIELD X8%, X7% AS X7$; 1 AS X9$:"                       2500b  */
    LSET Char$= c$;
    /* "LSET X9$= VA$(0)"                                       2500c  */
    CharPos[file#}++;
    /* "SY%(X8%,2)= X7%+1:"                                     2502a  */
    TotalCharCount[file#]++;
    /* "SY!(X8%,2)= SY!(X8%,2) + 1"                             2502b  */
    if CharPos[file#] == RECORD_END {
    /* "IF SY%(X8%,2) < 256 RETURN"                             2504   */
        PUT file#, RecordCount[file#];
        /* "X8!= SY!(X8,1): PUT X8%,X8!:"                       2506a  */
        RecordCount[file#]++;
        /* "SY!(X8%,1)= X8! + 1:"                               2506b  */
        CharPos[file#]= 0;
        /* "SY%(X8%,2)= 0:"                                     2506c  */
```

```
    }
/* "RETURN"                                          2506d  */
}
```

This may be a good time to review what all those SY%()s and SY!()s refer to. In SKELETON/BAS, I've set up three multi-dimensional arrays, SY%() SY!() and SY$(), to hold "system" values such as buffers, file names, and various pointers within a file buffer. Here's a partial lowdown on what each array reference means:

PseudoCode	SKELETON ref	Notes:
argc	SY%(0,0)	
argv[n]	AV$(n)	
Buffer$[file#]	SY$(n,1)	stored in temp X9$
CharPos[file#]	SY%(n,2)	FIELD CharPos AS Spacer$
CurrentCharCount[file#]	SY!(n,0)	
file#	VA%(0)	file or device code
FilesOpen	SY%(5,0)	total files fOpen()ed
KbdByteStreamEOF$	SY$(5,0)	default: [Shift][UpArrow]
KbdTextStreamEOF$	SY$(6,0)	default: "."
newline$	CHR$(13)	TRS-80 newline= CR
NextRec#	X9!	temporary variable
OpenType[file#]	SY%(n,1)	
Result	VR%(0)	SUCCESS/FAIL
RecordCount[file#]	SY!(n,1)	*next* record to -- GET in getc() -- PUT in Append()
Return$	VR$(0)	
Spacer$	X9$	temporary variable
StdSlot[n]	SY%(n,0)	n in 1..3
TotalCharCount[file#]	SY!(n,2)	

Skeleton refs

==========

SKELETON ref	PseudoCode	Value(s)	Note(s)
SY%(file#,1)	OpenType[file#]	1..6	
SY%(0,2)	CharPos[keyboard]	1..256	for keyboard
SY%(file#,2)	CharPos[file#]	0..256	for disk
		1..256	for keyboard
SY!(file#,0)	CurrentCharCount[file#]	0..EOF	
SY!(file#,1)	RecordCount[file#]	0..EOF	next record to-- -GET in disk getc() when 256 chars read in or after file OPENed. -PUT in disk append putc() when sector full or file to be CLOSEd.
SY!(file#,2)	TotalCharCount[file#]	0..EOF	for disk only
SY$(0,1)	Buffer$[0]		for keyboard
SY$(file#,1)	Buffer$[file#]	""..256 chars	
SY%(5,0)	FilesOpen	0..FN DN%(0)	
			# of files fOpen()ed
SY%(n,0)	StdSlot[n]	1..3	stdin, stdout, stderr

Now that we have putc() implemented, we can write putchar(), which is “simply”

```
putc( c, stdout);
```

It’s actually not that simple, because we have to determine what “stdout” is. It could be the screen, the printer, or a file, as in these examples where “myprog” is invoked from the shell:

```
MYPROG
MYPROG >/DEV/LPR
MYPROG >MYFILE/DAT
```

Somehow, when SKELETON/BAS inputs the CMDLINE/SH file, it assigns the screen, printer, or file to "stdout". We haven't gotten to that initialization code yet to know *how* it does that, but we can assume that "file#" will contain either a negative integer specifying which device to putc() to, or a positive integer for a disk file. Here's the pseudocode for putchar():

```
putchar(    c$) {    /* write char or string c$ on stdout    */
    putc( c$, StreamTranslate( STDOUT));
    }
```

Wait. What's that "StreamTranslate()"? Oops. I forgot to mention: some streams can be redirected to other streams! An example of this is when we want the stdout to be sent to the same destination as the stderr, as in

```
    MYPROG 3>ERRFILE/TXT 2>3
```

which says, "Run 'myprog', sending all error messages to ERRFILE/TXT. Oh, and send all stdout there too."

The shell program SH/BAS takes care of setting the stderr line in CMDLINE/SH to ERRFILE/TXT, and it sets the stdout line in that file to a value of 100 x the stream number of stderr, or in this case, 300. For our example, this is what the four lines of CMDLINE/SH would look like:

(Line 1) MYPROG
(Line 2)
(Line 3) 300
(Line 4) ERRFILE/TXT

(An empty line, like line 2 above, indicates to SKELETON/BAS that it should use the default value. Line 2 assigns stdin, so the standard input will be taken from the default source: the keyboard.)
The stdout stream is set to a value of 300. StreamTranslate will interpret that for a file or device that putc() can understand:

```
StreamTranslate( StreamCode) {  /* translates "raw" Stream Code into stream handle     */
    if (StreamCode > 99)                                                    /*  64040  */
        Slot= StdSlot[ Int( StreamCode / 100)]
    else
        Slot= StreamCode;
```

```
    /* "DEF FN ST%( SL%)= ABS(SL% > 99) * SY%(ABS( INT( SL% / 100)),0) +
        ABS( SL% < 100) * SL%"                                          */
    return Slot;                    /* AKA "stream"                    */
    }
```

Notice that StreamTranslate() doesn't just divide the "300" value by 100 and return a "3". It returns the *Slot* (I/O stream *number*) that this value refers to: a number between 1 and 3. In other words, whatever value is *in that Slot.* In our example, StreamTranslate() will convert the 300 that's in Slot 2 (stdout) into a 3, and then return what's in Slot 3, ie whatever device # or file# is in Slot 3 (stderr). This becomes the file# that's passed on to putc().

getc() is a bit more complicated. Here it is in pseudocode:

```
getc( file#) {              /* Read 1 char/byte from file #    */
    Result= FAIL;
    if !(file# < 0) {
        /* Disk file */
        if (!OpenType[file#] == "G")
            /* not in "G"etc() mode    */
            return;
        if (GetcIsByteStream) {
            /*  ** Byte-Stream getc() */
            while (Result=
             (CurrentCharCount[file#] < TotalCharCount[file#]) {
                if (CharPos[file#] == RECORD_END) {
                    NextRec#= RecordCount[file#];
                    GET file#, NextRec#;
                    CharPos[file#]= 0;
                    RecordCount[file#]++;
                    }
                else {
                    FIELD file#, CharPos[file#] AS Spacer$, 1 AS ReturnChar$;
                    CharPos[file#]++;
                    CurrentCharCount[file#]++;
                    }
                }
            }
        else {
```

```
        /*  **   Text-Stream getc()      */
        while (CharPos[file#] > LEN( Buffer$[file#] {
            if (EOF( file#))
            LINEINPUT# file#, Temp9$;
            Buffer$[file#]= Temp9$ + newline$;
            CharPos[file#]= 1;
            }
        Return$= MID$( Buffer$[file#], CharPos[file#], 1);
        CharPos[file#]++;
        CurrentCharCount[file#]++;
        Result= !(CurrentCharCount[file#] > TotalCharCount[file#]);
        /* Result= SUCCESS if not at EOF  */
        if (!Result)
            DefaultFileParams( file#);
        }
    }
else {
    /*   ** Keyboard getc()      */
    if (file# == KEYBOARD) {
        if (GetcIsByteStream) {
            Return$= FastByteStreamKbdGetc();
            Result= !(Return$ == KbdByteStreamEOF$;
            }
        else {
            /* Text-Stream:     */
            Result= FAIL;
            if (CharPos[file#] > LEN( Buffer$[0])) {
                LINE INPUT Buffer$[0];
                if (Buffer$[0] == KbdTextStreamEOF$) {
                    Return$= "";
                    ResetKeyboard();
                    return;
                    }
                Buffer$[0]= Buffer$[0] + newline$;
                CharPos[file#]= 1;
                }
            Return$= MID$( Buffer$[0], CharPos[file#], 1);
            CharPos[file#]++;
```

```
                Result= SUCCESS;
                }
            }
        }
    }
```

Other than simply ensuring that the file or device is indeed OPENed in "G"etc Mode, this code is simply divided into two main blocks: one for disk files, and the other for keyboard input. Then each block is again divided into two options: "Byte-Stream" vs "Text-Stream" input.
"Text-stream" means simple lines of ASCII characters, terminated by "newline"—in the case of the TRS-80, a simple carriage return (CR, with an ASCII code of 13). "Byte-stream" on the other hand is a free-form river of uninterpreted bytes, like a data file or some program code.
We'll analyze the pseudocode, this time with the BASIC code interspersed:

```
getc( file#) {          /* Read 1 char/byte from file #              */
/*  VA%(0)
                        Temp variables:
                        X9%    =VA%(0)       file#
*/
    Result= FAIL;
    /* "VR%(0)= 0: X9%= VA%(0):"                          2000a  */
```

The comments after

```
getc( file#) {          /* Read 1 char/byte from file #              */
```

are a reminder of what argument array element—VA%(0)—corresponds to the pseudocode argument "file#". Then we note that this file# will be kept in the temporary variable X9%.
It's a good idea to start out as though our procedure is unsuccessful, and only change that if and when it's successfully done its job, so we set the standard "Succeed/Fail" return value VR%(0) to 0 ("Fail"). Now, on to the real work.

```
    if !(file# < 0) {
    /* "IF X9% < 0 THEN 2050"                             2000b  */
        /* Disk file                */
        if (!OpenType[file#] == "G")
        /* "IF NOT SY%(X9%,1) = 4"                                */
            return;
```

```
        /* "RETURN"   not "getc()" OPEN type, so exit.   2002    */
```

Disk file#s have a positive value, so "if !(file#<0)" is simply asserting "if the file# is not negative". Then we confirm that the file has been opened in "G" mode: if its OpenType is NOT "G", we simply bail out of the procedure with a "return". Note that we previously set VR%(0) to "Fail" so the calling program can see that we didn't getc() anything.

```
    if (GetcIsByteStream) {
    /* "IF NOT SY%(4,0) THEN 2020"                    2004    */
        /*   ** Byte-Stream getc()     */
```

getc() can be operating either in "byte-stream" or "text-stream" mode, as mentioned above. SY%(4,0) is a "system" variable that was set by SKELETON/BAS in its initialization phase, and defaults to "text-stream", or a value of 0, in which case "NOT SY%(4,0)" would return a -1, and the interpreter will jump around this block to line 2020. But here's what happens if it's set to "byte-stream" input:

```
        while (Result=
         (CurrentCharCount[file#] < TotalCharCount[file#]) {
         /* "VR%(0)= SY!(X9%,0) < SY!(X9%,2):
                IF NOT VR%(0) RETURN"                  2006    */
```

We'll continually step through the file, byte-by-byte, until the CurrentCharCount is the same as the TotalCharCount for this file: in other words, until we've reached the end of the file. TotalCharCount[file#] was set when the file was OPENed (See the previous article, "OPEN For Business"). Note that we set Result based on this condition: if we reach end-of-file, it reports "Fail", because getc() has no more bytes to return. In that case, we just RETURN from this procedure. On the other hand, if we haven't reached the end of the file yet,

```
            if (CharPos[file#] == RECORD_END) {
            /* "IF SY%(file#,2) = 256"                          2008a  */
                NextRec#= RecordCount[file#];
                /* "X9!= SY!(X9%,1)"                            2008b  */
                GET file#, NextRec#;
                /* "GET X9%, X9!"                               2008c  */
                CharPos[file#]= 0;
                /* "SY%(X9%,2)= 0"                              2008d  */
                RecordCount[file#]++;
```

```
        /* "SY!(X9%,1)= X9! + 1"                          2008e  */
        }
        /* "GOTO 2006"                                    2008f  */
```

The getc() procedure steps through each sector or "record", one byte at a time, from byte 1 to RECORD_E ND (ie, byte 256). CharPos is actually the position within the sector *just before* the next byte to read, so for the 10th byte within the record, CharPos will have a value of 9, and can be used in a FIELD statement like

```
FIELD 1,
        CharPos    AS Spacer$,          ie, skip these bytes
        1          AS Byte$             to access this byte
```

Now if CharPos has been incremented via multiple getc()s to the value of 256, we would be trying to access the 257th byte in a 256-byte sector! It would then be time to get the next sector, which is what this code does.
RecordCount[file#] keeps track of the *next* random-access record to GET, and is initially set by fOpen() to a value of 1. fOpen() had also set CharPos[file#] to 256, ensuring that this event gets triggered the first time we call getc(file#).
So now we GET NextRec#, and reset CharPos[file#] to 0 so that we'll fetch the 1st byte in the new record. Then we also update RecordCount[file#] so that the next time we need a new record, it will point to the next one.
This is part of a while() loop, so at this point the interpreter will jump back to the while() test (BASIC line 2006). That test should pass, because we haven't altered CurrentCharCount[file#}, and now when we test CharPos[file#] again, it's 0, so we'll pass down to the "else {" portion:

```
    else {
        /*"X8%= SY%(X9%,2):"                             2010a  */
        FIELD file#, CharPos[file#] AS Spacer$,
         1 AS ReturnChar$;
        /* "FIELD X9%, X8% AS X9$,  1 AS VR$(0):"        2010b  */
        CharPos[file#]++;
        /* "SY%(X9%,2)= X8% + 1:"                         2010c  */
        CurrentCharCount[file#]++;
        /* "SY!(X9%,0)= X8% + 1"                          2010d  */
        return;
        /* "RETURN"                                       2010e  */
        }
```

First, the BASIC code uses the temporary variable X8% simply to increase speed and reduce program memory usage by a little:

```
/*"X8%= SY%(X9%,2):"                                    2010a  */
```

Then we skip over the previously read bytes with Spacer$. The return variable VR$(0) gets the value of the next byte after Spacer$ / X9$. Now we update CharPos[file#[and the CurrentCharCount[file#] so that both can be tested next time through getc(). Finally, we RETURN with the new byte value in VR$(0).

But the default is “text-mode” I/O, which is used in all the Software Tools. This sees a file as a collection of ASCII character lines, each terminated by a “newline”, or Carriage Return on the TRS-80. We tested GetcIsByteStream, also known as SY%(4,0), above. If it’s *not* set to “byte-stream” and has a value of 0, we get to this branch:

```
else {
    /*  **   Text-Stream getc()     */
    Temp9$= Buffer$[file#];
    /* "X9$= SY$(X9%,1)"                                  2020a  */
    while (CharPos[file#] > LEN( Buffer$[file#] {
    /* "IF NOT SY%(X9%,2) > LEN( X9$) THEN 2030"          2020b  */
        if (EOF( file#))
            return;
        /* “IF EOF( X9%) RETURN”                          2024   */
        LINEINPUT# file#, Temp9$;
        /* “LINE INPUT #X9%, X9$:”                        2026a  */
        Buffer$[file#]= Temp9$ + newline$;
        /* “SY$(X9%,1)=X9$+FNNL$(0):”                     2026b  */
        CharPos[file#]= 1;
        /* “SY%(X9%,2)= 1:”                               2026c  */
        }
        /* "GOTO 2020"                                    2026d  */
    Return$= MID$( Buffer$[file#], CharPos[file#], 1);
    /* "VR$(0)= MID$( X9$, SY%(X9%,2), 1):"               2030a  */
    CharPos[file#]++;
    /* "SY%(X9%,2)= SY%(X9%,2) + 1:"                      2030b  */
    CurrentCharCount[file#]++;
```

```
/* "SY!(X9%,0)= SY!(X9%,0) + 1:"                          2030c  */
Result= !(CurrentCharCount[file#] > TotalCharCount[file#]);
/* "VR%(0)= NOT (SY!(X9%,0) > SY!(X9%,2)):"               2030d  */
/* Result= SUCCESS if not at EOF  */
if (!Result)
    DefaultFileParams( file#);
    /* "IF NOT VR%(0) GOSUB 60700"                        2030e  */
}
```

Once again, we start with a temporary variable to save time and space:

```
Temp9$= Buffer$[file#];
/* "X9$= SY$(X9%,1):"                                     2020a  */
```

Now with "text-stream" I/O, we're not scanning through each sector of a random-access file; we're scanning a "buffer" instead, which holds a string of 0 to 256 characters. They're similar, but the buffer is a variable-length string. So this time we scan from the 1st position in the string until the end of that string.

At this point we're ready to either continue scanning the buffer for the next character, or to get a new line from disk into the buffer and start at the beginning. fOpen() initialized the buffer to be an empty string with length 0, and CharPos[file#] to be 1, so that we're forced to get the first line of text from the disk file. But before we do, what if the file is empty? Or what if we've been merrily using getc(), and now we're at the end of the file? Shouldn't we test for that first?

```
if (EOF( file#))
    return;
/* "IF EOF( X9%) RETURN"                                  2024   */
```

If so, we RETURN from getc() with VR%(0) signaling "Fail", so the calling program knows it has no new character for us.

But if not, it's time to read in a new line of text.

```
LINEINPUT# file#, Temp9$;
/* "LINE INPUT #X9%, X9$:"                                2026a  */
Buffer$[file#]= Temp9$ + newline$;
/* "SY$(X9%,1)= X9$+FN   NL$( 0):"                        2026b  */
CharPos[file#]= 1;
/* "SY%(X9%,2)= 1:"                                       2026c  */
```

```
}
/* "GOTO 2020"                                    2026d  */
```

We read the next line into Temp9$, and add a newline (Carriage Return) to it before stuffing that result into the Buffer[]. Why the "extra" newline? Because LINE INPUT "strips off" the newline that terminated the string it just read in! (It makes sense: if you use LINE INPUT from the keyboard, as in

```
LINE INPUT A$
```

you'll terminate your entry by pressing the ENTER key (*AKA* newline, *AKA* Carriage Return). But if you look at A$, it will contain everything you typed *except* the ENTER key. That's just what LINE INPUT does, but in *nix (Unix, Linux, Software Tools, C, etc), getc() *will* retain the newline, *unless* you terminated your input with a Ctrl-D. So we just add a newline, which also works well if you type a bunch of ENTERs in order to indicate "a bunch of empty lines". What's that FN NL$(0)? It's just a constant value for "newline", and it returns CHR$(13), the Carriage Return.
Now we reset CharPos[] to character position 1. Notice that, in "text-stream" mode, CharPos doesn't tell us how much space is *before* the next character, but rather where in the buffer string the next character is located; so it's 1 more than its purported value in "byte-stream" mode.)
Now we loop back and test again: is CharPos[] greater than the length of the string? No—even with an empty line, there's at least 1 character in the buffer: the newline. So we drop down to the next bit of code.

```
Return$= MID$( Buffer$[file#], CharPos[file#], 1);
/* "VR$(0)= MID$( X9$, SY%(X9%,2), 1):"           2030a  */
CharPos[file#]++;
/* "SY%(X9%,2)= SY%(X9%,2) + 1:"                  2030b  */
CurrentCharCount[file#]++;
/* "SY!(X9%,0)= SY!(X9%,0) + 1:"                  2030c  */
Result= !(CurrentCharCount[file#] > TotalCharCount[file#]);
/* "VR%(0)= NOT (SY!(X9%,0) > SY!(X9%,2))"        2030d  */
```

This is fairly straightforward: the next character is MID$(Buffer$[]. CharPos, 1)—one character in the Buffer at CharPos. Then we increment CharPos and CurrentCharCount for the file. Finally, we test if CurrentCharCount is past the end of the file, and if so, we will report "Fail" so that the calling program won't ask for anything more from getc().

```
        /* Result= SUCCESS if not at EOF */
        if (!Result)
            DefaultFileParams( file#);
            /* "IF NOT VR%(0) GOSUB 60700"                    2030e */
        }
        /* "RETURN"                                           2032  */
    }
```

If we are at the end of the file, we call

```
DefaultFileParams( file#);
```

and in either case, we return from getc(). What does DefaultFileParams(file#) do? Well, duh—it sets up the default file parameters for this file! We looked at its code in "OPEN For Business" in the last issue of TRS8bit, but here it is again:

```
DefaultFileParams( file#, mode#) {    /* Set up default disk file parameters
        VA%(0), VA%(1)                    when opening in "G" mode          */
    OpenType[file#]= mode#;
    /* "SY%(X9%,1)= X9%:"                                     60700a*/
    CurrentCharCount[file#]= 0;
    /* "SY!(X9%,0)= 0:"                                       60700b*/
    TotalCharCount[file#]= TotalBytes( file#);
    /* "SY!(X9%,2)= FN DX!( X9%):"                            60700c*/
    ResetDiskBuffer( file#);
    /* "GOSUB 60810"                                          60700d*/
    Result= Success;
    /* "VR%(0)=-1:"                                           60790a*/
    }
    /* "RETURN"                                               60790b*/
```

Finally, we come to the other major block of code in getc(): keyboard input. (I hope eventually to provide for cassette and perhaps even serial I/O, but I'm not there yet.)

```
    else {
        /*  ** Keyboard getc()    */
        if (file# == KEYBOARD) {
        /* "IF NOT X9% = FNKY%(0) RETURN"                      2050  */
```

At present, our only valid input *device* is the keyboard.

```
        if (GetclsByteStream) {
        /* "IF SY%(4,0)"                                    2052a  */
            Return$= FastByteStreamKbdGetc();
            /* "GOSUB 1900:"                                2052b  */
            Result= !(Return$ == KbdByteStreamEOF$;
            /* "VR%(0)= NOT VR$(0) = SY$(5,0):"             2052c  */
            }
            /* "RETURN"                                     2052d  */
```

"Byte-stream" in the context of the keyboard simply means getting a keystroke without echoing it to the screen, using INKEY$. (This might be useful for entering passwords.) That's accomplished using the excessively long-named FastByteStreamKbdGetc():

```
FastByteStreamKbdGetc() {    /* get 1 keypress                    */
    for (temp9= 0; ; temp9+= 0) {
    /* "FOR X5%= 0 TO 1 STEP 0:"                            1900a  */
        key$= INKEY$;
        /* "X9$= INKEY$:"                                   1900b  */
        if LEN( key$) > 0 {
        /* "IF LEN( X9$) > 0 THEN"                          1900c  */
            return key$;
            /* "VR$(0)= X9$: X5%= 1"                        1900d  */
            }
        }
        /* "NEXT:"                                          1902a  */
    }
    /* "RETURN"                                             1902b  */
```

This is a modified INKEY$ loop. Normally we'd write something like

```
1900    X9$= INKEY$: IF X9$="" THEN 1900 ELSE  VR$(0)= X9$: RETURN
```

Why use FOR..NEXT? Confession time: *because it's faster.* FOR..NEXT doesn't need to look for a line number to GOTO, and every line number search in TRS-80 BASIC scans the program from

the beginning. The only reason for X5%'s existence is to keep FOR..NEXT looping around; once we've got a valid keypress, it gets set to 1, and the FOR..NEXT loop terminates.
Back to the keyboard "byte-stream" getc() code. Once we've gotten a keystroke from FastByteStreamKbdGetc(), we test it against the unique KbdByteStreamEOF$, which stored in SY$(5,0), set up when SKELETON/BAS is initialized, and defaulting to [Shift]{UpArrow], CHR$(27), the TRS-80's version of the [Esc] key. This part of getc() reports "Success" in VR%(0) if it isn't the EOF$ character, and returns the character in VR$(0).

But normally, keyboard input will be in "text-stream" mode. This code is quite similar to the disk "text-stream" getc() code:

```
else {
    /* Text-Stream:    */
    if (CharPos[file#] > LEN( Buffer$[0])) {
    /* "IF NOT SY%(0,2) > LEN( SY$(0,1)) THEN 2070"          2062   */
        LINE INPUT Buffer$[0];
        /* "LINE INPUT X9$:"                                 2064a  */
        if (Buffer$[0] == KbdTextStreamEOF$) {
        /* "IF X9$ = SY$(6,0) THEN"                          2064b  */
            Return$= "";
            /* 'VR$(0)= "":'                                2064c  */
            ResetKeyboard();
            /* "GOSUB 60800:"                               2064d  */
            return;
            /* "RETURN"                                     2064e  */
            }
        Buffer$[0]= Buffer$[0] + newline$;
        /* "SY$(0,1)= X9$ + CHR$(13):"                       2066a  */
        CharPos[file#]= 1;
        /* "SY%(0,2)= 1"                                    2066b  */
        }
    Return$= MID$( Buffer$[0], CharPos[file#], 1);
    /* "VR$(0)= MID$( SY$(0,1), SY%(0,2), 1):"               2070a  */
    CharPos[file#]++;
    /* "SY%(0,2)= SY%(0,2) + 1:"                            2070b  */
    Result= SUCCESS;
    /* "VR%(0)= -1:"                                        2070c  */
    }
    /* "RETURN"                                             2070d  */
}
```

There are only a few differences between "text-stream" keyboard getc() code and that for disk files or "byte-stream" keyboard input.

1. End-of-file: instead of testing EOF(file#) and TotalCharCount[file#], we have to test for a special KbdTextStreamEOF$ character, similar to "byte-stream" keyboard input. Notice that this is a different character than KbdByteStreamEOF$, which was [Shift][UpArrow]. This is initialized to a single period, and it must appear alone at the beginning of a line, as in

   HERE IS SOME TEXT

   HERE IS SOME MORE
   END OF FILE COMING UP:
   .
2. When end-of-file is encountered, ResetKeyboard() is called, end then getc() exits. Here's the pseudocode for ResetKeyboard():

```
ResetKeyboard() { /* reset Keyboard Input Buffer after newline in getc()          */
    Buffer$[0]= "";
    CharPos[0]= 1;
    /* 'SY$(0,1)= "": SY%(0,2)= 1"                                          60800 */
    }
```

Why do this? In case the user program wants to get more from the keyboard further on in the program.

Of course, just as putchar() uses putc() to write to "stdout", the standard output, so getchar() uses getc() to obtain one character at a time from "stdin".

```
getchar() {          /* Read 1 char/byte from stdin                              */
    getc( StreamTranslate( STDIN));
    /* "VA%(0)= FN ST%( SY%(1,0)): GOTO 2000"                               2100 */
    }
```

Like putchar(), getchar() uses StreamTranslate() in case of I/O redirection.

Next episode: A CLOSE Call. We'll see how SKELETON/BAS CLOSEs its files, and then look into additional procedures available to help run Software Tools in BASIC.

If you ever want to comment on this series of articles, you can find Vince Otten in any of the TRS-80 or trs80gp groups on Facebook.

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

The amazing

"FreHD"

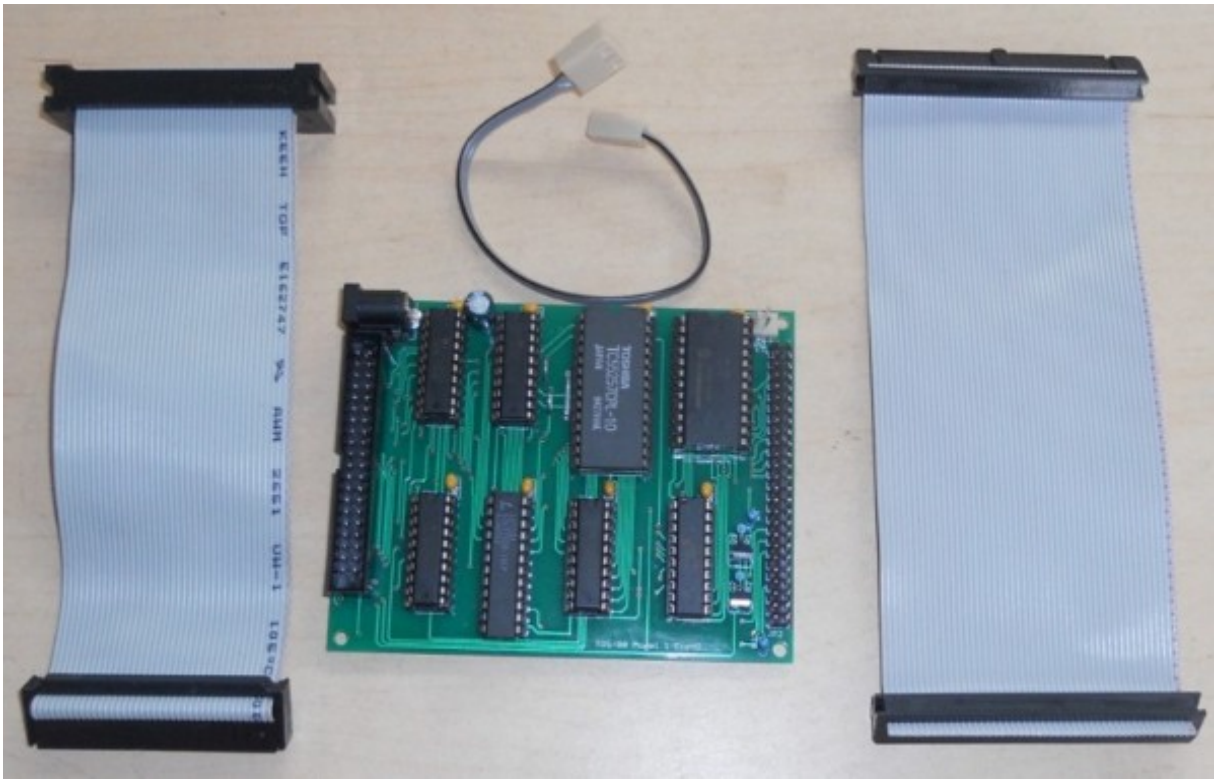


-Emulates a TRS-80 hard drive, but faster than any hard drive!
-Works with your favourite DOS (LS-DOS, LDOS, CP/M, Newdos/80 2.5)
-Uses SD card for storage medium
-Bonus free Real Time Clock function!
-Designed in Belgium and proudly built and shipped from Australia
-Kit form or fully assembled

Order yours today
http://members.iinet.net.au/~ianmav/trs80/

“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”

Thirsting for adventure?
Brass lantern getting dusty?



Take your Model 1, 3 or 4 on exciting new adventures never available until now....

M4ZVM enables your TRS-80 Model 1,3 or 4 to play Infocom and post-Infocom adventures, taking full avdantage of your hardware. Supports old favourites like Zork and Seastalker (including Sonarscope), as well as new-to-Model-4 games including Trinity, Border Zone and Sherlock, plus hundreds more available in the Interactive Fiction archive.



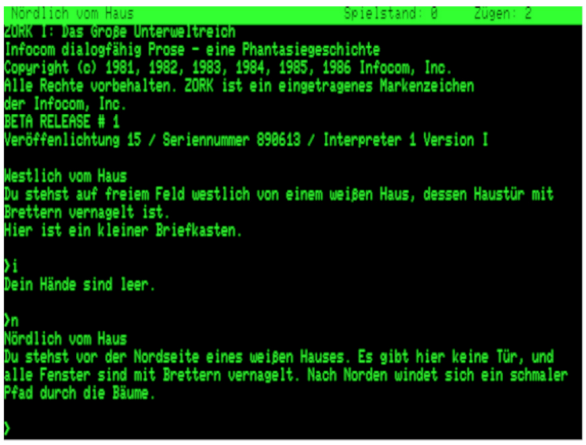
M4ZVM is available for download from https://gitlab.com/sijnstral/m4zvm (includes feature details and source)
Enhanced versions ready for download!

*128K recommended

Actual Screenshots of games new to the model 4, running M4ZVM (on trs80gp)



Beyond Zork



Zork 1 (German)

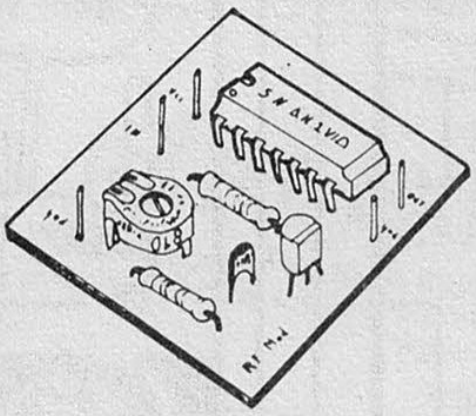
M4ZVM features include:
* z1-z8 game execution (z6 support is experimental)
* named game save/load
* accented characters
* 128K recommended - NEW Minimised 64K version
* Optionaly set the infamous "Tandy bit"
* NEW Model 1 and Model 3 versions



LOW COST R.F. MODULATOR

Only £ 2.—

includes p.&p.!



Converts ordinary T.V. into Video-Monitor. Use it for computing or video-games. This board has a special Video-IC and is easy to build (ten minutes). No special adjustments, works always. Tuning by turning potentio-meter.
Low Power 5 volt. Kit includes all parts, epoxy p.c. board and assembly instructions written in English.

How to order: Send your order and remittance to the address below. Or, send us your undersigned Barclays or Eurocheque for a value of £2.

Address to: Musicprint Computer Products B.V.

P.O. BOX 410
ARKELSE ONDERWEG 31
4200 AK GORINCHEM
THE NETHERLANDS

Your video R.F. Modulator will be shipped the same day we receive your order

dealer inquiries invited

Take your screen printer and 80 to the land of the low for less than $10.

Lower Cost Lowercase

Philip M. Van Praag
PO Box 35667
Tucson, AZ 85740

This article describes a lowercase modification and a KB fix for the Radio Shack Screen Printer, as well as a companion lowercase display modification for the TRS-80. These hardware modifications, unlike Radio Shack's, should cost less than $10.

Background

Both the TRS-80 and the Screen Printer contain character generator ICs that are capable of producing lowercase characters. What prevents them from doing this is the manner in which they are addressed.

Before we can describe the address control, however, let's explain the ASCII code used to select character generator outputs.

Table 1 contains the ASCII bit code for numbers, symbols and uppercase/lowercase characters. Each binary bit is either a zero (logic level low) or a one (logic level high). These highs and lows provide unique addresses which tell the character generator which character is needed. The bits to note are 5 and 6.

Notice that the only difference between the uppercase and lowercase versions of the same letter is that bit 5 is always a zero (low) for the uppercase and a one (high) for the lowercase.

Also, bit 6 is always high for letters, and low for numbers and symbols.

In the TRS-80 design, this eliminates the need for a bit 6 video storage IC (called a RAM).

TRS-80 ROM ICs decode each keyboard entry via an internal lookup table, and send the ASCII code to video RAM. Since the ROM output to video RAM never contains lowercase letters, bit 5 is high only when a number or symbol is present. Thus, bit 6 is not needed in this "uppercase only" application.

The Screen Printer prints exactly what you see on the monitor screen; that is, it prints the contents of the video RAM memory. Since the TRS-80 initially does not contain stored lowercase data (does not store bit 6), the Screen Printer synthesizes its own bit 6 for addressing its character generator IC. Since it only has to print uppercase letters and numbers/symbols, it synthesizes bit 6 by inverting bit 5.

The Scripsit program apparently contains its own lookup table to provide ASCII addressing for video RAM, thus providing lowercase letter addresses. All we need to do to modify the TRS-80 for lowercase display is to add a RAM IC to enable storage of bit 6 information.

Due to the bit 6 decoding technique used in the TRS-80, enabling the normal ASCII bit 6 function disables the graphics function. This is overcome by providing a switch to select uppercase/graphics or uppercase/lowercase.

Modifications

The Screen Printer lowercase modification involves disabling the bit 5-to-bit 6 inversion and replacing it with the real bit 6 address line. Implementing the lowercase disables the graphics feature, so again we provide a switch to select graphics or loer case. The printer must latch the incoming data from video RAM. We are now bringing in bit 6, and we must provide a data latch for

6_5	0_{10}	0_{11}	1_{00}	1_{01}	1_{10}	1_{11}
3210	Numbers/ Symbols		Uppercase Alpha		Lowercase Alpha	
0000	(SP)	0	@	P	\	P
0001	!	1	A	Q	a	q
0010	"	2	B	R	b	r
0011	#	3	C	S	c	s
0100	$	4	D	T	d	t
0101	%	5	E	U	e	u
0110	&	6	F	V	f	v
0111	'	7	G	W	g	w
1000	(	8	H	X	h	x
1001	)	9	I	Y	i	y
1010	*	:	J	↑	j	z
1011	+	;	K		k	
1100	,	<	L		l	
1101	−	=	M		m	
1110	.	>	N		n	
1111	/	?	O		o	

Table 1. ASCII code for numbers/symbols and uppercase/lowercase letters.

PMC-80 Expanded



Use all standard peripherals and existing software

When you buy PMC-80 you get hardware and software compatibility with the most popular microcomputer system in the world—that means thousands of disk and cassette based programs and all kinds of peripherals are instantly available!

PMC-80 has configurations that give the computer enthusiast a way to grow from a STARTER system in affordable increments. Begin at a low $675 for the basic 16K level II system and grow to the complete 48K memory system pictured above with two floppy disks for less than $3000.

FASTLOAD option inputs short programs as fast as "disk" from ordinary, standard format cassettes. Fast, reliable and economical!

PMC-80 COMMUNICATOR option provides interface to modems and parallel port printers. Take your pick of peripherals for communication with electronic bulletin boards and low cost timeshare services via phone lines from your home or business.

PMC-80 EXPANDER option provides the most powerful configuration with a total of 48K memory, provision for 4 mini-floppies, printer interface, RS-232C communications interface, plus a slot for the popular S-100 boards.

Sold through computer stores.

Personal Micro Computers, Inc. ↙422

475 Ellis Street, Mountain View, CA 94043 (415) 962-0220

↙See List of Advertisers on page 306

this bit. Rather than adding a new device to the circuitry, alter the operation of an existing data latch, one that is not used for non-graphics functions.

One additional Screen Printer/TRS-80 modification must be made to print with programs containing keyboard debounce (such as Scripsit).

Upon receiving a print request, the printer initiates the following sequence: TRS-80 CPU activity stops; CPU data and address outputs are disabled; the printer's next video RAM location is provided on the address bus; the video RAM output data is latched in the printer; and, finally, data is fed through the character generator and out to the print head.

The printer goes through this cycle for each of the 1024 screen character positions (16 lines of 64 characters each) every time a print is initiated.

Normal printer cycle operation disables the CPU's output to the data bus. It does not disable main RAM's output. Main RAM plays an active role in keyboard debounce routines. It is frequently addressed by the CPU when the printer takes control of the bus structure.

Once a print command has been initiated, the printer must print something at every character location. If the printer cannot seize control at the right instant, such as when the CPU is addressing main RAM, the print cycle will not occur for that character. Thus the character generator will print a null (a special character consisting of all low ASCII bits). To correct this problem, make two simple circuitry changes.

First, in the printer, alter the requirement that the CPU be in the ROM-read state in order for the printer cycle to begin. This is done by disabling a couple of diodes allowing the print cycle to begin even if the CPU is addressing RAM. Second, eliminate main RAM's contributing to the data bus (when all we want is video RAM), by utilizing an unused gate in the TRS-80. This disables the main RAM output buffers at the proper time.

Before proceeding with the modifications, check that you have all the parts and tools listed in Table 2. If you complete the mod and reassemble the equipment in one session, you will increase your chance of success.

Most Difficult

The most difficult task in the printer modifications is gaining access to the main logic board.

Although well-constructed with ample connectors, disassembling the board conjures up analogies to the old Chinese puzzle trick. All you have to do is follow this step-by-step procedure:

1.) Disconnect AC power and the interface cable.

2.) Open the hinged lid and remove the paper roll and two spindles.

Parts
SPDT Switch, miniature toggle
4PDT Switch, miniature toggle
21L02 IC, static RAM 1024 x 1
1N914 or 1N4148 Diode, silicon
1.2 kohm Resistor, 1/4 watt
10 feet wire, #30 AWG sold
Tools
Miniature diagonal cutters
Needle-nose pliers
35 Watt soldering iron, fine tip
.031 inch solder
Drill and bits (for switch mtg.)
Phillips screwdriver, #1
Xacto knife
Hex drivers, 3/16 inch and 1/4 inch

Table 2. Parts and Tool Requirements

3.) Turn the unit upside-down, and remove the six 3/16-inch hex screws.

4.) Turn the unit right-side-up, and remove the four Phillips screws which attach the print mechanism housing to the full-length main chassis.

5.)Slowly lift the housing. This will uncouple a fixed multi-pin connector. Reach underneath the housing, and unplug the small cable connector. Set the print mechanism aside.

6.) Remove the two 3/16-inch hex screws and brackets which attach the plastic power supply cover. Remove this by lifting it up and toward the rear.

7.) This exposes two 1/4-inch hex screws on the main chassis next to the power supply board. Remove these.

8.) Slide the main chassis to the rear until the print toggle switch lever clears the bottom housing. Then slowly lift the front of the chassis until you can disconnect the small LED connector on the underside.

9.) Lift the chassis rear so you can disconnect the remaining two wires: one at the fuseholder, the other at the power switch.

The main chassis should now be completely separated from the cabinet.

10.) Remove the Phillips screws which attach the logic board to the chassis. Slide the board out to the rear, and disconnect the cable connector. Disassembly is now complete.

To reassemble the printer once the modifications are done follow these steps in the reverse order.

The actual printer modifications are described below in four steps. The first three steps are performed before reassembling the printer. The last step involves only the interface cable.

Unfortunately, construction of the printer is such that you cannot test to determine whether your modifications are correct until everything is reassembled. So please take your time, and double-check your connections. Fig. 1 shows a schematic diagram of all printer modifica-

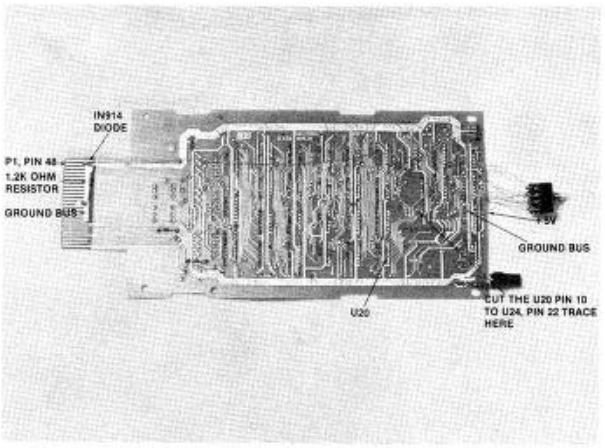


Photo 1. Underside of Printer Logic Board. Cut the pin 48 circuit trace at the ground bus before soldering diode in place.

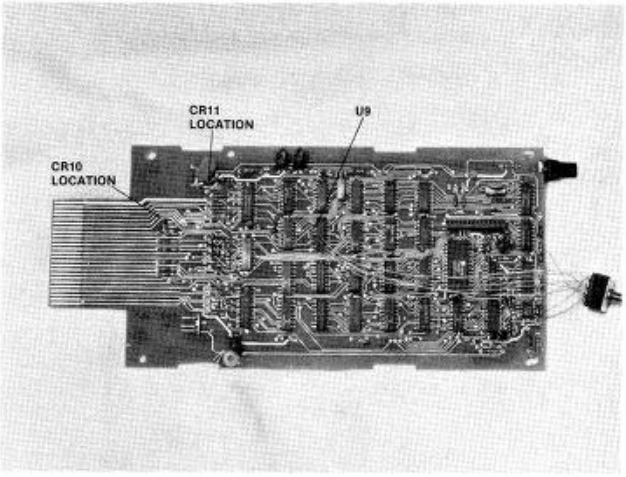


Photo 2. Top of Printer Logic Board. U9 pins 2, 5, and 6 must be cut 1/16-inch below IC body before wiring to switch.

TRS-80™ Compatible "carbonless" Continuous Statements



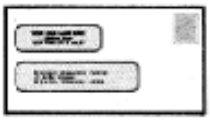
small quantities, low prices, fast delivery

Order as few as 500 statements imprinted with your firm name and address.

Only $27.95

NEBS 9062 Statements are software compatible with the TRS-80, Model 1, Level II, Accounts Receivable package #26-1555.

TRS-80 is a Trade Mark of the Radio Shack Co., Subsidiary of the Tandy Corp.



SPEED COLLECTIONS
Product 772 DU-O-VUE® Envelope (3⅝"x 6⅝") eliminates envelope addressing.

Product 9062 — Size 6"x 8⅝" detached. Prices include your firm name, address and phone in top section, plus your name only in lower section. Printed in black ink. Available in single (white) or duplicate (white, canary) continuous sets.

QUANTITY	SINGLE Product 9062-1	DUPLICATE Product 9062-2	Product 772 DU-O-VUE® Envelopes
10,000	$192.00	$355.00	$138.00
6,000	128.00	228.00	92.00
4,000	99.00	169.00	64.50
2,000	59.00	99.00	36.25
1,000	38.75	61.00	20.75
500	27.95	39.95	12.25

ORDER TODAY! MONEY-BACK GUARANTEE.
FAST SERVICE BY MAIL or PHONE TOLL FREE 1+800-225-9550 (Mass. residents 1+800-922-8580). It is our policy to ship within 6 working days following our receipt of your order.

Please ship: Date_____19____ **CODE 460**

_____ **9062-1 STATEMENTS (Single)**
_____ **9062-2 STATEMENTS (Duplicate)**
_____ **772 DU-O-VUE® Envelopes**
_____ **Information on continuous checks and other computer forms.**

HEADING TO BE PRINTED ON FORMS: (Please type or print)

STREET

CITY and STATE ZIP

PHONE

AUTHORIZED SIGNATURE
If you wish us to BILL and SHIP differently from above please indicate.

Nebs Computer Forms

78 Hollis Street, Groton, Mass. 01450
A division of New England Business Service, Inc.

MARK GORDON COMPUTERS

DIVISION OF MARK GORDON ASSOCIATES INC.
P.O. Box 77, Charlestown, MA 02129
(617) 491-7505

COMPUTERS

Mar 800 W/16K	799.00
Level II 16K System	659.00
Model II 64K System	3499.00
16E Model III	**879.00**

DISK DRIVES

40 Track 5¼ Inch drive	314.00
80 Track 5¼	544.00
4 Disk Drive Cable	39.00

PRINTERS

Centronics 730	599.00
Epson MX80B	Call for price
Epson MX70	Call for price
Centronics 737	849.00
Okidata Microline 83	999.00
Okidata Micro line 82	729.00
Integral Data 440G	999.00
NEC 5510 w-tractor	2679.00
Okidata Microline 80	499.00
Diablo 630	2495.00

MISC HARDWARE

Expansion Int. TRS-80 (Dx)	269.00
Novation Cat modem	159.00
16K Memory Kit	41.99
Leedex Monitor	119.00
Leedex 80 G	139.00
Printer Cable for above	49.00
ISO-2 Isolator	54.00
AC LINE FILTER	24.00

STORAGE MEDIA

Verbatim box 10 5¼	25.00
Memorex box 10 5¼	22.00
Plastic Storage Box	5.00

OPERATING SYSTEMS

NEWDOS by APPARAT INC	49.00
NEWDOS+ by APPARAT INC	99.00
MMS FORTH DISKETTE PRIMER	39.95
NEWDOS 80	149.00

BUSINESS SOFTWARE FOR THE TRS-80® BY SBSG

Free enhancements and upgrades to registered owners for the cost of media and mailing. 30 day free telephone support. User reference on request.

Fully Interactive Accounting Package. General Ledger, Accounts Payable, Accounts Receivable and Payroll Report Generating

Complete Package (requires 3 or 4 drives)	**$475.00**
Individual Modules (requires 2 or 3 drives)	$125.00
Inventory II (requires 2 or 3 drives)	$ 99.00
Mailing List Name & Address II (requires 2 drives)	$129.00
Intelligent Terminal System ST-80 III	$150.00
The Electric Pencil from Michael Shrayer	$150.00
File Management System	$ 49.00

FINE PRINT

TRS-80 is a Tandy Corporation trademark. Use of above operating systems may require the use of Radio Shack® TRS-DOS. Radio Shack equipment subject to the will and whim of Radio Shack.

ORDERING INFORMATION

We accept Visa and Mastercard. We will ship C.O.D. certified check or money orders only with a 40% deposit on orders over $300. Massachusetts residents add 5 percent sales tax. Shipping charges not included in prices.

To order call toll free 1-800-343-9206
For information call 617-491-7505

The Company cannot be liable for pictorial or typographical inaccuracies.

See List of Advertisers on page 226

COMPUTER EQUIPMENT & SOFTWARE BARGAINS



EVERY MONTH

BUY, SELL OR TRADE ALL TYPES OF COMPUTER EQUIPMENT AND SOFTWARE (pre-owned and new) among 20,000 readers nationwide.

FEATURES:

- Low classified ad rates - 10¢ a word
- Hundreds of ads from individuals
- Categorized ads so you can find them instantly
- Large (11 by 14") easy to read pages

Subscribe now for $10 and receive 13 issues/year (one FREE plus 12 regular issues). After receiving your first issue if you're not completely satisfied you may have a 100% refund and you still keep the first issue free. Bank cards accepted.

BONUS: *If you have something to advertise (pre-owned or software) send in a classified ad with your subscription and we'll run it FREE.*

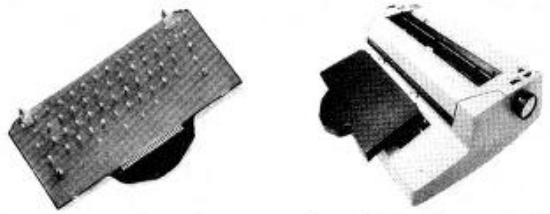
The Nationwide Marketplace for Computer Equipment

COMPUTER SHOPPER ↙212

P.O. BOX F27 • TITUSVILLE, FL 32780 • 305-269-3211

MasterCard & VISA subscriptions only, call TOLL FREE 1-800-528-6050 Ex. 184

At last --- the DYNATYPER TYPEWRITER INTERFACE!



Turn your electric typewriter into a low cost, high quality hard copy printer. 1 year warranty

DYNATYPER – Rochester Data's patented* Computer/Typewriter Interface is the industry standard for typewriter output.

- 2 minutes to initially install and 5 seconds to remove or replace.
- You do not have to modify your typewriter. All factory warranties and maintenance agreements on your typewriter will be honored.
- Compatible with all power carriage return typewriters having standard U.S. keyboard. The Dynatyper works with Selectrics (model 1) and most non-Selectrics (model 2). Please specify. Typewriter conversion between models takes 2 minutes and the kit (17 plungers) is available for a nominal fee.
- The Dynatyper is compatible with all major word processing software. (Scripsit, Pencil, Applewriter, Easywriter, Magic Window, Visi-calc CCADBM, Supertext, Write On)
- Interfaces available for TRS-80, APPLE, PET/CBM, OSI, Northstar, HP-85, H-89. Weighs only 3 lbs. Extremely portable.
- Delivery: Stock to two weeks. Price $499 for complete system. F.O.B. Rochester, Domestic. VISA and Master Card accepted. Call Ken Yanicky at 716-244-7804

*Pat. Pending

↙468

ROCHESTER DATA incorporated

3000 Winton Road South, Rochester, N.Y. 14623

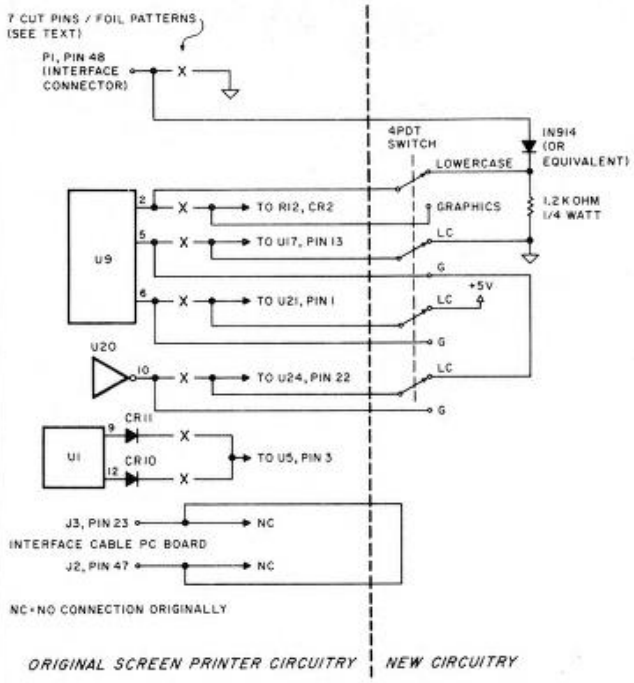


Fig. 1. Screen Printer Modifications

tions.

1.) Using an Xacto knife or small sharp screwdriver cut through the connector pin-48 circuit trace where shown in Photo 1. Using a small diagonal cutter, cut the following IC pins (approximately 1/16 inch below the pins' entrance to the IC body): U9 pins 2, 5, and 6, U20 pin 10. Use Photo 2 as a guide to locate the IC pins. Use a small screwdriver to separate the remaining pin stubs and the circuit board. Cut on end of CR10 and CR11 where shown in Photo 2.

2.) Connect the 1N914 diode and the 1.2 kohm resistor as shown in Photo 1. Now connect the four-pole switch as shown, in Fig. 1, allowing about three inches of wire beyond the front board edge to the switch. This will allow you to insert the switch in the cabinet as the chassis is reinstalled.

3.) Drill a hole in the front of the cabinet to accomodate the additional switch. Reassemble the printer.

4.) Disassemble the printer interface cable circuit board assembly by carefully prying up one end of the plastic cover with a small screwdriver. Using Photo as a guide, solder a short jumper wire from J3 pin 23 to J2 pin 47. When finished, snap the cover back over the circuit board.

TRS-80 Modifications

The TRS-80 modifications involve two circuit changes. One is the addition of a switchable storage facility (a 2102 RAM) for bit-6 data bus information. The other adds an unused gate in order to disable main RAM output to the data bus during Screen Printer access of video RAM. First, however, disassemble the keyboard assembly.

Gaining access to the TRS-80 logic board is considerably easier than the Screen Printer disassembly. Disconnect all cables and place the keyboard upside-down on a soft towel. Remove the six Phillips screws from the bottom cover. Then turn the keyboard right-side-up. Lift off the top cover. Hinge the keyboard upward, away from the logic board, to a vertical position. While holding the keyboard with one hand, remove the five rubber spacers from the plastic posts.

Cut the circuit trace connecting Z73 pins 9 and 12 shown in Photo 4. Set the keyboard down lightly onto the logic board.

Now carefully lift *both* the keyboard and logic board out of

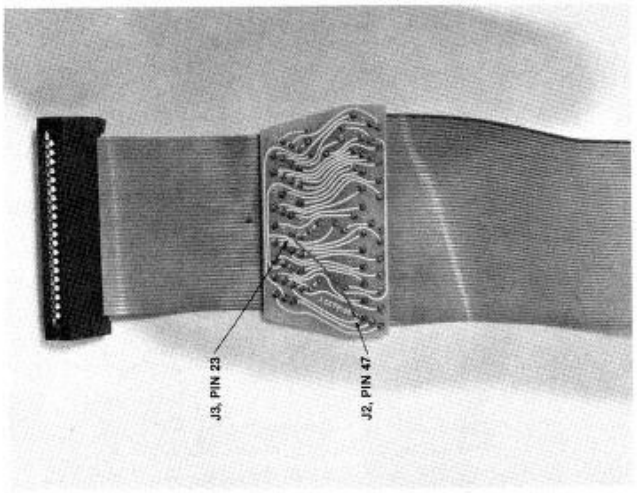


Photo 3. Printer Interface Cable PC Board. Install single jumper wire as shown.

the cabinet bottom as an assembly. Avoiding any bending of the logic board. Turn the two boards upside-down on the towel, using great care to avoid stress on the short interconnecting ribbon cable.

All of the remaining modification work involves the component side of the logic board.

It's best to use both the schematic in Fig.2 and Photo 5 to avoid confusion.

To install the lowercase mod, begin by "piggy-backing" the 2102 RAM on top of Z45. Solder all pins of the 2102 to the corresponding pins of Z45 except pins 11 and 12, which should be bent outward as shown in Photo 5.

Next, cut pin 13 of Z30 using the method described in the Screen Printer mod. Now drill a hole to mount the SPDT toggle switch where desired (I found the space between the tape and video DIN jacks convenient.) To complete the video RAM portion of the mod, connect four wires as shown from Z30, Z44, the new RAM, and the SPDT switch.

To complete the main RAM portion of the mod, cut the trace below Z74 where shown in Photo 5. (This trace was the original connection to the main RAM enable/disable buffers Z67 and Z68.) Finally, connect three wires from Z52, Z73, Z74, and the previously cut circuit trace.

Double-check all of your wiring against Photo 5. When reassembling, don't forget the five rubber spacers to isolate the logic board from the keyboard.

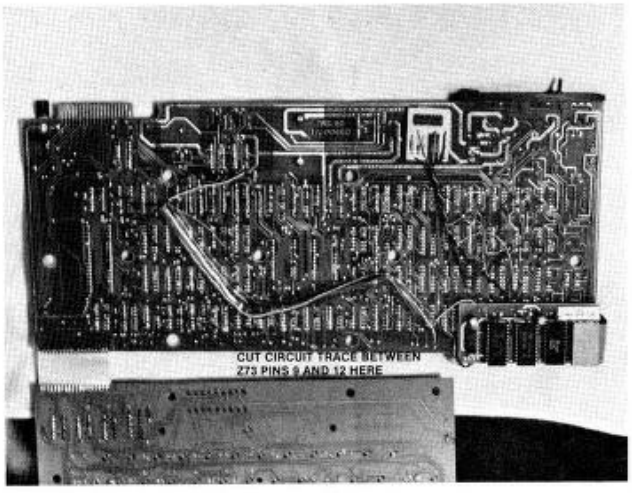


Photo 4. Underside of TRS-80 Logic Board

See List of Advertisers on page 306

TRS-80 OWNERS
BASIC SLOWING YOU DOWN?
TAKE
ZBASIC
FOR
FAST RELIEF!

Introducing SIMUTEK'S ZBASIC. The truly interactive BASIC COMPILER for your TRS-80! FINALLY! People that don't have the time or the inclination to learn complicated assembly language, have a chance to write PROFESSIONAL QUALITY SOFTWARE in machine language using a subset of LEVEL II BASIC!!

What does interactive mean? It means you have ZBASIC and your BASIC program resident at the same time! You may compile a BASIC program, run it or save it without destroying your resident BASIC program! In fact, jumping back and forth between the compiled code and the BASIC code is one of its finest features!

ZBASIC allows saving your COMPILED PROGRAM as a system tape, (tape version), or as /CMD file, (disk version). THE COMPILED CODE IS VERY EFFICIENT Z80 OBJECT CODE. THE LEVEL II ROMS ARE USED ONLY FOR I/O ROUTINES!!

FACTS ABOUT ZBASIC

1. 16K ZBASIC will compile a 4.8K program. (tape only)
   32K ZBASIC will compile a 17K (tape), 10K (disk) pgm
   48K ZBASIC will compile a 17K program. (disk only)
   (These are approximate values depending on program efficiency etc.)
2. ZBASIC DOES NOT support disk or tape files.
3. BASIC programs compiled with ZBASIC are between 10-200 times faster than interpreted BASIC!!
4. NO ROYALTIES ON ZBASIC COMPILED PROGRAMS!!
5. ZBASIC programs are only about 1.1 times larger than the average basic program.
6. ZBASIC programs may be used as USR routines from basic.
7. ZBASIC uses INTEGER MATH ONLY to increase speed and decrease compiled program size. Use of Single or Double precision would destroy the beauty of the first "INTERACTIVE COMPILER" on the market!
8. Limited variables: A-Z, A1-Z1, A2-Z2, A$-Z$. Arrays are not supported to decrease memory demands and speed up compiling of programs.
9. COMPILE TIMES ARE TYPICALLY 1 TO 10 SECONDS! THERE IS NO NEED TO USE COMPLICATED COMPILE TIME MODULES!
10. ZBASIC comes with a HIGHLY DETAILED manual describing all important memory locations, commands, variables, warm/cold start entry points and many useful sub-routines for emulating unsupported commands!!
11. Existing programs may be loaded from tape or disk and compiled as long as unsupported commands or variables are not used.

ALL COMMANDS DIRECTLY SUPPORTED BY ZBASIC

FOR	NEXT	STEP	IF	THEN	ELSE	PEEK
SET	RESET	POINT	CHR$	RANDOM	RND	POKE
DATA	READ	RESTORE	END	GOTO	GOSUB	CLS
INPUT	INKEY$	LET	STOP	OUT	INP	RETURN
PRINT	LPRINT	PRINT@	USR	SGN	INT	ABS
SQR	LEN	ASC	VAL	STR$	POS	ON GOTO
ON GOSUB	REM	NOT	AND	OR		

INTEGER MATH: *MULTIPLY /DIVIDE +ADD −SUBTRACT +/− 32767

NOTE: Some commands do not act exactly as BASIC commands act.

TRS-80 MOD I and III or PMC-80 Computers. (Level II Only) NEW LOWER PRICES!

ZBASIC 16K/32K LVII Tape Version + Manual	**$79.95**
ZBASIC 32K/48K Disk Version + Manual	**$89.95**
Both Versions + Manual	**$99.95**
ZBASIC Manual Only (Fully Refundable)	**$25.00**
Model III Version Available 3/1/81	

Credit Card or C.O.D. Call **Toll Free: (800) 528-1149**
or send check or money order to:

SIMUTEK COMPUTER PRODUCTS™ ↙19

P.O. Box 13687 Tucson, AZ 85732 (602) 323-9391
(C.O.D. Available $3.00 Extra)

TRS-80 is a TM of Tandy Corp.

TEXAS COMPUTER SYSTEMS
Radio Shack
Authorized Sales Center
All Radio Shack merchandise available at a discount. Ask for our price list.
We offer the lowest prices on ®
TRS-80
COMPUTERS
MODEL II 64K $3349 (Plus shipping)
All accessories for Model II available — disk expansions, printers and software
EPSON MX-80 $ CALL List $645
Letter quality matrix similar to the Line Printer IV and the Centronics printer but has full software control of 40, 80, 66, or 132 columns.
80 cps bidirectional printing, tractor feed, disposable printhead. $300 less than the nearest competitive printer, it's the best buy in a letter quality multi-strike printer. Our price is too low to advertise.
★ Anadex 9500/9501 ★ Line Printer V ★ The New Daisy Wheel II
★ Payment: Money Order, Cashier's Check, Certified Check. Personal Checks require 3 weeks to clear. VISA MASTERCHARGE - Add 3%
★ No tax out of state. TX add 5%
★ All items new, guaranteed by manufacturer
★ Delivery subject to availability
★ Prices subject to change at any time.
TEXAS COMPUTER SYSTEMS
An Authorized RADIO SHACK® Sales Center F701
Box 1174, Brady, Texas 76825 ✓25
TOLL FREE Number 800-351-1473
Texas Residents 915-597-0673

PACIFIC OFFICE SYSTEMS
COMPUTER INTERFACES & PERIPHERALS

- **POS-100 NRZI TAPE DRIVE CONTROLLER/FORMATTER** — Now your micro can read and write IBM/ANSI compatible NRZI format 9-track magnetic tapes. The POS-100 consists of S-100 bus card, 6' ribbon cable, tape drive controller card, cable to Pertec-Standard NRZI Tape drive, plus documentation and Z-80 or 8080 software (specify). Power is derived from tape drive and S-100 bus. Ship Wt.: 10 lbs. Suggested Retail Price $995.00
- **POS 103/202 "MIX or MATCH" MODEM** — Unique POS control design permits use in one housing of both Bell-compatible 103 (0-300 baud) and 202 (0-1200 baud) modem modules originally made by VADIC Corp. for a telephone company subsidiary. FEATURES: RS-232 serial interface, auto-answer, auto-dial, LED display, telephone line interface via acoustic coupler, manual DAA, or auto-answer DAA (sold separately). FULLY ADJUSTED; no special tools required. 3,000 mile range over standard dial-up telephone lines. Ship wt.: 15 lbs.
  PRICES . . . . POS 103 Modem — $199.95; POS 202 Modem — $299.95; POS 202 Modem w/Auto-Answer — $349.95; POS 103/202 Modem — $499.95; FCC-Approved Auto-Answer DAA — $125.00; Acoustic Coupler — $29.95.
- **POS DAISY-WHEEL PRINTER INTERFACE for TRS-80** — Will drive Diablo HyType I, HyType II, and Qume Q and Sprint 3 printers. Includes 1K user-available memory for custom print routines (such as graphics, bidirectional printing, etc.). Programmed to respond to print commands from BASIC ELECTRIC PENCIL™, and SCRIPSIT™ software. Draws its power from printer. Ship wt.: 5 lbs. Price $250.00
  Cables, each (Specify HyType I, HyType II, or Qume) . . . . . $ 25.00
- **POS ASCII INTERFACE for IBM I/O SELECTRIC** — This Centronics-style parallel printer interface will drive an IBM Model 731 or 735 I/O typewriter (EBCD and Correspondence codes). No software needed. Features on-board EPROM which holds up to 8 ASCII-to-IBM code tables for different type spheres. Closed-loop operation runs at maximum printer speed; stops and starts on a single character without loss of data. Requires +12VDC and ±5VDC power source. Ship wt.: 5 lbs. Price $249.95
  Power Supply (+5VDC, +12VDC, +24VDC for Solenoids on Printer) . . . . $ 49.95
- **CONVERT OFFICE SELECTRIC TO I/O TYPEWRITER** — Kit includes assembled solenoids, switches, wire harness, magnet driver PCB plus instructions for installation and mCPU interface. Price $150.00
- **"FORMALINER" Variable Width Forms Tractor** for 15" Selectrics . . . $95.00
- **GTE Model 560 ASCII SELECTRIC I/O Terminal** — With RS-232 Serial Interface and digital cassette deck for use as memory typewriter. Ship wt.: 100 lbs.
  Price, tested and adjusted $1,195.00
- **POS ASCII IBM SELECTRIC PRINTER** — 15" Selectric from GTE terminal cleaned and adjusted with POS Centronics-style ASCII printer interface. UC/LC, carbon and fabric ribbons. Compatible with TRS-80, Apple, SOL and other CPU parallel printer ports. Ship wt.: 75 lbs. Price $895.00

PACIFIC OFFICE SYSTEMS ✓153
918 Industrial Avenue Palo Alto, CA 94303 (415) 493-7455

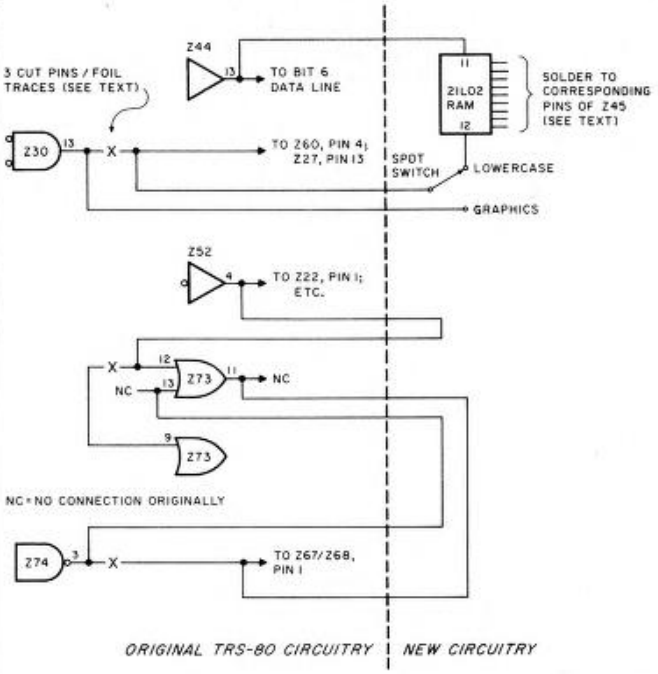


Fig. 2. TRS-80 Modifications

Operating Tips

The Screen Printer and TRS-80 modifications allow completely unmodified operation when the two new switches are off. This is probably the best way to begin after reconnecting all the cables. Thus, if normal operation is not achieved, don't go any farther. Shut off the power and find out what's wrong.

If everything is okay, load the Scripsit program, activate the printer and TRS-80 mod switches. Then depress the shift and control keys to enter Scripsit lowercase mode. Type some text into the display. Voila! You should now have an upper/lowercase display.

The Screen Printer will now function on many programs that just produced jibberish before, specifically, programs which contain the keyboard debounce routine. However, never initiate a print during program loading or execution involving anything other than "awaiting a keyboard entry." You can set or reset either of the mod switches during the "awaiting keyboard entry" state without adverse effects.■

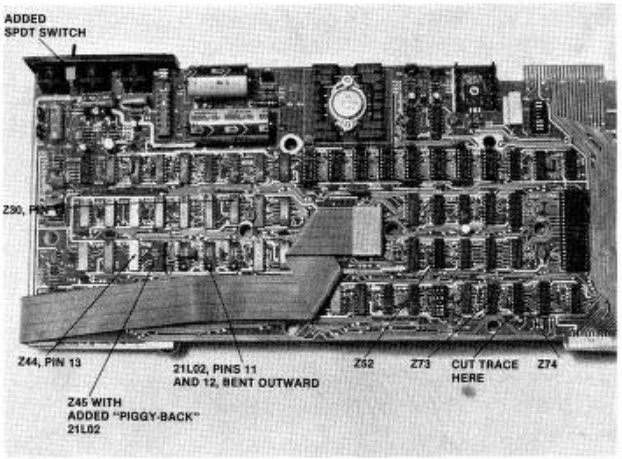


Photo 5. Top of TRS-80 Logic Board

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

EBAY.CO.UK 'BARGAINS'

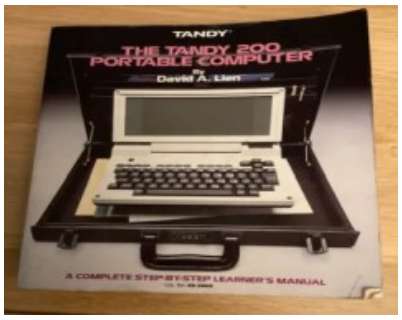
HERE ARE JUST SOME OF THE MORE INTERESTING TANDY (AND EVEN A UK101) EBAY LISTINGS, SINCE OUR LAST ISSUE.
WERE YOU ONE OF LUCKY ONES?



£
£262



£20



£2



£408



£126



£123



£102



£188



£211



£195



£137

EBAY.CO.UK 'BARGAINS'

HERE ARE JUST SOME OF THE MORE INTERESTING TANDY EBAY LISTINGS,
SINCE OUR LAST ISSUE. WERE YOU ONE OF LUCKY ONES?
HOW ABOUT LETTING US ALL KNOW HOW YOU'RE GETTING ON WITH YOUR BUY?



£
£166



£17



£155



£219



£33



£87



£37



£72

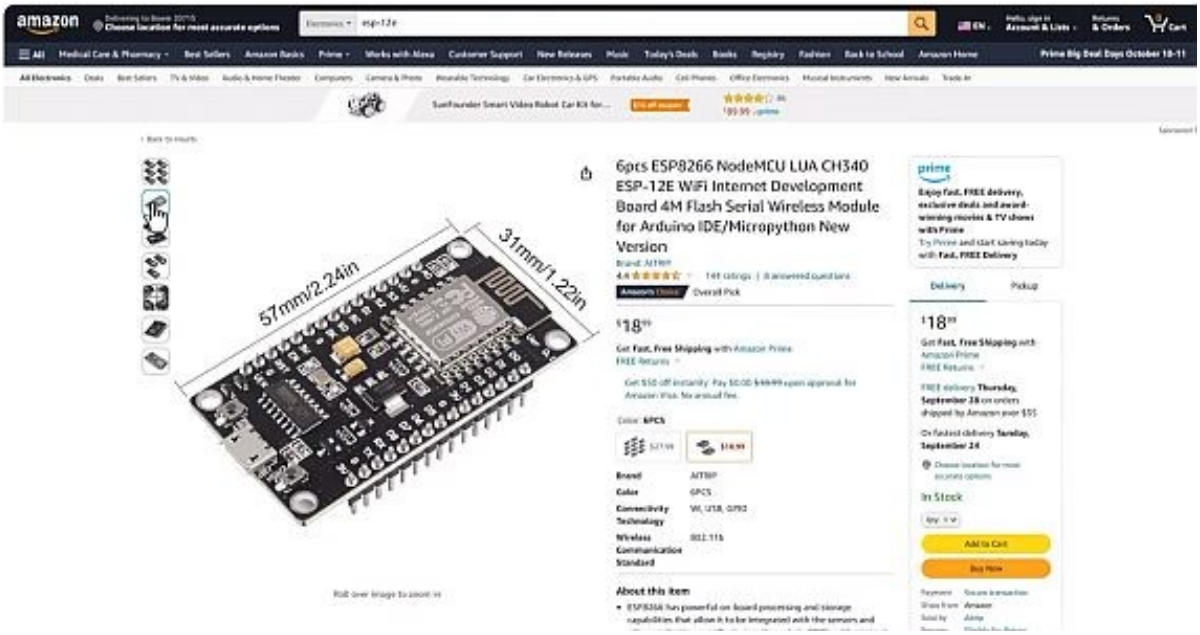


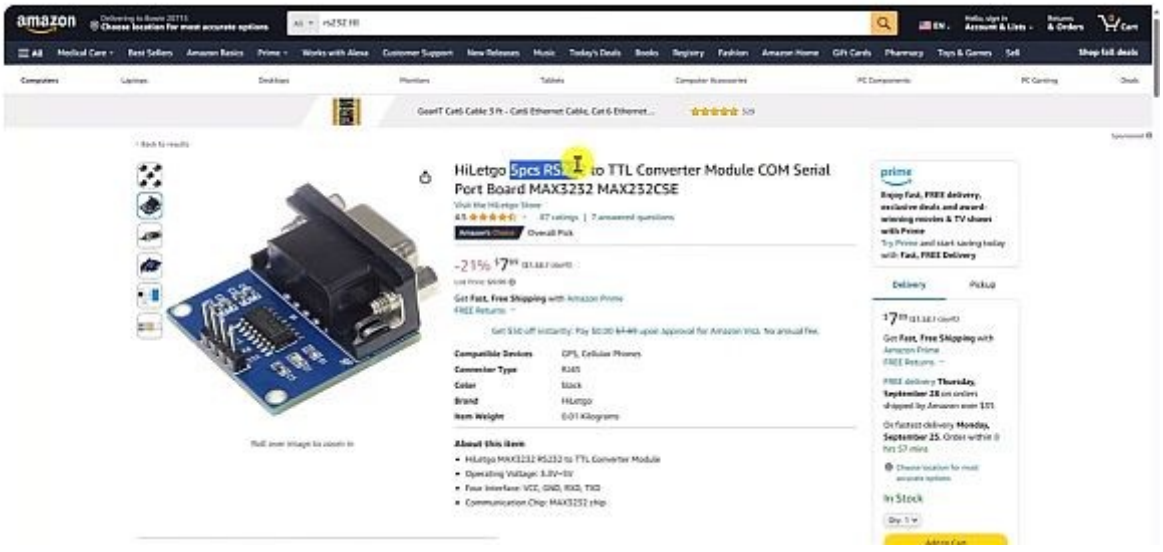
Not
sold

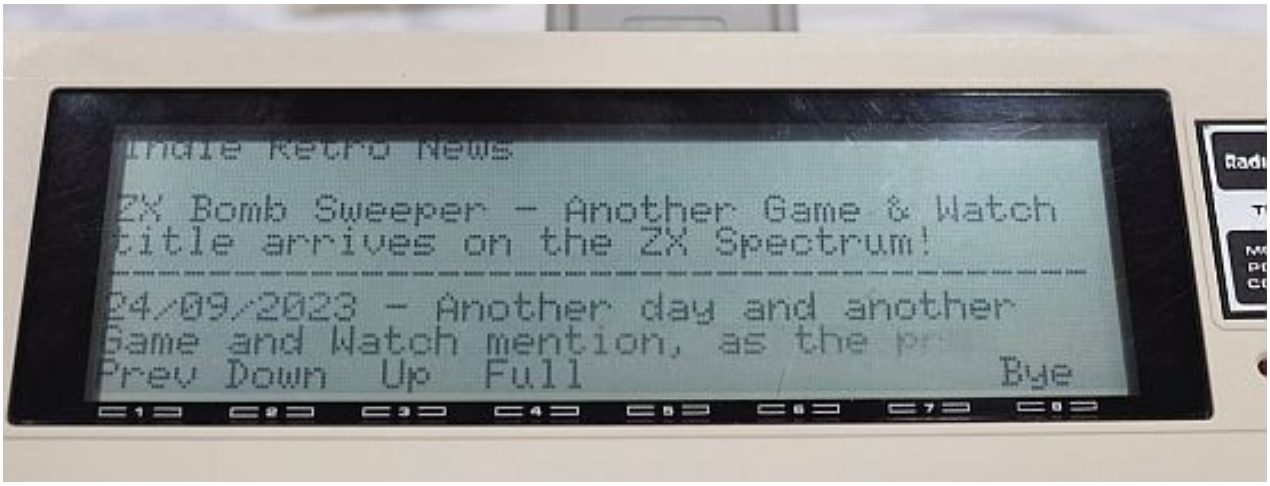


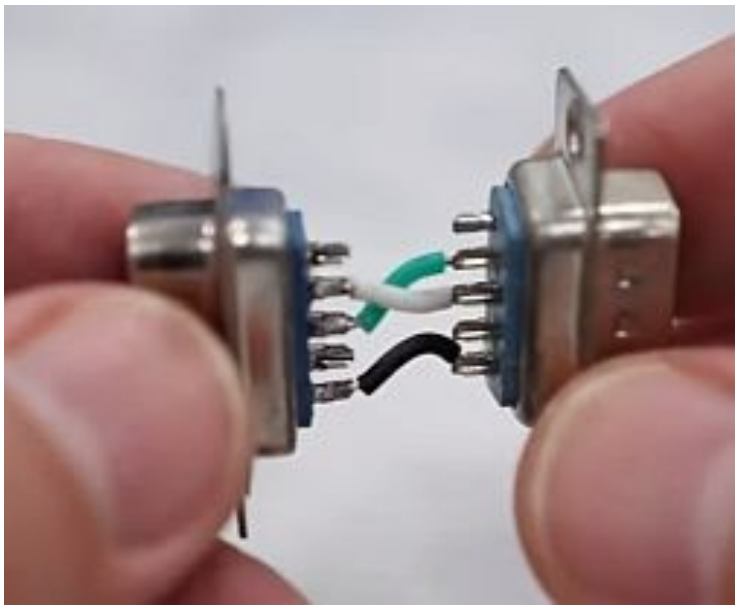
https://www.youtube.com/watch?v=ZE3nMxTLgjk

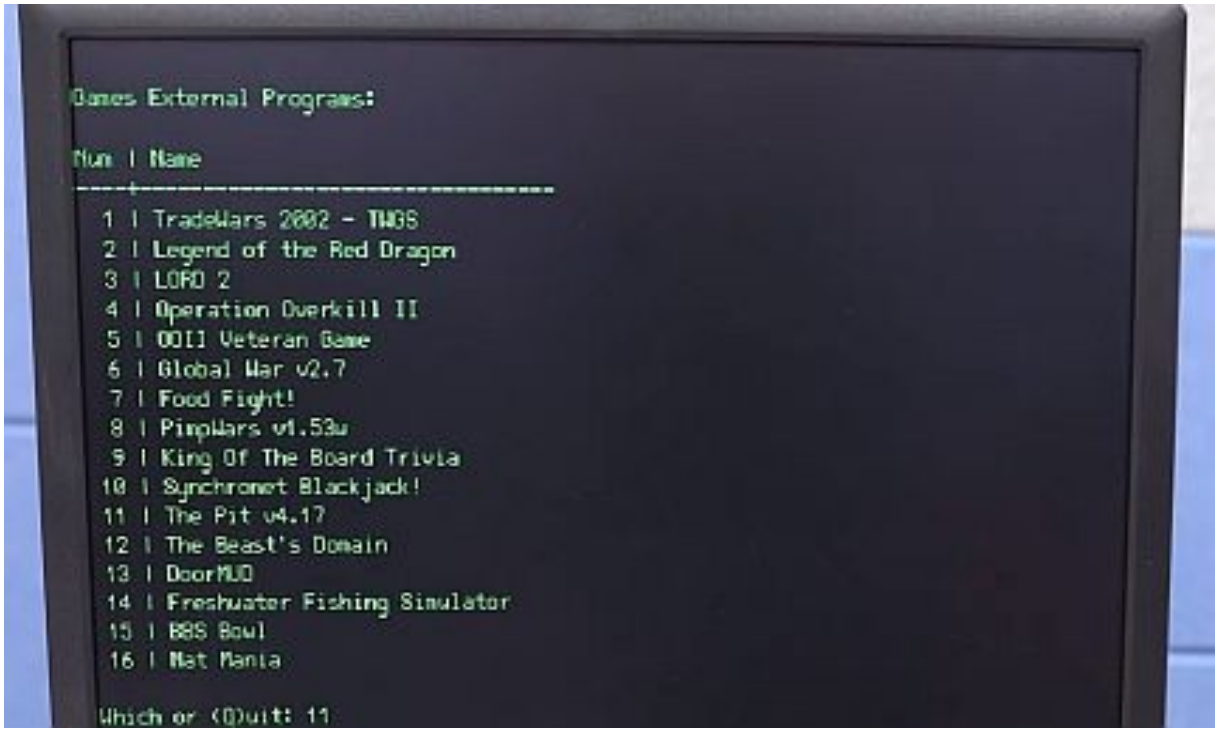
Surfin' The Net









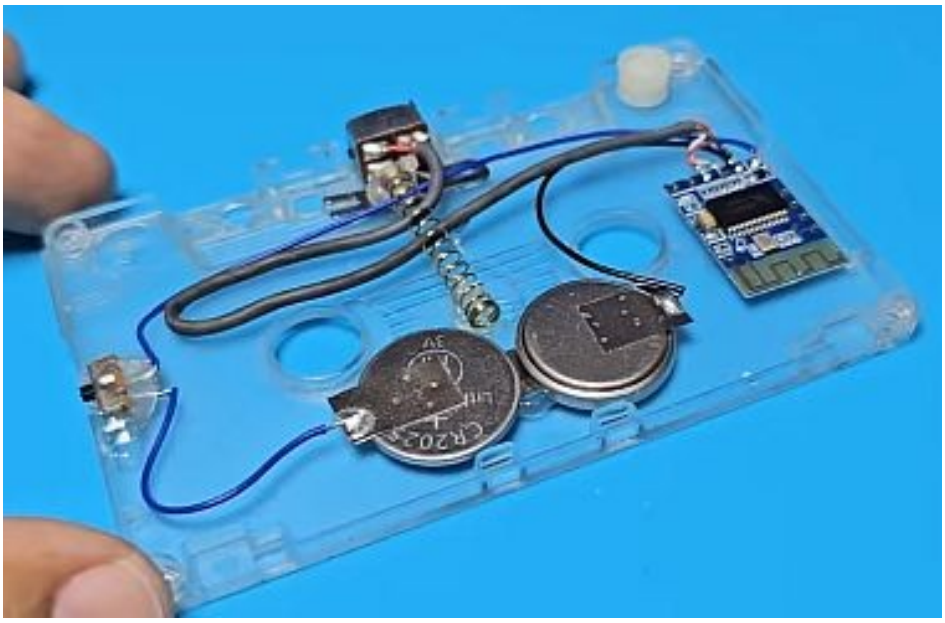


https://www.youtube.com/watch?v=tCyuzZlxT3w



https://www.youtube.com/watch?v=f0Joy7sKPa4&list=LL&index=1





IF, LIKE ME, YOU STILL HAVE A CASSETTE PLAYER IN YOUR CAR, HERE'S A NOVEL IDEA TO USE YOUR CASSETTE PLAYER TO PLAY MUSIC FROM YOUR MOBILE VIA 'BLUETOOTH'

CATSYNTH TV

https://www.youtube.com/watch?v=BJiLvXKjQX8





CONNECTING A MODEL 102 TO THE INTERNET VIA THE SERIAL PORT.



Jake Commander

Jake Commander is a notable figure in the history of early computing, particularly remembered by the Tandy community, for his contributions to the 80-Microcomputing magazine. His articles provided invaluable insights and practical advice to enthusiasts and professionals during the formative years of personal computing.

Background and Career

Jake Commander's career began at a time when personal computing was in its infancy. The late 1970s and early 1980s were marked by a burgeoning interest in microcomputers, with various models like the TRS-80 becoming household names among tech aficionados. 80-Microcomputing magazine, later known as 80 Micro, was a key publication dedicated to the TRS-80 line of computers produced by Tandy Corporation. It served as a vital resource for users, offering reviews, tutorials, programming tips, and industry news.

Commander, with his technical expertise and clear writing style, quickly became a prominent contributor to the magazine. His articles often bridged the gap between complex technical concepts and practical application, making them accessible to a broad audience. This ability to demystify technology for the average reader was one of his hallmark strengths.

Contributions to 80-Microcomputing

Throughout his tenure at both 80-Microcomputing and 80-US, Commander wrote extensively on a variety of topics. His pieces ranged from in-depth software reviews and hardware assessments to hands-on tutorials and programming guides. One of his key contributions was in the area of software development. Commander frequently provided code snippets and full-fledged programs that readers could input into their own TRS-80 systems, enhancing the functionality of their machines and expanding their computing skills.

His articles were often accompanied by detailed explanations, helping readers understand the logic behind the code.



This educational approach not only enabled users to replicate the programs but also empowered them to modify and create their own software, fostering a culture of learning and innovation.

Impact on the Computing Community

Jake Commander's work had a significant impact on the early computing community. At a time when resources were limited and

information was not as readily available as it is today, his articles served as a beacon of knowledge. Many readers credit Commander with helping them overcome technical challenges and deepen their understanding of computing.

Moreover, his contributions helped build a sense of community among TRS-80 users. 80-Microcomputing was not just a magazine but a forum where enthusiasts could share ideas, solve problems, and celebrate their shared passion for technology. Commander's approachable style and comprehensive coverage made him a trusted voice within this community.

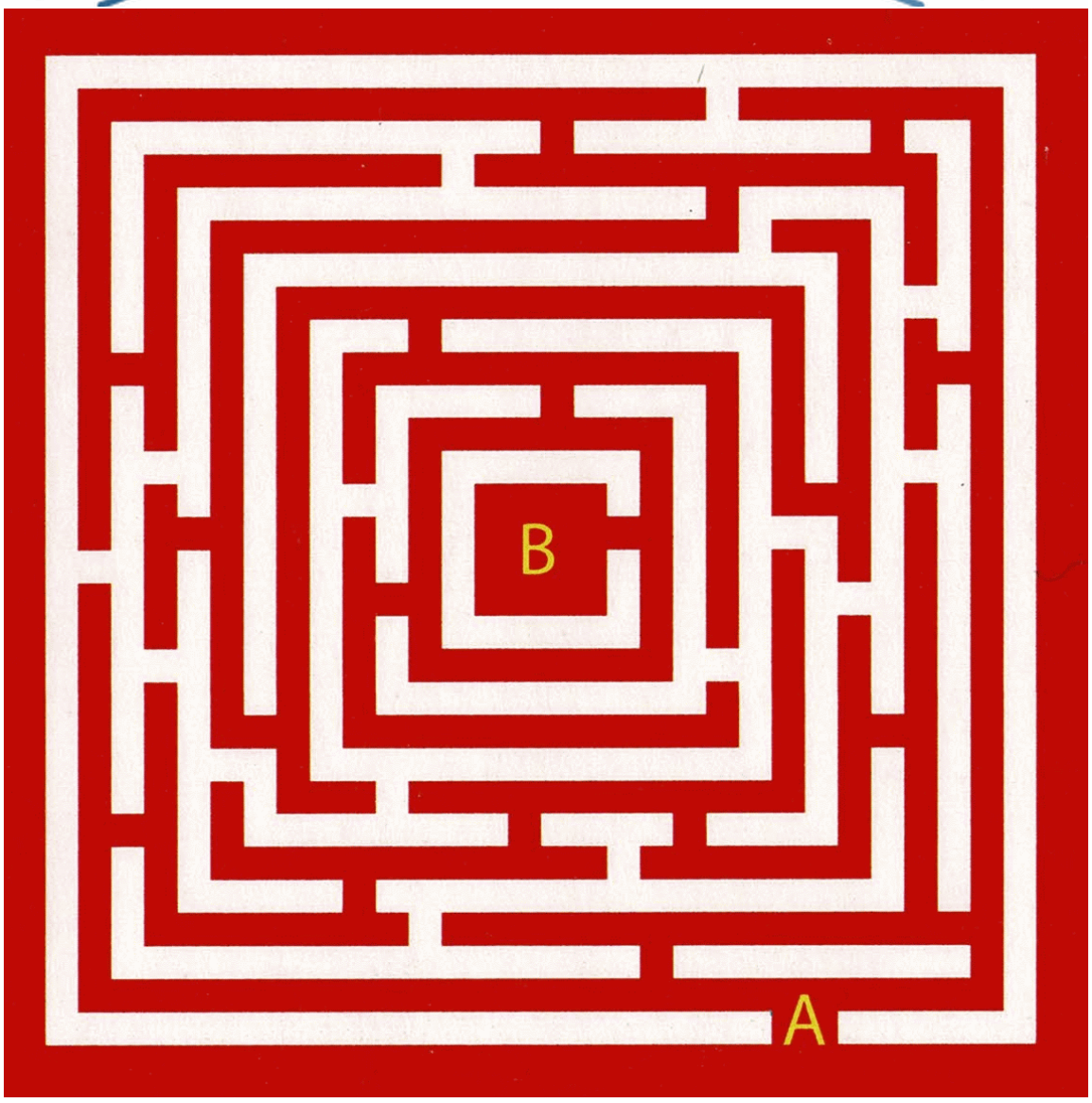


Legacy

While 80-Microcomputing eventually ceased publication as the early TRS-80 lines were phased out, the legacy of contributors like Jake Commander endures. His articles remain a testament to the early days of personal computing, capturing the excitement and curiosity of a generation of computer users. For those who experienced the rise of the microcomputer revolution, Commander's work is remembered with fondness and respect.

Today, as we look back at the pioneers of personal computing, Jake Commander stands out as a key figure who helped shape the early landscape of technology journalism. His contributions not only educated and inspired a generation of users but also laid the groundwork for the tech-savvy world we live in today.

AMAZED



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 HOW LONG IT TAKES TO GET FROM A TO B, FOLLOWING THE RED ROAD

I Love Editor Assembler Plus!
Vince Otten

There. I admit it. For over 40 years, I've had a crush on this one-off product from—of all people—Microsoft.

Back in the late 1970s, I wanted to learn assembly-language programming on my 16K Model I TRS-80, but the thought of loading Radio Shack's Editor/Assembler, editing source code, saving it to cassette, assembling the source, saving the machine code to cassette, exiting the assembler to BASIC, using SYSTEM to load the machine code, and running that ... I hoped I would be successful. If not, I'd have to get T-Bug, load that, load the machine code with that, set some breakpoints, and try running my program with that. Nah.

Why couldn't I develop assembly-language programs like I did with BASIC? Enter the source. CSAVE it. RUN it. If it bombed, take a look at some variables, or the source code, and make some fixes. RUN it again. Over and over, as often as I wanted.

It turned out that I could! Iin 1979 Microsoft brought out their enhanced version of the Editor/Assembler. It sold for about the same as getting the Radio Shack version and a copy of T-Bug. But it did so much more:

1. It assembled my source code *into memory*.
2. It had an integrated debugger, so I could test-run my program without leaving Editor Assembler Plus (EA+).
3. It not only let me set breakpoints, it let me *single-step* through my code.
4. The debugger, called Z-bug, was *symbolic*. That is, it referred to the symbol table produced by the assembler. If I had a variable named VAL1 which happened to assemble to memory locate 7000H, I didn't have to examine 7000H; I could just refer to VAL1.
5. The debugger also had a disassemble which made use of the symbol table as well.
6. I could exit the debugger *back into* the assembler and tweak my source. Then I could re-assemble and test again.
7. All this—inside one program that I only had to load via one SYSTEM command.

It also had some nice extra features. I'll take a look at them as I show some examples of EA+ in action.

```
01000 ;       "SIGN" -- RETURNS SIGN( A) IN BC
01010 ;                A       BC
01020 ;                >0       1
01030 ;                 0       0
01040 ;                <0      -1 (0FFFFH)
01050 SIGN    LD       BC,0
01060         OR       A       ; SET FLAGS ACCORDING TO A
01070         RET      Z       ; BC= 0 IF A == 0
01080         INC      BC
01090         RET      P       ; BC= 1 IF A > 0
01100         DEC      BC
01110         DEC      BC      ; BC= -1 (0FFFFH), SINCE
01120         RET              ; A MUST BE < 0
01130
01140 TEST    CALL     SIGN    ; TRY IT OUT
01150 DONE    JP       5786H   ; AND RETURN TO EA+_
```

Here, I'm developing a little subroutine that will load the BC register pair with 1 if the accumulator A is greater than zero; with 0 if it's exactly 0; and with -1 (or FFFFH) if A is less than zero. It's something like BASIC's SGN() function.




```
                                        EDTASM-PLUS/1.01   14:52 26 MAY 24  PAGE 1


                01000 ;         "SIGN" -- RETURNS SIGN( A) IN BC
                01010 ;                  A       BC
                01020 ;                  >0       1
                01030 ;                   0       0
                01040 ;                  <0      -1 (0FFFFH)
90B4 010000     01050 SIGN      LD       BC,0
90B7 B7         01060           OR       A          ; SET FLAGS ACCORDING TO A
90B8 C8         01070           RET      Z          ; BC= 0 IF A == 0
90B9 03         01080           INC      BC
90BA F0         01090           RET      P          ; BC= 1 IF A > 0
90BB 0B         01100           DEC      BC
90BC 0B         01110           DEC      BC         ; BC= -1 (0FFFFH), SINCE
90BD C9         01120           RET                 ; A MUST BE < 0
                01130
90BE CDB490     01140 TEST      CALL     SIGN       ; TRY IT OUT
90C1 C38657     01150 DONE      JP       5786H      ; AND RETURN TO EA+
                01160
0000            01170           END
00000 TOTAL ERRORS

DONE    90C1
SIGN    90B4
TEST    90BE
```

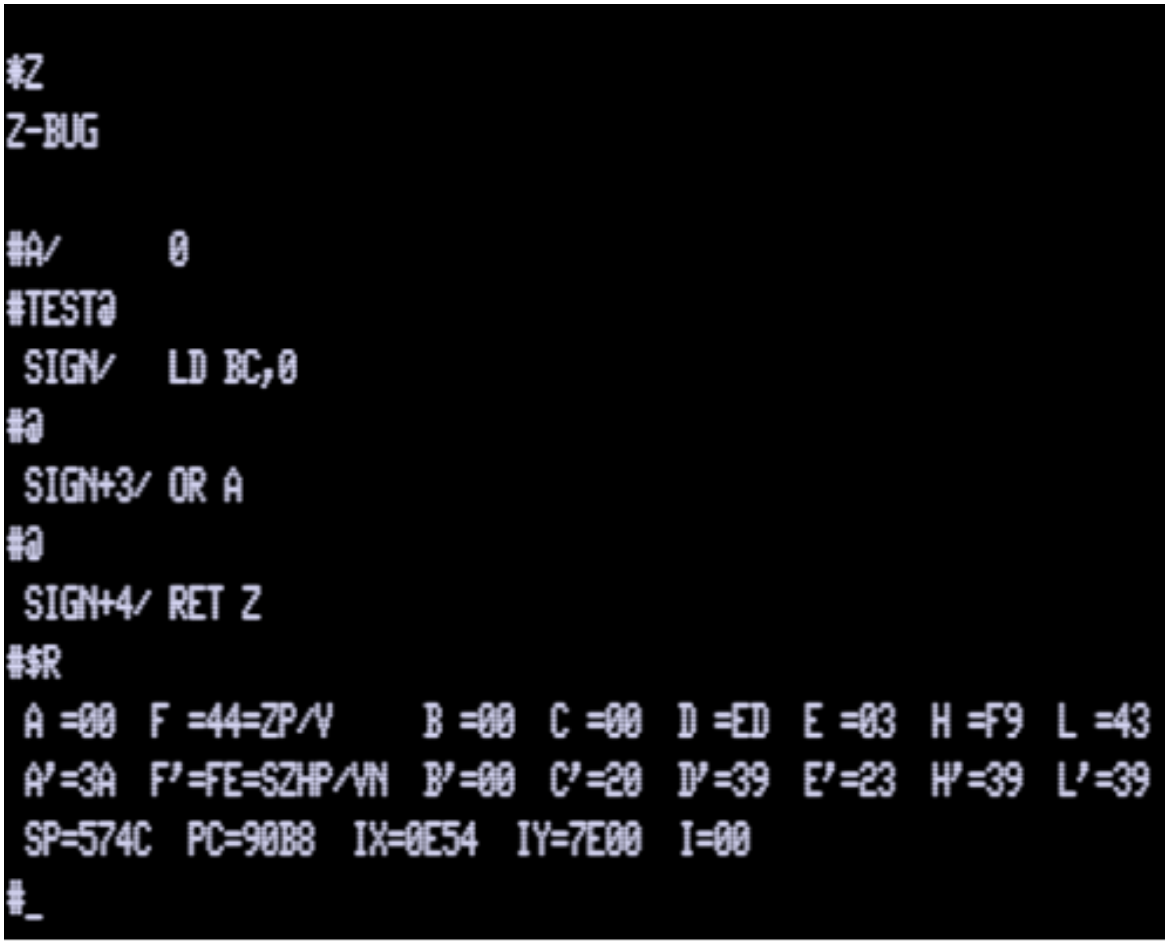
Now I Assemble the code. The command to do this is

A/IM/LP

which means "Assemble Into Memory and List to the Printer". Seasoned assembly-language programmer might have noticed that the source code has no

ORG

directive. By default, EA+ decides where in memory to place the machine code. We can see from the printout that the SIGN subroutine starts at 90B4H, and the main program label TEST is at 90BEH. EA+ allows us to ORG like any normal assembler if we want to, but typically when assembling into memory, especially to test your code, it's easiest just to let it decide where to place machine code where it wants to.
Now, we can try and test the subroutine. Lets try it first with the accumulator A containing 0.



From the Editor, we enter the

Z

command to enter Z-Bug. This debugger lets us refer to registers symbolically as well, so we examine A with

A/

and it tells us the contents: "0". If we wanted to put something else like a "3" into A, we simply type a new value and press [ENTER], like

A/ 0 3 [ENTER]

Now we can single-step through the main program "TEST". This is done by entering an address or a label and pressing the "@" key. It will then display the *next* instruction to be executed, so in this case

```
TEST@
SIGN/       LD  BC,0
```

"TEST @" executed the first line in our test program, which was

```
            CALL    SIGN
```

and now it tells us that if we press "@" again, it will execute the first instruction in the SIGN subroutine. So we go ahead and press "@", and the "LD BC,0" is executed, and Z-Bug now displays

```
SIGN+3/     OR A
```

There's no label in the source code for the line containing "OR A", so Z-Bug just "labels" it as "SIGN+3". We press "@" one more time, and "OR A" is executed. Let's take a look at the registers with

$R

(This isn't actually the result of pressing [$][R]. The "$" here is what Z-Bug displays when we type the TRS-80 version of the Escape key, which is [Shift][UpArrow], so we actually typed

[Shift][UpArrow][R]

to display the Registers. A is still 0. The flags are now set, and are displayed as

Z =44=ZP/V

so we know that the Z flag is set, and the next instruction

RET Z

will indeed RETurn to the calling program, TEST.

Here's another example Z-Bug run. In this case, I've set A to -1 (or FFH), and single-stepped like above, but up to SIGN+4. We join our session, already in progress:

```
#@
 SIGN+5/ INC BC
#@
 SIGN+6/ RET P
#F/      0AC=SP/V
#@
 SIGN+7/ DEC BC
#@
 SIGN+8/ DEC BC
#@
 SIGN+9/ RET
#$R
 A =FF  F =AC=SP/V     B =FF  C =FF  D =ED  E =03  H =F9  L =43
 A'=3A  F'=FE=SZHP/VN  B'=00  C'=20  D'=39  E'=23  H'=39  L'=39
 SP=574A  PC=90BD  IX=0E54  IY=7E00  I=00
#_
```

This time the SIGN subroutine won't "RET Z" because the Z flag isn't set, so BC gets incremented to 1 in anticipation of the possibility that A, a non-zero value, might be positive. If it is, the P flag will be set, and we'll "RET P" (at line "SIGN+6"). But no: we set A to a negative value. So if A wasn't zero, and it wasn't positive, it must be negative, and we should set BC to -1 (or FFFFH). Two "DEC BC"s should do the trick. Just before we "RET" at address "SIGN+9", let's take a look at our registers to confirm that everything is as expected.

F =AC=SP/V

Indicates that the "S"ign flag was set—A contains a value whose 8th bit is set, indicating a "negative sign".

B =FF C =FF

so BC contains FFFFH, or -1: just what we wanted.
If at this point we wanted to go back to the Editor, we simply type [Shift][UpArrow][E], and we're returned to enter or edit more source code.

A subject of interest for me is making as much use of ROM calls as possible. I use EA+ to develop and test these calls. Once I know my code is reliable, I'd like to be able to make it into a statement-like package that I can use without having to remember to "load HL with the address of your variable, and BC with the number of times you want to print it out" or something like that. I wish I could write something like

```
PRINT  VAR1,5        ; PRINT OUT "VAR1" 5X
```

instead of

```
LD     HL,VAR1
LD     BC,5
CALL   PRINTX        ; PRINT OUT "VAR1" 5X
```

The macro capabilities of EA+ allow me to do just that. Here's an example:





```
                                        EDTASM-PLUS/1.01   09:12:27 MAY 24  PAGE 1


                01000 CLS       MACRO
                01010           CALL    01C9H           ; ROM CLS ROUTINE
                01020           LD      A,0FH           ; "CURSOR OFF" CODE
                01030           CALL    033AH           ; PRINT 1 CHAR IN A
                01040           ENDM
                01050
                01060 ;
                01070 ;         VPRINS -- PRINT 0-TERMINATED STRING POINTED
                01080 ;                   TO BY HL TO SCREEN
924B 7E         01090 VPRINS    LD      A,(HL)          ; WHILE *HL
924C B7         01100           OR      A
924D C8         01110           RET     Z               ; (REACHED 0-TERM)
924E E5         01120           PUSH    HL              ; SAVE STRING PTR
924F CD3A03     01130           CALL    033AH           ; PRINT 1 CHAR IN A
9252 E1         01140           POP     HL
9253 23         01150           INC     HL              ; INCREMENT STRING PTR
9254 C34B92     01160           JP      VPRINS          ; :WEND
                01170
                01180 VPRINT    MACRO   #STRING
                01190           ; PRINT 0-TERMINATED #STRING TO SCREEN
                01200           ; IE. BASIC'S "PRINT STRING$;"
                01210           LD      HL,#STRING
                01220           CALL    VPRINS
                01230           ENDM
                01240
9257            05000 START     CLS
9257 CDC901                     CALL    01C9H           ; ROM CLS ROUTINE
925A 3E0F                       LD      A,0FH           ; "CURSOR OFF" CODE
925C CD3A03                     CALL    033AH           ; PRINT 1 CHAR IN A
                                ENDM
925F            05010           VPRINT  HELLO           ; PRINT HELLO$;
                                ; PRINT 0-TERMINATED HELLO TO SCREEN
                                ; IE. BASIC'S "PRINT STRING$;"
925F 216E92                     LD      HL,HELLO
9262 CD4B92                     CALL    VPRINS
                                ENDM
9265            05020           VPRINT  NL              ; PRINT
                                ; PRINT 0-TERMINATED NL TO SCREEN
                                ; IE. BASIC'S "PRINT STRING$;"
9265 217C92                     LD      HL,NL
9268 CD4B92                     CALL    VPRINS
                                ENDM
926B C38657     05030 DONE      JP      5786H
                05040
                05060
926E 48         10000 HELLO     DEFM    'HELLO, WORLD!'
926F 45
9270 4C
9271 4C
```

The first macro, "CLS", is just a wrapper for a ROM call. But the second, "VPRINT", allows me to use "arguments"—not disagreements, but different values every time I invoke "VPRINT".
In my main program "HELLO" I first invoke CLS, and then use VPRINT twice: first, to print the "HELLO WORLD" message, and then to print a newline. It makes my source code a bit more readable, and certainly easier to write.
The printout shows the "expanded" macro as CLS and VPRINT are used in HELLO. You can see how the VPRINT macro replaces

```
        LD      HL,#STRING
```

with

```
        LD      HL,HELLO
        CALL    VPRINS
```

the first time it's invoked by

```
        VPRINT  HELLO
```

and then replaces #STRING with

```
        LD      HL,NL
        CALL    VPRINS
```

the second time when we write

```
        VPRINT  NL
```

Again, seasoned assembly programmers won't be very impressed with EA+'s limited macros, because one macro can't make use of another. But they're perfect for my use: I just want to do some incremental developing and testing, similar to Forth.

Another useful feature of EA+ is "include" files. You can put a bunch of source code in a file and then just tell EA+ to "read the contents" of that file along with your main code. As I've been developing and testing my routines and macros, I've placed them in files like MATH/INC, STRINGS/INC, and IO/INC for use as needed while I do more programming. Here's an example:

```
00100      ; TSTACC2U/ASM -- TEST S.P. ACC TO UNSIGNED INT
00110
00120 ;            MAY 16, 2024
00130 ;            TESTED 5/16/24
00140 ;
00180 ;
00190 ;            * PROLOGUE *
00200
00205              INCLUDE ATTRIB/INC
00210              INCLUDE ENVIRON/H
00220              INCLUDE TRS80ROM/INC
00230              INCLUDE IO/INC
00240              INCLUDE MATH/INC
00250              INCLUDE STRINGS/INC
00270
00280 *MLIST  OFF
00290
00300              TITLE   'TSTACC2U/ASM'
00970 ;
00980 ;            ** START PROGRAM **
00990
02000 START        INIT
02010               VPRINT  TITLE          ; PRINT TITLE$;
02020              VPRINT  NL             ; PRINT
02025
02040
02100              VPRINT  PROMPT         ; PRINT PROMPT$;
02120              LINPUT  NUMSTR         ; LINE INPUT NUMSTR$
02130
02140              USES     MATH
02150
02160              A2ACC   NUMSTR         ; CONVERT NUMSTR TO NUMERIC
02165              CALL     0AB1H         ; CONVERT TO S.P.
02170              SP2UI    WORD1         ; WORD1= UINT( NUMSTR$)
02180
48970 ;
48980 ;            ** EXIT PROGRAM **
48990
49000              VPRINT  CURON          ; TURN ON CURSOR
49010                                     ; BECAUSE LINPUT TURNS OFF
49015
49020              EXIT
49032
49990 ;
49992 ;            ** DATA DIVISION **
```

```
49994 ;
49996 ;          * USER DATA *
49998 ;
50000            STRLIT   TITLE,'TEST UINT 2 ACC'
50010            STRLIT   PROMPT,'ENTER S.P. INT '
50090
50100 NUMSTR     DEFS     16
50110
50500 WORD1      DEFW     1           ; LOW 16 BITS OF 32
50510 WORD2      DEFW     0           ; HI 16 BITS OF 32
50520
59070 ;
59080 ;          * SUPPORT DATA *
59090 ;
59100 NL         DEFB     0DH         ; NEWLINE
59110            DEFB     0
59120 CURON      DEFB     0EH
59130            DEFB     0
65520            END
```

We INCLUDE a number of files I've previously developed:

ATTRIB/INC	"Attributions"—where I copied some of my code from
EN VIRON/H	Macros like INIT and EXIT
TRS80ROM/INC	Setup routines, particularly for MATH operations
IO/INC	printing to the screen, inputting strings, etc.
MATH/INC	All my math routines and macros
STRINGS/INC	String copying, appending, etc.

Except for ATTRIB/INC, these files turn off listing, so their contents don't clutter up the assembler's listing output. Similarly, the main program turns off "macro expansion listing' with the directive

```
*MLIST     OFF
```

again to reduce clutter. Here's a test run:



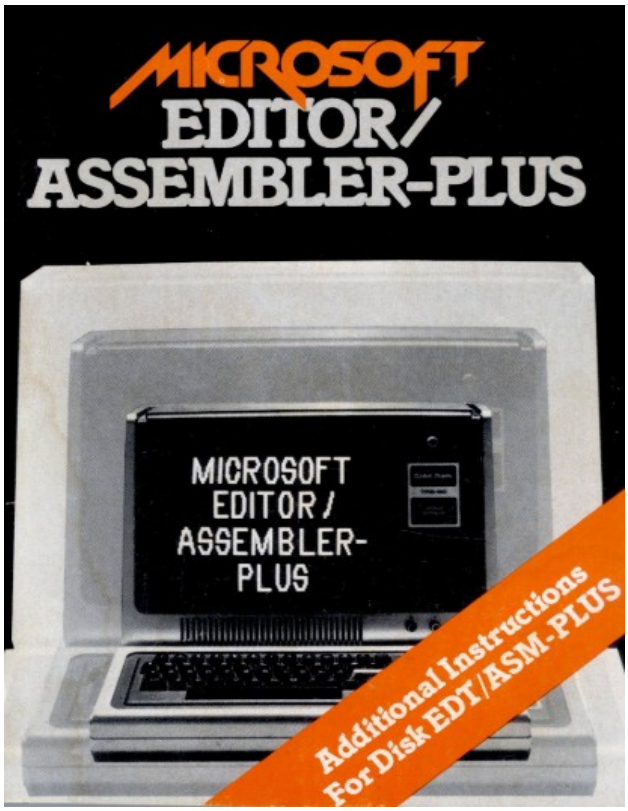

```
'TSTACC2U/ASM'                    EDTASM-PLUS/1.01   09:23 27 MAY 24  PAGE 1

00100          ; TSTACC2U/ASM -- TEST S.P. ACC TO UNSIGNED INT
00110          ;
00120          ;               MAY 16, 2024
00130          ;               TESTED 5/16/24
00140          ;
00180          ;
00190          ;       * PROLOGUE *
00200          ;
00205          ;       INCLUDE ATTRIB/INC
00090          ;*LIST  OFF
00098          ;
00100          ;       ATTRIB/INC
00110          ;               MAY 17, 2024
00130          ;
00300          ;       TITLE   'ATTRIBUTIONS'
00310          ;
01000          ;               CODE ADAPTED FROM SAMPLE CODE IN
01010          ;
01020          ; 'GP'          GEORGE PHILLIPS
01030          ;               "ADVANCED Z-80 PROGRAMMING"
01040          ;                       AND OTHER TIPS & HINTS FROM
01050          ;                       SOCIAL MEDIA
01060          ;
01070          ; 'IGRA"        "ROM ADDRESSES" IN IRA GOLD'S
01080          ;               EXCELLENT TRS-80.COM WEBSITE
01090          ;
01100          ; 'MBD'         "MICROSOFT BASIC DECODED" BY
01110          ;               JAMES FARVOUR
01120          ;
01130          ; 'T8AL"        "TRS-80 ASSEMBLY LANGUAGE MADE SIMPLE" BY
01140          ;               EARLES MCCAUL, SAMS 1981
01150          ;
01160          ; 'TRRD"        "TRS-80 ROM ROUTINES DOCUMENTED" BY
01170          ;               JACK DECKER, THE ALTERNATE SOURCE 1984
01180          ;
01200          ; 'WBTLS"       "TRS-80 ASSEMBLY LANGUAGE SUBROUTINES" BY
01210          ;               WILLIAM BARDEN, JR. PRENTICE-HALL 1982
01220          ;
65520 *LIST    ON
65522          ;
00210          ;       INCLUDE ENVIRON/H
00090 *LIST    OFF
65520 *LIST    ON
65522          ;
00220          ;       INCLUDE TRS80ROM/INC
00050 *LIST    OFF
65522 *LIST    ON
65524          ;
00230          ;       INCLUDE IO/INC
```

This program tests my routine to convert a BASIC-style Single Precision number into a 23-bit Unsigned Integer. You may recall that the BASIC ROM's "Integer" is *signed,* and can only have values from -32768 to 32767. I wanted to be able to manipulate and print out *unsigned* values like 65535 or FFFFH, typical 16-bit numbers used for addresses, etc. Here we can test my conversion routine "SP2UI", "Single Precision To Unsigned Integer". I use a number of macros I've previously developed:

INIT	Initializes the stack, etc.
VPRINT	prints a string to the screen
LINPUT	"LINE INPUT" a string and store it in a variable
USES	initializes Math vectors and code segments
A2ACC	"Alpha To Accumulator" converts a numeric string to a Single Precision number in BASIC's 'Accumulator 1'
SP2UI	the "Single Precision To Unsigned Integer" macro we're testing
EXIT	restores normal DOS Exit vectors, etc.
STRLIT	"String Literal" macro creates a short 0-terminated named string

The version of EA+ I'm using on the trs80gp emulator is disk-based. It's not especially fast, but there's something retro and satisfying about hearing the disk drives whirr and click as it assembles my code. Thanks, George and Peter, for the sound effects!
I expect that, once I get all my macros in order, I'll switch to something like "zmac" for "production code". But maybe not. After all, at least one of the main game programmers back in the 1980s used Microsoft Editor Assembler Plus to write some enduring code!



IN MAV'S WORKSHOP

Nip Mostek Madness In The Bud on the TRS-80 Model II (Here we go again...)



by Ian Mavric

Back in the December 2023 edition of TRS8Bit I talked about the Model II which I had recently obtained. In that article I explained how to disable the Rifa mains filter capacitors so you can examine and test-run your machine without them letting out the magic smoke. (Always replace them at your earliest convenience, but it's OK to run the machine for a little while without mains filter caps.)

After an initially promising start I noticed the machine getting less and less reliable, namely in the memory department with the amusingly cryptic "BOOT ERROR MF". :-)

MOSTEK CORPORATION SEMICONDUCTOR: BEST TO AVOID

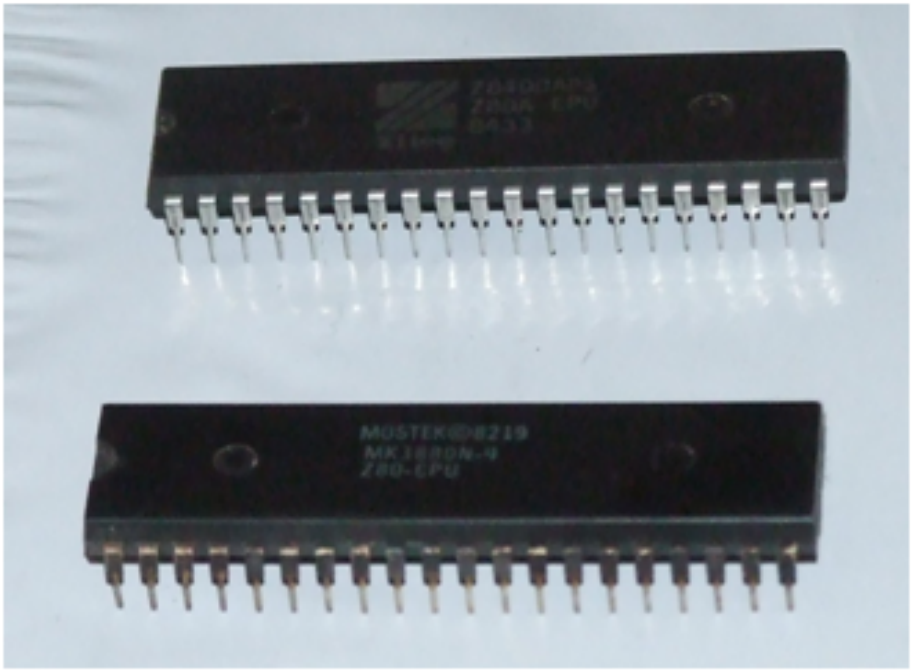


Photo: Mostek and Zilog Z80 CPUs, they are the same age but Mostek has oxidised legs.

After the smile on your face recedes and thoughts of Samuel L. Jackson in Pulp Fiction voice stating "BOOT ERROR MF" gives way to the realisation that MF in this instance stands for Memory Fault, you can narrow the diagnosing to the 64K memory card in the Model II. For those not familiar, the 64K card has 4 rows of 8 type-4116 16K DRAM chips. That 32 individual chips for those who are slow at math. The usual process is to remove each chip and put them in a DRAM tester, and find the bad chip(s). A time consuming process even if you have all the best testing equipment.

Then I noticed something concerning, and odd at the same time. A visual inspection of the DRAM chips showed them all having black oxidation occurring on chip legs. I looked for the brand of the chips, and they are Mostek. Eye-roll time. I hav seen this before, and perhaps you have too. On the Model III microcomputer a lot of them came with the Mostek version of the Z80 CPU (called MK3880), and they regularly had black oxidised chips. In fact these days when I get a Model III which has a Mostek CPU in it I replace it with a Zilog or NEC chip even if the Mostek is working properly.

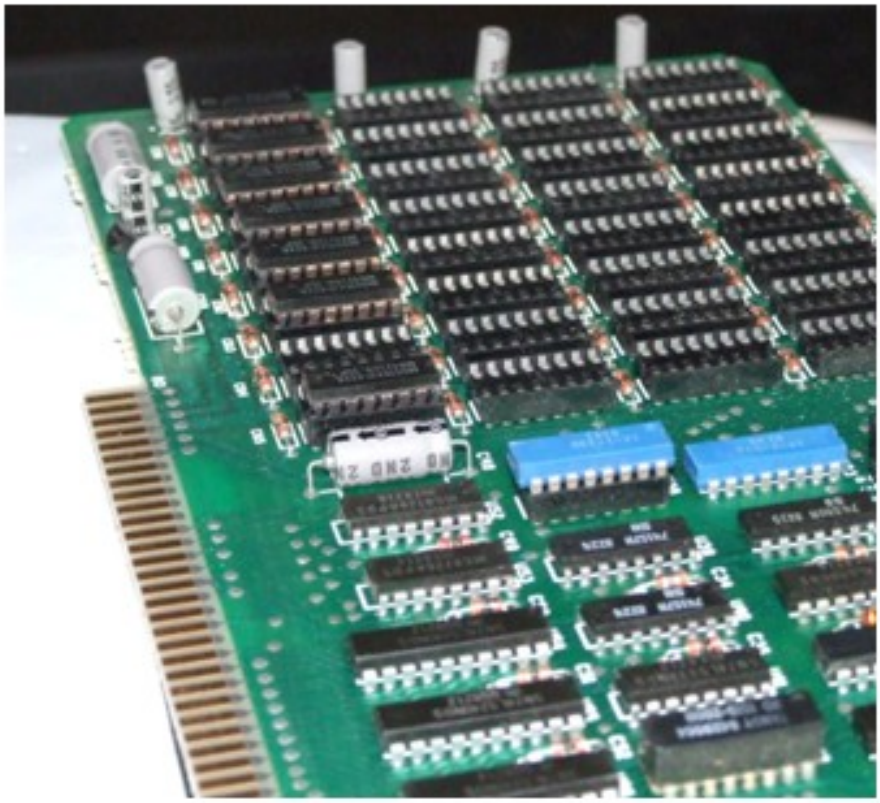


Photo: Oxidised DRAM chips, compare the legs of the chips in the foreground to the Ram chips in the background.

I want to say Mostek was a terrible company which made terrible chips but my limited research online (thanks to Google) seems to imply that Mostek started making semiconductors in 1969 and did a pretty good job at it, was taken over by United Technology in 1979, then later went onto being owned by Thomson-CSF and the current owners of Thomson are STMicroelectronics. All these names are well regarded in the field of semiconductors, especially Thomson and ST. So I can't point the finger at Mostek and accuse them of manufacturing junk, but I will say that whenever I find a chip with oxidised legs on it, 100% of the time it is a Mostek. That is all I will say about it.

BETTER CHECK THE REST OF THE COMPUTER

Since the Model III was open and I had already pulled out the 64K Ram card it seemed like a good idea to inspect the rest of the cards in the system to see exactly what I had. I found a Rev C. CPU board, a 2nd generation floppy disk controller board, and the regular video/keyboard board all machines from the Model II to the Tandy 6000 came with. Upon closer inspection I saw that every LSI (large scale integration - chips of 24 pins or more in this case) chip was made by Mostek and every leg had oxidation. A disaster just waiting to happen. I was even surprised the computer could run well enough to tell me the cursed BOOT ERROR MF.

It seems to me that sometime during late-1980 into mid-1981 perhaps the suppliers of chips normally used to make TRS-80s were in short supply and Radio Shack cut a deal with Mostek. I suggest this because it wasn't very long before you no longer saw *any* Mostek chips being used in TRS80s.

My Model II must have been one of the "lucky" ones where practically every chip happened to be by Mostek. The CPU, PIO, SIO, CTC, RAM all had oxidised legs. I am lucky in that I had on hand Zilog versions of all those chips, and a spare 64K card from another computer which was fully populated with 32 Motorola DRAM chips. I tested those DRAMs anyway and all of them passed on the chip tester so I re-installed the 4 cards after getting rid of all the Mostek chips and the computer started with the friendly INSERT DISKETTE prompt.

GO CHECK YOUR MODEL II NOW

If your Model II (and heaven knows, this might also apply to the Model 16 but I don't have any here to check but they are similar computers and some production overlap might have happened) has been running ok but sometimes exhibits undesirable traits like crashing, I would check for Mostek chips. If you are scratching your head with errors like BOOT ERROR MF not going away after changing some memory DRAM chips, change them all...

CONCLUSION

The Model II was a very reliable system when new, many ran 8-to-18 hours per day in offices, for years and years with hardly showing any problems. Now these machines are over 40 years old problems with chips may show up, and it's good to know what to look for. Like the Rifa mains filter caps covered in the previous article, they may not have given problemS back in the day, but after all this time are apt to cause chaos now.

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.



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

http://members.iinet.net.au/~ianmav/trs80/





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

WHERE 8 BITS AND 16K STILL RULE OK

TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

SEPTEMBER 2024 - VOLUME 18 ISSUE 3

PAGE 02 TRS8BIT 2024 COMPETITION
BEAT "DEFENCE COMMAND"
HAVE YOU SUBMITTED YOUR ENTRY?

PAGE 04 TA 2024 ROUNDUP
IAN MAVRIC

PAGE 13 TRS-80 BASIC
WHAT ARE FLOATING POINT NUMBERS VALUE RANGE?
JIM ADAMS

PAGE 22 NEWS FROM FRANCE
PASCAL HOLDRY
PASCAL REPORTS THE PROGRESS AS HE HAS BEEN WORKING ON 68000 BOARDS

PAGE 31 IN MAV'S WORKSHOP
IAN MAVRIC
TYPE-11 HDA BOARDS

PAGE 42 'AT THE READY' PROMPT
ODD OR EVEN, DON'T FORGET ZERO!

PAGE 44 CO-CO RANDOMISING TUNE UP
GEORGE PHILLIPS

PAGE 50 THE WAY WE WERE
A LOOK BACK AT THINGS 'FROM THE DAY'

PAGE 53 EBAY U.K. BARGAINS
WERE YOU ONE OF THE LUCKY ONES?

PAGE 55 'WHERE ARE THEY NOW?'
JAMES GARON

PAGE 57 AMAZED
JUST FOR FUN

PAGE 59 CHECK DIGITS
E.T. FONEHUME
BASIC ONE-LINERS FOR INDUSTRY STANDARDS

PAGE 64 SURFIN' THE NET
SOME SUPER IDEAS I RECENTLY CAME ACROSS

PAGE 70 MY FREHD UPDATE STORY (PART ONE)
MARK CASTELEIJN

PAGE 78 FREHD AND TRSVID (PART TWO)
MARK CASTELELJN

PAGE 84 IN MAV'S WORKSHOP
IAN MAVRIC
TANDY 1200'S

PAGE 92 ROM CALLS FROM TRSDOS 2.3 WITH ERROR TRAPPING
VINCE OTTEN

TRS8BIT 2024 Competition

Beat “Defense Command”

The objective is to write a program that can play the best game of Big Five Software's “Defense Command”. The program with the highest average score over a series of attempts will win. In the case of ties (e.g., programs can play indefinitely or are not obviously superior to each other) the earlier entrant will be declared the winner.

The programs themselves will be run externally to the emulator running the game. The positions of all game objects and other status information will be sent to an entry program which will respond with updated controller inputs to move the player's ship. The setup will simplify testing and allow competitors to write their program in any language including BASIC.

There are two categories and hence two prizes to be won. The limited category is for programs that run on a TRS-80 itself. It can be any TRS-80 emulated by trs80gp – Model 1, 2, 3, 4, 16, Colour Computer, etc. The unlimited category allows any program you like as long as it can be tested by the competition organizers. Ideally some form of Windows executable or as a service accessible “in the cloud”.

The winners will have their pick of any one of Ian Mavric's fabulous products for the TRS-80 line.

Do not over fit; don't try and predict the game's random number generator as the judging test runs will use different random sequences. Entries must be deterministic – always giving the same response if the input is the same.

Competitors are encouraged to get in touch early in order to receive timely updates on leader board changes, rule clarification and supplementary materials. You are welcome to submit improvements to your entry multiple times.

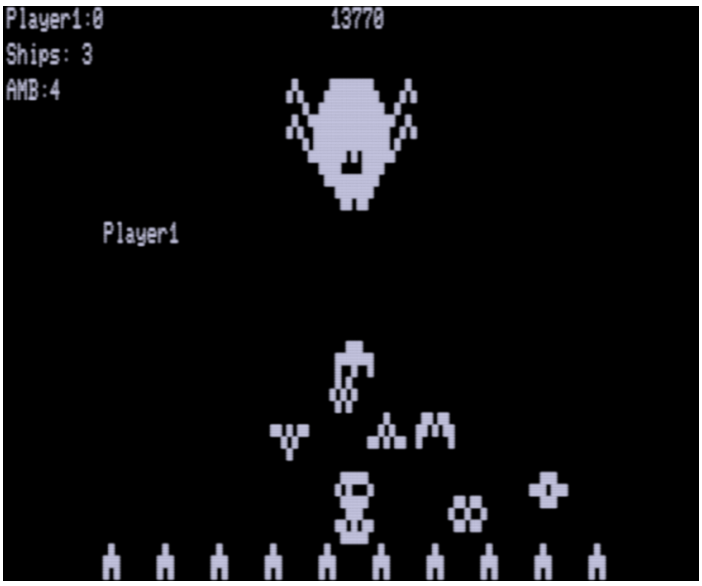
For details on how to talk to the game and a running leader board please consult:

http://48k.ca/comp2024.html

All entries will be judged by George Phillips and
should be sent to george@48k.ca

Competition closing date is 15:00 hrs GMT, 2 November 2023.





LATEST UPGRADE AVAILABLE SHORTLY
VERSION 2.5.4

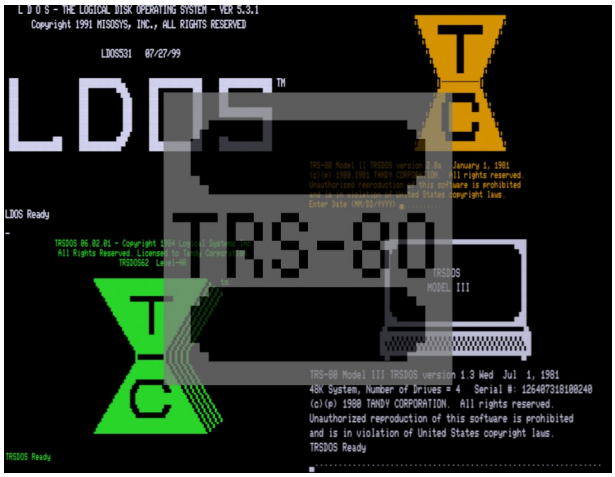
DON'T HAVE A TRS-80?

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

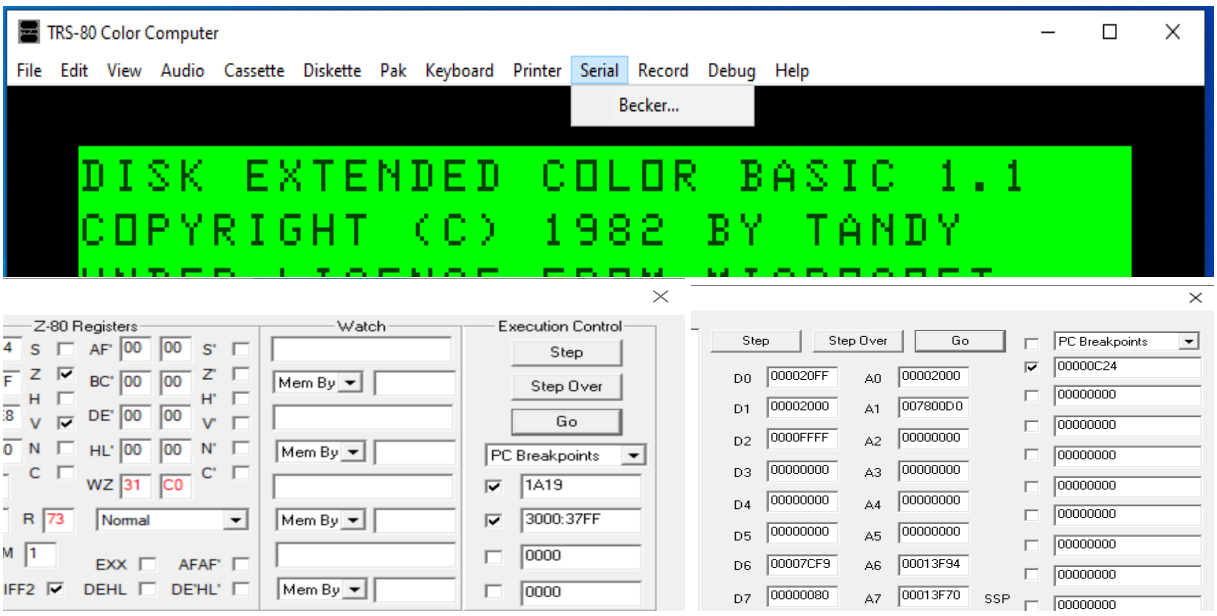
RUNS ON WINDOWS, MACOS, LINUX & RASPBERRY PI.
EMULATES MODELS 1, 2, 3, 4, 12, 16, 6000,
MC-10, COLOR COMPUTER, DT-1 AND VIDEOTEX.

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

AN UPDATED VERSION, 2.5.4, WILL BE RELEASED SHORTLY:

BETTER BREAKPOINT GUI WITH SINGLE INPUT WINDOWS.
BECKER PORT SUPPORT FOR THE COLOR COMPUTER.
HARDWARE SERIAL PORT ACCESS ON LINUX.



MAV'S TANDY ASSEMBLY QUARTERLY COLUMN

TA24 NEWS

by Ian Mavric



Brining you the latest news you need to know about the greatest Tandy TRS-80 show on the planet. This time I talk about some of the planned speakers at the show. But first the big news...!

KEYNOTE SPEAKER: STEVE LEININGER - TRS-80 MODEL I DESIGNER

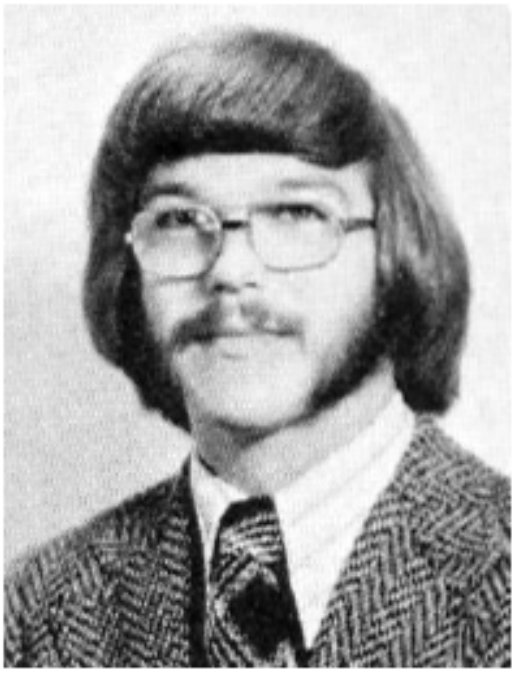


Photo: Steve Leininger at the time of designing the Model I

We are pleased to announce that Steve Leininger, the main designer of the TRS-80 Model I, will be the first talk on Saturday morning. His presentation "The REAL story of the TRS-80" is a not-to-be-missed event. Starts at 10am sharp! If you only make it to one Tandy Assembly in your lifetime, this is the one to attend.

OTHER PRESENTATIONS AND TANDY TECH TRACK FRIDAY

After Steve's talk on Saturday the next presentation at 1pm is by Arno Puder, Matt Boytim & Lawrence Kesteloot "TRS-IO++: The Swiss Army Knife for TRS-80 Enthusiasts". The continuing improvement of the TRS-IO++ is always fascinating to see what the boys can achieve.



Photo: Arno Puder, Matt Boytim & Lawrence Kesteloot

Saturday afternoon and evening will follow the traditional format of an Auction, then after dinner the Trivia Game and evening entertainment. This year the trivia game will be super exciting to watch as I pit Steve Leininger against Coco aficionado Neil Blanchard, and vintage computer specialist and collector Brendan Bergie. There

is bound to be fireworks. Read the TA24 report from the last issue of TRS8Bit to see how I have improved the game for 2024.

SUNDAY MORNING: RICK ADAMS ~ COCO PROGRAMMER

Rick was one of the speakers at the original Tandy Assembly back in 2017, so we are pleased to have him back to talk more about his involvement in writing software (mainly games) for the Color Computer. He has been with the Coco since it's beginning in 1980, and wrote popular games Temple of ROM and Shanghai, as well as the updated VIM version of TSEDIT and a Basic Pre-programmer app.



Photo: Rick Adams has humorous stories about Coco game programming

He will also give an update on what he has been working on since his previous talk at TA17. Rick's been far from idle..

FRIDAY IS TANDY TECH TRACK DAY CLASSES

A set of 5 great tutors (including myself) will present a wide range of interesting topics and a history lesson.

TTT101

Title: Using a Logic Analyzer to Understand and Troubleshoot TRS-80s

Instructor: Tim Halloran

Home: Pittsburgh, PA



Abstract: You want to understand the hardware in your TRS-80 better or perhaps fix it. What can you do? This talk will introduce using a logic analyzer -- to help diagnose retro computer problems and better understand your TRS-80. My objective is to introduce you to this powerful tool, pass along a bit of what I've learned, and jumpstart you toward effective logic analyzer use.

TTT102

Title: Programming Tandy Pocket Computers

Instructor: Jeff Birt (Hey Birt!)

Home: Rolla, Missouri, USA



Abstract: When getting started with pocket computers it is a chore to discover how to get programs into and out of the machine. Using a cassette tape for mass storage is fun and nostalgic but not the only solution available. Transferring programs and data to a PC and/or SD card is now a possibility for most systems. Finding the information, software and interfaces for transferring programs and data can be tricky.

This presentation will cover the basics of programming Tandy branded pocket computers. The tools, tips, software, etc. that make the job easier. Special emphasis will be given to the PC-2 as it is the most powerful and expandable of the line-up.

TTT103

Title: 1984: Tandy Joins the Big Boys: From Sophisticated Failure to Unexpected Success in 12 Short Months

Instructor: Ian Mavric

Home: Melbourne, Australia



Abstract: I am going to talk about Tandy's first products into the MS-DOS world in 1984. The market failure of the Tandy 2000, and the success of the much less powerful Tandy 1000. I'll cover the intro of the T2K in Dec'83 (arrived in stores in Jan 84), it's failure to meet expectations, and RS quick return to the drawing board to re-think their MS-DOS future.

TTT104

Title: Model III and 4 Floppy Disk I/O Deep Dive

Instructor: Amardeep Chana

Home: Lafayette, Indiana



Abstract: We're going to talk about how the floppy disk storage subsystem in a TRS-80 Model III and 4 works. There are several challenges in programming the low level hardware which we'll cover in detail. Using this information we'll create a game disk that boots on both a TRS-80 Model I and III.

TTT105
Title: Interfacing with the Color Computer
Instructor: John Strong

(At the time of writing, John hasn't submitted the specifics of his talk, but anything around hardware interfacing to a computer is bound to be fascinating the tech-types.)

OTHER NEWS REMINDERS BEFORE THE SHOW

At the time of writing, the 24th August, the TA24 show is around a month away and all the important aspects of the show have been locked in. In other words, what I have described above is what you will see at the show. The following items are reminders you would be keen to remember if attending to make your visit to TA a great one:

1. People who attended the "Last" CocoFEST in May this year can get free entry to TA24 just by presenting their door pass which looks like this:



Photo: one of at least two CocoFEST 2024 door pass designs

2. The day before TA starts (so that's Thursday 26th September) an informal gathering of those who are around is planned for the USAF Museum in Dayton, followed by a late lunch TBA. Its a popular event and the more the merrier. People who arrive on Wednesday night after long travel periods are most welcome to join in to help lick jetlag before the show starts the next day. We meet in the breakfast room at the hotel between 8:30 and 9:30 before getting heading out. There is always room for more in cars. Contact any of the TA team members for more details. If you are on the Tandy Discord (and if not, why not?) it's under the # TAUT-2024 folder in the Events section.

3. The trivia game, like in the last two years, will have an audience game running concurrently with the 3 experts on the panel, so bring a pen to answer the 10 questions on the audience game to see who is the cleverest in the crowd.

4. More than half the exhibitors at TA24 are also vendors selling products they make and sell, or spare parts to keep old TRS-80s running, or software and accessories. If you have your heart set on something why not reach out now to them and place your order for pick-up at the show? That way you won't miss out, especially if the item you are after is something popular and likely to sell out quickly at TA. Vendors love to hear from you to place orders and save on shipping for their customers.

See you all at Tandy Assembly 2024! Spend up big at the auction, shoot the breeze with old friends, buy me a drink in the bar (I never say no), and I'm Australian so I'm always thirsty.

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.

TRS-80 BASIC: What are the Floating Point Numbers Valid Ranges?

Jim Adams, v1.0 September 2024

A reasonable and sensible question to ask, and the answer should be easy to find, all it takes is to dig up a manual and look it up. After all, most TRS-80 BASIC manuals list the valid BASIC line numbers as 0 to 65529, a string variable may be up to 255 characters and the range of valid integers as -32768 to 32767. So surely the ranges of valid single precision floating point numbers and of valid double precision floating point numbers must be just as easy to find.

Indeed, a quick look at *TR-80 LEVEL II BASIC Reference Manual*, first edition 1978, shows the ranges of valid single and valid double precision floating point numbers on page A/16. Great, we found the answer and we are done, right? *Well*, not long after you read the answer in the manual, you will realize there are at least few typos *{the manual lists "Single Precision -1.701411E+38 to +1.701411E+38 inclusive"; "Double Precision -1.701411834544556E+38 to +1.701411834544556E+38 inclusive"}*.

No problem, after all, that was the first edition and in the rush of things, mistakes happened. Let's check the *second edition*, second printing of 1979; and page A-17 does correct the Single Precision's typos but not the double precision's typos *{Single Precision -1.701411E±38 to +1.701411E±38 inclusive; Double Precision -1.701411834544556E+38 to +1.701411834544556E+38 inclusive}*. Third time's the charm, let's try the *Model III Operation and BASIC Language Reference Manual*, second edition, February 1981, and on page 220, all typos have been corrected, Single Precision range is listed as **E±38** and Double Precision range is listed as **D±38**. Hooray, now we are done, right? *Well, not really…*

To begin with, anyone who knows about two's complement, will recognize ±38 is erroneous, since the value 0 reduces the + range by one, i.e. -38 to +37 or -39 to +38. Apparently, more reference manuals are needed and I searched as many manuals as I could get my hands on, starting with the Altair BASIC 2.0 1975 through GW-BASIC 3.23 1988, as they are decedents of the same Microsoft family of BASICs.

And there lies the problem, each manual either repeats the inaccuracies or lists different ranges altogether. *Houston, we have a problem.* I cannot find the ranges that show the **largest** and **smallest** numbers showing **all the significant digits** for both the single and the double precision numbers. The manuals, at best, show subsets of the valid ranges; or worse, have erroneous ranges *{see last section}*.

Please keep in mind this dilemma when representing the infinite real numbers with most digital computers: how to squeeze a two pound *bologna or veggies* into a one pound sandwich? A) *real numbers* are infinite, ie given an integer n, there exists n+1 or –n there exists –n-1; and infinite fractions between any two integers, ie given m<n, m/n there exists m/n+1 or –m/n there exists –m/n+1. B) Floating point numbers are a limited finite subset approximation of real numbers; with finite set of non-contiguous digitized binary fractions, ie unrepresented gaps exist between two digitized fractions.

Now that we understand the problem, let's quickly jump to the ranges I calculated after lots more digging and a little bit of programming.

Floating Point Ranges

{Those who are smarter than I, please correct my calculations and share them with the rest of us, I love to learn the correct ranges; and those who already knew the ranges, I should have asked you first!}

GW-BASIC and TRSDOS 6.x BASIC allow the entire MBF format ranges via the BASIC assignment statement. All earlier BASICs [TRS-80, CP/M MBASIC 5.21 through Altair BASIC 3.0] assignment statements accept smaller subsets of the entire valid MBF ranges. However, POKE, VALPTR() or machine language routines can store the values directly into memory and BASIC will evaluate them correctly.

For example:
MBF internally, GW-BASIC and TRSDOS 6 Disk BASIC assignment statements allow:
Normalized real 8digit: -1.7014117E+38 to -2.9387359E-39, 2.9387359E-39 to 1.7014117E+38
CP/M MBASIC 5.21 through Altair EXTENDED BASIC 3.2 assignment statements allow:
BASIC Assignment: -1.7014109E+38 to -1.0000047E-38, 1.0000047E-38 to 1.7014109E+38
Yet, A!=1.0:POKE VALPTR(A!)+3,1 allows storing 2.9387359E-39

Microsoft Binary Format Valid Ranges

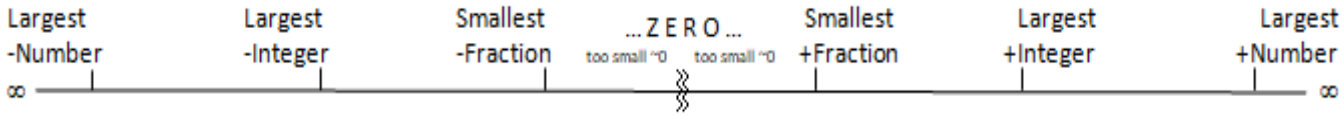
Z80, i8080, i8088
zero all zero's: **0**
4byte normalized real 8digit as *integer*: **-16,777,216** to **16,777,216**
normalized numbers with implicit "0.1", effectively 2^24=16777216
PRINTed as 7digit: -1.677722E+07 to 1.677722E+07
4byte normalized real 8digit: **-1.7014117E+38** to **-2.9387359E-39**,
2.9387359E-39 to **1.7014117E+38**
PRINTed as 7digit: -1.701412E+38 to -2.938736E-39, 2.938736E-39 to 1.701412E+38

8byte real 17digit as integer: **-72,057,594,037,927,936** to **72,057,594,037,927,936**
normalized numbers with implicit "0.1", effectively 2^56=72057594037927936
PRINTed as 16digit: -7.205759403792794D+16 to 7.205759403792794D+16
8byte normal real 17digit: **-1.701411834604692D+38** to **-2.938735877055719D-39**,
2.938735877055719D-39 to **1.701411834604692D+38**
PRINTed as 16digit: -1.70141183460469D+38 to -2.93873587705571D-39,
2.93873587705571D-39 to 1.70141183460469D+38

MC6809, MC6803, MC6800, MOS6502
zero all zero's: **0**
5byte normalized real 10digit as integer: **-4,294,967,296** to **4,294,967,296**
normalized numbers with implicit "0.1", effectively 2^32=4294967296
PRINTed as 8digit: -4.2949673E+09 to 4.2949673E+09
5byte normalized real 11digit: **-1.7014118345E+38** to **-2.9387358771E-39**,
2.9387358771E-39 to **1.7014118345E+38**
PRINTed as 9digit: -1.70141183E+38 to -2.93873588E-39,
2.93873588E-39 to 1.70141183E+38

Integer mantissa values represented by MBF are 100% accurate up to 2^24 and 2^56; or 2^32. All other numbers, integers or fractions beyond the significant digits, are close approximations of the true values, due to the "binary fraction intervals" verses the infinite real number line.

BASIC Interpreters Valid Floating Point Ranges



MBF Using 4Byte and 8Byte Little-Endian
GW-BASIC 3.23 through 1.04

4byte, 8digit -1.7014117E+38 PRINTed as 7digit -1.701412E+38	-16777216 -1.67722E+07	-2.9387359E-39 -2.938736E-39	2.9387359E-39 2.938736E-39	16777216 1.67722E+07	1.7014117E+38 1.701412E+38
8byte, 17digit -1.7014118346046923D+38 PRINTed as 16digit -1.701411834604692D+38	-72057594037927936 7.2057594037927940+16	-2.9387358770557188D-39 -2.938735877055719D-39	2.9387358770557188D-39 2.9387358770557190-39	72057594037927936 7.2057594037927940+16	1.7014118346046923D+38 1.701411834604692D+38

Model 4 TRSDOS 6.x Disk BASIC, LS-DOS 6.x Disk BASIC, DOSPLUS 4 Disk BASIC

4byte, 8digit -1.7014117E+38 PRINTed as 6digit -1.70141E+38	-16777216 -1.6772E+07	-2.9387359E-39 -2.93874E-39	2.9387359E-39 2.93874E-39	16777216 1.6772E+07	1.7014117E+38 1.70141E+38
8byte, 17digit -1.7014118346046922D+38 PRINTed as 16digit -1.701411834604692D+38	-72057594037927936 7.2057594037927940+16	-2.9387358770557177D-39 -2.938735877055719D-39	2.9387358770557177D-39 2.9387358770557190-39	72057594037927936 7.2057594037927940+16	1.7014118346046922D+38 1.701411834604692D+38

Model III/II/I TRSDOS Disk BASIC, *All DOSs Disk BASIC*, Model III BASIC, Model I LEVEL II, MBASIC 5.x

4byte, 8digit -1.7014117E+38 PRINTed as 6digit -1.70141E+38	-16777216 -1.6772E+07	-1.0000048E-38 -1.00001E-38	1.0000048E-38 1.00001E-38	16777216 1.6772E+07	1.7014117E+38 1.70141E+38
8byte, 17digit -1.7014118346046922D+38 PRINTed as 16digit -1.701411834604692D+38	-72057594037927936 7.2057594037927940+16	-1.0000000000000005D-38 -1.0000000000000001D-38	1.0000000000000005D-38 1.0000000000000001D-38	72057594037927936 7.2057594037927940+16	1.7014118346046922D+38 1.701411834604692D+38

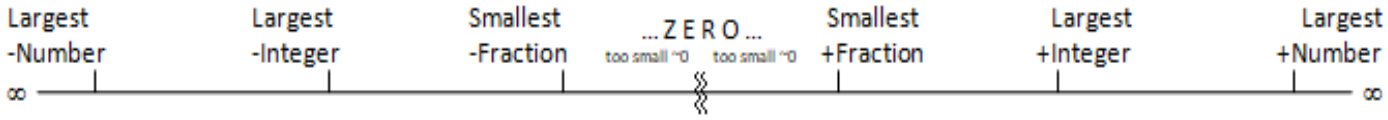
Model III/I LEVEL I

4byte, 7digit -1.701411E+38 PRINTed as 6digit -1.70141E+38	-16777216 -1.6772E+07	-2.938741E-39 -2.93873E-39	2.938741E-39 2.93873E-39	16777216 1.6772E+07	1.701411E+38 1.70141E+38

MBASIC 4.x through ALTAIR BASIC 3.0

4byte, 8digit -1.7014109E+38 PRINTed as 6digit -1.70141E+38	-16777216 -1.6772E+07	-1.0000048E-38 -1.00001E-38	1.0000048E-38 1.00001E-38	16777216 1.6772E+07	1.7014109E+38 1.70141E+38
8byte, 17digit -1.7014118346046922D+38 PRINTed as 16digit -1.701411834604692D+38	-72057594037927936 7.2057594037927940+16	-1.0000000000000005D-38 -1.0000000000000001D-38	1.0000000000000005D-38 1.0000000000000001D-38	72057594037927936 7.2057594037927940+16	1.7014118346046922D+38 1.701411834604692D+38

BASIC Interpreters Valid Floating Point Ranges (Continued)



MBF Using 5Byte Big-Endian
CoCo 3/2/1 SDECB through CB, MC-10 MCB

5byte, 11digit -1.7014118345E+38 PRINTed as 9digit -1.70141183E+38	-4294967296 -4.2949673E+09	-2.9387358771E-39 -2.93873588E-39	2.938735771E-39 2.93873588E-39	4294967296 4.2949673E+09	1.7014118345E+38 1.70141183E+38

MBF Using 8Byte and 4Byte BCD Representation
XENIX MBASIC, Model 200/102/100 DVI, Model 200/102/100 BASIC

8byte, 14digit - 9.9999999999999D+62	-9.9999999999999D+13	-9.9999999999999D-64	9.9999999999999D-64	9.9999999999999D+13	9.9999999999999D+62
4byte, 6digit -9.99999E+62	-9.99999E+5	-9.99999E-64	9.99999E-64	9.99999E+5	9.99999E+62

Additional MBF Comments

a. Floating Points valid ranges are presented in number line order, from negative to zero to positive
b. Values shown as allowed by the BASIC interpreters assignment statement
c. Default precision is listed first, ie top row
d. MBF developed by Monte Davidoff as a contractor for Bill Gates and Paul Allen. Binary representation inspired by DEC PDP-11 BASIC-PLUS [1.4E-39 to 1.7E38, 4Byte or 8Byte big-endian].
e. LEVEL I Floating Point reals developed by Steven W Leininger, in i8080+Z80. Microsoft already contracted for LEVEL II; so in MBF compatible format but coded by Leininger.

MBF Internal Representation

For the sake of brevity, we will focus on the floating point values only. However, here is a glimpse of the internal binary representation 1975-1988 till Microsoft/Intel switched to the IEEE 754-1985 standard: the byte order of the Mantissa, Sign bit, Exponent, and the Exponent's bias. Fortunately, plentiful resources online discuss the details of the Microsoft Binary Format internal representation.

Format

4Byte Single	e7..e0 EXP	s0f22..f16 MSB	f15..f8 2SB	f7..f0 LSB	
8Byte Double	e7..e0 EXP	s0f54..f48 MSB	f47..f40 2SB	f39..f32 3SB	
	f31..f24 4SB	f23..f16 5SB	f15..f8 6SB	f7..f0 LSB	
5Byte	e7..e0 EXP	s0f30..f24 MSB	f23..f16 2SB	f15..f8 3SB	f7..f0 LSB

Value

v = 0 if EXP = 0
v = (-1)^s0 × mantissa × 2^(EXP - 128)
otherwise, v is a binary fraction mantissa, v = 0.1 f54, f30, or f22 f21 f20 ... f0

Little-Endian Order

Memory Low..High 4Byte single = LSBlsb 2SB s0MSB EXP
Memory Low..High 8Byte double = LSBlsb 6SB 5SB 4SB 3SB 2SB s0MSB EXP

Big-Endian Order

Memory Low..High 5Byte = EXP s0MSB 2SB 3SB LSBlsb

MBF-DUMP/BAS

As the old saying goes, if you can't find it, write it; so I started writing a *line or two* of BASIC to verify my understanding of the internal binary formats and the valid ranges. It does not find the valid ranges, I *pre-calculated* the valid ranges and the program stores the pre-calculated values into single precision and double precision variables, then displayed both the PRINT results as well as the raw memory contents. With trial and error, tweaking my calculations and repeated runs, the largest and smallest values were validated.

So here is the listing for MBF-DUMP/BAS in TRS-80 BASIC LEVEL II.

```
1 REM By J A, 25 Jan 2024, BASIC LEVEL II 1978 .. Rev 1.3 1981, Altair 1977..75
10 REM MICROSOFT BINARY FORMAT (MBF) FLOATING-POINT 4BYTE 8BYTE LITTLE-ENDIAN
20 A!=16777215:A%=VARPTR(A!):CLS:'LARGEST SINGLE INTEGER = (2^24) -1
30 REM VARIABLE TYPE/LENGTH AND VARIABLE NAME1(A-Z) AND NAME2(A-Z,0-9)
40 PRINT"MICROSOFT BINARY FORMAT (MBF) FLOATING-POINT":PRINT"4BYTE AND 8BYTE LITTLE-
ENDIAN":PRINT:PRINT "SINGLE TYPE= "; PEEK(A%-3); "   ";
41 'TRSDOS 6.2-> CLS:PRINT "SINGLE TYPE= "; PEEK(A%-4); "   ";
50 PRINT "NAME= "; CHR$(PEEK(A%-1)); CHR$(PEEK(A%-2)); "   "; PEEK(A%-1); PEEK(A%-2); "    ";
51 'TRSDOS 6.2-> PRINT "NAME= "; CHR$(PEEK(A%-3)); CHR$(PEEK(A%-2)); "   "; PEEK(A%-3); PEEK(A%-2); "    ";
60 REM BASIC VALUE AND VARPTR() ADDRESS (POINTS TO EXP BYTE)
70 PRINT "VARPTR= ";A%
80 PRINT "LARGEST SINGLE INTEGER= ";A!:GOSUB 1000
90 POKE A%+2,255:PRINT "LARGEST NEGATIVE SINGLE INTEGER= ";A!:GOSUB 1000
100 INPUT "PRESS ANY KEY FOR SINGLE PRECISION";A$
110 POKE A%+2,127:'RESET SIGN BIT FOR POSITIVE MANTISSA
120 POKE A%+3,255:PRINT "LARGEST SINGLE NUMBER= ";A!:GOSUB 1000
130 POKE A%+2,255:PRINT "LARGEST NEGATIVE SINGLE NUMBER= ";A!:GOSUB 1000
140 POKE A%+2,127:'RESET SIGN BIT FOR POSITIVE MANTISSA
150 POKE A%+3,1:PRINT "LARGEST SINGLE FRACTION= ";A!:GOSUB 1000
160 POKE A%+2,255:PRINT "LARGEST NEGATIVE SINGLE FRACTION= ";A!:GOSUB 1000
170 A!=1.0:POKE A%+3,1:PRINT "SMALLEST SINGLE FRACTION= ";A!:GOSUB 1000
180 POKE A%+2,128:PRINT "SMALLEST NEGATIVE SINGLE FRACTION= ";A!:GOSUB 1000
190 INPUT "PRESS ANY KEY TO CONTINUE";A$:CLS
200 A!=0.0:PRINT "ZERO= ";A!:GOSUB 1000
210 A!=1.0:PRINT "ONE= ";A!:GOSUB 1000
220 A!=-1.0:PRINT "-ONE= ";A!:GOSUB 1000
230 A!=1234567:PRINT "VALUE= ";A!:GOSUB 1000
240 A!=1.234567:PRINT "VALUE= ";A!:GOSUB 1000
250 INPUT "PRESS ANY KEY FOR DOUBLE PRECISION";A$:CLS

310 REM MBF DOUBLE
320 A#=72057594037927935:A%=VARPTR(A#):'LARGEST DOUBLE INTEGER = (2^56) -1
330 REM VARIABLE TYPE/LENGTH AND VARIABLE NAME1(A-Z) AND NAME2(A-Z,0-9)
340 PRINT "DOUBLE TYPE= "; PEEK(A%-3); "   ";
341 'TRSDOS 6.2-> PRINT "DOUBLE TYPE= "; PEEK(A%-4); "   ";
350 PRINT "NAME= "; CHR$(PEEK(A%-1)); CHR$(PEEK(A%-2)); "   "; PEEK(A%-1); PEEK(A%-2); "   ";
351 'TRSDOS 6.2-> PRINT "NAME= "; CHR$(PEEK(A%-3)); CHR$(PEEK(A%-2)), PEEK(A%-3); PEEK(A%-2),
360 REM BASIC VALUE AND VARPTR() ADDRESS (POINTS TO EXP BYTE)
370 PRINT "VARPTR= "; A%
380 PRINT "LARGEST DOUBLE INTEGER= ";A#:GOSUB 2000
390 POKE A%+6,255:PRINT "LARGEST NEGATIVE DOUBLE INTEGER= ";A#:GOSUB 2000
```

```
400 INPUT "PRESS ANY KEY TO CONTINUE";A$:CLS
410 POKE A%+6,127:'RESET SIGN BIT FOR POSITIVE MANTISSA
420 POKE A%+7,255:PRINT "LARGEST DOUBLE NUMBER= ";A#:GOSUB 2000
430 POKE A%+6,255:PRINT "LARGEST NEGATIVE DOUBLE NUMBER= ";A#:GOSUB 2000
440 POKE A%+6,127:'RESET SIGN BIT FOR POSITIVE MANTISSA
450 POKE A%+7,1:PRINT "LARGEST DOUBLE FRACTION= ";A#:GOSUB 2000
460 POKE A%+6,255:PRINT "LARGEST NEGATIVE DOUBLE FRACTION= ";A#:GOSUB 2000
470 A#=1.0:POKE A%+7,1:PRINT "SMALLEST DOUBLE FRACTION= ";A#:GOSUB 2000
480 POKE A%+6,128:PRINT "SMALLEST NEGATIVE DOUBLE FRACTION= ";A#:GOSUB 2000
490 INPUT "PRESS ANY KEY TO CONTINUE";A$:CLS
500 A#=0.0:PRINT "ZERO=";A#:GOSUB 2000
510 A#=1.0:PRINT "ONE= ";A#:GOSUB 2000
520 A#=-1.0:PRINT "-ONE=";A#:GOSUB 2000
530 A#=1234567890123456:PRINT "VALUE=";A#:GOSUB 2000
540 A#=1.234567890123456:PRINT "VALUE=";A#:GOSUB 2000
550 INPUT "PRESS ANY KEY TO END";A$:CLS
560 END:'DONE

1000 REM SINGLE PRECISION 4BYTE LITTLE-ENDIAN MEMORY CONTENT
1010 PRINT "MEMORY= ";
1020 FOR I%=0 TO 3
1030 PRINT PEEK(A%+I%);
1040 NEXT I%
1050 PRINT
1060 RETURN

2000 REM DOUBLE PRECISION 8BYTE LITTLE-ENDIAN MEMORY CONTENT
2010 PRINT "MEMORY= ";
2020 FOR I%=0 TO 7
2030 PRINT PEEK(A%+I%);
2040 NEXT I%
2050 PRINT
2060 RETURN
```

MBF-DUMP/BAS, with few minor changes, is easily adaptable for the Model 4, CP/M MBASIC, GW-BASIC, etc. I am confident many inquisitive readers can eloquently enhance my simple brute force program for both speed and portability.

Please see the comments lines for changes to adapt it to different BASICs, they are along the lines:

TRSDOS 6.2 DISK BASIC:	40 PEEK(A%-4), 50 PEEK(A%-3); PEEK(A%-2)
MSDOS GWBASIC 3.23:	same as TRSDOS 6.2 DISK BASIC
CP/M MBASIC 5.28:	same as GWBASIC 3.23 but remove all CLS
ALTAIR BASIC 4.1-3.0:	same as LEVEL II but remove all CLS

BASIC Manuals Floating Point Ranges Discrepancies

None of the manuals accurately list the full range, including largest and the smallest values specified using all the significant digits.

GWBASIC 1991 3.20-2.11 1984, *Microsoft GW-BASIC User's Guide*, July 1987, page 50
- 3.0 x 10^-39 to 1.7 x 10^38
- Single precisions are stored 7 digits, only 6 accurate
- Double precisions are stored 17 digits, only 16 accurate

GWBASIC 1984 2.11-1.04 1982, *MINDSET Microsoft GW-BASIC Reference Manual*, 1984, page 2-6
- 10^-38 to 10^+38
- Single precisions are stored 7 digits, only 6 accurate
- Double precisions are stored 17 digits, only 16 accurate

IBM BASIC 1984 3e 3.0-1.10 2e, *IBM BASIC Manual v1.10 2e*, May 1982, page 3-10
- 2.9E-39 to 1.7E+38
- Single precisions are stored 7 digits, only 6 accurate
- Double precisions are stored 17 digits, only 16 accurate

MBASIC 5.0 1979/BASCOM 5.3 1981, *Microsoft BASIC-80 Reference Manual 5.0*, 1979, page 1-5
- 10^-38 to 10^+38
- Single precisions are stored 7 digits, only 6 accurate
- Double precisions are stored 17 digits, only 16 accurate

TRS-80 Model III/4 Operation and BASIC Reference Manual 2e, February 1983, page 220
TRS-80 Model III Operation and BASIC Reference Manual 2e, February 1981, page 220
- Single Precisions -1.701411E+-38 to 1.701411E+-38
- Double Precisions -1.701411834544556D+-38 to 1.701411834544556D+-38

TRS-80 Model I LEVEL II BASIC Reference Manual 2e, 1979, page A-17
- Single Precisions -1.701411E+-38 to 1.701411E+-38
- Double Precisions -1.701411834544556D+38 to 1.701411834544556D+38

TRS-80 Model I LEVEL II BASIC Reference Manual, 1978, page A/16
- Single Precisions -1.701411E+38 to 1.701411E+38
- Double Precisions -1.701411834544556E+38 to 1.701411834544556E+38

TRS-80 Model I User's Manual for LEVEL I, 1978, page 38, page 69, page 186
- Single precisions are stored 8 digits, only 5 to 6 accurate
- accepts scientific notation n.nE+-nn

MITS Altair BASIC Reference Manual
4.1, July 1977, page 11, page 13: 7 digits and 17 digits; 2.9387E-38 to 1.70141E+38
4.0 January 1977, page 11, page 13: 7 digits and 17 digits; 2.9387E-38 to 1.70141E+38
EXTENDED 16K 3.2, December 1975, page 81: 16 digits; 10^-38 to 10^+38
3.0, August 1975, page 6: 7 digits but only 6 accurate; 2.93875 x 10^-39 to 1.70141 x 10^+38

I hope you have found the journey fun and useful. *Happy* computing.

Dedication

To all the TRS-80 enthusiasts out there, who have been sharing their knowhow with new generation of Trash80 users. 8bit computing is timeless and provides complete hands-on to the bare metal, *putting you in full control of entire computer and best of all, in direct control of the CPU. I wish all of you* Happy Computing!

Acknowledgments

I am indebted to those who are kind enough to share their knowledge and their love for 8bit computing, they are too many to thank by name but please allow to thank the few whom I annoyed the most with my questions yet received their abundance support:

Vernon B Hester for his plethora of software culminating with Esoteric/MultiDOS, and his generous insight into BASIC, BASICG, Z80 and any/all topics of technology. *George and Peter Phillips* for their truly amazing trs80gp emulator. *Ira Goldklang* for reviving the TRS-80 and all things TRS-80, especially the Model I, III, 4, 4P, 4D. *Peter Cetinski* for his preservation of the Model II family of computers (M2/M16/M12/M16B/6000). *Chuck(G)* on vcfed.org forums for his many CP/M insights and early computers. *Dusty Miller* for editing and publishing the *TRS8BIT newsletter*, with special gratitude for his encouragement and support with which all my articles would not have been possible. And many, many more enthusiasts. I thank *all* of you for your help over the decades, and sincerely wish you all the best.

Further Readings

ROM Routines Documented. Jack Decker. 1984.
MBF Single and Double precision representation: pages 23-25. Excellent ROM details book.

More about Arithmetic Operations (PDF). Hubert Howe. *COMPUTRONICS* March 1980.
MBF representation, BCD representation: pages 545-548.

Fast BASIC – Beyond TRS-80 BASIC USR EDTASM. George Gratzer. 1982.
Variable Table: pages 52-55; MBF Single and Double precision details: pages 227-229.

Microsoft BASIC Decoded and Other Mysteries. James Farvour. 1981.
MBF Floating Point Accumulators WRA1+WRA2: page 12; VLT, VARPTR(): pages 44-46.

LEVEL 1 Disassembly. Andy Vaught Mesa. 2018.
4Byte variables, 5Byte FPACC, https://github.com/andyv/level1/blob/master/memory.txt

Machine Language Programming for the "8008". Nat Wadsworth. SCELBI 1975.
Identical format to MBF and LEVEL I: page 5-13; Excellent Arithmetic: pages 5-1 through 5-45.

Microsoft KB35826 (PDF). Microsoft. 2006.

MBF Format, https://en.wikipedia.org/wiki/Microsoft_Binary_Format

William Kahan, IEEE 754, http://www.cs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF

LLL BASIC Floating Point Math Package. David Mead. *Interface Age* February 1977.
Public Domain floating point implementation: pages 104-108, 120.

Michael Maples, LLL Floating Point, https://www.retrotechnology.com/herbs_stuff/float.html

Monte Davidoff, https://www.theregister.com/2001/05/11/microsoft_altair_basic_legend_talks/


BRITAIN'S BEST-SELLING DIGITAL CULTURE MAGAZINE
PIXEL ADDICT
£6.50 ■ 2024 ■ ISSUE 19
Tandy
The Story of an Electronics Giant
PLUS
NEIL WEST OF SEGA POWER
40 YEARS OF KING'S QUEST
MULTITECH'S APPLE II CLONE
20 GOTO 10 BOOK REVIEW
20 GOTO 10



PASCAL HOLDRY

Bonsoir Dusty

If I had these .jed files of the PALs U6, U36, U40, U45 and U48, I could try this card for your publication on TRSBIT. Because I'm not going on vacation
I have a lot of promises, but I'm still waiting!!!!
I was making the CPU 68000 6MHz card long (I had a card with these PALs at home), but two people dissuaded me and assured me that they would provide me with the PALs, if I cloned the CPU68000 8MHz card and I have been waiting since 15 days !!!! I already have all the components and I invested a hundred hours of work because of this promise and now I don't want to invest in manufacturing without these PAL files.
I sent this email with these people in copy so that they understand that cloning a card is a lot of work and research of components
If the card works, I will modify it with components that are easier to find
So if you can find these (files and/or PALs)

Bonsoir à tous,

(traduction Google)

THANKS,
I plan to send this card for manufacturing tomorrow, with another 32K/64K MII card, if I can make the transport as profitable as possible.
Ian is sending me a CPU 6800 8MHz card with a PAL and not a HAL, I should receive it in a fortnight (depending on the Olympic games????)
I look for errors with a "stabilo" to check all the tracks and naturally I still found small non-binding errors that the software could not detect.
If the card works, as usual I will send GERBERS to the people who helped me clone the card!!!
I'm thinking of cloning a 128K/256K memory card and or a 6000 512K/1MB card (but I don't have this card and I don't know if this card works on an M16??? plus the delay line is special and not found 240nS in 8/10 outputs of 30nS each I think to use 2 delay lines of 150nS in series ????

What do you prefer? the 128/256K or 512/1MB card ????
I'm crossing my fingers so that my 8MHz CPU card works, then I will replace the components not found (DP8303, MC6882 A or B with 74LSxx or others
I don't promise anything about the result, but with Ian's card I will be able to carry out continuity tests track by track with my multimeter in the event of a problem.

A+

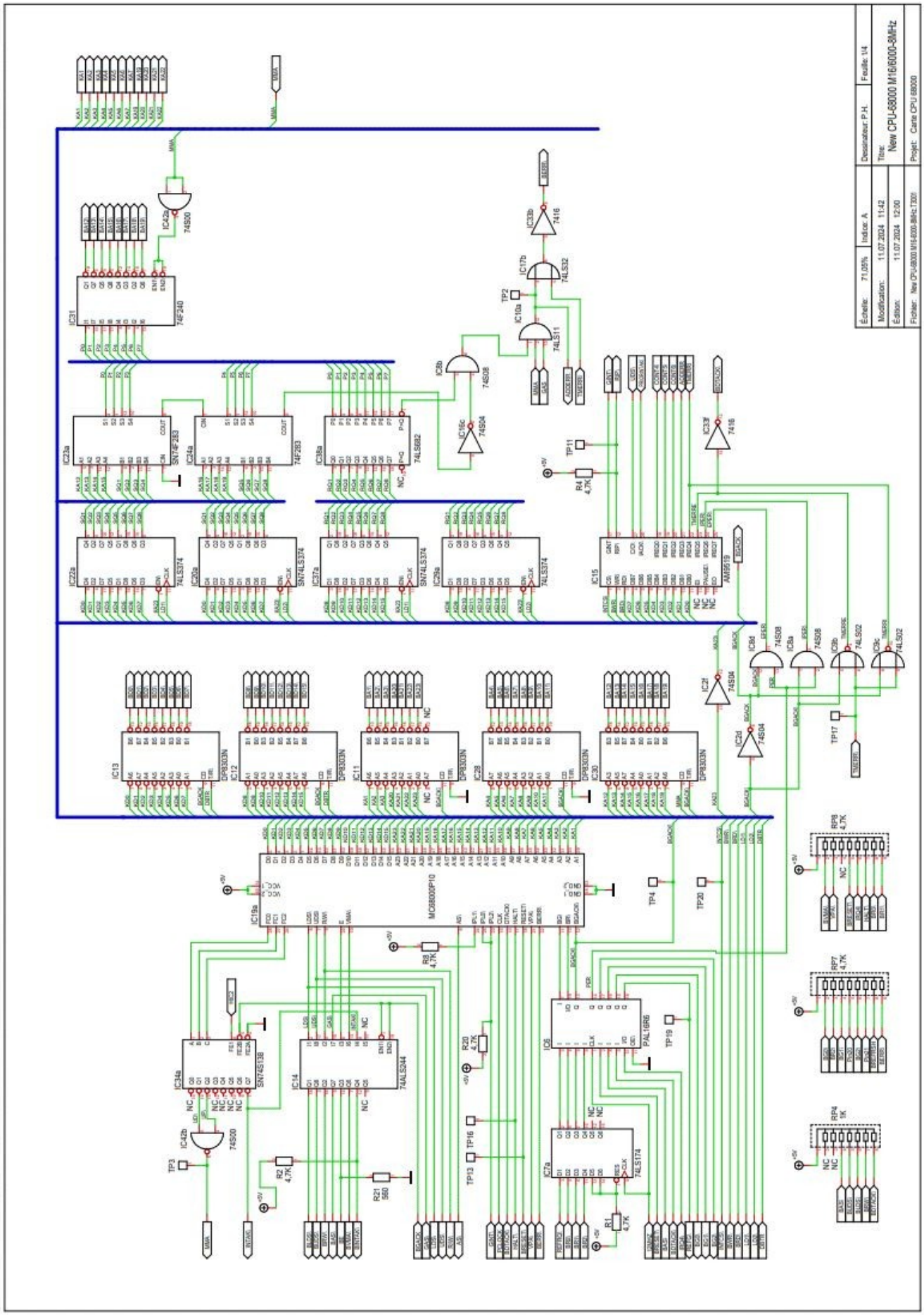
Pascal

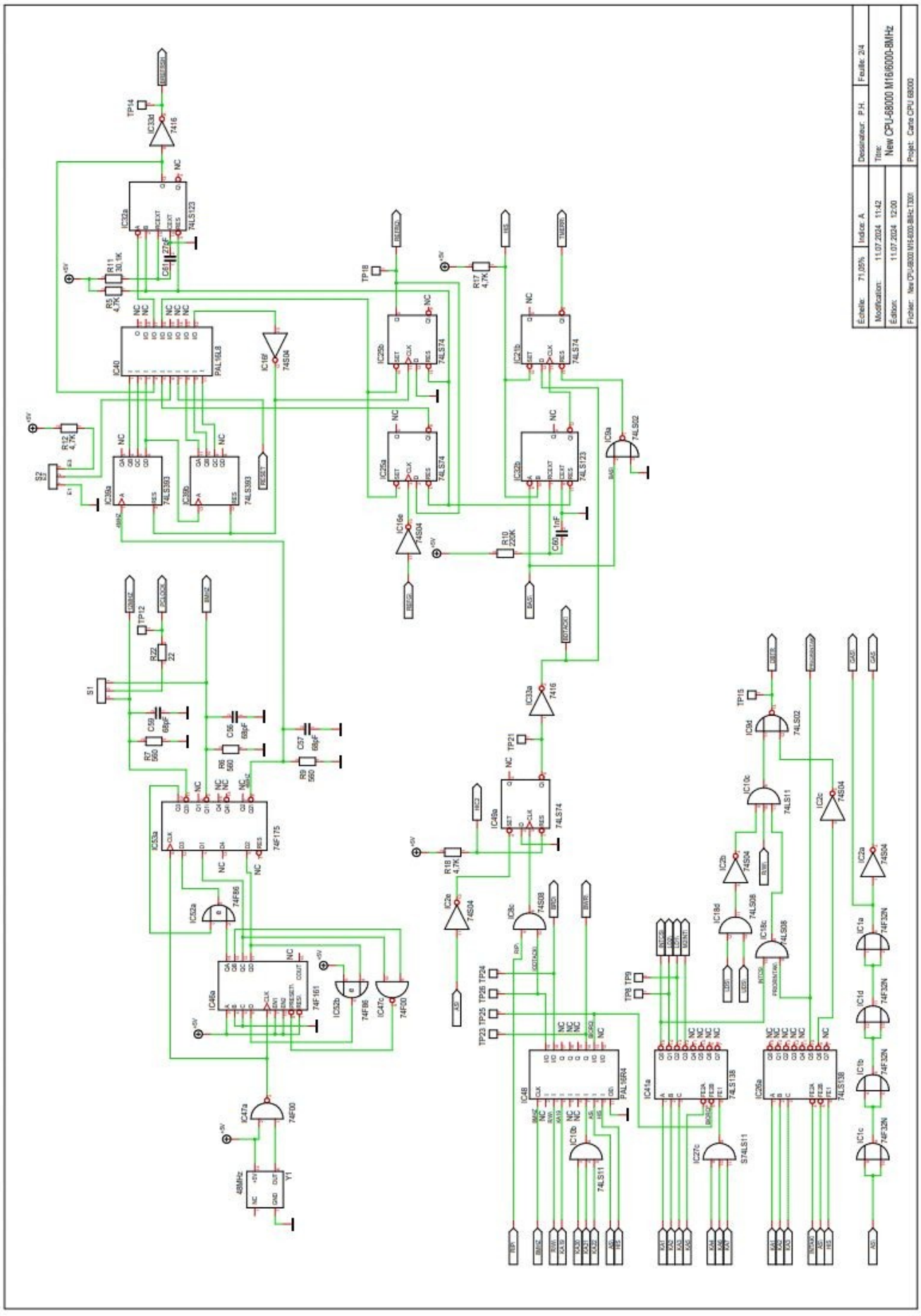
Well, I will try to clone the 512/1MB card soon
I have to study it to check the feasibility
If you have photos for me to study and to post on my site (the nomenclature does not always correspond to the reality of the components placed on the card
I finish the CPU 68000 and its development with my fingers crossed and I start on the 512/1MB card
I always try to be as close as possible to reality, because certain software uses the subtleties of the card that I clone.
The larger TRS-80s seem to use expanded memory and replacing 4116s with 4164s like on the TRS-80 M1 doesn't seem wise to me.
The delay lines are often there to delay the CAS and RAS signals between the memory banks and to use "it seems to me" only one refresh signal for several memory banks

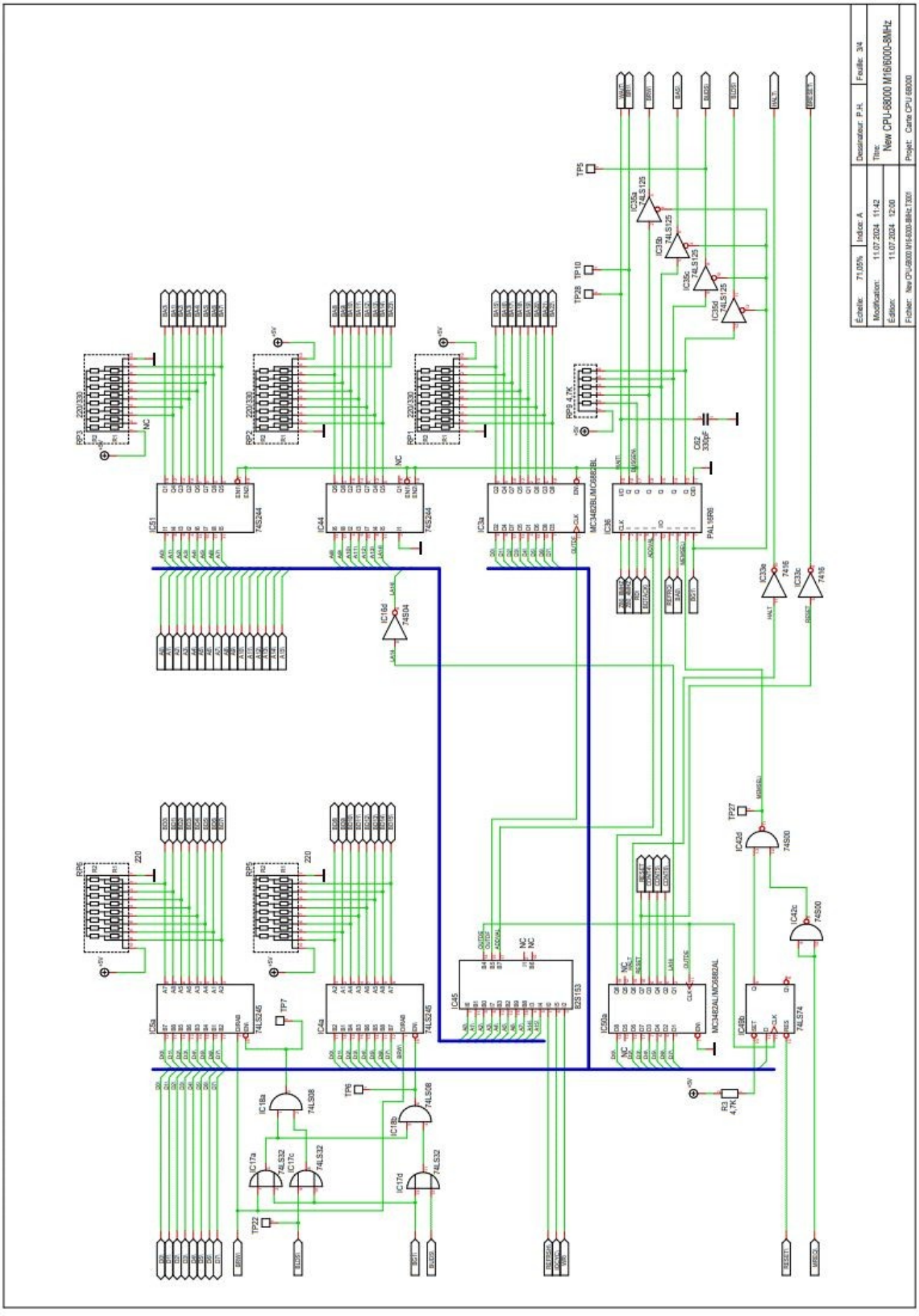
A+

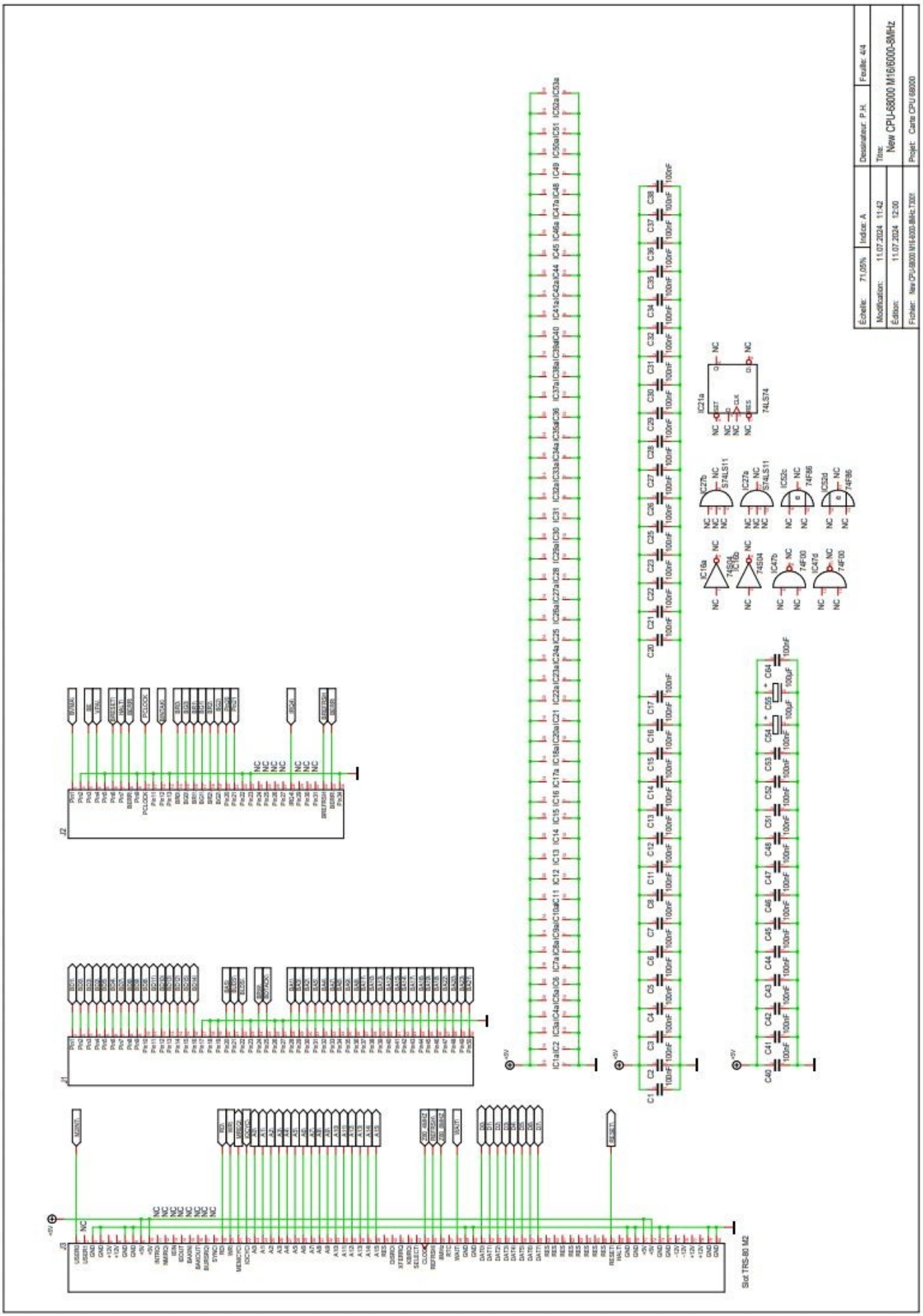
Pascal

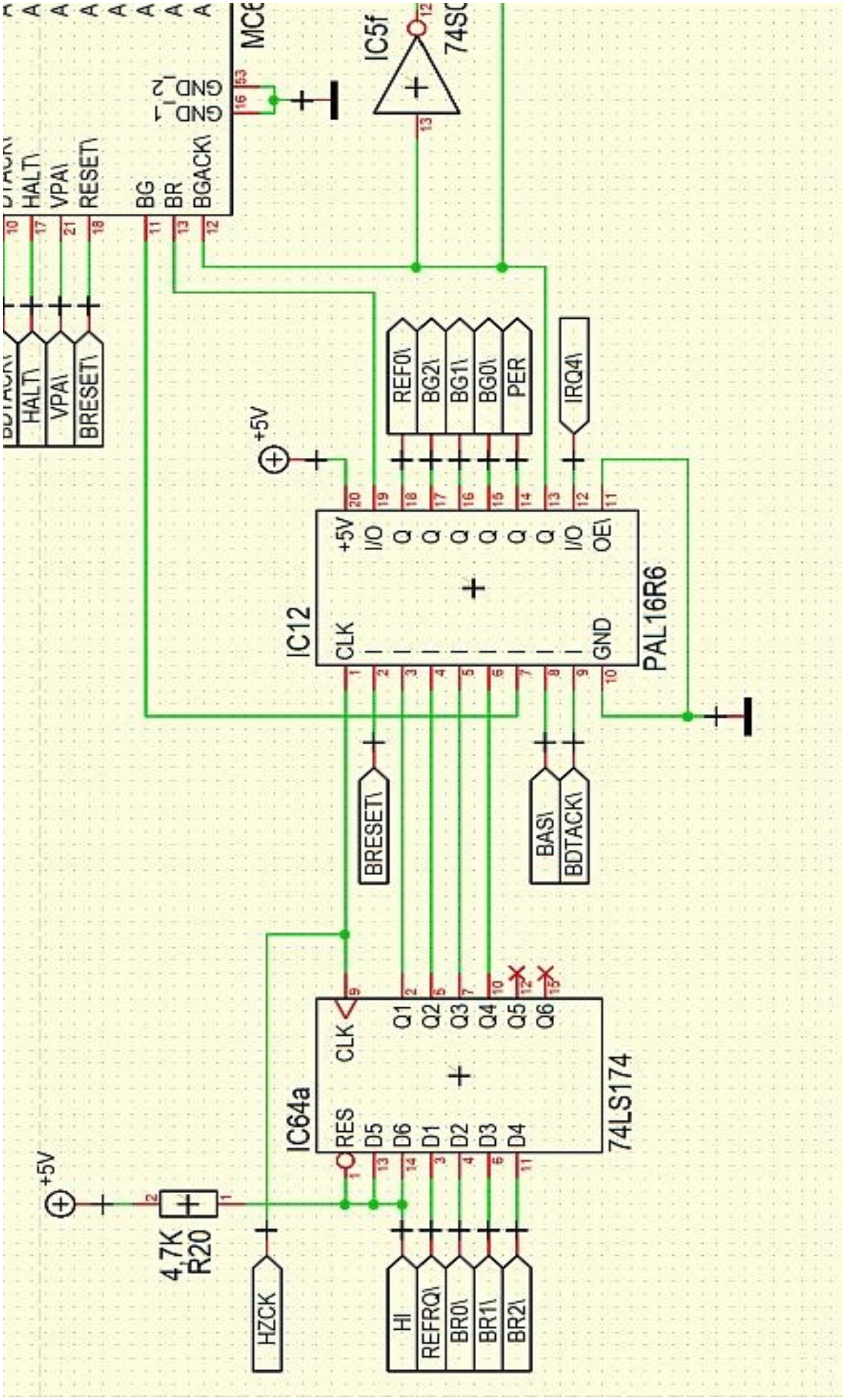



Echelle: 71.05%
Indice: A
Dessinateur: P.H.
Feuille: 1/4
Modification: 11.07.2024 11:42
Titre: New CPU-68000 M16/8000-8MHz
Edition: 11.07.2024 12:00
Fichier: New CPU-68000 M16-8000-8MHz.T3000
Projet: Carte CPU 68000


Echelle: 71,05%
Indice: A
Dessinateur: P.H.
Feuille: 2/4
Modification: 11.07.2024 11:42
Titre: New CPU-68000 M16/6000-8MHz
Edition: 11.07.2024 12:00
Fichier: NewCPU-68000 M16-6000-8MHz.T3000
Projet: Carte CPU 68000


Echelle: 71.05%
Indice: A
Dessinateur: P.H.
Feuille: 3/4
Modification: 11.07.2024 11:42
Edition: 11.07.2024 12:00
Titre: New CPU-68000 M16/6000-8MHz
Fichier: New CPU-68000 M16-6000-8MHz.T3000
Projet: Carte CPU 68000


Echelle: 71.05%
Indice: A
Dessinateur: P.H.
Feuille: 4/4
Modification: 11.07.2024 11:42
Titre: New CPU-68000 M16/8000-8MHz
Edition: 11.07.2024 12:00
Fichier: New CPU-68000 M16-8000-8Mhz.T30!
Projet: Carte CPU 68000
Std TRS-80 M2


IC5f
74S
MC6
GND_1
GND_2
BG
BR
BGACK\
HALT\
VPA\
RESET\
BRESET\
+5V
REF0\
BG2\
BG1\
BG0\
PER
IRQ4\
IC12
CLK
I/O
Q
OE\
GND
PAL16R6
BAS\
BDTACK\
IC64a
RES
D5
D6
D1
D2
D3
D4
Q1
Q2
Q3
Q4
Q5
Q6
74LS174
4,7K
R20
HZCK
HI
REFRQ\
BR0\
BR1\
BR2\

MMU extender - thanks to Peter Cetinski for the image



My project to clone the CPU-68000 8MHz card is in its final phase. [New-CPU 68000 8MHz (prof-80.fr)]

Thanks to the cooperation of Stephe, David and Ian who sent me from Australia a CPU-68000 8MHz card for comparison with my project. I think my card works correctly,.
My card passes the tests correctly. I can not load XENIX on my hard drives because of the heat wave in Paris which disturbed the alignment of all my hard drives.
Steve seems to make a card and will thus be able to test this card completely at home.
The wiring of this card is difficult for a novice of the soldering iron, and the choice of the I.C. of capital importance.
This NEW CPU-68000 card is made on a 4-layer "PCB" and the supply of expensive and difficult components, another version is under study with less rare and therefore less expensive components. The different PALs have been replaced by 5 GALs of the same type "GAL16V8" to avoid a difficult supply of virgin PAL, very expensive and thus allow the programming of these components by a programmer type "made in China". The cost price of this card depends on the number of cards in manufacture, I therefore advise against making this card individually and without good knowledge in electronics. See Ian if he wants to build this card, I will make a 512/1MB memory card, to save space in the basket of an M16

I am looking for a THT video card for M12, (my card works randomly) !!! I redid all the soldering and I suspect a micro-cut in the THT transformer
Accueil (prof-80.fr)


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


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

IN MAV'S WORKSHOP

Type-II HDA (Hard Disk Adapter) for 8in TRS-80s EXPLAINED

by Ian Mavric



HISTORY AND NOMENCLATURE

When the TRS-80 Model II business computer was designed and released in 1979, hard disk drives were still a little while off being miniaturised sufficiently to be used on microcomputers. By 1981 the problems of interfacing the newly designed small (compared to their mini-computer cousins) Shugart 1004 8in 8Mb hard drive had been resolved. The solution for most early computers (no just those from Radio Shack) was: a hard disk interface and controller card, the hard

drive itself, and an updated operating system with drivers to make the hard drive user friendly. Radio Shack accomplished this with the enormous 8 Meg disk system. The hard disk interface eventually became known as the Type-I interface, once a successor was needed and available. Type-I interfaces can only work with the 8 Meg disk system. An interesting aspect of the Type-I interface was an additional 16K Ram was added, more on that later.

After the Model III was released a hard drive was also considered to be an essential peripheral so one was designed. Improvements in technology meant a much smaller Tandon TM-602 5Mb MFM hard drive could be used and the unit itself made smaller and cheaper to produce. The design also allowed easy upgrade to future hard drive capacities of 12Mb, 15Mb, 35Mb and 70Mb as development continued through the 1980s. As the Model II was coming to it's end-of-life a decision was made moving forward that it would be easier to make the Model III hard drive only, and make a new Hard drive adapter for the Model II and all future 8in TRS-80s, appropriately called the Type-II Hard Disk Adapter (or HDA).

The final hard drive controller board (called the Type-4), and this was indeed a full hard drive controller and not an adapter, was designed for system with the 15Mb Tandon TM-503 hard drive mounted internally. This includes the 16B (hard disk version), Tandy 6000HD, and Model 12 systems sufficiently upgraded. The rest of this article will concentrate on the Type-II HDA.

HDA, a hard disk adapter board, talks to a hard drive controller, which is in the TRS-80 hard drive enclosure. A hard disk controller board needs nothing else in the hard drive enclosure other than the MFM hard drive itself (and a power supply (PSU)). We know them as Primary hard drives (key on the front, hard drive controller, MFM hard drive, and PSU) connect to the Type-II HDA. Secondary hard drives (no key on the front, and only an MFM hard drive and PSU inside) connects only to a Type-4 controller as a secondary hard drive to the main hard drive mounted inside the computer.

NOW LET'S LOOK AT THE TYPE-II HDA

The Type-II HDA, while it looks complicated, really does only two things: 1. Connects to a Primary hard drive through a bus that mimics what you find on a Model III/4/4P. 2. Adds 16K of Ram.

The HDA talks to the HDC inside the primary hard drive, and that's how the magic of high speed hard disk storage is achieved. The additional 16K is needed because the aforementioned drivers to control the whole system do take up a decent chunk of memory, and that would be taken away from the 64K present in the Model II. Much software uses all or nearly all of the 64K so would struggle to work on the Model II. Radio Shack's solution: add 16K more and put the hard disk drivers there. The ploy works, and it's also why the Model 12 (the successor to the Model II) came with 80K Ram as standard; RS figuring many 12s would eventually be upgraded with hard drives.

Looking at the HDA you can see the 16K bank on the left side, and the 50-pin bus connector at the top. A couple of 74LS244s just below buffer the 50-pin bus connector. The largest chip, a Z80A CTC is a counter timer used in hard disk operation. The rest of the chips are mainly about interfacing the memory and Mod III bus to the computer. Many of the chip types are common on numerous Model II cards, eg. the Model II 64K Ram card has four LS240s, two 8T26s, two 74157s, an S64, S74, S08, and LS133, among others.

WHEN IT ALL GOES WRONG: NOW IT DOESN'T WORK :-(

It can be daunting looking at the HDA when it doesn't work, where do you start? Fortunately there is a memory diagnostic called MEMII which will help out. It's on the Diagnosics Disk for Model II, keep on eye on eBay, they come up from time to time. Run MEMII and you will see this screen:

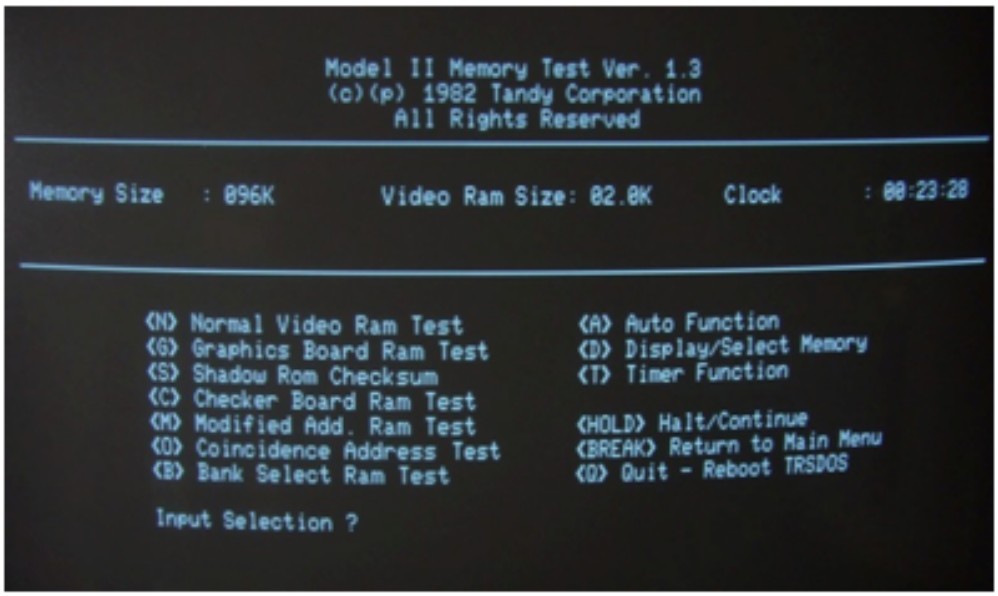


Photo: Notice 96K reported? Normal for the Type-II HDA

The first thing you notice is the memory size. If it reads 96K then the memory on the HDA is being seen. Don't concern yourself with why, but the 16K was mirrored in the design to appear as two 16K banks so the computer reports 96K. If it says 64K then the memory on the HDA has a problem. In my experience with these I do these steps to determine where the problem lies:

1. Check the jumpers are set up correctly: A-B near U19, V-W near U17, A-B near U22. However the important jumper is at the block near U34. On a Model 12 AG-AL need to be jumpered. On all other systems AK-AP need to jumpered. AG-AL disables the 16K because 16K is already installed on the M12 motherboard. AK-AP enables the 16K.

2. Remove the 16K and test the chips individually which a Dram tester, or put them in another 16K computer (like a Model I 16K Level II) and see if that computer runs. Sub out each chip one at a time to determine the failed chip(s).

3. U20 and 26 are 8T26 chips, which are bus transceivers. They buffer the memory and were ancient chips when new and even at the time considered a bit junky, hence they are socketed. If you can't find 8T26 chips to replace them with, use Motorola MC6880 which are the same thing just newer and more robust.

4. Now it gets tricky. I have seen the 74ALS240s fail and they are not socketed. Using a CRO scope around for missing signals. Remember when you find the dead one to replace is with a 74ALS240-1N version. U43 is particularly sensitive to this.

5. Check the DDU at U11. It's job is to properly time the signals on the 4116 DRAM chips. When this fails the 16K memory will

absolutely not work. Unfortunately the Valor DL1993 unit hasn't been available anywhere since the 1980s (trust me, I have tried), so if that has failed a modern replacement (mentioned later) will be needed. BTW if anyone knows of a source of vintage Valor DL1993 DDUs please get in touch with me, they will make life much easier when producing the red repro of the Type-II HDA I make.

With MEMII what we are ideally looking for is a full test using the <A> Auto Function. This test runs all the tests one after another. It takes some 30 minutes to run, and at the end a fully working HDA (from the memory perspective) should give this display:

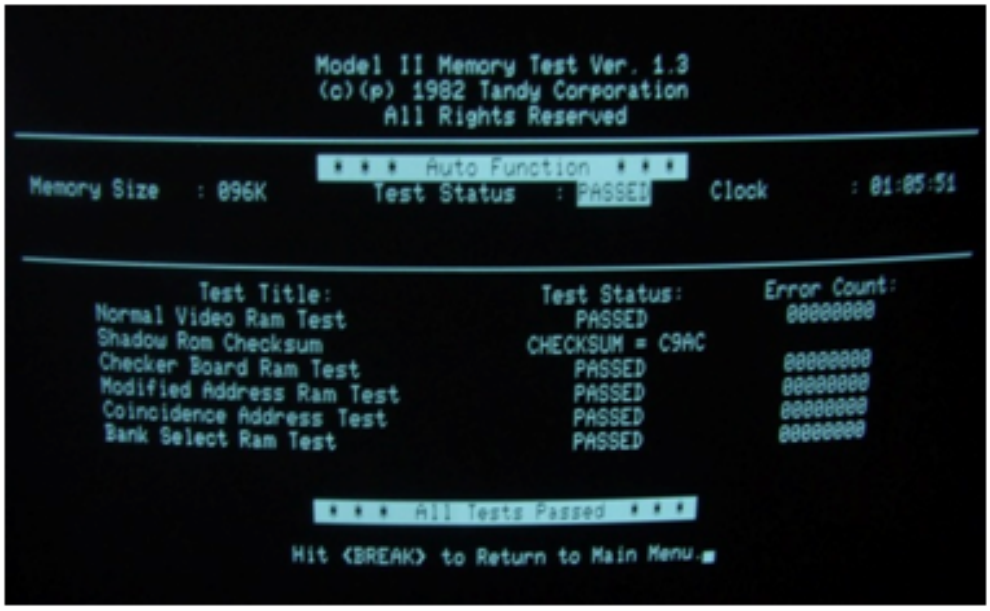


Photo: Completed full test of the 16K Ram section of Type-II HDA

Sometimes however you don't want to run the whole Auto Function test and just want a quick way to see if the 16K is present and behaving. I use <C> Checker Board Ram Test which takes about 30 seconds. Passing this test will mean a 99% chance the HDA will pass

the whole <A> Auto Function test. I use this when I am changing one 4116 DRAM chip at a time to isolate a bad one. A successful checker board ram test screen looks like this:

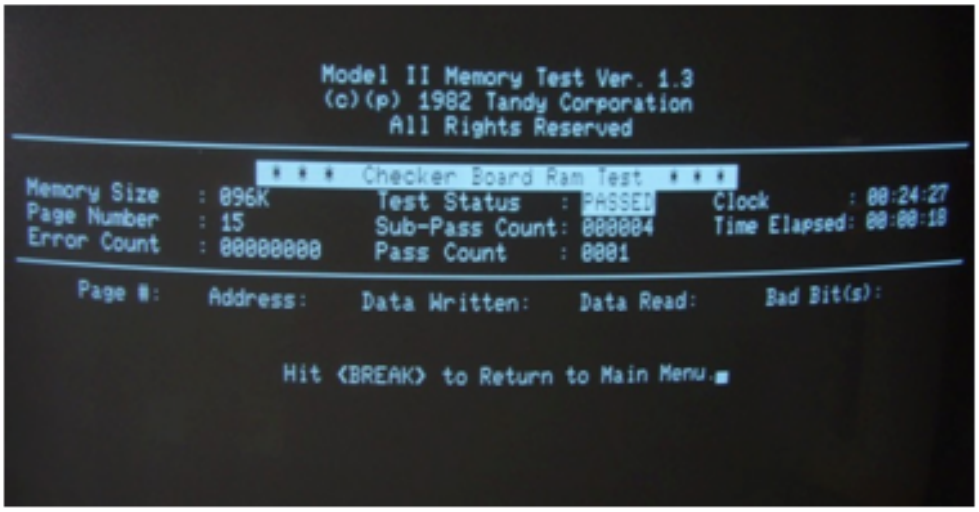


Photo: Successful Checker Board Ram Test only takes 30 seconds

THE HARD DRIVE BUS IS A LITTLE HARDER TO TEST

The connection to the primary hard drive, as stated earlier, is through a 50-pin bus which mimics the expansion bus on a Model III/4. Timing is critical and that is accomplished with a Z80 CTC chip. Fortunately it's socketed and also reasonably easy to procure new even today, so that is the easiest place to start. I have only ever seen one go bad though. (Also, you might think being the biggest chip on the board it has something to do with the 16K memory bank but it has nothing at all to do with it. I ran MEMII with the Z80 CTC removed and it did the whole 30 min <A> Auto Function test without any errors.)

U13 is a DP8304 which is another ancient device going back to the 70s which also hangs off the 50-pin bus. It's also socketed and I have

seen it fail. More likely however are U14 and U15, which are 74LS244 buffers. The have a surprisingly hard and busy life so I find they are just plain worn out often. Being soldered straight to the board you are wise to scope around with a CRO to determine which one (or both) have failed, or if you just like desoldering remove them both, put in some sockets and nice new LS244s. The only other chip I have seen fail related to the 50-pin interface is U9 which is a 74S04. Remember if you decide to replace it to use a 74S04, not LS04. Timing is critical with this chip.

THE REPRO TYPE-II HDA a.k.a "RED BOARD"

So useful is the Type-II HDA that a reproduction had been deemed necessary for some considerable time. With the release of F48, which is an updated version of the FreHD hard drive emulator especially made for 8in TRS-80s, the need for a repro became more urgent. It didn't take long for those who already owned the original HDA to spread the word about F48 and from then when original HDAs came up for sale they commanded high prices on places like eBay.

In 2020 Pascal Holdry from Prof-80 did a repro of the Type-II HDA very faithful to the original. In fact the only point of difference is the DDU mentioned earlier. A substitution using a DS1100Z-100 DDU on a small adapter board works well in place of the DL1993 until we can find a supply of them. All the important fail-point components are socketed on the repro, and it has a nice gold-plated connector for reliability. Where possible, these are build up from new parts, however several parts on the repro Type-II HDA have not been manufactured since the 1980s, so NOS parts have been fitted.

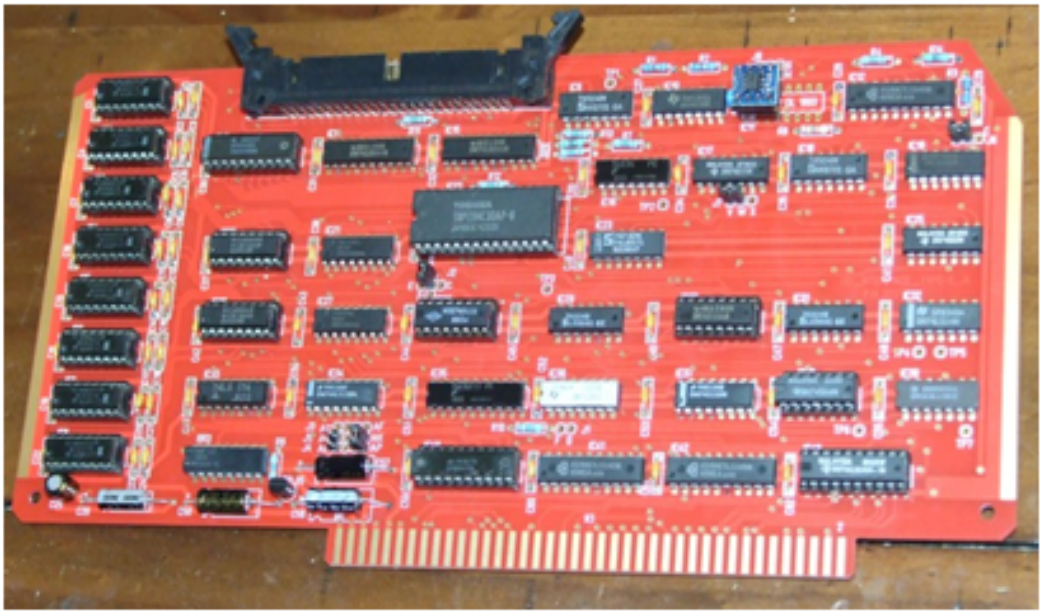


Photo: Repro Type-II HDA is $US199.99 and sells very rapidly

WHY HAS IT TAKEN UNTIL 2024 FOR IT TO BE AVAILABLE?

A glitch in my ability to produce these things appeared soon after I started making my first batch in 2021.. only 2 out of 5 I made would work properly. It took me a while to chase down the problem, which I thought was related to the DDU, but turned out to be that certain chips needed to be of the -1N variety in this application. Once they were identified and changed in 2024 every repro board I have made has worked perfectly.

If you are after one of these I suggest you contact me to place your order sooner rather than later. At the moment I have parts on hand to build around 10 more of these at the current price, after which some of the more exotic components I have on hand will be exhausted, meaning I need to buy them on the open market, adding to the cost of the repro HDA. Get your order in before the price rise in 2025!

A WORD ABOUT THE F48: FreHD FOR 8in TRS-80s

Just a timely reminder that F48 has become the de-facto standard solid state hard drive device for 8in TRS-80s, just like the regular FreHD did for the Model I/III/4. F48, at the time of writing, supports four operating systems for nostalgia and experimentation: TRSDOS-II 4.2.6, CP/M 2.2, CP/M 3.0, and Xenix. If you have a fully functional Model 16, 16B or 6000 this is often the quickest way to get Xenix running on your system. Model II Z80 buffs will get the most use out of TRSDOS-II and the vast array of software for those systems released on 8in floppy disk.

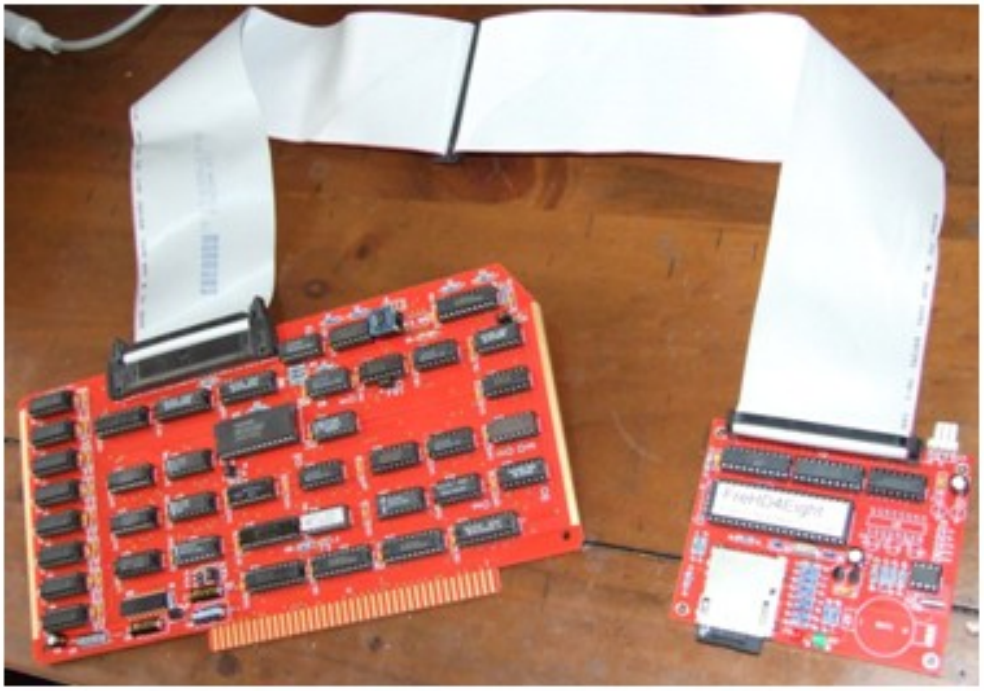


Photo: Type-II HDA coupled with a F48

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.

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)

AT THE READY PROMPT >



I CAME ACROSS SOME BASIC CODE, FROM **BARRY NELSON**, TO FIND IF A NUMBER IS ODD OR EVEN. I THOUGHT IT WAS RATHER NEAT.

```
X = (INT(N/2)=N/2)
```

X WILL EQUAL -1 IF 'N' IS AN EVEN NUMBER OR Ø IF ODD.
REALLY NICE I THOUGH.
THEN, WITHIN A COUPLE OF MINUTES I FOUND THIS BIT OF CODE FROM **RICHARD KELLY**.

```
X = 1 AND N
```

X WILL EQUAL Ø IF N IS EVEN OR 1 IF ODD.

TRULY AMAZING!
DON'T YOU FEEL LIKE SAYING, WHY DIDN'T I THINK OF THAT!
MIND YOU, ONE SLIGHT PROBLEM MIGHT BE PRESENT. SHOULD 'Ø' BE EVEN? SOMETHING ELSE I'VE NOT EVER THOUGHT ABOUT. WHAT DO YOU THINK? AFTER ALL, IT IS AN ODD NUMBER, DON'T YOU THINK?

```
>LIST
0 N=RND(999):X=1ANDN:IFX=0THENPRINTN;"EVEN",:GOTOELSEPRINTN;"ODD
",:GOTO
```

```
READY
>RUN
 296 EVEN        941 ODD         895 ODD         324 EVEN
 291 ODD         618 EVEN        752 EVEN        848 EVEN
 976 EVEN        928 EVEN        153 ODD         733 ODD
 201 ODD         909 ODD         238 EVEN        632 EVEN
 142 EVEN        283 ODD         807 ODD         757 ODD
 412 EVEN        885 ODD         814 EVEN        175 ODD
 918 EVEN        839 ODD         589 ODD         853 ODD
 476 EVEN        471 ODD         830 EVEN        459 ODD
```

Thirsting for adventure?
Brass lantern getting dusty?



Take your Model 1, 3 or 4 on exciting new adventures never available until now....

M4ZVM enables your TRS-80 Model 1,3 or 4 to play Infocom and post-Infocom adventures, taking full avdantage of your hardware. Supports old favourites like Zork and Seastalker (including Sonarscope), as well as new-to-Model-4 games including Trinity, Border Zone and Sherlock, plus hundreds more available in the Interactive Fiction archive.



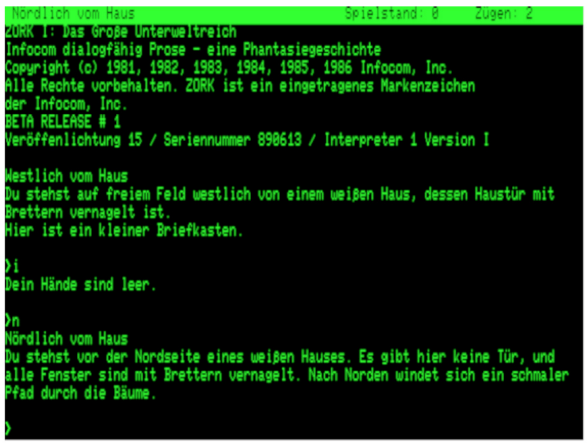
M4ZVM is available for download from https://gitlab.com/sijnstral/m4zvm (includes feature details and source)
Enhanced versions ready for download!

*128K recommended

Actual Screenshots of games new to the model 4, running M4ZVM (on trs80gp)



Beyond Zork



Zork 1 (German)

M4ZVM features include:
* z1-z8 game execution (z6 support is experimental)
* named game save/load
* accented characters
* 128K recommended - NEW Minimised 64K version
* Optionaly set the infamous "Tandy bit"
* NEW Model 1 and Model 3 versions

CoCo Randomization Tune Up

George Phillips

BASIC games with a random element share a common problem. BASIC's random number function `RND()` will give the same sequence when the computer is first turned on or reset. Players will end up playing with the same shuffled deck or other random events each time they start a play session.

Level II BASIC on the Model 1 gives us the `RANDOM` statement to change the seed of the random number generator in a more-or-less random fashion. Color Computer BASIC has no such statement. Instead it has the more general notion that if `RND()` is passed a negative number it will be used as the random number seed. That's great but does leave one wondering where to come up with a random seed that is random. An `A=RND(-3893)` call at the start of the program will have exactly the opposite effect of giving variety to our random numbers.

One reasonable solution is to use the special `TIMER` variable. It continuously cycles from 0 to 65535 stepping up by one every 60th of a second. Since players will take a variable amount of time to type in the commands to load and run the game the `TIMER` value will be different when it runs. A simple `A=RND(-TIMER)` at the start of the game program will give a reasonable variety in random number sequences.

At first glance this should give 65535 different random sequences. William Astle on the CoCo Discord pointed out that BASIC doesn't use the seed value directly. It converts it to a floating point number and then only uses the mantissa. The mantissa of a floating point number is normalized which means that 1, 2, 4, 8 will all result in the same seed. How can we avoid this problem?

It helps to understand normalization but it isn't essential. Skip over this paragraph and the next if you like. For most purposes when we write numbers we'll only use a decimal point when needed. 1, 50, 12354 are fine as they are. Fractions like 1.5 or 0.001 need a decimal point. But when numbers get really big we switch to scientific notation. Instead of 1,000,000,000 we can write 1×10^9 or write 3.7×10^{-4} for 0.00037. That power of 10 is called the exponent and the fractional number is the mantissa. The mantissa is always between -10 and 10. If it were 10 or greater then dividing it by 10 and adding 1 to the exponent gives us the same number. Scientific notation is more precise when the numbers are normalized in this way as there is only one way to write down any number.

Computers use scientific notation when storing floating point numbers except that instead of base 10 they use base 2 (binary). If we have the number 16 (base 10) that's 10000 in binary and can be written as 1×2^4. If we have 18 that's 10010 and is written as 1.001×2^4. 1, 2, 4 and 8 are 1×2^0, 1×2^1, 1×2^2 and 1×2^3. They all have the same mantissa and since BASIC is only using the mantissa they will give the same random number sequence. As a side note, in binary scientific notation the mantissa is always between -2 and 2. So except for 0 (which is treated as a special case) the first digit of the mantissa is always 1. When storing a floating point number the computer will not waste space with the leading digit of the mantissa since it knows it is always 1.

William suggested using `RND(-ABS(SIN(TIMER)))` which would be something like a hash function scrambling the values to give the greatest range of seeds. I speculated that `RND(-TIMER-65536)` would do the trick as the values will all have the same exponent and normalization will have no effect. Another suggestion was to use `RND(-TIMER/65535)`.

I'm sure someone more clever than I could analyze these possibilities and determine whether they solve the problem or not and even how well they do. Luckily there's no need to be clever. For a modern computer running a CoCo emulator the 65535 different values is not a big number. It takes only a few minutes to output the first random number after each seeding technique to a printer and no time at all for a little perl script to count the unique occurrences. The term of art here is "exhaustive testing" which is funny because it may be exhausting for the computer but easiest for us.

I started by writing little BASIC programs to test out each technique.

bad.txt – the technique with known flaws

```
10 POKE 150,1
20 FORI=1TO65535:A=RND(-I):?#-2,RND(0):NEXT
```

sin.txt – use `SIN()` to scramble the values

```
10 POKE 150,1
20 FORI=1TO65535:A=RND(-ABS(SIN(I))):?#-2,RND(0):NEXT
```

div.txt – remap the values with division

```
10 POKE 150,1
20 FORI=1TO65535:A=RND(-I/65535):?#-2,RND(0):NEXT
```

sub.txt – remap the values with subtraction

```
10 POKE 150,1
20 FORI=0TO65535:A=RND(-I-65536):?#-2,RND(0):NEXT
```

I could run each program under trs80gp, capture the printer output and run it through a little perl script to evaluate it:

```
while (<>) {
     $line{$_} = 1;
     $n++;
}
print scalar keys(%line), " of $n\n";
```

For all the files I did:

```
trs80gp -mc -batch -haste file.txt
ren trs80-printer.txt out-file.txt
perl uniq.pl out-file.txt
```

The POKE in each program changes the printer baudrate to the maximum which allows the test to complete faster. `-haste` makes trs80gp run as fast as possible. Mac or linux users can use a similar sequence of commands but will want `mv` instead of `ren`. And maybe even use the native unix `uniq` command. Mine is only one of many approaches.

Here are the results:

Program	Randomizer	Unique seeds
bad.txt	-TIMER	32768
div.txt	-TIMER/65535	32768
sin.txt	-ABS(SIN(TIMER))	61975
sub.txt	-TIMER-65536	65536

The subtraction technique slightly edges out SIN() and, somewhat surprisingly, the division makes no improvement at all. Only half the values give a different sequence.

William was a little concerned that outputting a single random number wouldn't necessarily distinguish the sequence. It is possible for two floating point numbers to differ slightly but appear the same when printed by BASIC. I created sin3.txt to test this idea:

```
10 POKE 150,1
20 FORI=1TO65535:A=RND(-ABS(SIN(I))):?#-2,RND(0);RND(0);RND(0):NEXT
```

And, indeed, it did make a difference. The SIN() technique outputs 61977 unique seeds – two more than discovered before. None of the other results changed.

Not content to leave well enough alone I did some comparisons of the output files to find the two seeds that look the same but are different. They are 7750 with 32727 and 3855 with 39320. This program demonstrates it:

```
10 READ T:IF T=0 THEN END
20 A=RND(-ABS(SIN(T))):?RND(0)RND(0)T:GOTO10
30 DATA 7750,32727,3855,39320,0
```

which outputs:

```
 .620938931   8.6989639E-03   7750
 .620938931   .769241416     32727
 .916264301   .456744844     3855
 .916264301   .963773146     39320
```

Notice how the first random number generated is the same but the second differs.

It is funny that -TIMER generates exactly half of the unique values we hoped. Why is that? I spent more than a little time trying to figure out what values get the same mantissa after normalization but finally saw a simple explanation. Normalization works on the absolute value so for our purposes we only need to consider 0 .. 65535. The process is to multiply the number by 2 until it is >= 32768 (i.e., the high bit is set). If a number N is divisible by two then the normalization of N will be equal to the normalization of N / 2. That is where we lose half our values. Conversely as we normalize an odd value it will either stay the same or become an even number. It can never equal another odd number so all the odd numbers generate unique values. So we will keep exactly half.

Go forth and use A=RND(-TIMER-65536) instead of A=RND(-TIMER).

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

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

The amazing

"FreHD"

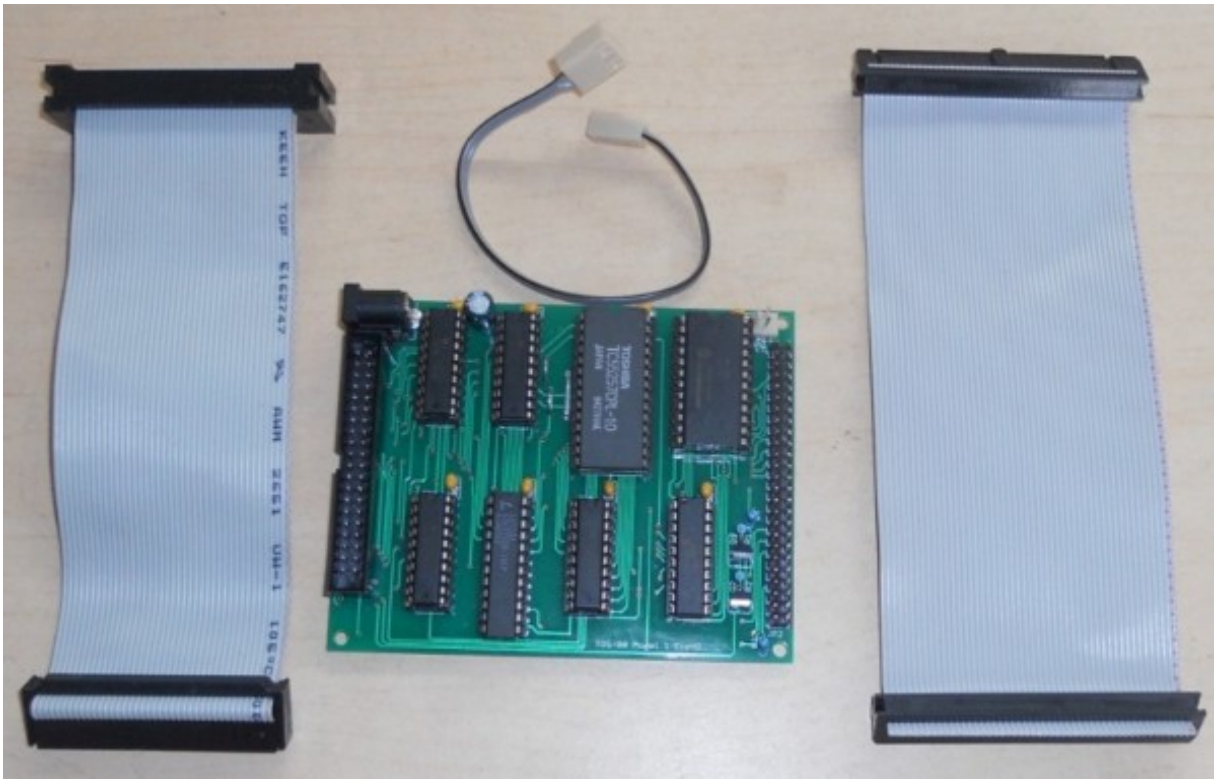


-Emulates a TRS-80 hard drive, but faster than any hard drive!
-Works with your favourite DOS (LS-DOS, LDOS, CP/M, Newdos/80 2.5)
-Uses SD card for storage medium
-Bonus free Real Time Clock function!
-Designed in Belgium and proudly built and shipped from Australia
-Kit form or fully assembled

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.

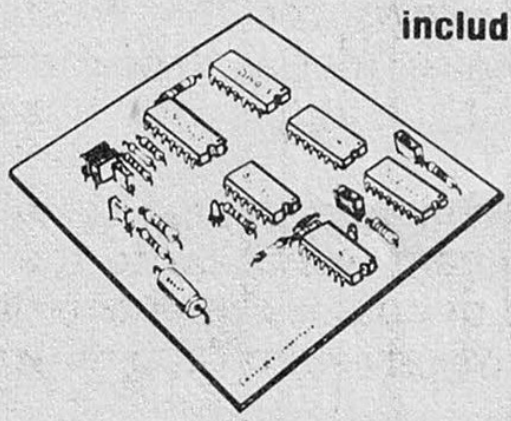
U.K. And Europe, contact Bas. at BetaGamma Computing.
U.S.A. And rest of the world, contact Mav. At “The Right Stuff”

THE WAY WE WERE

1982
1985
1977
1979
1978
1984
1976
1983
1980
1981

LOW COST CASSETTE INTERFACE

includes p.&p.! **Only £ 8.75**



Convert your cassette-recorder into a digitalrecorder. Easy to build, no adjustments! Uses Kansas City Standard. High speed, till 1200 Baud. Full Duplex, which means that you can (if neccessary) use the interface with 2 cassetterecorders at one time. Audio in- and outputs connects directly to cassette-recorder. Data in- en outputs are T.T.L.-compatible. Low Power, only 5 volt needed. Kit includes all parts, heavy epoxy p.c. board with edge connector and clearly written manual in English. Board also available assembled and tested for £ 11,75 incl. p.&p.

How to order: Send is your Barclays, Eurocheque or other remittance in an envelope, put your name in capitals on the back. Kit £8.75 includes P&P. Assembled and tested (guaranteed for 90 days) £11.75 incl. P&P. Send your order to: Musicprint Computer Products b.v., P.O. Box 410, ARKELSE ONDERWEG 31, 4200 AK GORINCHEM, THE NETHERLANDS. Phone 010-31 18 30-24693. Your cassette interface will be shipped the same day we receive your order.

dealer inquiries invited

Stringy Floppy Encore

Last month we reviewed the 6800/S-100 Stringy Floppy. This month—the TRS-80 version.

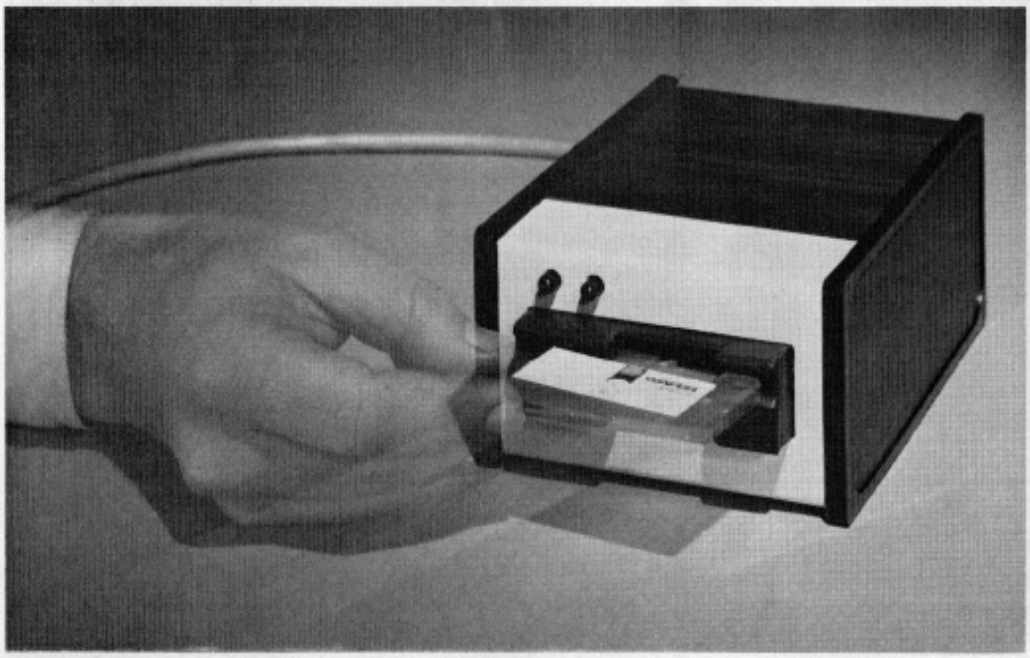
Jim Perry
Microcomputing
Projects Editor

The Exatron Stringy Floppy has been available for some time, for 6800 and S-100 systems, but a version designed for the Radio Shack TRS-80 has only recently been developed. A review, by Bill Harvey, of the 6800 version appeared in the October 1979 issue; this month we review the TRS-80 version, identical mechanically—but even easier to use as all the software needed is in a 2708 EPROM built into the Stringy Floppy itself.

For the benefit of those readers who missed last month's article, a short description of the Stringy Floppy mechanics is in order. The biggest drawback in using any small computer system is the data storage medium, an audio cassette recorder of dubious reliability and performance. If God had intended cassette recorders to store computer data He (or She?) would have given them far greater bandwidth, as the only way to store programs and data on an audio cassette is to do it very slowly—with your fingers crossed. Another alternative is to use a floppy disk unit, which will record and retrieve data orders of magnitude faster than any cassette-based system, but the cost is also orders of magnitude greater. The third way is to use a Stringy Floppy, nearly the speed of a floppy disk at nearly the cost of a cassette recorder.

The basis for the Stringy Floppy is an endless-loop magnetic tape contained in a housing about the size of one of the new micro-sized dictating cassettes (approximately half the size of a normal cassette). This whole unit is called a "wafer"; they come with 5, 10, 20 or even 50 feet of tape inside. Obviously, the longer the tape the more data the wafer can store; with 5 feet of tape, the wafer can hold a 4K BASIC program. Because the device is specifically designed for computer use, it can record information much more densely than an audio system modified for use with computers. Technically inclined readers will be interested to learn that the data is encoded with parity bits and uses a bi-phase self-clocking technique. What this means to nontechnical readers (and all users) is amazing reliability!

Because the unit is designed for a specific purpose it is not suitable for use in the house or office (unlike the normal cassette recorders, which can be used as ordinary cassette recorders), but as a direct result, there are no superfluous controls or knobs—in fact, there are *no* controls or knobs, just two red lights to indicate what the unit is doing. All control is under software instruction, so making mistakes is difficult (how many programs have you saved without pressing the record button?).



The exterior of a Stringy Floppy. *(Photograph courtesy of Exatron)*

Checking It Out

Connection of the Stringy Floppy to the TRS-80 is easy—it just plugs into the back of the keyboard. Multi-way adapter cables are available for systems that need additional devices plugged into the keyboard (screen printers, etc.). After powering-up the system, enter the SYSTEM command, followed by the instruction /12345. In a twinkling of an eye the display comes up with 'EXATRON STRINGY FLOPPY VERSION 3.1', the software for its operation having been patched into the Level II operating system. As far as BASIC is concerned, it now has three extra commands: @NEW, @SAVE and @LOAD.

All the new commands are suffixed with a single digit in the range 1 through 9. This enables selective erasing of old data and

storage of up to nine separate programs per wafer. The @NEW command writes a regular pattern on the wafer, and then reads it back to verify the integrity of the wafer. If a suffix is not used, the software defaults to a 1 and writes over the whole wafer. If a suffix is used, then the file named and all files after it are deleted. For example, on a wafer with eight files, the command @NEW5 erases files 5, 6, 7 and 8.

The @SAVE command is the Stringy Floppy's equivalent of CSAVE—it always needs a suffix digit. When executed, the program in memory is written to tape, then the wafer is read back and compared to the program still in memory . . . automatic verification! One thing to watch is that no file numbers are missed out; this causes the unit to hang up.

If you have three files on a wafer, and want to save a fourth, you must use @SAVE4. For example, if you asked the computer to @SAVE6, with only three files existing, the software would continue to search for the end of file 5—and never find it . . . instant hang-up. As a result you must always keep a record of what files are on each wafer; filing cards are ideal.

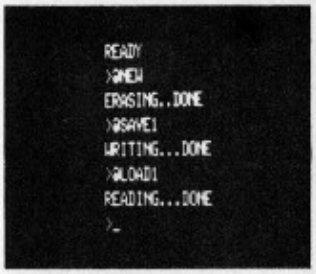
The @LOAD command is the Exatron equivalent of CLOAD, with no buttons to press. If no number is specified, then the next file encountered is loaded; if a file is specified, then the specified file is loaded. If you ask for a nonexistent file, the unit will continue to search as with the @SAVE hang-up. The only way out of both these conditions is to press the BREAK key and try again, so it really is important to keep a record of each wafer's contents.

Both the @SAVE and @LOAD commands use parity bits on each byte of data and a checksum for the entire file. There are only four error messages from the software, but they are spelled out in full. When you use the @LOAD command, PARITY ERROR indicates that one or more bits of data are incorrect—try again. CHECKSUM ERROR also indicates that some of the data is corrupt—try again.

In my six weeks' use of the unit, neither of these messages ever appeared; every load was good. When I used @SAVE, only two messages ever turned up: TAPE TOO SHORT, when a program was far too long for the space left on the wafer, and WRITE PROTECTED, when the reflective disk had been removed to protect a wafer. The reflective disk is the Stringy Floppy's main safeguard against erasure of wanted files, without it the unit will not @SAVE or @NEW the wafer.

Data and Assembly Language

For the intrepid assembly-language programmer, the Stringy Floppy has five operations, found between locations 3000 and 3780 (hex), that can be accessed. These are covered fully in the operating manual that comes with the unit. Briefly, address 3000 is the location of the wind-tape-to-beginning subroutine, 3003 is the start of a read-data routine but needs certain other information specified to read data. 3006 is the routine to write data; two registers must contain data length and load location. Address 3009, when called, will write an end-of-data mark, and 300C is for writing data files onto a wafer. BASIC programs and data cannot be mixed on the same wafer.



The Stringy Floppy sign-on message.



All the commands can be seen in this photograph.



Comparison between the wafer and a conventional cassette.

Wrap-up

If you consider the cost (around $250) and the speed (4K load in seconds), the Exatron Stringy Floppy seems to be a useful addition to the expanding range of TRS-80-related products. In the six weeks that I used the unit, not once did any data/program material misload or become corrupted; reliability was excellent.

The unit comes with a comprehensive instruction manual, one-year warranty and, best of all, a 30-day, unconditional, money-back-if-not-happy guarantee.

Exatron Corporation, 3555 Ryder Street, Santa Clara CA 95051. ■

MEMOREX
Floppy Discs

Lowest prices. **WE WILL NOT BE UNDERSOLD!!** Buy any quantity 1-1000. Visa, Mastercharge accepted. Call free (800)235-4137 for prices and information. All orders sent postage paid.



PACIFIC EXCHANGES
100 Foothill Blvd.
San Luis Obispo, CA
93401. (In Cal. call
(805) 543-1037.)

✓P66

TRS-80™ SOFTWARE



Send for Free Software Info Packet.
* D-Disk System Only * TD-Tape or Disk Level II *

Compress-IT[D] $24.95
2 Programs: Remove Spaces & REM statements. Compresses program into max multiple statement lines. Reduce MEM needs up to 50%.

Simplify-IT[D] $24.95
3 Programs: Lists in single statement lines. Searches & Finds requested text or key commands. Lists all variables used in program.

TEXT[D] $49.95
A Combination Text Processor & ELECTRIC FILING CABINET. WRITE EDIT FILE Reports, Information Files; Auto Directory, DISK store with auto backup options. File searches by Title or Content on 1 to 4 Disk Drives. Formats & Prints. A SELF INDEXING QUERY SYSTEM and Word Processor.

List 'n File: Names & Things[D] $34.95
Create and maintain NAME (customer) files. Sort using any information category. Print lists or labels.

All programs supplied on TAPE or DATA DISKETTE
Indicate your preference and RAM size.

VISA — *Bluebird's Inc.* — master charge
1441 Greenview Ave.
East Lansing, Mi. 48823
Michigan residents add 4% sales tax.

TRS-80™ SOFTWARE



Send for Free Software Info Packet.
* D-Disk System Only * TD-Tape or Disk Level II *

Math/Stat Pac[TD] $39.95
7 Programs: Linear, multiple, polynomial, geometrical, exponential regression analysis. Simultaneous equations, Bargraph with auto axis. Plot any polynomial over any range (Reduce or Enlarge) with auto axis.

DISK[D] $14.95
The HOW TO rudiments of your Disk System. Interactive with lots of dynamic examples. Helps a beginner use the disk system.

Hangman/Hangfile[TD] $19.95
2 Programs: Educational. Create your own dynamic word files. User option multiple clues & subjects with graphics. Really FUN.

Secret Words-Game[TD] $17.95
3 Programs: Educational & Fun & Mind boggling. Guess a 2,3,4,5 letter word. Review & Status options. 3 Versions: Easiest gives lots of clues, hardest no clues.

All programs supplied on TAPE or DATA DISKETTE
Indicate your preference and RAM size.

VISA — *Bluebird's Inc.* — master charge
1441 Greenview Ave.
East Lansing, Mi. 48823 ✓B50
Michigan residents add 4% sales tax.

✓ Reader Service—see page 227

EBAY.CO.UK 'BARGAINS'

HERE ARE JUST SOME OF THE MORE INTERESTING/UNUSUAL TANDY EBAY LISTINGS, SINCE OUR LAST ISSUE.
WERE YOU ONE OF LUCKY ONES?



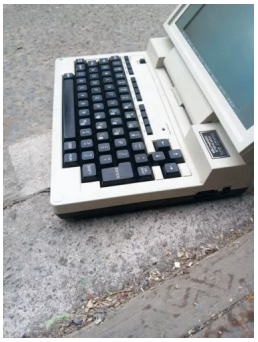
£160



£32



£99
NS



£69



£120



£139



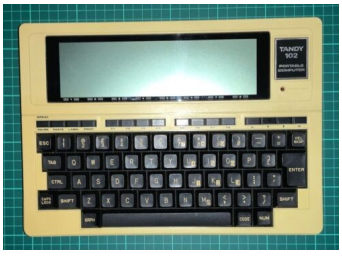
£41



£10
NS



£100



£110

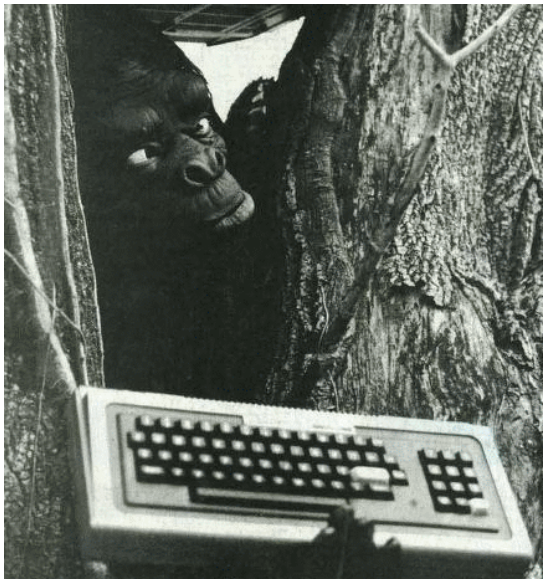


£88



https://www.youtube.com/watch?v=ZE3nMxTLgjk

WHERE ARE THEY NOW ?



James Garon

James Garon, an innovative programmer and influential figure in the early days of personal computing, made significant contributions through his work featured in SoftSide Magazine.

SoftSide was a prominent publication during the late 1970s and early 1980s, catering to enthusiasts of microcomputers such as the TRS-80, Apple II, and Atari 8-bit family. Garon's programs and writings in SoftSide played a crucial role in shaping the home computing revolution.

SoftSide Magazine, launched in 1978, quickly became a favorite among computer hobbyists, offering a variety of articles, tutorials, and program listings that readers could type into their own machines.

James Garon emerged as a prolific contributor, known for his creativity and technical expertise. His programs often combined entertainment with education, showcasing the potential of these early computers.
One of Garon's earliest contributions was his game "Android Nim," which appeared in the June 1979 issue of SoftSide. "Android Nim" was a digital adaptation of the classic mathematical game of Nim, where players take turns removing objects from distinct piles.
The game was designed for the TRS-80 and highlighted Garon's ability to transform traditional games into engaging computer programs. His version added a science fiction twist, with players competing against the computer.

The program was well-received, praised for its clever use of graphics and demonstrated Garon's talent for making educational content enjoyable.

In addition to games, Garon also wrote many utilities that showcased his versatility as a programmer. For instance, his "Super Disk Bonus" program, provided users with a set of tools to manage their floppy disks more efficiently. This utility was particularly valuable at a time when storage space was limited and expensive. "Super Disk Bonus" included features such as disk formatting, file copying, and sector editing, offering a comprehensive suite of tools that enhanced the functionality of the TRS -80.

Another remarkable program by Garon was "Biorhythm. This program calculated and plotted biorhythms, which were believed to predict various aspects of a person's physical, emotional, and intellectual cycles based on their birth date. While the scientific basis of biorhythms was debatable, the program was popular among readers and demonstrated Garon's knack for combining fun with a touch of pseudoscience. "Biorhythm" was user-friendly and visually appealing, making complex calculations accessible to non-technical users.

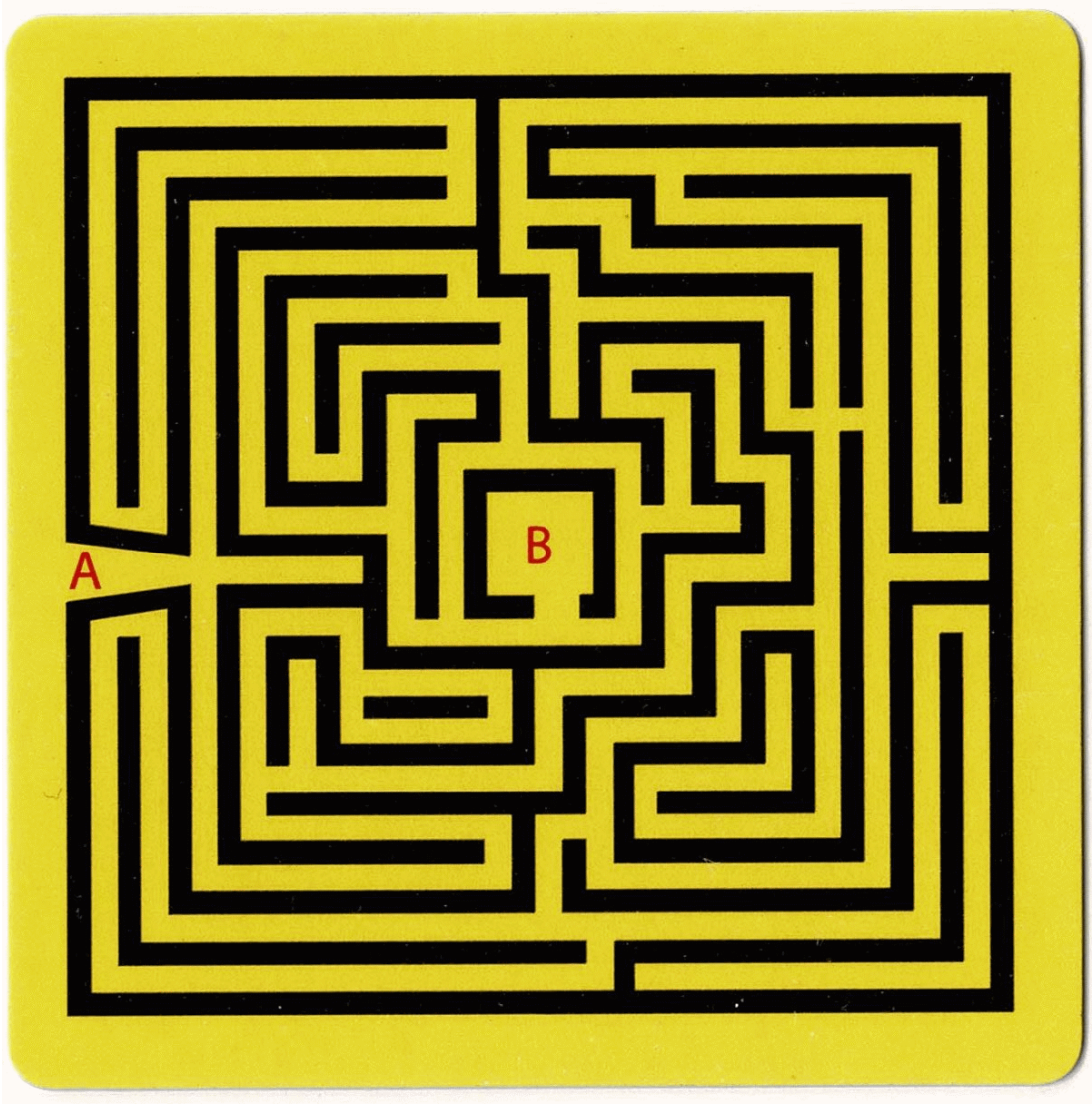
Garon's contributions to SoftSide were not limited to individual programs. He frequently wrote articles explaining the inner workings of his code, providing readers with valuable insights into programming techniques and logic with his "Garon's GOODIES" series. These articles served as educational resources, empowering readers to modify existing programs or create their own. Garon's clear and engaging writing style made complex concepts understandable, fostering a community of budding programmers. He was also a keen 'crossword' designer, providing many for the publication.

The impact of James Garon's work in SoftSide Magazine extended beyond the programs themselves. He inspired a generation of computer enthusiasts to explore the possibilities of their machines, sparking creativity and innovation. His contributions helped demystify programming, showing that with a bit of imagination and effort, anyone could harness the power of their microcomputer.

In summary, James Garon's programs and articles in SoftSide Magazine were instrumental in the early days of personal computing. Through his engaging games, practical utilities, and educational writings, he left a lasting legacy that influenced countless hobbyists and future professionals in the field. His work exemplified the spirit of experimentation and learning that defined the home computing revolution.

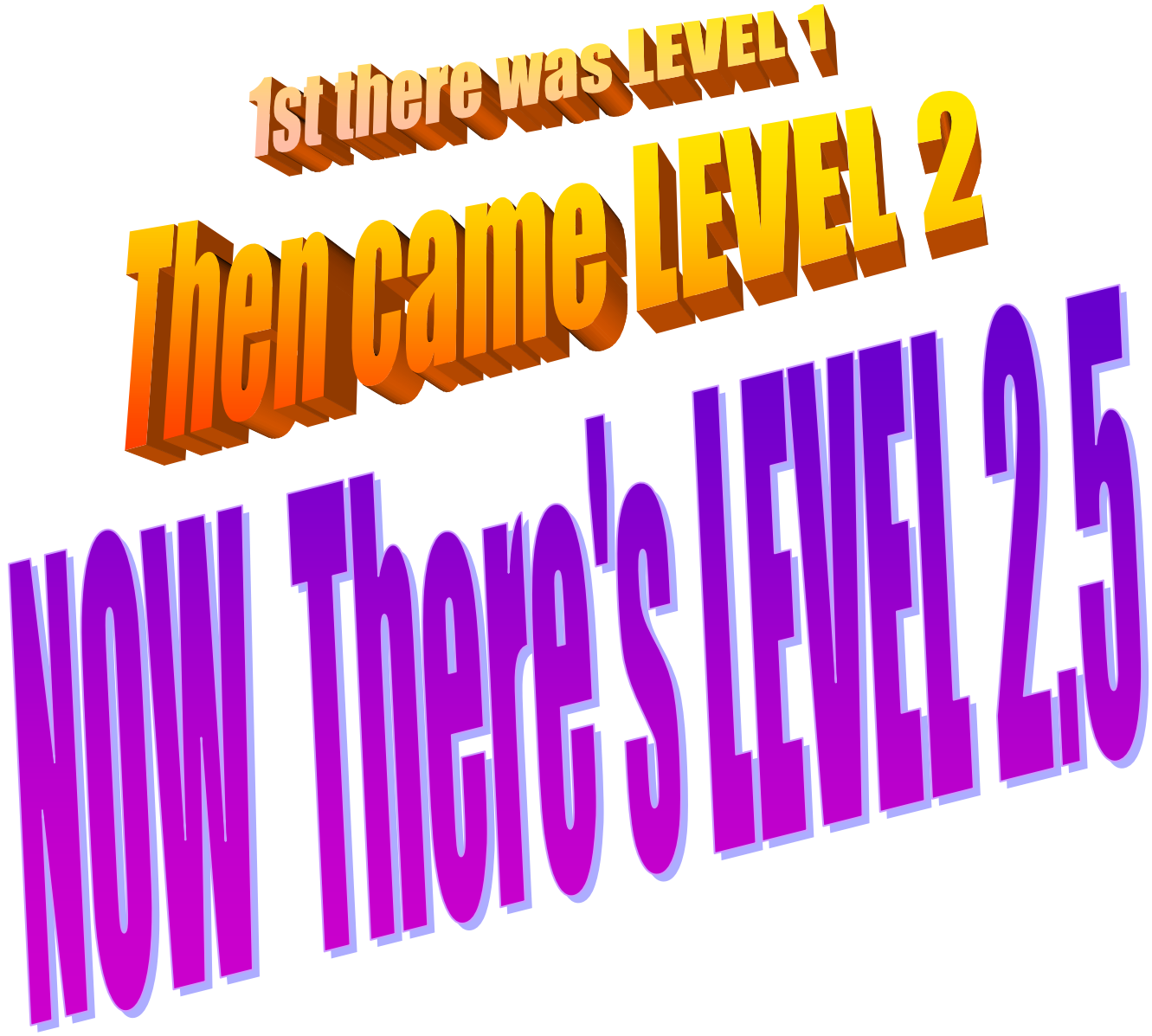
MY APOLOGIES FOR THE PHOTO! JAMES WAS THE PERSON IN THE 'GUESS WHO'S IN THE GORILLA SUIT' COMPETITION AND IT SEEMS TO BE THE ONLY PHOTO I COULD FIND OF HIM. THANKS TO EVERYONE WHO HELPED IN THIS ENDEAVOUR - ED

AMAZED



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 HOW LONG IT TAKES TO GET FROM A TO B, FOLLOWING THE YELLOW ROAD.



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



E.T. FONEHUME

HI DUSTY. THE OTHER NIGHT, IN BED, I SUDDENLY STARTED THINKING (LORD KNOWS WHY!) ABOUT CHECK DIGITS AND HOW THE GIRLS IN THE PUNCH ROOM HATED NUMBERS PROTECTED WITH A MOD 11. (I'VE JUST REALISED, WHILE TYPING, THAT THIS WAS NEARLY 50 YEARS AGO!)

THEY WERE AMAZINGLY FAST, USING THEIR LEFT HAND TO TURN OVER PILES AND PILES OF CREDIT OR PAYMENT SLIPS AND THEIR RIGHT HAND TO OPERATE A NUMERIC KEYPAD. BUT WHEN THE 'MOD 11 X' APPEARED, THEY THEN HAD TO MOVE THEIR RIGHT HAND OVER TO THE 'NORMAL' KEYBOARD TO HIT THAT PESKY 'X' AND THEN MOVE BACK TO THE NUMERIC KEYPAD WHICH UPSET THEIR NORMAL RHYTHM. THEY ALWAYS SAID IT WAS MOST ANNOYING AND REDUCED THEIR OPERATING SPEED NOTICEABLY.

I SEEM TO REMEMBER THAT EITHER YOU OR ONE OF THE ENGINEERS MADE A SOFTWARE PATCH SO THAT THE 'STAR KEY' ON THE NUMERIC KEYPAD COULD BE USED INSTEAD OF THE MAIN KEYBOARD'S 'X'.

ANYWAY, AS I WASN'T FEELING SLEEPY BY THEN, I DECIDED TO CHECK OUT CHATGPT AND SEE WHAT IT SAID ABOUT CHECK DIGITS. I FOUND THAT IT TURNED OUT A BIT MORE INTERESTING THAT I FIRST THOUGHT AND NEEDLESS TO SAY I JUST HAD TO WRITE A COUPLE OF ONE-LINERS TO CREATE AND 'CHECK' MOD10 CHECK DIGITS!

Understanding Check Digits: A Vital Component of Data Validation

The Mod10 check digit, also known as the Luhn algorithm or Luhn formula, is a simple yet effective checksum formula used to validate a variety of identification numbers. It was developed by German computer scientist Hans Peter Luhn in 1954. The primary purpose of this algorithm is to detect errors in sequences of numbers, such as mistyped or transposed digits, which can commonly occur during data entry processes.

How Mod10 Check Digits Work
The Mod10 check digit method involves a straightforward calculation. Here's a step-by-step breakdown of the process:

Reverse the Digits: Begin by reversing the order of the digits in the number you want to validate, excluding the check digit itself.

Double Every Second Digit: Starting from the left, double the value of every second digit. If doubling a number results in a value greater than 9, add the two digits of the resulting number together (e.g., 7 becomes 14, which then becomes 1 + 4 = 5).

Sum All Digits: Sum all the digits from the previous step, including those that were not doubled.

Calculate the Modulo 10: The total sum is then taken modulo 10 (i.e., the remainder when the sum is divided by 10).

Determine the Check Digit: The check digit is the amount needed to make the total sum a multiple of 10. If the modulo result is 0, the check digit is also 0.

For example, consider the number 7992739871 (where 1 is the check digit). When reversed and processed through the steps outlined above, the sum of the digits yields a result of 67. Since the last digit of 67 is 7, subtracting it from 10 gives the check digit 3, matching the provided check digit, thus validating the number.

Applications of the Mod10 Check Digit
Mod10 check digits are widely used in various industries for validating data. Some common applications include:

Credit Card Numbers: Almost all credit and debit card numbers use the Mod10 algorithm to ensure that the number entered is valid. This helps in reducing the occurrence of erroneous transactions caused by incorrect card number entry.

Identification Numbers: Several identification systems, such as those for social security, tax identification, and some barcodes, implement Mod10 check digits to validate the integrity of the entered data.

International Standard Book Numbers (ISBN): Although the ISBN-13 format predominantly uses a different algorithm, the older ISBN-10 format employs a variant of the Mod10 algorithm, specifically designed for error checking.

Benefits and Limitations

The primary advantage of the Mod10 check digit system is its simplicity and ease of implementation. It can quickly validate large sets of numbers without requiring extensive computational resources. This efficiency makes it particularly valuable in automated systems, where speed and accuracy are critical.

However, the Mod10 algorithm is not foolproof. While it is effective at detecting simple errors like single-digit mistakes and adjacent digit transpositions, it may not catch more complex errors, such as multiple digit transpositions or random digit changes. As a result, while it adds a layer of security and accuracy, it should not be solely relied upon for critical systems where higher data integrity is required.

Conclusion

In summary, Mod10 check digits serve as a vital tool in data validation, providing an efficient means to detect and prevent common errors in numerical data entry. While it has its limitations, the Mod10 algorithm remains a cornerstone in the fields of finance, identification, and beyond, ensuring accuracy and reliability in everyday transactions and data processing.

Exploring Mod11 Check Digits: Ensuring Data Accuracy and Integrity

The Mod11 check digit system is a versatile and widely used method for verifying the accuracy of numerical sequences. Unlike the Mod10 system, which is often used for simple checksums, Mod11 is employed in more complex data validation scenarios. It provides a robust mechanism for detecting a range of common errors, making it a valuable tool in various fields including banking, government identification, and library systems.

Understanding the Mod11 Algorithm

The Mod11 algorithm involves calculating a check digit that is appended to the end of a number sequence. This check digit helps to verify the correctness of the entire sequence. The calculation process is as follows:

Assign Weights: Assign a set of weights to each digit in the number sequence. Commonly, weights start from a specific number and decrease in a linear fashion. For example, if we have a number sequence of six digits, the weights might be 6, 5, 4, 3, 2, 1.

Multiply and Sum: Multiply each digit by its corresponding weight and sum the results.

Compute Modulo 11: The total sum is then taken modulo 11 (i.e., the remainder when the sum is divided by 11).

Determine the Check Digit: The check digit is determined by subtracting the modulo result from 11. If the result is 10, the check digit is often represented by an 'X' (in the case of systems like ISBN-10) or adjusted according to specific rules. If the result is 11, the check digit becomes 0.

For instance, to calculate the check digit for the number 123456, you might assign weights 6 through 1 to each digit. The weighted sum would be
$1\times6+2\times5+3\times4+4\times3+5\times2+6\times1=21+10+12+12+10+6 =71$.
The modulo 11 of 71 is 5, so the check digit is
11-5=6. Thus, the full number with the check digit is 1234566.

Applications of Mod11 Check Digits
The Mod11 system is particularly useful in environments where data integrity is critical. Some of the most common applications include:

International Standard Book Numbers (ISBN-10): The ISBN-10 system uses Mod11 to verify the accuracy of book identifiers. In this context, the check digit can be an 'X', representing the value 10.

Bank Account Numbers: Many banking systems use Mod11 to validate account numbers, helping to prevent errors in transactions caused by mistyped account details.

National Identification Numbers: Various countries employ Mod11 in their national ID numbers to ensure the authenticity and accuracy of identification data.

Vehicle Identification Numbers (VINs): In some regions, VINs use Mod11 to ensure the correctness of the code, which helps in tracking vehicles and preventing fraud.

Benefits and Considerations
The Mod11 check digit system is robust in detecting common data entry errors, such as single-digit mistakes and some transpositions. This makes it a preferred choice in systems where the consequences of errors can be significant, such as financial transactions and personal identification.

However, Mod11 is not infallible. It may not catch more sophisticated errors, such as when multiple digits are transposed or altered in a way that does not change the weighted sum. Therefore, while it provides a useful layer of error detection, it is often used in conjunction with other verification methods to enhance overall security and data integrity.

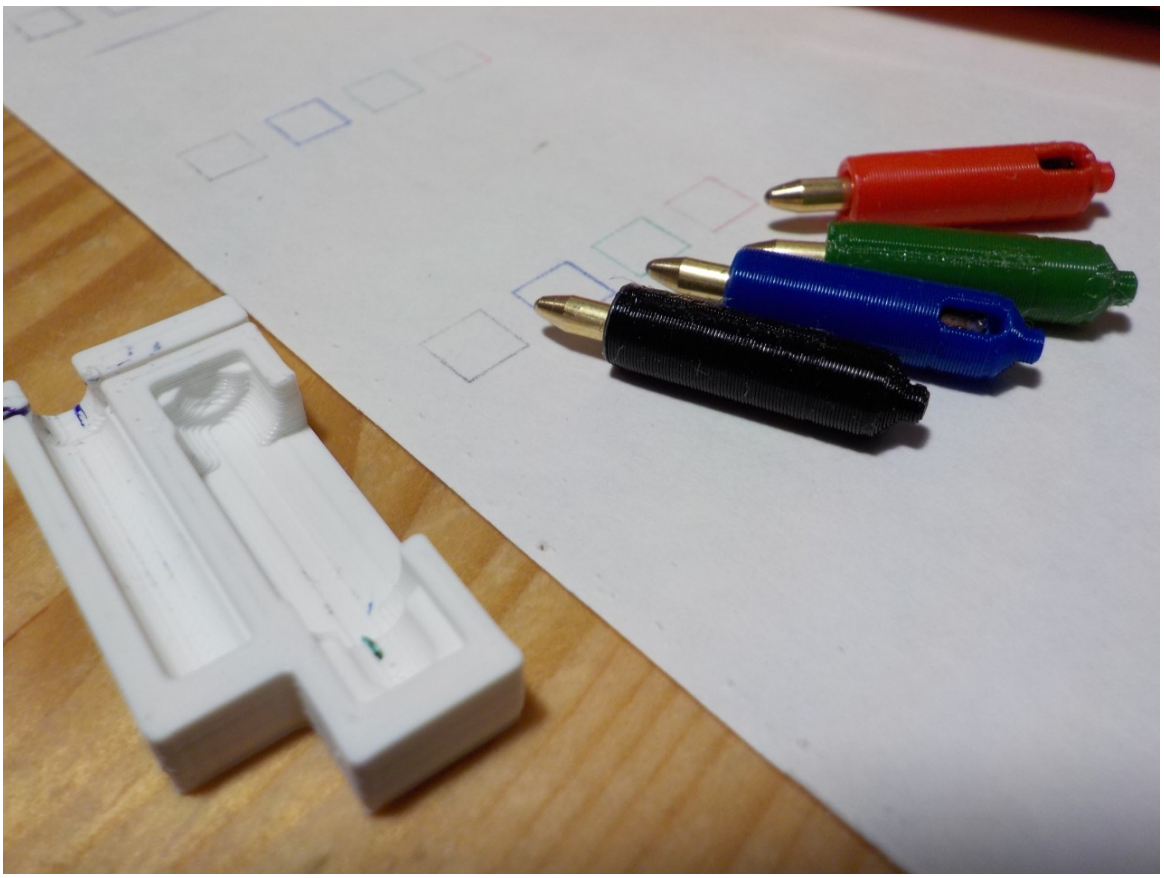
Conclusion
In summary, Mod11 check digits are a critical component in maintaining data accuracy and integrity across various systems. Their ability to detect a wide range of errors makes them invaluable in fields that handle sensitive or high-stakes information. By understanding and implementing Mod11, organizations can significantly reduce the risk of errors in their numerical data, ensuring smoother and more reliable operations.

```
0 CD=0:INPUT N$:FOR X=1TOLEN(N$)STEP2:CD=CD+VAL(MID$(N$,X,1)):NE
XT:CD=CD*7:PRINT" ";:FOR X=2TOLEN(N$)STEP2:CD=CD+VAL(MID$(N$,X,1
)):NEXT:IFINT(CD/10)*10=CDTHENCD=0:PRINTCDELSECD=INT(CD/10+1)*10
-CD:PRINT"**";CD;"**":PRINT:GOTO
2 CD=0:INPUTN$:Y=LEN(N$):C=VAL(MID$(N$,Y,1)):FORX=1TOY-1STEP2:CD
=CD+VAL(MID$(N$,X,1)):NEXT:CD=CD*7:FOR X=2TOY-1STEP2:CD=CD+VAL(M
ID$(N$,X,1)):NEXT:CD=INT(CD/10+1)*10-CD:CC=CD-C:IFCC=10ORCC=0THE
NPRINT"OK":GOTO2ELSE PRINT"ERROR!":GOTO2
```

THANKS FOR THAT E.T. I HAD FORGOTTEN ABOUT THE 'X' PROBLEM. YES, THOSE PUNCH GIRLS WERE REMARKABLY FAST AT DATA ENTRY AND COULD STICK AT IT FOR HOURS ON END.
I ALSO REMEMBER THAT THEY WERE ALSO VERY USEFUL WHEN THEY WERE PUNCHING COBOL PROGRAMS TO TAPE. THEY WOULD QUITE OFTEN CONTACT THE PROGRAMMER POINTING OUT CODING ERRORS IN THEIR DATA SHEETS TOO!!

HAPPY DAYS EH!

Surfin' The Net



ALPS PLOTTER PEN REPLACEMENTS KIT (SOON WITH CAD FILES)
3D-PRINTABLE HOLDER AND CUTTING KIT TO REPLACE ALPS PLOTTER PENS, WHICH ARE BECOMING RARE AND EXPENSIVE. THESE ALLOW USING CHEAP BIC 4-COLORS BALL-PENS INSTEAD. FOR USE IN MCP-40 (ORIC, SEGA/YENO.).AND OTHER SIMILAR PRINTERS.

IT SEEMS SOME PEOPLE ALREADY TRIED PREVIOUSLY, BUT THE ONLY ATTEMPT I FOUND WAS NOT FUNCTIONAL. SO I DESIGNED A COMPLETE ONE, AND AFTER SEVERAL ITERATIONS WE SHOULD SOON HAVE A USABLE ALTERNATIVE TO OVERPRICED NEW OLD STOCK PENS.

DESIGNED IN OPENSCAD.

LICENSE ISN'T CHOSEN YET.

https://github.com/mmuman/ALPS-plotter-pen-replacement

ALPS plotter pen replacement kit User manual

For use with MCP-40 & other compatible printers

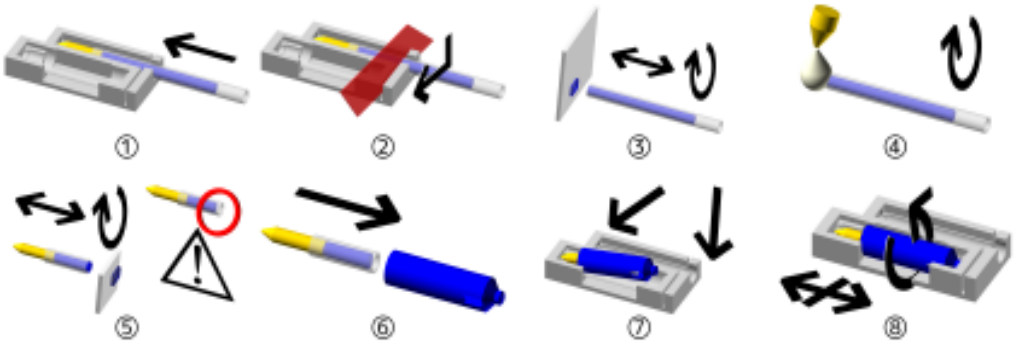
François Revol

Description

This 3D-printed kit allows using *Bic® 4-colors* pens or recharges as stylus for ALPS-based printers like the MCP-40.

It compries four color holders, two extra holders (in case one breaks, or for other colors), a cutting jig, a bent paperclip to help extract the ink tube from the holder.

Usage



Prerequisites

- cutter or exoknife or other sharp blade
- hot-glue gun to seal the remainder of the ink tube
- gloves or a good soap for the ink spills
- paper tissues or wipes

Extracting empty ink tube from the holder

When the installed ink tube is empty, use the bent paperclip as a lever into the side hole to push the ink tube outwards.

Note: It is possible to use pliers on the metallic part, but you risk separating it from the ink tube and spilling a lot of ink.

Scavenging ink tubes from a complete 4-colors pen

Unscrew the bottom part (usually blue), and pull on the transparent ink tubes to detach them from the top of the pen.

Setup from complete ink tube

① Insert the ink tube by the side hole on the cutting jig, ball first. The ball must touch the other end of the gutter. ② Using a sharp blade, cut the ink tube at the slit.

Note: The slit being larger than the required precision for the assembly (1mm), it is advised to do the first attempts aligning your blade on the exterior side on the slit, and extract the ink tube and re-cut it if too long. If the ink tube is cut too short, it risks going too far into the holder regardless the friction, and not print at all, and result in a too short stylus. But if the assembly is too long, it risks blocking the mechanism and damaging the printer.

③ Remove the protruding part of the tube, wipe off the ink from the border of the tube. ④ Now seal the cut end of the tube with the hot glue gun, only this side. A pearl-sized drop is enough, it is only required to have an air-tight seal. Do not seal the other end of the tube, doing so would risk leaking the ink from the other side.

⑤ Remove the useful part from the cutting jig, the tube part with the ball end, and wipe off enough ink from the border of the tube so it doesn't leak, about a millimeter, and wipe the sides of the tube as well.

⑥ Progressively push the cleaned tube inside the holder, the ball point on the outside. ⑦ When the total length of the assembly approaches the size of the test spot on the cutting jig, put it inside, with the ball first, and push again to adjust it, until it fits correctly. ⑧ It must be able to turn around inside, but not move along the axis. If it doesn't fit, remove the ink tube from the holder (see above), and retry assembly after cutting the tube shorter.

Refill from cut ink tube leftover

(**untested**) It is possible to reuse the ball assembly and the rest of the unused ink tube for refill several times.

Remove the metallic part from the empty tube after disassembly.

Carefully remove the hot glue cap from the unused ink tube, and if needed trim the end to have the ink right at the end of it.

Fit the metallic part at the end of the tube, and wipe off the excess ink around it.

Then proceed to setup as with a complete ink tube.

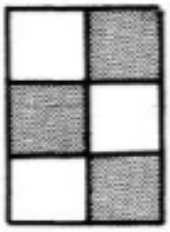
TRS-80 PROGRAMMING HINT

Suppose while writing a program, you want to display some specific graphics. How can you determine how to find them? Here's a way to figure them out:

Number the square with binary decimal equivalents as shown in the diagram at right. Find the decimal number equivalent to the graphic desired; add this number to 128. This gives you the CHR$[-] needed to produce your desired result.

1	2
4	8
16	32

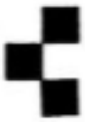
EXAMPLE:



2 + 4 + 32 = 38

plus 128

CHR$(166) **results in**





IT IS NOT A DETAILED REPRODUCTION OF THE CASE, BUT IT PROVIDES A GENERAL TRS-80 FEEL AND IS DEFINITELY RECOGNIZABLE. THERE IS A RECESS THAT HOLDS A WIRELESS KEYBOARD/TRACKPAD. THERE ARE LEDS IN THE FAUX FLOPPY DRIVES AND AT THE MOMENT THEY ARE ALWAYS ON (AS LONG AS THE POWER SWITCH IS ON). I PLAN TO MAKE THEM BLINK SOMEWHAT RANDOMLY WITH A SMALL SCRIPT RUNNING ON THE PI. IT WOULD BE REALLY GREAT IF TRS80GP HAD A WAY OF SIGNALING DISK ACTIVITY TO AN OUTSIDE PROCESS. IF IT DID, I WOULD SYNCHRONIZE THE LEDS WITH THAT. ANYONE KNOW IF THAT'S POSSIBLE?

I WILL BE RELEASING THE STLS SOON (NEXT COUPLE OF WEEKS), BUT I HAVE ANOTHER ROUND OF PRINTING TO DO TO ENSURE THAT THE LAST ROUND OF TWEAKS DIDN'T INTRODUCE ANY PROBLEMS.

https://youtu.be/sNZiHpACEHM?t=17



https://www.youtube.com/watch?v=OfundIpIgQY

THE PARADE OF MODERN HARDWARE FOR THE TRS-80 MODEL II/12/16/16B/6000 SYSTEMS CONTINUES WITH THE TANDY EMERITUS KEYBOARD ADAPTER! CHRIS SHOW OFF THE ADAPTER AND ENABLES VIRTUAL CONSOLES IN XENIX 3.3. HE ALSO GIVES A SHORT RECAP OF HIS TIME AT VCF SOUTHWEST 2024. FINALLY, THERE'S A LITTLE SHOW-AND-TELL OF HIS TANDY DT-100 TERMINAL.

Jim McGinley has start up a GAMES I USED TO PLAY webpage.

The site uses TRS80GP to highlight 30 of his favourite TRS-80 games (6 so far are posted). A snapshot feature is present which allows the games to be saved and loaded, so people can start at higher levels.

GAMES I USED TO PLAY

1 OUTHOUSE
2 DEFENSECOMMAND
3 CRAZYPAINTER
4 SCARFMAN
5 PENETRATOR
6 DANCINGDEMON

CONTROLS

Esc	Break
\	Clear
Ctrl+P	Pause
Ctrl+S	Save
Ctrl+L	Load
Ctrl+C	Copy
Ctrl+V	Paste

WELCOME

Greetings you absolute weirdo. My name is Jim McGinley and I played these games 42 years ago. Click a game, watch the video, **then click a level to play right now**! A TRS-80 Model III emulated in your browser? Truly it's 2024! Most games start boring so I recommend the higher levels. I love these games, and now you'll love them - all 16 kilobytes.

You can reload your game (**Ctrl+L**) and save (**Ctrl+S**) - 3 slots per game. You can copy a game (**Ctrl+C**), paste it into an e-mail (**Ctrl+V**) and send it to a friend. They can paste it into this site and continue **your** game! Should you reach higher levels or find interesting moments e-mail them to me (jim@bigpants.ca) for main menu consideration.

Consider buying Endlight, our chaotic action game with a TRS-80 heart.

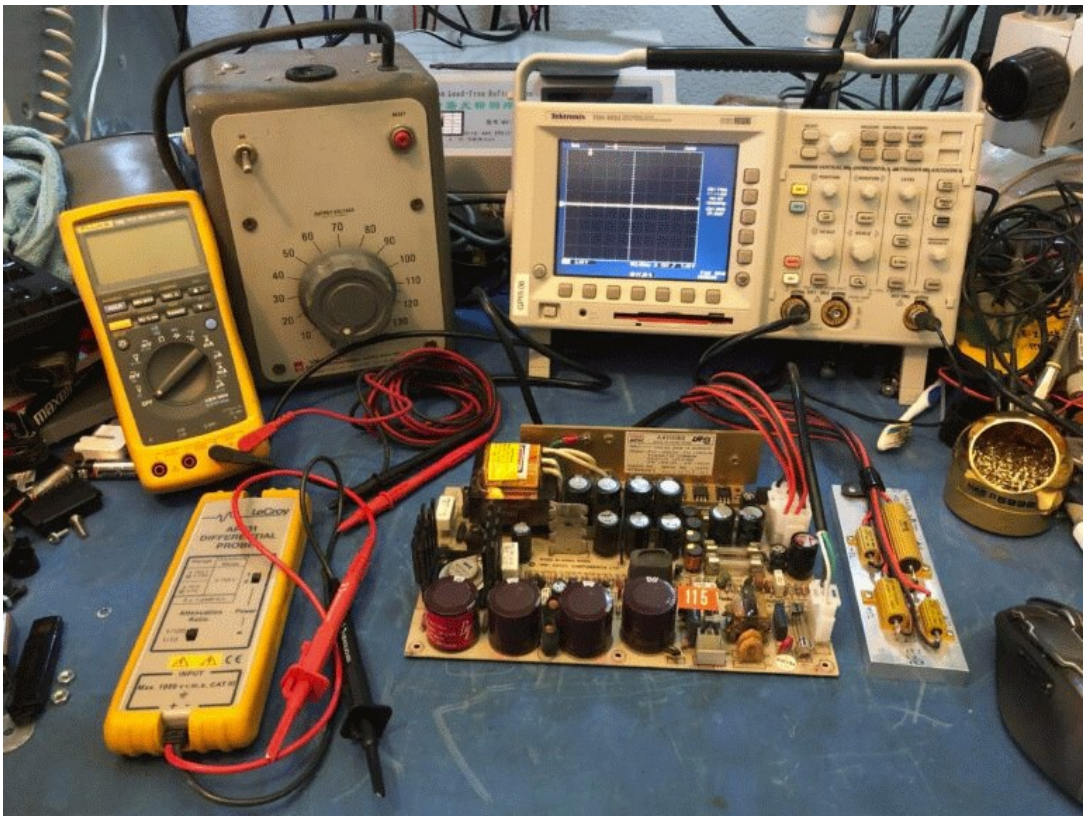
NEWS
ABOUT
CONTROLS
RESOLUTION
HISTORY
CHALLENGES

MARK'S TECH JOURNAL

MARK J. BLAIR'S BLOG OF TECHNICAL STUFF

Tandy 6000 Power Supply Replacement





My FreHD Update Story

Mark Casteleijn

To quote "Hitch Hikers Guide to the Galaxy" – The story so far.....

Opening Comment: The **"FreHD"** is an amazing machine! Frederic and all the other contributors have my congratulations and thanks, as does Ian for his excellently built units!

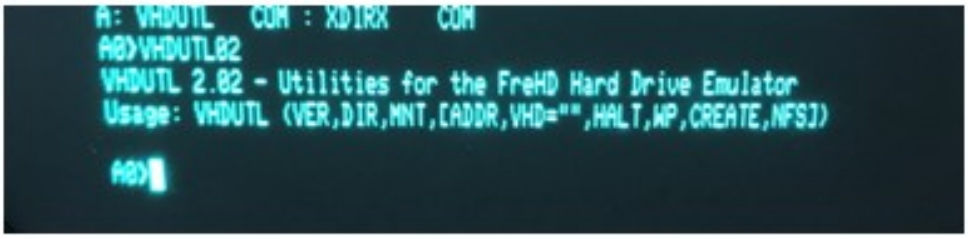
Part One – All FreHD Utilities (tools) have been updated and bug fixed (V 2.05)

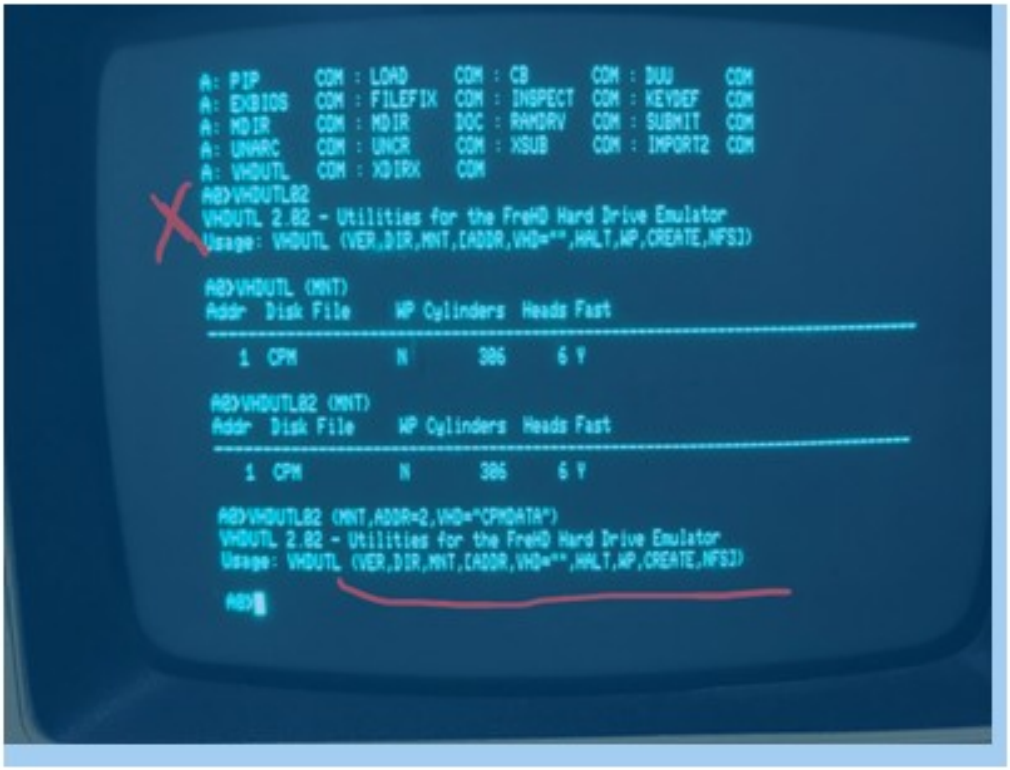
My little project started out with a very simple goal! To locate the "FreHD" CP/M utilities that "I assumed" were just missing from the "stock" CP/M image that came with my **FreHD**.

I found that:

1, There was no way to "import" to the CP/M image (no 'import2.com').

2, The version of 'vhdutl.com' (v2.02) could "directory" images on the SD Card (DIR) & "list" the mounted images (MNT) but it was "NOT" able to mount them.

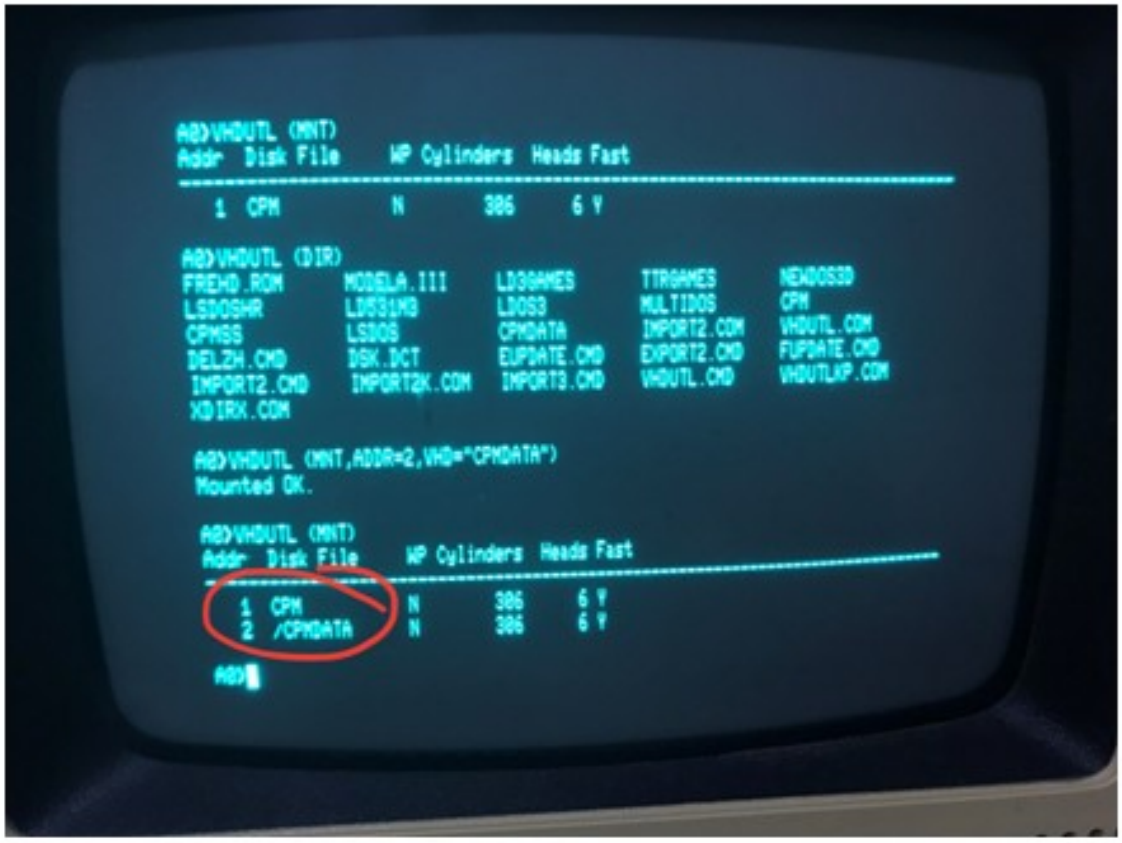




FreHD – My Updates Story – Part One

After this I spent a great deal of time - both directly and indirectly – working with many members of the "Tandy" discord (What a great group of people), with Matt (@maboytim) in particular. With their "great" assistance I have "curated" multiple bugfixes and code corrections into the codebase for the FreHD Utilities for all the stock OS'es. All the bugfixes have been noted / commented in the code base and referenced to the contributor.

In short, these corrected the issues I was having with the CP/M utilities and many others.



VHDUTL.COM – Mounts drive images correctly now under CP/M

A Note on Version Numbers – (firmware) This is jumping ahead a little, but I thought it best to cover it now – in brief – there was in the past a great deal of ***"grief"*** in relation to version numbers and the confusion that it caused. I do hope I have not added to this, in fact my goal is to help get everyone back on an even keel. A "tiny" summary of the back story – I don't know the exact timing here or full details but, suffice it to say – at some point after firmware **2.13** additional hardware support was added and this may have inadvertently caused some issues in its operation on the TRS-80 and – may have - caused "errors" in compiling it from source. One of the releases of *2.14* would appear to have been the culprit here! The successfully compiled version that came after this is **2.15** and will be found on "most" *pre-built* units. General opinion is that **2.13** is **"rock"** solid, so if you are not affected by any issues and **"do not"** wish or need to compile your **"own"** firmware – feel confident in sticking with **(v2.13)** or **(v2.15)**.

(Utilities) – I don't think I need to go into great detail here (most drive images I was able to locate on the Internet contained **v2.00 – v2.03**) - **The Utilities** (Tools **v2.05**) have been

updated & bug fixed significantly and I would "generally" recommend using the later versions - These **(v2.05)** utilities contain some "missing" tools, are compliant with and will work with all **2.13** firmware and on.

After finalising and testing – as best I was able – the utilities, I spoke with Frederic (@veco) and he was very pleased to pull these into the main repository.

Frederic's repository now contains these updates: **https://github.com/veco/FreHDv1**

FreHD Utilities (tools) v2.05 – *Release files* (if you are not compiling your own) can be found here:
https://github.com/ontheslab/FreHDv1CPM/releases

Utilities Update Notes are here:
https://github.com/ontheslab/FreHDv1CPM/blob/main/notes.alltools

Available Files:

cpm_tool205.zip – CP/M Only Utilities (tools) - (original project that started it all 😉)

all_tool205.zip – Utilities for all stock OS'es

TOOLS205.DSK – Disk Image with the above Utilities

Part Two (a) – FreHD firmware has been updated and bug fixed (V 2.16 & V 3.68)

Like the work that has been done on the Utilities, the "firmware" has also been bug fixed. Given the history of issues, Matt (@maboytim) incorporated these fixes into the development of his "TrsVid" firmware and from this he has patched the stock (2.xx) firmware. The goal here was to make bug fixed firmware that would compile cleanly and was compliant with "fupdate". Firmware **(v2.16)** is now the current version for **Model I/III/4** if you are compiling it yourself. Firmware **(v3.68)** is the current version for the **Model II**.

This process showed up another small bug in the "fupdate" utility with the final version display missing a digit – This is now fixed and is already in the "utilities / tools" release on the main GitHub.

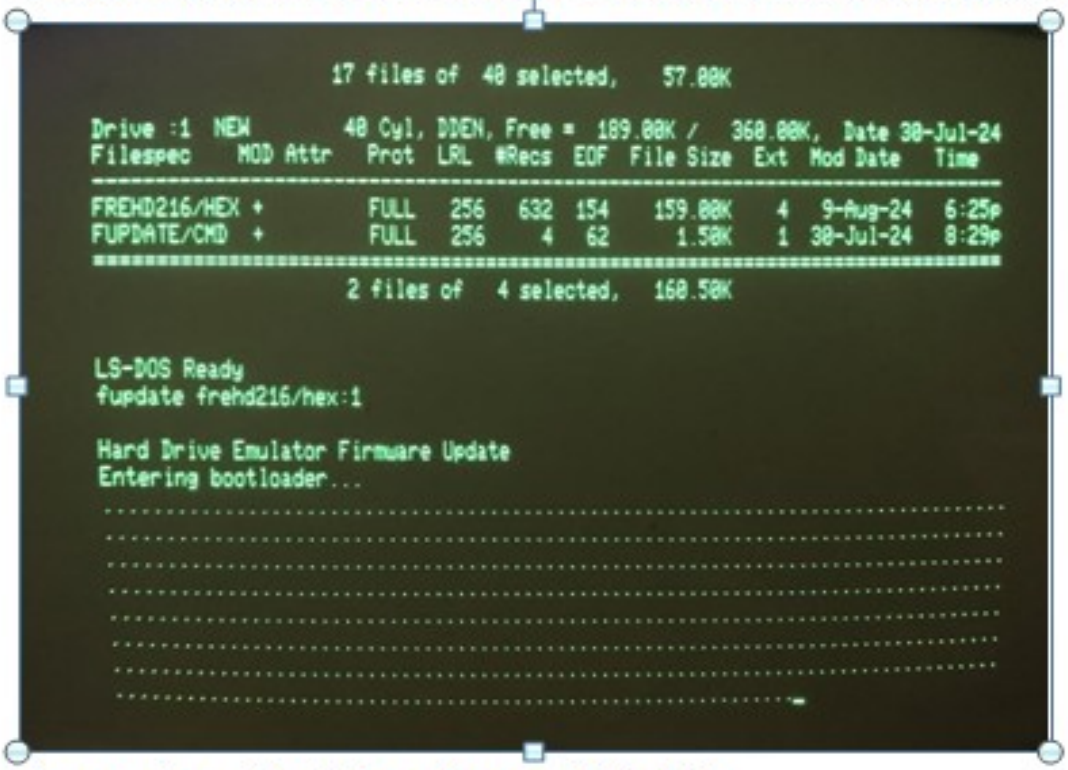
**** In the following example (v3.14) refers to the "TrsVid" Bootloader Compliant firmware.**

**** How to use this software updatable firmware will be covered in my next article (Part Two).**

"fupdate" loading of the new **(2.16 & 3.14)** and old firmware **(2.15)** is confirmed on the **4p**.



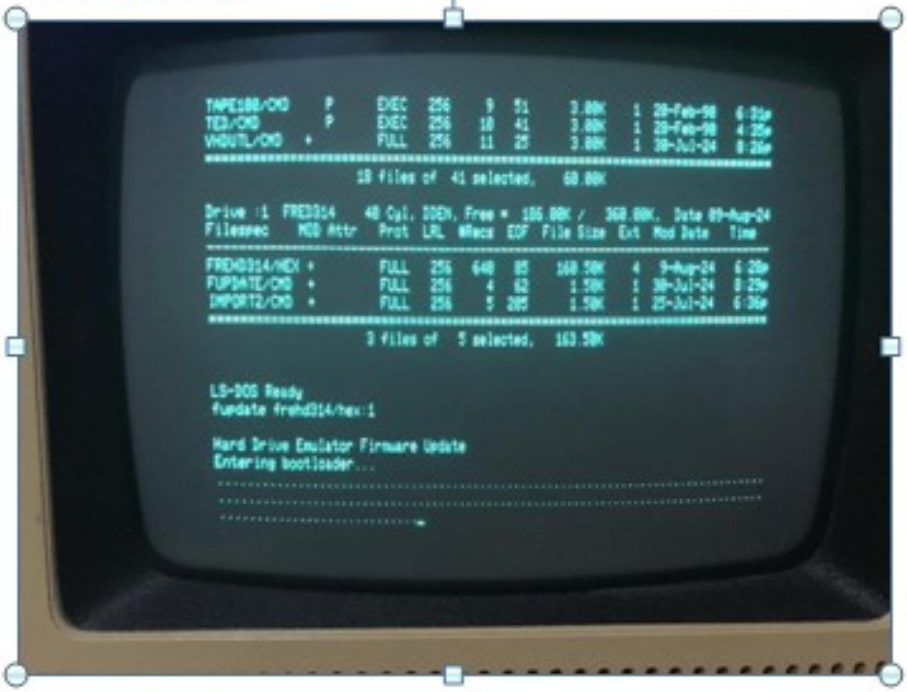
Bug **(now fixed)** in the display after update (dropping a digit (2.6 instead of 2.16) etc.

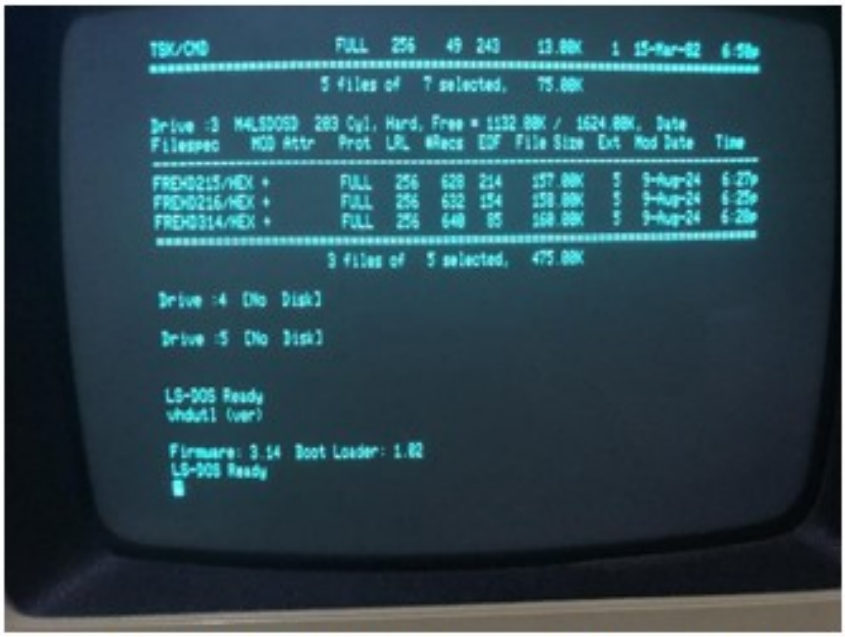


Programming and Confirming software update to 2.16

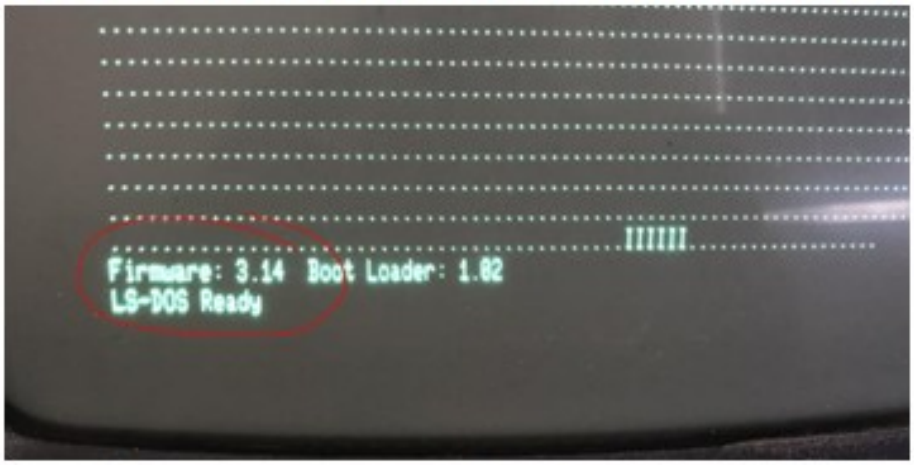


Process confirmed.





Programming and Confirming software update 3.14 – (with TrsVid support).



"fupdate" version display bug also confirmed fixed – version display corrected.

Secondly(b) – FreHD rom has been updated and bug fixed (V 1009)

A version display bug that would cause a crash was also identified in **"frehd.rom"** this patch is also now incorporated into the main repository and release (binary) files.



New "frehd.rom" functional in all – **above** - tested firmware versions without crash (details of the bug are not covered here but can be found in the source and on the Tandy discord).

Available files:

FreHD_2.16.zip – FreHD Firmware **(v2.16)** and frehd.rom **(v1009)**

Release files are here:
https://github.com/ontheslab/FreHDv1CPM/releases/tag/v2.16-firmware

Repository firmware notes:
https://github.com/ontheslab/FreHDv1CPM/blob/main/notes.firmware

Frederic's repository now contains these updates:
https://github.com/veco/FreHDv1

In Closing! I hope this is of help and use to others! 🙂

Summary:

- FreHD Utilities updated to 2.05 and pulled into "mainstream" by Frederic.
- FreHD Firmware bug fixed and updated to 2.16 – pulled into "mainstream" by Frederic.
- FreHD frehd.rom bug fixed and updated to v1009H – pulled into "mainstream" by Frederic.

FreHD – My Updates Story – Part One

Novice Instructions – for people like me 😳 – are in the next document – **Part Two**.

****** Warning – "flashing" or updating firmware – Warning ******

****** "can" and does go wrong – so only proceed with caution ******

Please note: **Multidos** and **TrsVid** support is not present in the above software.

References for these:

Tim's MultiDos -
https://github.com/hallorant/bigmit/tree/master/etc/multidos_frehd

Matt's FreHD Stream:
https://github.com/maboytim/FreHDv1

Matt's 3.14 TrsVid firmware:
https://github.com/maboytim/FreHDv1/tree/main/sw/trs_hard

FreHD – My Updates Story – Part One

Happy – *almost* – "SepTandy"! – My FreHD Update Story – TrsVid!

To quote "The Phantom" – And for those who came is late……

"TrsVid" is amazing! Hats off to George Phillips! Being a "late" returner to the TRS-80 world, it was a new discovery for me, I just had to get it going!

First - The "hurdle"

To quote Georges page "While getting a TRS-80 set up to play videos is not easy, trsvid itself is simple to use" **http://48k.ca/trsvid.html#manual**

To run it on "real" hardware - Requires updated / custom firmware that has to be installed via flashing a pic chip! (Hardware programming the FreHD's ROM externally).

For many reasons that I don't need to bore you with, this was sadly **NOT** for *me*!

Second - The "solution"

Enter the unexpected merger of this and my last articles subject – **FreHD Utilities and firmware update**.

As I started the discovery and patching process – initially as explained in part one – I discovered that Matt (@maboytim) had been working on "amazing" expansion ideas for the FreHD firmware. Chief among these was the development of "TrsVid" FreHD firmware that did **NOT** require alteration to the Bootloader and therefore would be "software" updateable!

FreHD firmware Version 3.14 – https://github.com/maboytim/FreHDv1

To quote @maboytim on discord:

> *…FreHD trsvid firmware that is compatible with the frehd bootloader allowing it to be programmed with FUPDATE instead of requiring a PIC programmer. I took a slightly different approach which reduces the changes from the original code and I thought eliminated some instructions from the most critical path that would offset the bootloader overhead, but it is after the most critical section. The timing seems okay but you can be the judge. This shouldn't brick your FreHD because it should stay in the bootflasher if the application crc is bad allowing you to reprogram again, and if you don't like it for some reason you can downgrade back to 2.x…*

FreHD – My Updates Story – Part Two (TrsVid)



Shazam! We have Video on my TRS-80 4P! – *No pic flashing required!*

How to do it!

****** This should go smoothly as suggested above – but ******

****** Warning – "flashing" or updating firmware – Warning ******

****** "can" and does go wrong – so only proceed with caution ******

If you are "familiar" with the process, by all means continue under you own steam!

You will need (and where to find it):

The update **(v3.14)** firmware – "FreHD_3.14.hex" – located here:

https://github.com/maboytim/FreHDv1/blob/main/sw/trs_hard/FreHD_3.14.hex

I would recommend renaming it to something SD/LSDOS compliant – **"FREHD314.HEX"**

Optionally, updated (version display corrected) "fupdate" utility – located in the "tools" collection:

https://github.com/ontheslab/FreHDv1CPM/releases/tag/v2.05a-alltools

How to do it – a novice guide!

FreHD – My Updates Story – Part Two (TrsVid)

Another warning, from the *FreHD Utilities Guide* (Thank you Ian):

Warning:

Do not attempt to load the firmware file from a drive mounted on the FreHD emulator or with the system drive running from the FreHD emulator.

The first step in the firmware update is to enable the FreHD emulator boot loader and erase the currently installed firmware. This stops the hard disk functionality and the update will fail.

The FreHD emulator will not operate as an emulated hard drive until new firmware is loaded.

Well if I have not scared you off! Here goes…

Please note: these are "short-form" instructions and assume a working knowledge of creating and coping to and from Floppy disks! (*I feel pretty safe is this assumption given the TRS-80 user base*).

You will need:

1. One bootable floppy disk for your system (LSDOS in my case).
2. One Data floppy disk of sufficient size to hold the firmware (a separate floppy is suggested given the size of the update file – typically 163k).

I then carried out the following steps:

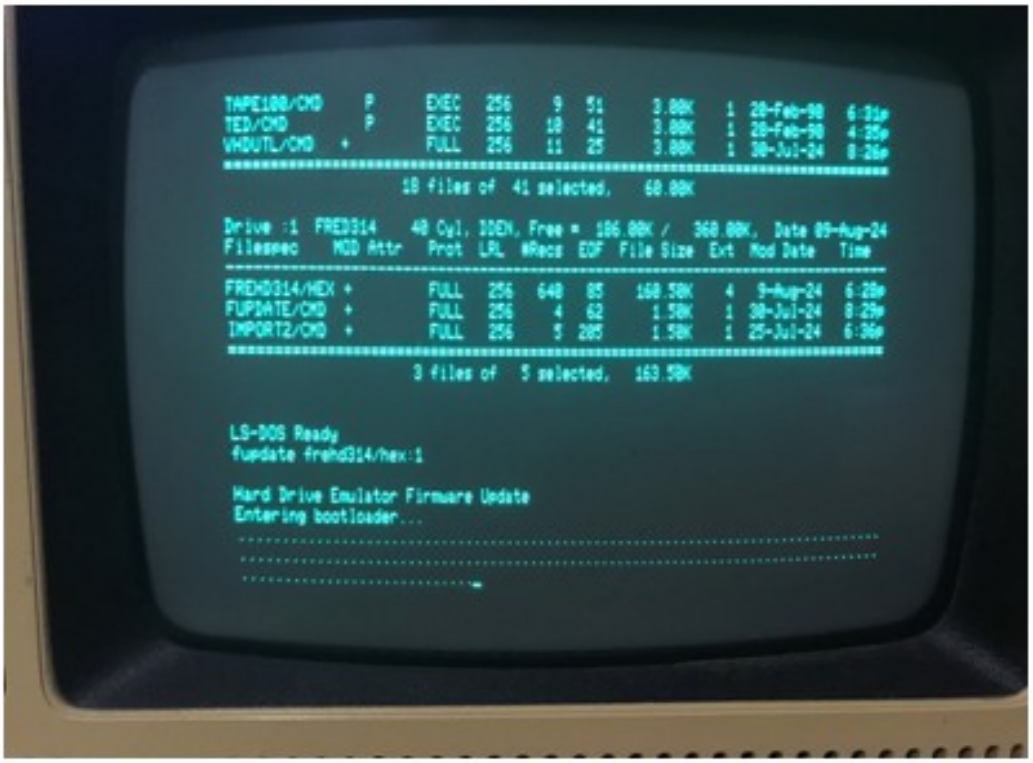
1. With my 4P booted into LSDOS from the FreHD, I used an existing "bootable" LSDOS floppy and copied the "FUPDATE/CMD" to this using the normal copy commands.
2. With a freshly formatted floppy disk in my second drive (DRIVE 1) - I used "IMPORT2/CMD" to copy my firmware file from the FreHD SD Card to drive 1:

   IMPORT2 FREHD314.HEX FREHD314/HEX:1

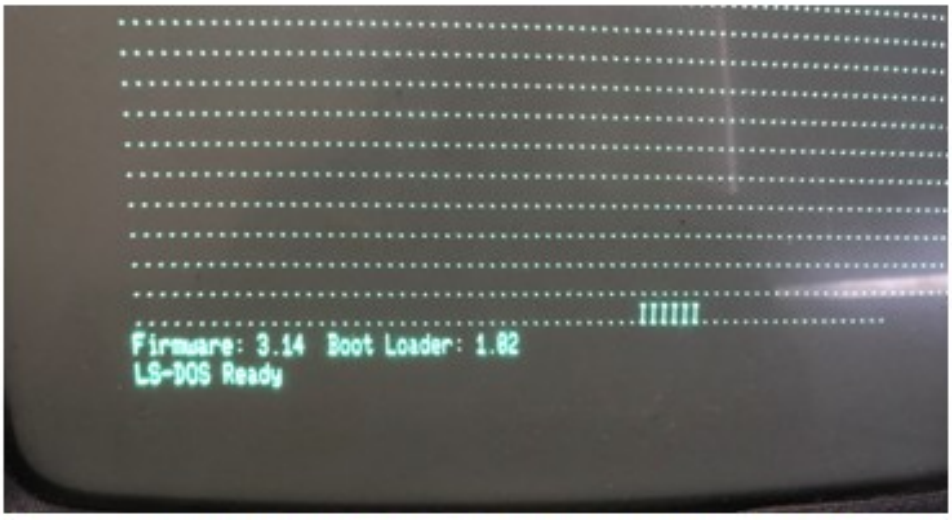
3. **** Important Step **** reboot your system from the "bootable" floppy disk and NOT the FreHD.
4. Next run the "FUPDATE/CMD" Utility and start the update:

   FUPDATE FREHD314/HEX:1

5. This will put the FreHD into Bootloader Mode and the 2 (red & green) LED's will illuminate to show its entered programming mode.



Dots will be displayed until the programming is complete – be patient – this takes a couple of minutes.



When programming **successfully completes** the 2 LED's will turn off. Signalling that the process was successful. The version number will be displayed confirming the update.

If they **do not** turn off you may need to try the process again.

FreHD – My Updates Story – Part Two (TrsVid)



You can now reboot back off your FreHD and you are ready to go!

At this time you will want to load some of the excellent example video's onto your SD card from the "TrsVid" main page: **http://48k.ca/trsvid.html**

Example Video of My TRS-80 4P playing the Dr Who demo:

https://youtu.be/RqysAKHT9Mg

I have found no "noticeable" video playing issue so far – but as Matt says – I will let you be the judge! I'm still in the – I'm blown away that I can even do this on my old 4P stage 😊

** If you should wish / need to go back to your original firmware, simply repeat the above steps using your choice of originally supplied 2.xx firmware – I have tested the update on my 4P with the following versions – **2.15, 2.16 & 3.14** without issue.

Final Note:

I would strongly suggest having a good read of all the TrsVid information:
http://48k.ca/trsvid.html

If you are comfortable with PIC programming (you can join the *"cool kids"*):
http://48k.ca/trsvid.html#frehd

Once again – HUGE thanks to George for **TrsVid** and Matt for making software updatable **TrsVid** firmware a living thing!

FreHD – My Updates Story – Part Two (TrsVid)

In Closing! As with part one - I hope this is of help and use to others! 😉

Reference material:

George's TrsVid Site: http://48k.ca/trsvid.html

Matt's FreHD Stream: https://github.com/maboytim/FreHDv1

Matt's 3.14 TrsVid firmware: https://github.com/maboytim/FreHDv1/tree/main/sw/trs_hard

My updated Utilities: https://github.com/ontheslab/FreHDv1CPM/releases/tag/v2.05a-alltools

Sample Videos:
In ".tv1" Format: http://48k.ca/trsvid/tv1.zip

In ".tv3" Format: http://48k.ca/trsvid/tv3.zip

In ".tv6" Format: http://48k.ca/trsvid/tv6.zip

IN MAV'S WORKSHOP

What *Was* The Tandy 1200?



by Ian Mavric

As collectors and retro computing enthusiasts become interested in the Tandy's MS-DOS computers a common question arises when searching through eBay.... What is a Tandy 1200? The popular and most interesting Tandy MS-DOS machine is undoubtedly the Tandy 1000 series, and the Tandy 2000 was Tandy's high-performance lemon. Educated people know of the Tandy 3000 and 4000, so where does this Tandy 1200 fit into the line-up of DOS systems? This article aims to explain what the 1200 was, who it was aimed at, and why you would want one now.

THE TRS-80 MODEL 2000 (T2K): TANDY'S APPLE LISA

Condensed history: In 1981 IBM released the IBM Personal Computer (PC), and by the end of 1982 companies were making IBM clone and compatible PCs. IBM upgraded the PC in 1983 to the PC/XT (extended Technology), and soon clones and compatible of that computer came along. Radio Shack came the conclusion that MS-DOS computers were the future and decided if they were

going to release one it was going to be the best darn MS-DOS computer available. And it was. Just so long as the software was written specifically for the special T2K hardware, otherwise the software wouldn't work. Coupled with the fact that the T2K was very expensive, especially if you added a colour monitor and hard drive, it was a hard sell. Like the Apple Lisa (also released in 1983, as well as the T2K), it was too expensive for home users, and not compatible enough to appeal to office users, which needed all computers to be on the same page as far as software is concerned.

Radio Shack's solution three months after the T2K's release in December 1983 was to go back to the drawing boards and design something more compatible, which we know today as the Tandy 1000 (T1K). However during that time an opportunity to plug a sales gap came along to sell to customers who needed a fully IBM PC/XT compatible machine, cheaper than IBM but 100% compatible. Enter the Tandy 1200... by Tandon.

TANDON: MAKERS OF FINE FLOPPY AND HARD DRIVES

Jugi Tandon, the owner and CEO of Tandon Corporation was good friends with the higher-ups in the computer division of Radio Shack. Tandon's products were in many TRS-80 products. The 5Mb, 12Mb, 15Mb hard drives were all from Tandon, the 8in disk drives in all 8in computers after the Model II were sourced from Tandon. The 4P had Tandon drives. The disk drives marked TP (Texas Peripherals, sometimes referred to as Tandy Production) in the Model III and 4, and external disk drives of the same era all come from Tandon. It was a lot of drives! At the time of the PC compatible and clone expansion in 1982/3 Tandon's talented engineers decided they could design a compatible. But how to market it?

By this time the market was saturated with clones and compatibles and other than computer manufacturers no one really knew much about the Tandon

name, certainly not enough to dump a bunch of cash on a PC compatible. A conversation with Radio Shack cut a deal to allow Tandon to get it's product out, as we call it, the Tandy 1200 (initially 1200HD then just plain 1200). This allowed Radio Shack a 3-pronged MS-DOS product line: The ultra-high performance Tandy 2000, the PCjr compatible budget-friendly Tandy 1000 at the other end and in the middle and pure or 'mirror image' IBM PC/XT compatible to allay any software or hardware incompatibility anxiety.



Advertising: Tandy's 3-prong MS-DOS approach - MS-DOS for everyone

TANDY 1200 HD SPECS: IT READS JUST LIKE AN IBM PC/XT

Same....same... could be the best way to describe the Tandy 1200HD, the spec sheets are almost identical. Tandy's advertising even gave the 1200 the nickname "Mirror Image" for those who might have been unsure if their expensive MS-DOS software would work (it all did) or their full length 13in expansion cards would fit and work (they also did, more on that later). So with the Tandy 1200HD you got a 4.77MHz 8088 CPU, 256K Ram (expandable to 640K), 5 expansion slots, a 360K 40-track double-sided double-density floppy

drive, a 10Mb hard drive (both Tandon drives, of course), an 84-key keyboard in PC/XT layout, and even the case was metal with the same colour paint as the IBM PC/XT. Even the TANDY and 1200 badges are in roughly the same places on the case and keyboard as on the IBM. Choice of video cards were a monochrome display adapter (MDA) and a colour graphics adapter (CGA). The monitors recommended for the system were the VM-3 green screen (IBM equiv. 5151) and the CM-2 colour RGB (IBM equiv. 5153).



The only compelling reason to buy it was the price at introduction which was $US 2,999, and the equivalent IBM PC/XT was $US 4,395. If you had a growing business and needed to buy, say, 30 computers, that could mean saving some $US 41,880. Enough to get the IT team members of a company interested. Likewise in a school if you wanted to outfit a Computer Lab for 30 students and save 41 grand. It was a pretty good deal, especially since most 'Unbranded' (a.k.a. what would be known as white-box compatibles) were only slightly cheaper but didn't have a big brand name like Tandy behind it. For comparison the Tandy 2000 HD cost $US 3,950 and the Tandy 6000HD Xenix computer was a whopping $US 5,499. The Tandy 1200HD was even cheaper than a TRS-80 Model 4 system outfitted with a hard drive ($US 1,299 + $US 1,999). It was certainly competitive in the hard disk market.

Period reviews of the Tandy 1200 were hard to find, but one in 80micro[1] magazine gave the computer a generally favourable write-up since the two

things it needed to do, which was to be 100% compatible with the IBM PC/XT and cheaper than it were both achieved. Preference was given to the VM-3 green screen over the CM-2 colour monitor as it was easier to read for long hours at the computer.

"MINE DOESN'T SAY 1200 'HD' AND LACKS A HARD DRIVE"

Although initially offered as a hard disk based computer, you have to remember that in the mid-1980s lots of people, especially home users, got by happily on their XT-compatibles using dual disk drives. A user could buy a dual-drive machine and when finances allowed upgrade with a hard drive. It also helped that hard drives got cheaper by the year so even if you used your dual-drive XT-compatible for 18 months then upgraded to a hard drive, chances are you saved several hundred dollars for being patient. Recognising this Tandon offered Radio Shack a dual-drive version of the machine, which they went on to sell for $US 1,999. It even had the space next to the dual slim-line 360K floppy drives to fit a hard drive when the time arrived.



By the time the Tandy 1200 dual-drive machine came around the Tandy 1000 had been recently released and was on it's way to becoming quite a success in its own right, making the dual drive Tandy 1200 quite rare. Let's see how they stack up.

TANDY 1000: BUDGET MS-DOS COMPUTING

It's easy to forget that the original Tandy 1000's purpose was to go head-to-head with the IBM PCjr which had been released around the time the design requirements for the T1K were being locked down. This made Radio Shack's aforementioned 3-prong sales strategy thusly: T2K was up against IBM's AT (and out performed it), the Tandy 1200 HD was the 'mirror image' cost saving machine against the IBM PC/XT, and the T1K was cheaper and up against the PCjr.

An interesting thing happened which was that the market didn't much like the IBM PCjr, and cheap clone manufacturers made the PCjr economically pointless. For the same money (or less) you can get a full IBM PC/XT compatible on mail order or from the newly emerging white-box PC builders.

Too far in to change anything on the T1K Radio Shack had to hope they'd landed on a successful formula. Turns out they did and were able to put the fiasco of the T2K behind them. The T1K kept the better graphics and 3-voice sound of the PCjr, as well as it's small desktop footprint, but still had the 4.77MHz 8088 CPU, a 360K floppy drive with space for a second drive, 128K Ram, 3 expansion slots, and use a version of the very nice T2K keyboard with a different connector. It was no fire-breather; the Ram size was too small and had no DMA chip until you added a memory expansion card, but was an impressive 95% software compatible, and what was important was that all the

important business apps and popular games ran great on it... once you upgraded the Ram to 640K and a 2nd floppy disk drive.

It's price of $US 1,199 and the PCjr graphics and sound made it a run away success. By the time you added a 2nd disk drive and memory to match the Tandy 1200 dual drive computer the cost was almost the same but you got better graphics and sound, and it took up less space on your table. The Tandy 1200 became irrelevant, and once the R&D cost for the T1K was recovered and/or nearly all the Tandy 1200 HDs had been sold Radio Shack stuck a 10Mb hard drive and 256K Ram with DMA into the T1K and christened it the Tandy 1000HD, with a bargain price of just $US 1,999. Clearance prices cleared out the last of the Tandy 1200 and Tandy 1200 HDs in 1986, and disappeared into history as quietly as they were introduced.

IT IS COLLECTABLE AND A USABLE RETRO CLASSIC?

Tandy 1200s are well built and long lasting machines built for long hours of running in office buildings. The hard drive, being an early MFM design has a MTBF (mean time between failure) of 3000 hours, which in a workplace is around 5 years. If the hard drive is still running after nearly 40 years it's a miracle and I wouldn't trust it to last. The rest of the computer is no more or less reliable than any other IBM PC/XT compatible. Check the power supply for bulging electrolytic capacitors and Rifa paper-in-oil mains filter capacitor and change them as needed.

The 1200 does a great job of running MS-DOS software from the period 1981-1986. The keyboard is nice to use. One plus with the Tandy 1200 is that it takes long 13in ISA expansion cards. The T1K can only take cards up to 10in long. Radio Shack didn't consider this to be a limitation at the time the T1K was being designed, figuring if buyers had a 13in card they needed to use they would be buying a Tandy 1200 anyway. The T2K uses a different bus connection that doesn't even use ISA cards, so it's interesting to note that

Radio Shack didn't make an MS-DOS system of it's own that would accept 13in long cards until 1987's Tandy 3000 and 3000HL.

CONCLUSION

Radio Shack were lucky Jugi Tandon came along at the right time with the Tandy 1200 which allowed them to quickly recover from the embarrassing failure of the T2K. I like to think that Tandy learned a lot from the T2K and the Tandy 1200 that the minimum requirement to be successful in the MS-DOS compatible world was indeed to be compatible. If you see a Tandy 1200 in your travels, give it a good home, it surely deserves one.

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.

References:

1. 80micro, March 1985 pp.31-35, "Through the look glass: The Tandy 1200 HD"

Rom Calls From TRSDOS 2.3 With Error Trapping

Vince Otten

The books I've read which detail how to call TRS-80 ROM routines from assembly language ("Microsoft BASIC Decoded", "The B00K", "Disassembled Handbook", "Inside Level II", "Level II ROM Reference Manual", "Pathways Through The ROM", "ROM Routines Documented", "Supermap")— all assume something. They don't often come out and say it aloud, but they expect you to be programming with *cassette* versions of an assembler or debugger. How do I know? Because whenever I tried out their examples of ROM calls that required any math from a DOS platform like Microsoft's Editor Assembler Plus, they would crash horribly.

CALL 0716 Single Precision Add

Add the single precision value in (BC/DE) to the single precision value in WRA1. The sum is left in WRA1

```
        LD      HL,VAL1         ; ADDR. OF ONE SP VALUE
        CALL    9B1H            ; MOVE IT TO WRA1
        LD      HL,VAL2         ; ADDR. OF 2ND SP VALUE
        CALL    9C2H            ; LOAD IT INTO BC/DE REGISTER
        CALL    716H            ; ADD VALUE 1 TO VALUE 2
        .                       ; SUM IN WRA1
        .
        .
VAL1    DEFS    4               ; HOLDS A SP VALUE
VAL2    DEFS    4               ; HOLDS A SP VALUE
        .
        .
```

24/314

Why? It became apparent that, when my program CALLed ROM code like 9B1H or 9C2H or 0716H, the processor ended up jumping somewhere unexpected. But it must have worked for James Farvour and the other authors. Surely they didn't write all this sample code without ever testing it. What was going on?

Over time, I began to realize that the code worked for them because they had tested it with something like the original cassette-based Editor Assembler and/or T-Bug. To run these cassette programs, *you first have to start up BASIC.*

When a TRS-80 is started, it almost immediately checks for disk drives. If there's none, it initializes a number of patches, memory addresses, and subroutines before it gets to the "BASIC Ready" prompt. That's all been set up before you type SYSTEM, load and run your Editor Assembler or T-Bug. But *it never gets set up if you go to TRSDOS.*

There were hints in some of those books that you had to initialize things like the "Division Support Routine", and only "The BOOK" recommended error-handling alternatives (in case you try to add two Single Precision values of "1E38", for example). But in the end, it took me a fair amount of reading through the TRS-80 startup code, and single-stepping through BASIC math routines using the trs80gp emulator, to fish out all the necessary support subroutines and patches needed to access ROM routines if you're writing an assembly language program to run under DOS (TRSDOS, Newdos, LS-DOS, etc) for the Model I. Here's what I found out.

1. There's a group of patches, vectors, and routines that need to be set up before you CALL these ROM routines.
2. In order to prevent an error from sending the processor into never-never land, it will be necessary to replace the BASIC error reporting code with our own "ON ERROR GOTO" vector and some code to safely return the processor to our program.
3. It will be necessary to save versions of the program counter PC and stack pointer SP registers *just before* we call the ROM routine.
4. Optionally, after we call the ROM routine, we can check if there was an error, and take action.

Here's how I implemented each set of procedures.

Part 1: the group of patches, etc. There are five that I've identified, each presented as a separate subroutine:

1. The "Division Support Routine". When either Level II or Disk BASIC is initialized, a 22-byte long code fragment is copied from ROM into RAM. Although this is called "Division Support", it's even used in such math as Addition, and so it's run from RAM instead of the slower ROM.

```
;
;     DIVSUP -- "DIVISION SUPPORT ROUTINE" INIT
;
DIVSUP    LD    HL,18F7H      ; COPY OF CODE @ 18F7..190DH
          LD    DE,4080H      ; PATCH 4080H..4096H
          LD    BC,22
          LDIR
          RET
```

(Of course, all labels are arbitrary: choose your own, more readable versions.)

2. If a Disk BASIC statement (like "LINE INPUT" or "OPEN") happens to pop up in a Level II BASIC program, the processor will jump out to one of a set of RAM locations that will redirect it into the appropriate extension that Disk BASIC loaded when it was invoked from TRSDOS. But of course that code doesn't exist if we're not running Disk BASIC, so the Level II initialization code just patches 21 "RET" return instructions in those RAM locations. This subroutine does the same:

```
;
;            PCHRET -- PATCH "RET" FOR DISK BASIC ROUTINES
;
PCHRET   LD      HL,41A6H      ; PATCH 41A6H..41E2H
         LD      B,15H         ; 21X WITH
I0LOOP   LD      (HL),0C9H     ; "RET"
         INC     HL            ; ORIGINAL CODE @
         INC     HL            ; 00A1H
         INC     HL
         DJNZ    I0LOOP
         RET
```

3. Along with those RETurn patches, Level II reports "SN ERROR L3" by patching jumps to that error message report code in 28 locations. We need to do the same:

```
;
;    PTCH28 -- PATCH 28 COPIES OF "JP 012DH" INTO
;         4152H..41A5H (DISK BASIC VECTORS)
;         TO PRODUCE "SN ERROR L3" IF
;         THEY ARE CALLED (FROM L2 BASIC)
;
PTCH28   LD      DE,012DH      ; "012D"
         LD      B,1CH         ; 28 COPIES
         LD      HL,4152H      ; START @ 4152H
L00R96   LD      (HL),0C3H     ; "JP"
         INC     HL
         LD      (HL),E        ; "2D"
         INC     HL
         LD      (HL),D        ; "01"
         INC     HL
         DJNZ    L00R96
         RET
```

Why bother with Disk BASIC patches (steps 2 and 3) when we're only going to CALL Level II ROM routines? A single-step trace through code like the Single Precision Add at CALL 0716H—which I had to try several times in the trs80gp emulator while following the detailed notes at Ira Goldklang's *TRS-80 Revived Site* (https://www.trs-80.com/wordpress/) in order to hunt down these patches—will reveal some fascinating insights into Microsoft's programming practices at the time. On the one hand, the code is remarkably clever; but on the other hand, it's a prime example of "spaghetti code". If an overflow error is detected, for example, the code will begin preparing several aspects of BASIC error reporting before checking the error trap vector I'll cover below. The code will actually pass through the vectors set up in steps 2 and 3 here as part of that preliminary setup.

4. When the processor encounters a RST instruction in the ROM code—and they show up a lot—it performs a subroutine call to each of the corresponding addresses: 08H, 10H, 18H..38H. Each of these immediately jumps to one of the 7 "DOS Exits" at 4000H..4015H. Again, each of these vectors is just an address to "jump back" into the appropriate ROM routine. However, they're not the same in BASIC as they were in DOS. So we need to
    a. Save the existing set of vectors so that we can restore them after we're done using the ROM calls;
    b. Insert the vectors that Level II BASIC places there.

Here's how we save the vectors:

```
VS          EQU   7*3            ; 7 INSTANCES OF 3-BYTE INSTRUCTIONS
;
;     V4000S -- SAVE VECTOR(S) AT 4000H+
;
V4000S      LD    HL,4000H
            LD    DE,V4000D
            LD    BC,VS
            LDIR
            RET
V4000D      DEFS  VS             ; SAVE DOS VECTORS HERE
```

(Here's how we restore them later:

```
;     V40000R -- RESTORE SAVED VECTORS TO 4000H+
;
V4000R    LD    HL,V4000D
          LD    DE,4000H
          LD    BC,VS
          LDIR
          RET
)
```

V4000S gets CALLed by this routine, which does the actual work of setting up the "Level II DOS Exits":

```
L2DOSX    CALL  V4000S        ; A. SAVE EXISTING
                              ; DOS EXITS
                              ; B. INSERT L2 EXITS
          LD    HL,V4000M     ; LIST OF "JP"S FOR MATH
          LD    DE,4000H
          LD    BC,VS
          LDIR
          RET

;     V4000M -- DISK BASIC DOS EXITS AT
;               4000H..4012H
;
V4000M    JP    1C96H         ; RST 08 / 4000H
          JP    1D78H         ; RST 10 / 4003H
          JP    1C90H         ; RST 18 / 4006H
          JP    25D9H         ; RST 20 / 4009H
          JP    4BA2H         ; RST 28 / 400CH
          JP    44B4H         ; RST 30 / 400FH
          JP    4518H         ; RST 38 / 4012H
```

5. Finally, there's a vector that BASIC sets up whenever a ROM routine "bails out" due to an error. It's located at 41A6H, and if we don't replace it with our own "ON ERROR GOTO" routine, any time a ROM routine encounters an error, it will send the processor

into BASIC never-never land, and we don't want that. Here's my code to replace that with my own error trap:

```
;
;     INITER -- INITIALIZE ROM ERROR TRAP VECTOR
;
INITER    LD      HL,41A6H      ; REPLACE EXISTING ERROR
          LD      (HL),0C3H     ; VECTOR WITH JUMP TO
          LD      DE,ERTRAP     ; "ERTRAP" ERROR-TRAP
          INC     HL
          LD      (HL),E        ; ROUTINE
          INC     HL
          LD      (HL),D
          RET
```

Part Two: Now we turn to trapping and recovering from errors. Step 5 has redirected BASIC's error routine to our own procedure "ERTRAP". My version of ERTRAP does just two things: it restores the Stack Pointer SP to its state just before we called the ROM routine, and restores the Program Counter PC to the address of the instruction following the ROM call. These two values have been stored in two addresses I have labeled SVSP00 ("Save Stack Pointer 00") and SVNX00 ("Save Next PC 00")—how they got saved there will be discussed below. Here's the code for ERTRAP:

```
;
;       ERTRAP -- "ERROR TRAP" SUBROUTINE
;
ERTRAP    LD        HL,(SVSP00)      ; RESTORE SP
          LD        SP,HL
          LD        HL,(SVNX00)      ; RETURN TO NEXT
          PUSH      HL               ; INSTRUCTION AFTER
          RET                        ; CALL TO ROM CODE
```

Now this is vital: *immediately before any ROM call which could result in an error*, like CALL 9B1H or CALL 9C2H or CALL 0716H in the example above from Farvour's "Microsoft BASIC Decoded" above, we insert

```
CALL    GETSPC
```

as in

```
    LD      HL,VAL1         ; ADDR. OF ONE SP VALUE
    CALL    GETSPC
    CALL    9B1H            ; MOVE IT TO WRA1
    LD      HL,VAL2         ; ADDR. OF 2ND SP VALUE
    CALL    GETSPC
    CALL    9C2H            ; LOAD IT INTO BC/DE REGISTER
    CALL    GETSPC
    CALL    716H            ; ADD VALUE 1 TO VALUE 2
    ..
```

GETSPC will do the following:

a) preserve some registers;
b) clear BASIC's "ERR" value, so we can test it for the occurrence of an error;
c) reset BASIC's "ON ERROR" flag so the ROM routine doesn't redirect our processor towards BASIC code rather than our routine;
d) save the the Stack Pointer to its state *just before* the following ROM call in SVSP00; and
e) save the address of the instruction following the ROM call in SVNX00.

```
    BERROR  EQU     409AH           ;  BASIC "ERR" HERE
    ;
    ;    GETSPC -- "GET SP & PC" SUBROUTINE
    ;
    GETSPC  PUSH    DE              ; DON'T CLOBBER S.P. ARG IN BC/DE
            PUSH    HL
            PUSH    AF
            XOR     A               ; CLEAR "BERROR"
            LD      (BERROR),A
            DEC     A
            LD      (40F2H),A       ; RESET ON ERROR FLAG
            LD      HL,8            ; GET SP BEFORE THIS CALL
            ADD     HL,SP           ; HL= SP
            LD      (SVSP00),HL     ; (SVSP00)= SP BEFORE
            DEC     HL
            DEC     HL
            LD      E,(HL)          ; DE= PC OF CALLER
            INC     HL
```

```
LD      D,(HL)
INC     DE
INC     DE
INC     DE              ; DE= PC FOR NEXT INSTR
LD      (SVNX00),DE     ; (SVNX00)= PC+3
POP     AF
POP     HL
POP     DE
RET
```

Part Three: Here's how we can tell if the ROM code detected an error (perhaps "overflow" when we tried adding 1E38 to 1E38). If it did, BERROR, aka the byte at 409AH, will contain a non-zero value that exactly corresponds to what ERR is in BASIC programs (in the case of this example, 10). This is an example of using that error flag:

```
CALL    GETSPC
CALL    716H            ; ADD VALUE 1 TO VALUE 2
LD      A,(BERROR)      ; GET ERROR CODE
OR      A               ; SET Z FLAG IF NO ERROR
JP      NZ, OOPS        ; GO IF AN ERROR DETECTED
..
```

To sum up, here's how you use ROM calls from your disk-based assembly programs:

1. Before using any, set up all vectors with

```
CALL    DIVSUP
CALL    PCHRET
CALL    PTCH28
CALL    L2DOSX
CALL    INITER
```

2. Ensure that

```
ERTRAP
```

is defined somewhere in your source code.

3. Insert

```
CALL    GETSPC
```

immediately before any ROM call.

4. (Optional) after the ROM call, inspect the byte value at

```
BERROR    ; 409AH
```

for any non-zero value indicating an error.

After you're done using ROM calls in your code, you may want to reset the DOS vectors at 4000H. I do that while testing my programs in the Editor Assembler Plus environment, for example. So, just before my assembly program exits back to EA+, I make sure it executes

```
CALL    V4000R          ; RESTORE DOS EXITS
```

as listed above.



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

http://members.iinet.net.au/~ianmav/trs80/







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

WHERE 8 BITS AND 16K STILL RULE OK

TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

DECEMBER 2024 - VOLUME 18 ISSUE 4

PAGE 04 — 2024 COMPETITION RESULTS
GEORGE PHILLIPS
BILL KEWLEY IS THE WINNER!!

PAGE 08 — BASQUE TRS-80 TOURNAMENT
EGOITZ CAMPO
DETAILS AND PHOTOS OF THE LATEST EVENT

PAGE 12 — TANDY ASSEMBLY 2024
IAN MAVRIC
IF YOU MISSED THIS YEAR'S TA MAV'S PROVIDED US WITH DETAILS OF EVENTS AND ANY AVAILABLE YOUTUBE VIDEOS THAT ARE AVAILABLE

PAGE 25 — SOFTWARE TOOLS IN BASIC - EPISODE 5
'A CLOSE CALL'
VINCE OTTEN
VINCE PRODUCES THE PENULTIMATE EDITION OF HIS EXCELLENT SERIES

PAGE 36 — TANDY 10
IAN MAVRIC
MAV REPORTS ON PETE'S RECENT ACQUISITION, OF THIS VERY RARE TANDY

PAGE 52 — U.K. EBAY BARGAINS
WERE **YOU** LUCKY WITH YOUR BIDS?

PAGE 54 — IGOR, IS IT A PRIME?
E.T. FONEHUME
HERE'S A ONE-LINER FROM ET, JUST TO CHECK IF YOUR NUMBER YOU'RE THINKING OF IS A PRIME!

PAGE 55 — THE WAY WE WERE
JUST A QUICK LOOK AT THINGS TANDY (ISH) BACK WHEN 8 BITS WERE THE 'THING'. (CHECK-OUT THE PICTURE OF A REALLY YOUNG BILL GATES).

PAGE 66 — IN MAV'S WORKSHOP
IAN MAVRIC
MAV SHOW US A PREVIEW OF A MODEL 1 BOOK HE'S WORKING ON. I THINK THIS WILL BECOME THE GO TO REFERENCE FOR ALL THE TANDY MODEL 1 REFERENCE NUMBERS. I CAN'T WAIT FOR THE FINISHED ARTICLE!

PAGE 75 — JUST FOR FUN!
ON THE LIGHTER SIDE OF LIFE. HERE'S A LITTLE PUZZLE FOR YOU TO TRY WHEN YOU'VE HAD TOO MUCH XMAS PUDDING!!

PAGE 77 — SURFIN' THE NET
INTERESTING PAGES FROM THE INTERNET

PAGE 79 — OBIT. - T.E. KURTZ

LATEST UPGRADE NOW AVAILABLE
VERSION 2.5.4

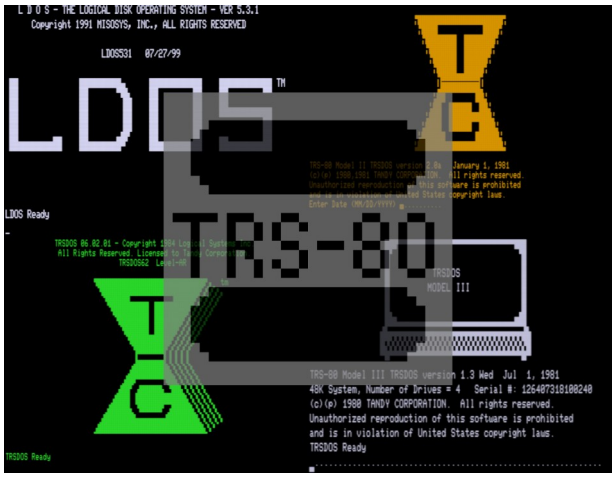
DON'T HAVE A TRS-80?

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



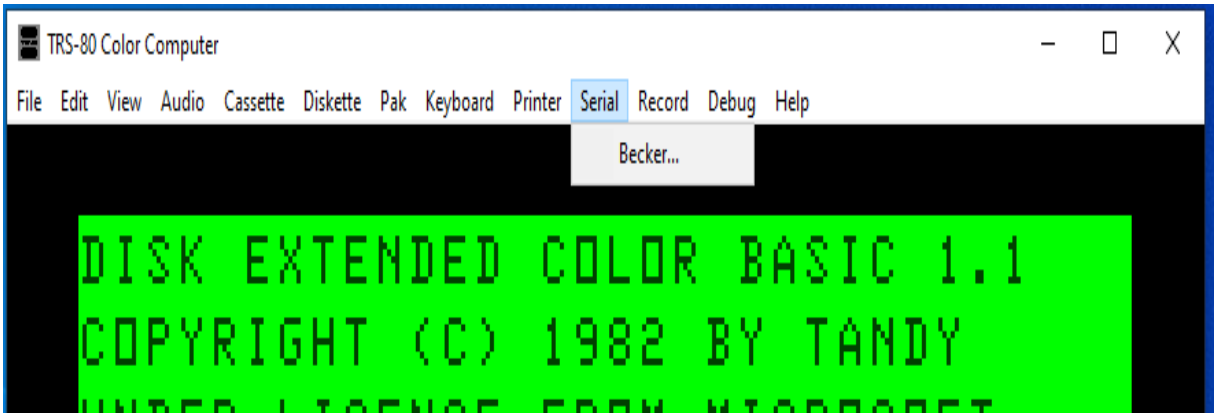
RUNS ON WINDOWS, MACOS, LINUX & RASPBERRY PI.
EMULATES MODELS 1, 2, 3, 4, 12, 16, 6000, MC-10,
COLOR COMPUTER, DT-1, VIDEOTEX AND NOW THE **TANDY 10.**

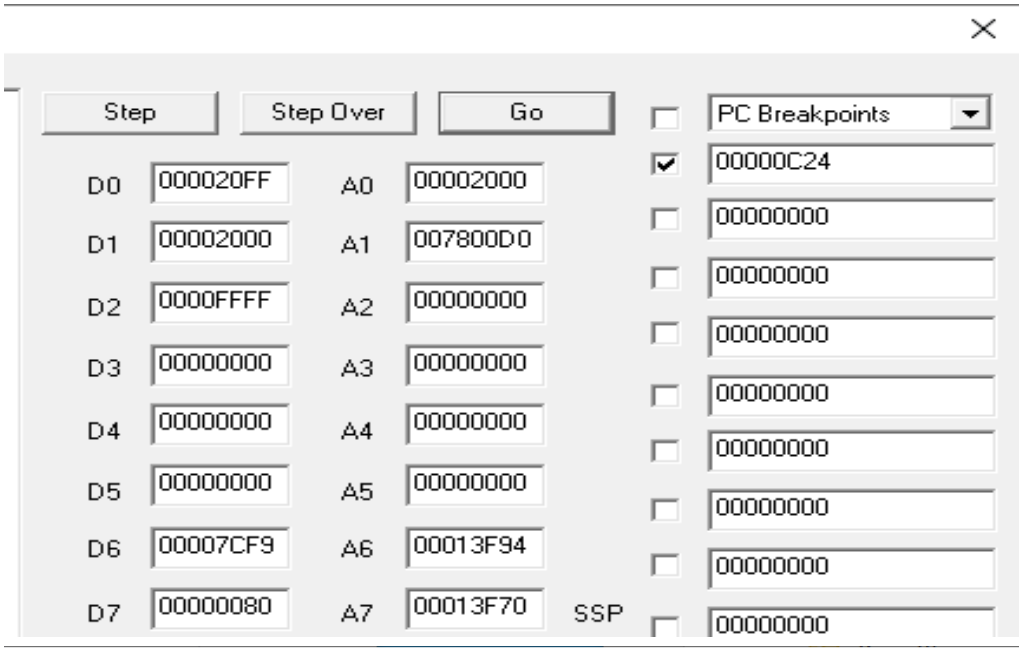
JUST LIKE A REAL TRS-80 WITH NONE OF THE HASSLE.
BUILT-IN DEBUGGER AND OTHER FEATURES, JUST GREAT FOR PROGRAMMERS.

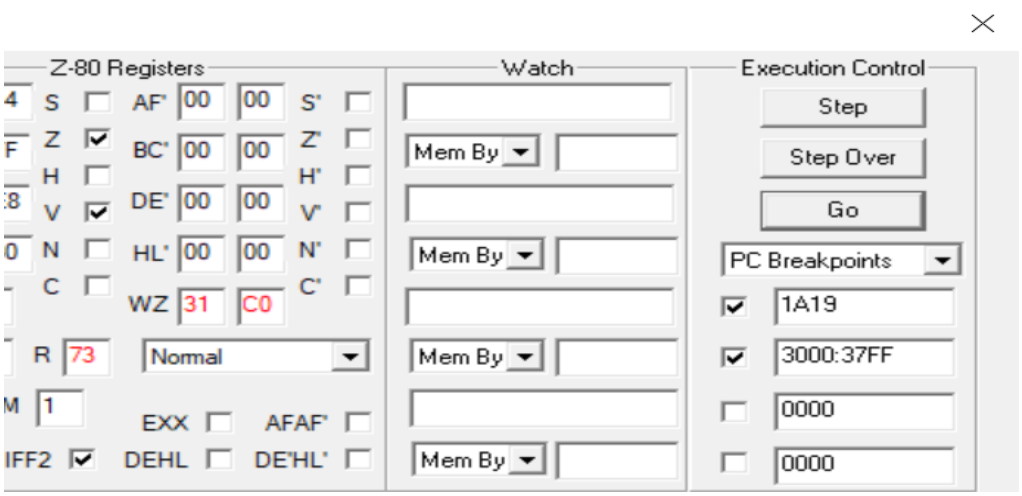
GET IT NOW AT HTTP://48K.CA/TRS80GP.HTML

UPDATED VERSION, 2.5.4, NOW OFFERS :

BETTER BREAKPOINT GUI WITH SINGLE INPUT WINDOWS.
BECKER PORT SUPPORT FOR THE COLOR COMPUTER.
HARDWARE SERIAL PORT ACCESS ON LINUX.
** TANDY 10 EMULATION **




Step
Step Over
Go
PC Breakpoints
D0 000020FF
D1 00002000
D2 0000FFFF
D3 00000000
D4 00000000
D5 00000000
D6 00007CF9
D7 00000080
A0 00002000
A1 007800D0
A2 00000000
A3 00000000
A4 00000000
A5 00000000
A6 00013F94
A7 00013F70
SSP
00000C24
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000


Z-80 Registers
Watch
Execution Control
S
Z
H
V
N
C
AF' 00 00
BC' 00 00
DE' 00 00
HL' 00 00
WZ 31 C0
S' Z' H' V' N' C'
R 73
Normal
EXX AFAF'
DEHL DE'HL'
IFF2
Mem By
Step
Step Over
Go
PC Breakpoints
1A19
3000:37FF
0000
0000


Tandy

2024 Competition Results

The 2024 competition was all about writing an "AI" to play Big 5's Defense Command. Though he did not have anyone nipping at his heels, our winner made a very good entry that can get staggeringly high scores in the 100,000 point range. Congratulations to:

Bill Kewley

I ran Bill's entry a number of times with different random number seeds to get an idea of how well it would do generally. Here are the scores for each seed I tried:

Seed	Score
9000	110960
60000	106240
1000	104790
3000	103050
30000	84410
5000	79790
40000	78880
7000	74110
6000	68190
10000	67410
2000	63940
4000	59850
8000	53960
50000	47950
20000	42970

The program displays good tactics. It can pick off attacking enemies almost always though it can get fooled when they dog-leg towards a cannister. In the early stage of the game it holds the enemies off and never loses a can. As the difficulty ramps up after 10,000 points or so the enemies start to steal cans. This is something of a critical point in the game but the programs "knows" that it must hold on to at least one can and manages to transition to an end game where it must fight off a continual attack by one can-grabbing enemy and one can-destroying slicer.

Generally it can keep up but if the two arrive at a bad time or are far separated the can will be picked up. The program will use an antimatter bomb if available to save it (otherwise the game would be over). Or it puts the highest priority on shooting the can-stealer and catching the dropped can.

If a slicer gets to the bottom or the program realizes it cannot be stopped it will use an antimatter bomb to destroy all the on-screen enemies and keep the game going. If none are available it immediately tries to get killed. As long as the slicer isn't too close to the can it can do so and save the can from destruction as the wave will reset before the slicer reaches the can. The lesser scores tend to occur when the timing is bad and the can can't be saved.

At 100,000 points or so the game becomes very cruel. The enemies start going at double speed. About all the program can do at that point is take what it can before its stockpile of ships and antimatter bombs are depleted.

A video of the best score seen so far and Bill's program are available on the competition page.

http://48k.ca/comp2024.html

Here's Bill's description of how the program works:

The basic loop of the program goes like this:

Map Enemies
Avoid Bombs
Avoid Enemies
Check Fuel Cans
Shoot Enemy
Flagship/Centre
Check Slicer
Check Swarm
Repeat

Map Enemies: For each enemy I work out where the ship needs to be placed to make the shot taking into account enemy direction and speed. I include fuel cans in this as they effectively become the target if being carried away.

Avoid Bombs: If a bomb is close by then move left or right to avoid it, also setting a flag for this frame so that other movement code doesn't override it.

Avoid Enemies: Similar to avoid bombs but if the enemy is bearing down quickly then a double step left or right is issued. If the enemy is directly above then fire a shot. If enemies are closing both left and right then fire an AMB "Panic Nuke".

Check Fuel Cans: Track location of fuel cans. Fuel cans being stolen to the very left or right are sacrificed as I found protecting fuel cans in the middle is much more efficient. If the enemy is just about to successfully steal the last can then an AMB is fired to try to rescue it.

Shoot Enemy: Select an enemy to target, the code basically selects the lowest enemy. Retreating enemies are ignored (unless carrying a fuel can) and enemies creeping along the bottom row are ignored. Once a slicer gets too low it is also ignored. At one point I tried to target the closest enemy rather than the lowest but there seemed little difference and in some

cases it was worse. If while moving toward the chosen target and it was still a distance away if we happened to be lined up with another enemy then a shot is taken.

Flagship/Centre: If we are not chasing an enemy then check for the flagship and chase otherwise we return to the centre of the fuel cans.

Check Slicer: If a slicer has got past the ship and we have enough then an AMB is fired. If we don't have any AMB's left and we have more than one ship left then we go into "Kamikaze" mode and take out the slicer with the ship. If a slicer makes it to the bottom row heading towards a fuel can "Kamikaze" mode is also enabled so that the ship will be destroyed by the next enemy hopefully saving the fuel can. This can be seen in action in the end game.

Check Swarm: Basically if there are more than seven enemies on the screen then fire an AMB if we have any left. This usually happens after shooting a flagship.

The loop now repeats by returning to Map Enemies until the game ends.

There was a lot of fine tuning of parameters, sometimes the tiniest change would be disastrous and occasionally an improvement. Once the AI was scoring decently each run would take 2-3 hours to complete, even in turbo (for some reason haste mode was unreliable on my machine). I ended up starting the run and recording the screen then I could come back later and see the final score, sometimes much less than the last run. I could then fast forward through the video to find the issue. So the tuning/test cycle allowed only a couple of changes in a day. No doubt there is a lot more that can be done to improve the AI to maximise points hopefully before the slicers get aggressive after 100,000 points at which stage it is all about survival. At least I can say the AI plays the game much better than I do :-)

Thanks, Bill for that informative description and all your work entering the competition. I'm very impressed at how well you've done when faced with hour long test runs.

I had hoped the competition would answer the question as to whether Defense Command can be “beaten”, that is, played indefinitely by a human player. That sudden difficulty increase at 100,000 points leaves me thinking it isn't possible. Though I feel a lot more optimistic about a human player reaching that level. The simple tactic of keeping one can alive has been proven to work well. It'd be quite a feat as a miss when targeting slicers can be disastrous yet it does not seem beyond human capabilities.

And there are riskier strategies to explore. A slicer reaching the bottom can be thwarted by letting a cannister be picked up. The slicer will only traverse the screen once or twice before it goes off screen and gives up. That may be one way to continue past the wall and has an additional advantage of scoring lots of points for cannister catching. Those points add up to bonus ships and antimatter bombs which serve as a hedge against situations where no other action is possible. Maybe some day the science of Defense Command will advance to answer such questions. For now we can only wonder at what the future may hold.

Thirsting for adventure?
Brass lantern getting dusty?



Take your Model 1, 3 or 4 on exciting new adventures never available until now....

M4ZVM enables your TRS-80 Model 1,3 or 4 to play Infocom and post-Infocom adventures, taking full avdantage of your hardware. Supports old favourites like Zork and Seastalker (including Sonarscope), as well as new-to-Model-4 games including Trinity, Border Zone and Sherlock, plus hundreds more available in the Interactive Fiction archive.



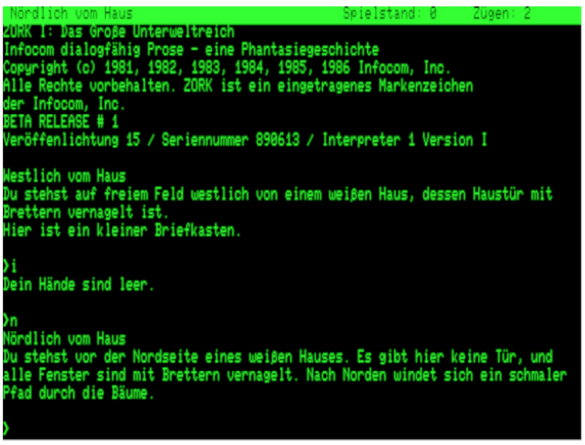
M4ZVM is available for download from https://gitlab.com/sijnstral/m4zvm (includes feature details and source)
Enhanced versions ready for download!

*128K recommended

Actual Screenshots of games new to the model 4, running M4ZVM (on trs80gp)



Beyond Zork



Zork 1 (German)

M4ZVM features include:
* z1-z8 game execution (z6 support is experimental)
* named game save/load
* accented characters
* 128K recommended - NEW Minimised 64K version
* Optionaly set the infamous "Tandy bit"
* NEW Model 1 and Model 3 versions

VIII. Basque Tournament of TRS-80
Egoitz Campo

On Thursday, 11th of July, the Euskal Retro organized the VIII. Basque Tournament of TRS-80. We met 5 players in Bilbao, in Café Berria (a bar located in Santutxu)

It was a fun afternoon. We plugged in the TRS-80 as soon as we arrived and started playing. Then later we would play other classic consoles such as the PC Engine, Play Station 2 or Commodore Amiga 500, but first we got into the American retro-computer. We played to these 5 games: Monster Maze, Canyon Climber, Demolition Derby, Clowns & Ballons and Skiing.

As I comment in our previus article with my new TV the games look better, the image has improved and we can list the music of the games. The step taken has been for the better.

Egoitz won the tournament after playing to these 5 games. But the other players played as Luis did a great job, and Luis he was about to win the tournament. They tied on points, but the tournament was won by Egoitz because he won 3 partial victories and Luis only two, but the equality was maximum. For the next tournament, possibly in april of 2025, we hope to have new games like Dragon Fire or Robot Battle.

Here you have the results:

Monster Maze

1-Luis Getxo 3 points (1.420)
2-Egoitz Campo 2 points (1.210)
3-Daniel Barrientos 1 points (200)

Canyon Climber

1-Egoitz Campo 3 points (17.400)
2-Luis Getxo 2 points (14.200)
3-Daniel Barrientos 1 points (2.600)

Demolition

1-Luis Getxo 4 points (2.900)
2-Galder Etxebarria 3 points (2.200)
3-Egoitz Campo 2 points (1.200)
4-Daniel Barrientos 1 point (1.000)

Clowns & Ballons

1-Egoitz Campo 4 points (2.180)
2-Luis Getxo 3 points (1.080)
3-Daniel Barrientos 2 points (250)
4-Galder Etxebarria 1 point (210)

Skiing + the penalty seconds; 1 second penalty for each door that was not passed

1-Egoitz Campo 5 points (1:18 +5)
2-Luis Getxo 4 points (1:15+13)
3-Daniel Barrientos 3 points (1:18+12)
4-Galder Etxebarria 2 points (1:30+6)
5-Mikel Torde 1 point (1:18+26)

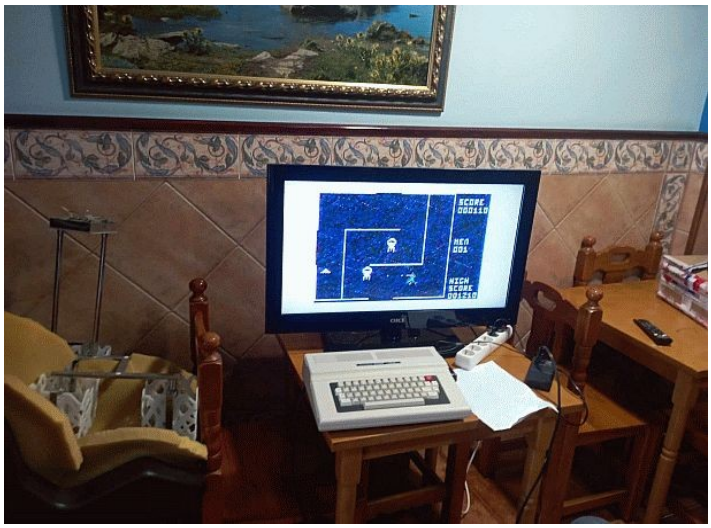
Classification

1-Egoitz Campo 16 points (Basque Country) 3 partial victories
2-Luis Getxo 16 points (Basque Country) 2 partial victories
3-Daniel Barrientos 8 points (Basque Country)
4-Galder Etxebarria 6 points (Basque Country)
5-Mikel Torde 1 point (Basque Country)





VIII. Basque Tournament of TRS-80














Joe Kay
Director
NORTH COMPUTER WEST MUSEUM
Create Experience Learn
joe@nwcomputermuseum.org.uk
01942 582826
Information Technology facilities & training
Workshops for programming, Electronics & VR
Community classes, clubs & retro games
IT Skills for the young & old
City & Guilds Creditation
Supporting education of local people
Creating opportunities & careers
Disable friendly
Volunteer- ed trainirg & museum operations
NORTH WEST COMPUTER MUSEUM
Create Experience Learn
www.nwcomputermuseum.org.uk
Leigh Spinners Mills, Park Lane, Leigh, WN7 2LB

MAV'S TANDY ASSEMBLY QUARTERLY COLUMN

TA24 NEWS

by Ian Mavric



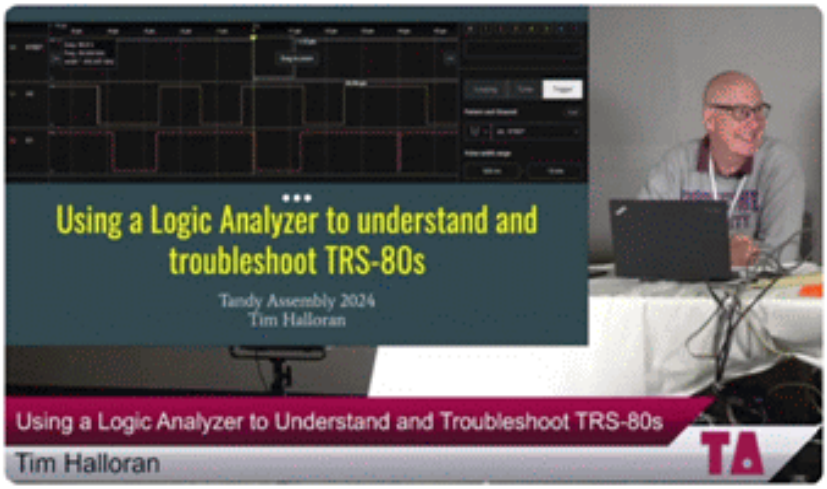
Bringing you the latest news you need to know about the greatest Tandy TRS-80 show on the planet. This time I talk about the 2024 show which has just finished.

THE 2024 SHOW HAS BEEN AND GONE - LET'S LOOK AT THE HIGHLIGHTS



Friday started with the usual Tandy Tech Track sessions and there was much to be learned during these popular tutorial presentations. If you missed them I have included links to each of the talks.

TTT01: Tim Halloran: Using a Logic Analyzer



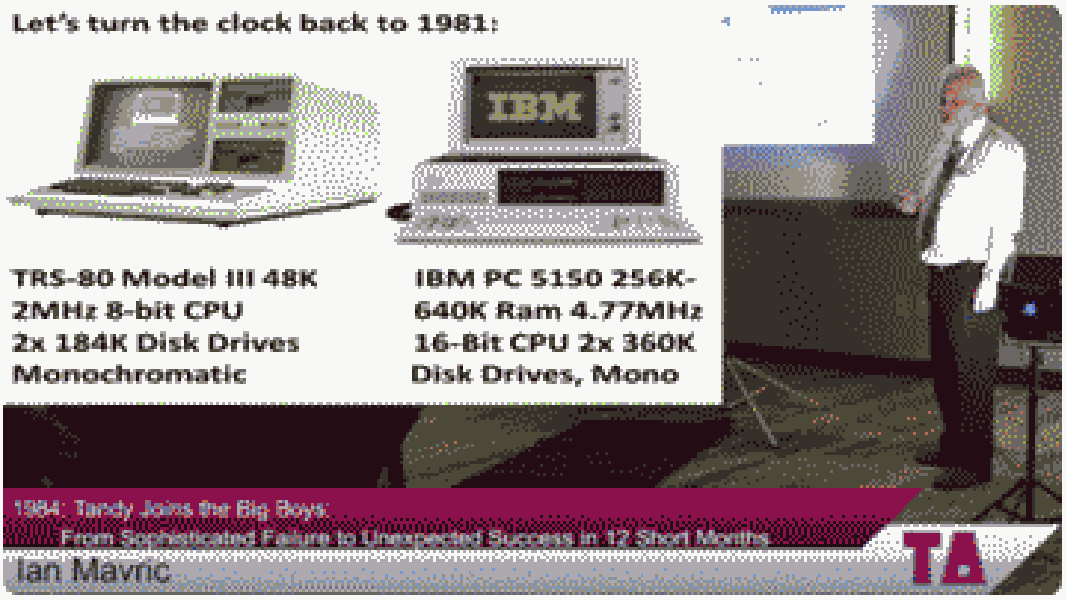
https://www.youtube.com/watch?v=qVE67v0ojfA

TTT02: Jeff Birt: Programming Pocket Computers



https://www.youtube.com/watch?v=YLOKHvXm7zQ

TTT03: I talk about The Tandy 2000 and Tandy 1000



https://www.youtube.com/watch?v=or94JMF6flU

TTT04: Amardeep Chana: Model III/4 Disk I/O Deep Dive



https://www.youtube.com/watch?v=m-JQ8E7tRUM

TTT05: John Strong: Interfacing to the Coco



https://www.youtube.com/watch?v=PuLk8Qqh2tE

SATURDAY: TA24 BEGINS IN EARNEST WITH KEYNOTE SPEAKERS

A full audience was in attendance for the main keynote speaker, Steve Leininger who gave a great talk about how he designed the TRS-80 Model I. As a bonus Don French (project manager for the Model I) dropped by and helped from the audience to explain some of the details from so long ago.



Photo: Steve (centre), with Don (right) and Peter Cetinski (left)

KEYNOTE SPEAKER: Steve Leininger: The Real Story of the TRS-80



https://www.youtube.com/watch?v=xv8Mh3h1dLg

2nd SPEAKER: Arno Puder & Lawrence Kesteloot: TRS-IO++

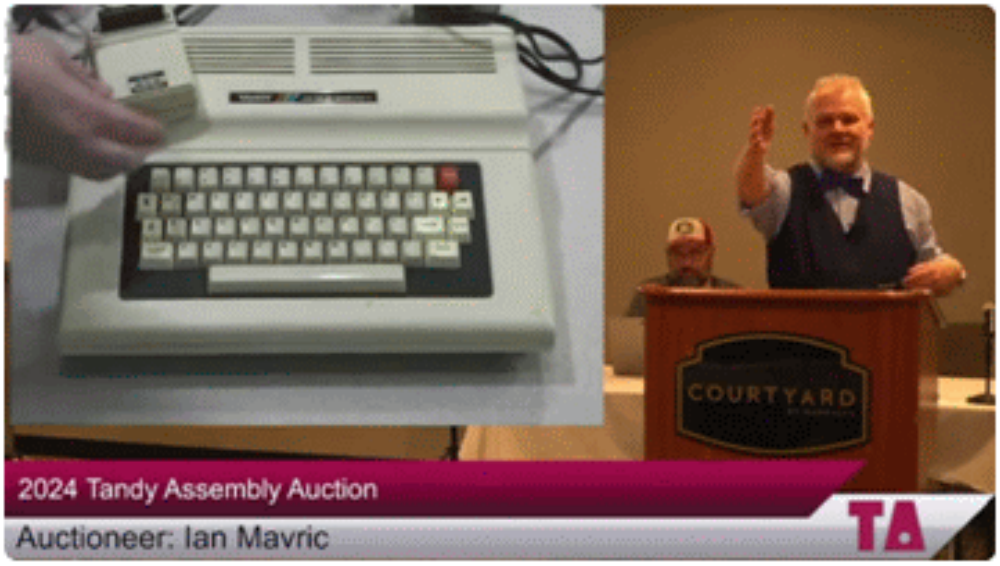


https://www.youtube.com/watch?v=tV7D6_IFSac



We then moved onto the entertainment part of the show. The auction of vintage TRS-80 items which generates funds for next years Tandy Assembly event was a success and fun to watch. It was followed by the TA trivia game which, like previous years, involves the audience playing a pub-trivia style simplified TRS-80 game.

SATURDAY AFTERNOON: AUCTION



https://www.youtube.com/watch?v=iY1phJERsvw

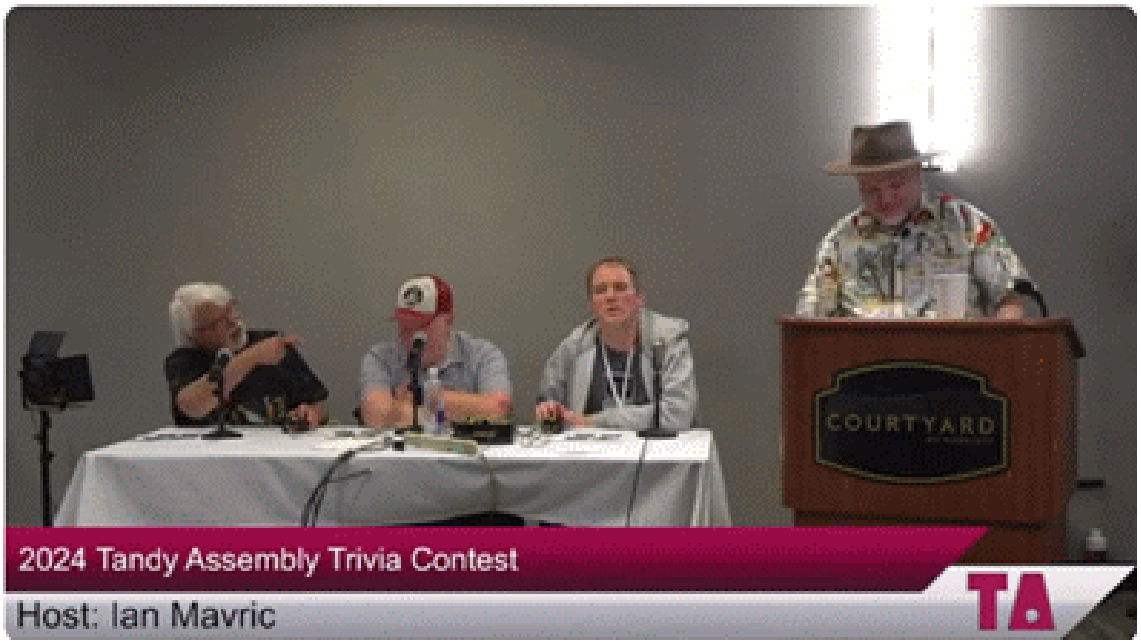
SATURDAY NIGHT: TRIVIA GAME

As is the tradition with the trivia game, I always aim to get the keynote speaker on the panel so I was pleased to have Steve Leininger join us. He was up against Brendan Bergie and Neil Blanchard, both of which have extensive knowledge of Tandy and TRS-80 computers. I'll let you watch the video to see who won. Likewise the audience trivia game was also played and the winner

this year scored and amazing nine out of 10 possible points. Again, you'll need to watch the video to see who that was. :-)



Photo: Steve, Brendan and Neil - Tandy trivia experts for 2024



https://www.youtube.com/watch?v=mU2vQw-uQD0

SUNDAY: FINAL SPEAKER AND THE SHOW WINDS DOWN

RICK ADAMS: COCO WAR STORIES



https://www.youtube.com/watch?v=zMl10_sUN6A

It's always good to have Coco content at TA and we were happy to have two presenters (John Strong and Rick Adams) give two presentations.

OTHER NOTEWORTHY MENTIONS ABOUT TA24

1. The offer to Coco enthusiasts who attended CocoFest earlier in the year free entry into TA24 was enthusiastically taken up by a few people simply by bringing their CCF card along. Thanks for making that a successful endeavour. We always like to try something new.

2. Speaking of new things, well done to Elizabeth Hayes of Ontario, Canada who was a first-time exhibitor with her "Pretty In Pink" themed table of custom painted computers. Everyone who saw them up close was amazed by the amount of effort and attention to detail she puts into these machines. Liz even donated one to the TA24 auction.



3. As predicted, lots of people brought their TRS-80 Model I's along for Steve to sign. I lost count but there must have been at least 20 and Steve was happy to sign them and have a chat with the owner about their particular machine. I thought about bringing my Model I all the way from Australia to get it signed, but worried it might be treated harshly by United Airlines baggage handlers left it behind. While at TA I purchased a 64K Coco 1 and had Steve sign that for me. The amusing thing (to me at least) was that the Coco wasn't one of his designs, in fact he had little to do with the design, but saw the humour in what I was doing. Think of getting Roger Moore to sign a picture of Sean Connery and you get the idea. It was great to see that Steve had a great sense of humour.



Photos: Steve signing my 64K Color Computer I

4. The fine tuning of the trivia game with expert rounds for each on the panel was extra fun especially when one contestant would steal an answer from the other and get double points. The phone-a-friend feature was also funny because when you phone a friend and they don't know the answer both people end up with egg on their faces.

ANOTHER TANDY ASSEMBLY DONE AND DUSTED

By mid afternoon on Sunday the show was over and we all slowly packed our tables and talked about how great the show was and who would be back for TA25 (next year's show). Many said with both Steve and Don present this was the best Tandy Assembly yet which begs the question, what will we do to make it even better next

year? The committee for TA which I am a part of have already started outlining next years show, but if you have any ideas, no matter how crazy you think they are, drop us a line at tandyassembly@gmail.com and tell us all about it. Better still if you know of veteran TRS-80 personalities from back in the day and think they would be interested in telling us their story we'd be all ears!

NEXT TIME: It will be the new year so let's start talking about TA25!

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.




BRITAIN'S BEST-SELLING DIGITAL CULTURE MAGAZINE
PIXEL ADDICT
£6.50 ■ 2024 ■ ISSUE 19
Tandy
The Story of an Electronics Giant
PLUS
NEIL WEST OF SEGA POWER
40 YEARS OF KING'S QUEST
MULTITECH'S APPLE II CLONE
20 GOTO 10 BOOK REVIEW
20 GOTO 10

Software Tools, Episode 5:

A CLOSE Call

Vince Otten

This month, we'll wrap up the discussion of SKELETON/BAS, an environment in which you can write "Software Tools" like those discussed in the book by Brian Kernighan and PJ Plauger, only in TRS-80 Model I Disk BASIC. Our next and last article in the series will outline SH/BAS, the "shell" program that lets us run Software Tools written in BASIC with "redirection", so we can enter lines like

```
$ COPY >FILE2/ASC
$ WC <FILE2/ASC >/DEV/PRINTER
$ TR A-J 0-9 <FILE2/ASC >CODED/ASC:3
```

But this month, we're polishing off some more mundane details: how SKELETON/BAS CLOSEs its files, and how it sets up in the first place before running your user program at line 40000.

CLOSEing Files

Your user program in SKELETON/BAS can be written between lines 40000 and 59979. Any program written using SKELETON/BAS will self-initialize (which we'll investigate toward the end of this article), and when it's done the initialization code will GOTO 40000, so your user program should start at that line. Normally, your program will "run off" into line 59980, which CLOSEs all previously OPENed files. If you want to place subroutines in this "user program space" (lines 40000..59979), you should have your main program "jump around" them and GOTO 59980 at its completion. Subroutines can also be located between lines 10000 and 39990, and this is probably a better location, since they'll run faster there, and won't interfere with the "run off into line 59980" just described.

(As well, once you've written your user program between lines 40000..59979 and 10000..39990, you'll want to edit the tile REMark in line 1 and then SAVE it under its new name, as in 'SAVE "COPY/BAS:2" '.)

So—line 59980 CLOSEs all files that have been OPENed by either the SKELETON/BAS initialization routines we're going to discuss below, or by the user program calling the "fOpen()" routine by GOSUBing Line 60500, as disussed in the previous two articles.

```
59990 GOSUB60920' FCLOSE-ALL()
```

Here's what "fCloseAll()" looks like:

```
fCloseAll() {   /* close all open files, preparatory to exiting                    */
    for (f= 1; f < MaxOpenFiles(); f++) {
    /* "FOR X5%= 1 TO FN DN%( 0):"                          60920a  */
        fClose (f);
        /* "VA%(0)= X5%: GOSUB 60910"                       60920b  */
        }
        /* "NEXT:"                                          60920c  */
    }
    /* "RETURN"                                             60920d  */
```

Duh: we simply loop over each file number from 1 to MaxOpenFiles and fClose() it. So here's fClose():

```
fClose( file#) {     /* close file#                                    */
/*   VA%(0)                                                            */
    if ((mode= OpenType[file#]) < INPUT_MODE)
    /* "X9%= VA%(0): X8%= SY%(X9%,1): IF X8% < 1 RETURN"    60910   */
        return;
    if (IsValidMode( mode)) {
    /* "IF FN DF%( X8%) THEN"                               60912a  */
        if mode == APPEND_MODE
        /* "IF X8% = 6 THEN"                                60912b  */
            AppendClose( file#);
            "GOSUB 60900"                                   60912c  */
        }
    else
    /* "ELSE"                                               60912d  */
        CLOSE( file#);
        /* "CLOSE X9%"                                      60912e  */
    OpenType[file#]= CLOSED_MODE;  /* mark this slot "closed/avaiable"      */
    /* "SY%(X9%,1)= 0"                                      60916   */
    /* "RETURN"                                             60918   */
```

First, we confirm that this is a file that ought to be CLOSEd because it had previously been OPENed with a proper Mode: if not, we simply exit fClose() without doing anything (line 60910). Now, we confirm that the Open Type or Mode is valid (a value of 1..6) for this file (line 60912a); if it is, we then check whether it's the special case of 'OPEN for "A"ppend (value 6)' in line 60912b, and if it is, we call AppendClose() in line 60912c; otherwise, we perform a simple CLOSE on the file (line 60912d/e). AppendClose(), discussed below, has to update the file's DCB to reflect the new "End Record Number" and "EOF Offset" to the last byte in the last record.

In any case, we now reset the Open Type or Mode of this file to CLOSED_MODE or 0, so that this file number becomes "available" to any later calls to fOpen() (line 60916).

What does AppendClose() look like?

```
AppendClose( file#) {  /* update file#'s DCB, then CLOSE file#                    */
/*        VA%(0)                                                                  */
    PUT file#, RecordCount[file#];
    /* "X8%= VA%(0): PUT X8%, SY!(X8%,1):"                          60900a  */
    ConvertCharCount( TotalCharCount[file#]);
    /* "VA!(0)= SY!(X8%,2): GOSUB 1910"                             60900b  */
    POKE EndByteLastRec( file#), Offset;                 /* "Offset" is VR%(2)  */
    /* "POKE FN DL%( X8%), VR%(2)"                                  60902a  */
    WordPoke( EndRecAddr( file#), RealAddrToSigned( RecordNo));
                                                         /* "RecordNo" is VR!(0)  */
    /* "VA!(0)= FN DE!( X8%): VA!(1)= FN SI%( VR!(0)): GOSUB 9500   60902b  */
    CLOSE file#;
    /* "CLOSE X8%: RETURN"                                          60908   */
    }
```

First, we make sure to PUT the last record we've been putc()ing to out to the disk file (line 60900a).

We discussed ConvertCharCount() in "Episode 3: OPEN For Business" (TRS8bit March 2024, p. 39). It's a multipurpose function used here and in AppendOpen(), and it splits up the TotalCharCount for this file into 3 values:

a) A Record # to GET or PUT or store in the file's DCB — VR!(0)
b) an offset in the record for the last byte to store in the DCB — VR%(1)
c) the size of the spacer field before the character to append to the file — VR%(2)

We call ConvertCharCount(TotalCharCount) in line 60900b. We're going to use its return values to update this file's DCB in order to reflect that we've appended some data to the end. We take the position of the last byte in the last record [VR%(2)] and POKE it into the DCB at the location provided by FN DL%(), in line 60902a. Next we're going to POKE the End Record Number with the updated value (a) that ConvertCharCount() provided as VR!(0). This is a multi-step process. We determine *where* to poke it using FN DE!(fle#) to set VA!(0), in the first part of line 60902b. That's simple. But ConvertCharCount() returned the End Record Number as a single-precision value in VR!(0), because it can have a 16-bit value of 1..65535. We have to

a) covert that value to a signed integer that BASIC understands, and
b) "WordPoke()" the value, ie, split it into two bytes and POKE them in successive memory locations.

So we use FN SI%(VR!(0)) to convert the single-precision value to a signed integer and then provide it as VA!(1) be "Word Poked" at the address VA!(0). This will have the result of a 16-bit End Record Number being updated in this file's DCB (line 60902b).
Lastly, we simply CLOSE the file, since its DCB has now been updated to reflect the appended bytes. Note that fClose() can also be used from the user program for any individual file.

SKELETON/BAS Initialization

Any program built using SKELETON/BAS will first "initialize": it will open the files specified by the "shell", SH/BAS, and then jump to line 40000. Here's how SKELETON/BAS begins:

```
1 GOTO65000'

    * SKELETON/BAS --
          MAY 24, 2024 * TESTED 9/26/23
```

So all the initialization code comes quite "late" in the program text, at line 65000. That way the BASIC interpreter can find all the "fast" subroutines, and your user code, more quickly at lower program line numbers. What's at 65000?

```
64999 '

    ** INITIALIZATION (65000..65529) **

    INITIALIZATION()

65000 SS%= 500 ' "STRING SPACE" IN BYTES
65002 CLEAR SS%
65004 X9%= 15 ' "LENGTH" OF SYSTEM ARRAYS
65006 DIM SY%(X9%,3), SY!(X9%,2), SY$(X9%,2)
65008 D0%= 1 ' SYSTEM-WIDE DEBUG FLAG
65020 GOSUB 65050 ' INITIALIZE
65048 GOTO 40000'
```

Three objectives are accomplished here. First, we reserve string space (65000..65002). Secondly, we set aside some "System Arrays"—integer, single-precision, and string arrays, all 2-dimensioned. These will mostly contain buffers, file records, and some miscellaneous constants. (They're stored in arrays named "SY()" so that every other variable name is available to the user.) Thirdly, we do all the rest of the initialization in a main subroutine at 65020, and then immediately jump to the user program which begins at line 40000.
This is what the main initialization routine looks like:

```
'
    * INIT SCHEDULER *

    INITIALIZE()

65050 GOSUB65100' INIT-VALS()
65052 GOSUB65150' INIT-CMD-LINE()
65054 GOSUB65200' INIT-USER-PRGM()
65056 GOSUB65250' INIT-STD-FILES()
65058 GOSUB65300' OPTIONAL-PACKAGES()
65098 RETURN
```

In other words, we initialize "values" with INIT-VALS() at 65100; parse the contents of CMDLINE/SH with INIT-CMD-LINE() at 65150; the user program with INIT-USER-PRGM() at 65200; the "stdin", "stdout", and "stderr" files or devices with INIT-STD-FILES() at 65250, and any "optional packages" (typically a program stub) with OPTIONAL-PACKAGES() at 65300. Looks a lot like COBOL ("PERFORM INIT-VALS THROUGH OPTIONAL-PACKAGES"): if you know, you know.

Here's "INIT-VALS()":

```
'
    INIT-VALS() -- INITIALIZE STD DEVS, DEF FNS

65100 GOSUB 60000' INIT-SSVS()
65102 GOSUB 60800' RESET-KEYBOARD()
65104 GOSUB 64000' DEFINE-FUNCTIONS()
65108 GOSUB 64500' INIT-IO-VALS()
65112 GOSUB 60020' INIT-CHAR-SETS()
65148 RETURN
```

Oh great. Another list of subroutines. This is FORTH-style programming at its finest: it's called "factoring", where you break down a large set of tasks into individual routines and call each one separately. It's slower, but this is initialization, so it's only done once. And it does make the code a little easier to understand, at least for me while I was developing it!

"INIT-SSVS()" at 60000 initializes the "Search State Variables" for various string search subroutines built into SKELETON/BAS. See TRS8bit March 2023, page 31 for the article that describes the string scanning system. Line 65112, "GOSUB 60020", calls "INIT-CHAR-SETS()" which also sets up properties used in string scanning.

"RESET-KEYBOARD()" should probably have been called "OPEN-KEYBOARD()", because that's technically what the subroutine at line 60800 does, so that any calls to "getc()" which look for characters from the keyboard buffer will start correctly.

The "DEFINE-FUNCTIONS()" subroutine starting at line 64000 sets up all the DEF FNs that we've discussed in previous articles.
"INIT-IO-VALS()", ie "65108 GOSUB 64500", sets up a number of constants and variables in SY() array elements pertaining to IO. Here it is:

```
'
    INIT-IO-VALS()

64500 SY%(1,0)= FN KY%( 0): SY%(2,0)= FN SC%( 0): SY%(3,0)= FN SC%( 0) ' DEFAULT DEVS FOR STDIN, STDOUT,
STDERR
64502 SY$(15,0)= "CMDLINE/SH" ' SEQ FILE CONTAINNG CMD LINE, STD FILE ASSIGNMENTS
64504 SY$(4,0)= FN NL$( 0): SY$(5,0)= CHR$( 27): SY$(6,0)= ".": SY$(7,0)= "." ' NL, EOF CHARS
64506 SY%(6,0)= -1 ' EXIT AFTER USAGE()
64508 RETURN
```

In line 64500, "stdin" is set to its default, the keyboard; "stdout" and "stderr" are set to the screen. After that, we save the default name for the file containing the output of the shell program, "CMDLINE/SH". We also set default newline and end-of-file characters. Line 64506 means that if the command line contains an option to print the "usage" help information for this program, as in

```
COPY -?
```

then the program will print out its help text and immediately exit.

Now back to the main initialization routine. Next is "INIT-CMD-LINE()", and here's its listing:

```
'
    INIT-CMDLINE() -- READ CMDLINE/SH,
            PARSE CMD LINE & PROGRAM NAME,
            ARGC, ARGV

65149 '

    1. TRY TO READ "CMDLINE/SH"

65150 VR%(0)= 0: ON ERROR GOTO 65500
65152 OPEN "I", 1, SY$(15,0)
65154 ON ERROR GOTO 0'

    2. IF CMDLINE FILE EXISTS:
```

```
65156 IF VR%(0) THEN VR%(0)= 0: RETURN ' NO IT DOESN'T

    3. READ SY$(0..3,0)

65158 FOR X9%= 0 TO 3: INPUT#1, SY$(X9%,0): NEXT: CLOSE#1'

    4. PARSE CMD LINE IN SY$(0..,0) INTO AV$(0..)
       ARGC -> SY%(0,0)

    4A. PROGRAM NAME -> AV$(0)

65160 SY%(0,0)= 0: VA$(0)= SY$(0,0): GOSUB 5160: IF NOT SY%(11,0) RETURN
65161 GOSUB 5600: IF NOT SY%(12,0) RETURN
65162 AV$(0)= VR$(0) '

    4B. ARGS -> AV$(1..N)

65170 GOSUB 5600: IF NOT SY%(12,0) RETURN
65172 SY%(0,0)= SY%(0,0) + 1: AV$(SY%(0,0))= VR$(0): IF SY%(11,0) THEN 65170
65174 RETURN
```

First (Comment "1.") we try to read CMDLINE/SH. An ON ERROR GOTO is used to "trap" the occasion where SKELETON/BAS tries to OPEN "CMDLINE/SH" only to find it doesn't exist—for example, if we tried to run SKELETON/BAS or a program like COPY/BAS that's built with it *without* using SH/BAS to invoke it.

Next ("2. IF CMDLINE FILE EXISTS:" and "3. READ SY$(0..3,0)") we continue normally, opening CMDLINE/SH and reading its contents, the four lines that contain

Line 1 command name and any arguments
2 stdin filename or device, or "" (empty)
3 stdout filename or device, or "" (empty)
4 stderr filename or device, or "" (empty)

Then we try to "parse" the first line in CMDLINE/SH. For example, if we had a COPY/BAS program that was invoked by SH/BAS with the following command

```
COPY/BAS <FILE1 >/DEV/PRINTER –OPTION1
```

then CMDLINE/SH would contain these four lines:

```
COPY/BAS –OPTION1
FILE1
-3
```

(The fourth line is empty.) Notice that the first line contains the original program specified to SH/BAS, which is "COPY/BAS", and its "option", "OPTION1", but the "file redirection" elements "<FILE1" and ">/DEV/PRINTER" have been "stripped out" and replaced in the appropriate lines in CMDLINE/SH. (In a later article, I'll describe how SH/BAS does this.) The initialization code in lines 65160..65174 read just that first line of CMDLINE/SH—in this example, "COPY/BAS –OPTION1". "4A", lines 65160..65162, use the String Scanning algorithm first to skip past any "whitespace" in Line 1 (GOSUB 5160 in line 65160), and then to "gather" any non-whitespace information (GOSUB 5600 in line 65161). This text—in our example, "COPY/BAS", will be placed in AV$(0), which is SKELETON/BAS's equivalent of C's argv[0] (line 65162).

In the section of INIT-CMDLINE() labeled "4B", we parse out any remaining text items (like "OPTION1") into the remainder of the command line argument list AV$(). The subroutine at 5600 pops out one whitespace-delimited "word" after another, reporting a -1 (T) in SY%(12,0) if it returned anything "meaningful'—ie, something other than whitespace or end-of-line (line 65170). The "argument count" (similar to C's argc) in SY%(0,0) gets incremented, the next element in AV$() gets that content, and we go back to look for any more arguments on the command line (line 65172).

INIT-USR-PRGM()—GOSUB 65200—is just a stub here. But if your program wants to set up some of its parameters based on the command-line arguments 1..SY%(0,0) now stored in AV$(), it could do that here.

INIT-STD-FILES—GOSUB 65250—will set up stdin, stdout, and stderr. This is what it looks like:

```
'
      INIT-STD-FILES() -- STDIN, STDOUT, STDERR REDIRECT

65250 VR%(0)= 0: FOR X7%= 1 TO 3: IF LEN( SY$(X7%,0)) < 1 THEN 65290 ' ONLY ATTEMPT TO OPEN NAMED FILES
65252 IF X7% = 1 THEN SY%(4,0)= -1' DEFAULT BYTE-STREAM INPUT FOR STDIN FILE
65254 IF INSTR( SY$(X7%,0), "-") > 0 THEN VR%(1)= VAL( SY$(X7%,0)): GOTO 65280 ' I/O REASSIGNED
65255 X6%= VAL( SY$(X7%,0)): IF X6% > 99 THEN VR%(1)= X6%: GOTO 65280 ' STD SLOT RE-ASSIGNED (IE, "3>2",
WHERE X7%=3=STDERR IS ENCODED AS "200")
65256 VA$(1)= MID$( "GPO", X7%, 1): VA%(1)= 0: VA$(0)= SY$(X7%,0): VA%(0)= 0
65257 X9%= INSTR( VA$(0), "+"): IF X9% THEN VA$(0)= LEFT$( VA$(0), X9% -1): VA$(1)= "A"' STDOUT/ERR APPEND
MODE SIGNIFIED BY "+" AFTER FILENAME
65258 GOSUB 60500
65280 SY%(X7%,0)= VR%(1) ' ASSIGN FD# TO STD FILE#
65290 NEXT: RETURN
```

Firstly, line 65250, we loop through each of the 3 standard files described in the last 3 lines of CMDLINE/SH and now stored in SY$(1..3,0). If that line is a blank (LEN(SY$(X7%,0)) < 1) we just skip it: the default for this standard file has already been set.

Line 65252: if we're presently looking at a stdin *file*, we want its I/O to be "byte-stream" instead of the default "text-stream" for the keyboard. See the discussion of getc() in a previous article for the difference between the two stream types.
Line 65524: we next test the text for a negative number, such as "-3". Such a number will at least tell us that this standard file is not a disk file, but a device (like a printer). We assign that device number to this "standard slot" in line 65280, and continue with the next standard file.
Line 65525: next, we test for positive numbers. A figure greater than 99 indicates a "standard slot re-assignment". What's that? For example, if the program was invoked from SH/BAS will the command line

```
COPY/BAS <FILE1 >/DEV/PRINTER 3>2
```

then SH/BAS would have created CMDLINE/SH with these lines

```
COPY/BAS
FILE1
-3
200
```

which tell COPY/BAS to assign FILE1 to stdin, the printer to stdout, and any error output from stderr to also be written instead to the printer (stdout). "3>2" is a "standard slot re-assignment": it re-assigns stderr to stdout. Line 65525 simply passes the number on to the current "standard slot" SY%(X7%,0) via VR%(0) in line 65280.
Line 65256: if we've gotten here, then this must be a disk file. In that case, we need to assign it an "OPEN type" or mode, depending on whether it's standard slot 1 (stdin), 2 (stdout), or 3 (stderr). For stdin, that should be "G" [getc()]; for stdout it should be "P" [putc()]; and for stderr it should be "O" [BASIC's "O" mode]. These mode values get set via the MID$("GPO"). After that, we set up some default arguments before we prepare to call fOpen().
But before we do, it's possible that one of the files was to be opened in "A"ppend mode. Here we want to append FILE1 to FILE2:

```
COPY <FILE1 >>FILE2
```

and SH/BAS would output CMDLINE/SH as

```
COPY/BAS
FILE1
FILE2+
```

The "+" added to FILE2 indicates that it should be opened in "A"ppend mode. Line 65527 tests for this, and corrects the Open Type argument for fOpen() to "A" (where it might originally have been "P" or "O"). It also strips the "+" off the filename before it's passed along as an argument to fOpen(). After all this, line 65258 calls fOpen() via GOSUB 60500.
In all cases (files, devices, or "standard slot re-assignment") the standard slot itself--SY%(slot#,0) – gets the file number, device number, or slot re-assignment number in line 65280, and INIT-STD-FILES() is done.

The final item to initialize, OPTIONAL-PACKAGES(), is normally a stub as well, but if the user wants some last-minute initialization to occur, especially if they are developing a set of Software Tools that use the same initialization, it should be written in lines 65300..65497.

With that, our initialization returns to line 65048, which immediately jumps to the user program starting at line 40000. As an example, here's the "user program area" code from COPY/BAS:

```
39999 '

    COPY -- COPY INPUT CHARACTERS TO OUTPUT (ST P. 8)

    "WHILE (GETC( C) != EOF)"

40000 GOSUB 2100:
        IF NOT VR%(0) THEN GOTO 50000

40009 '
            "CALL PUTC( C)"

40100       VA$(0)= VR$(0): GOSUB 2300:
            GOTO 40000
50000 '
    "STOP"

59990 CLOSE

59994 '
    RETURN TO SHELL

59995 RUN"SH/BAS"
```

Next episode: the "shell" program SH/BAS. If you ever want to comment on this series of articles, you can find Vince Otten in any of the TRS-80 or trs80gp groups on Facebook.


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


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

IN MAV'S HISTORY WORKSHOP

What *Was* The Tandy 10?



by Ian Mavric

As collectors and retro computing enthusiasts become interested in the Tandy's computers, one has been considered the holy-grail of vintage Tandys... The Tandy 10. The what? Many people haven't even heard of this machine or know what it looks like. Recently one of these extremely rare business computers was located so let's take a look at it.

WHY IS THE TANDY 10 A THING?

Let's have a little look at the timeline. It's well known that the TRS-80 Model I made it's debut in August 1977 and Radio Shack had to spend nearly the rest of the year playing catch-ups as the demand for the machine was so great. Things were not idle at the Shack and during the last months of 1977 the finishing touches were put onto Level II Basic by Microsoft for release in early 1978, as well as the Expansion Interface, disk drives and TRSDOS 2.0 Which when you

consider the short time and instant popularity of the Model I, all resources available were sent to producing the Model I and it's accessories.

Knowing they were onto a winner by early 1978 and needing information on what types of microcomputers were being bought and used by different user bases, Radio Shack set up an interesting side hustle. Selling many different brands of computers (just not Apples or Commodores) under the name of "Tandy Computers" to better understand the marketplace. They even had a special catalog printed.



While they didn't carry anything from Apple, Commodore or IBM, they did carry systems from Imsai, Intercolor, Processor Technology,

Lear Siegler, and SWTPC to name a few. Although the project didn't last very long (it was wound up by the end of 1979) Radio Shack did gain important buyer information which was mainly that the Model I was a good start but business customers were prepared to pay a much higher amount for more powerful business orientated systems. Thus plans were drawn up for what would eventually be the TRS-80 Model II. The problem here was that the Model II wouldn't be ready for another 18 months, and Radio Shack in the meantime thought it would be a good idea to have a business oriented machine to sell to buyers with deep pockets and accounting systems to run, ledgers to general and words to process...

It's not exactly clear how Radio Shack came to be friends with Applied Digital Data Systems (hereafter known as ADDS) but a deal was signed that they would sell a special version of the ADDS System 70 upgraded to Tandy 10 specifications with a new ROM among other things. They would go on sale for $US9995.00 and would only sell around 50 of them, hence why they are possibly the rarest Tandy computer that was actually released.



LET'S SEE WHAT YOU GOT FOR YOUR $US9995.00

- An 8080 processor (an Intel CPU which would be developed by Zilog into the Z80)
- 48K of RAM
- A 51 key keyboard with an 11 key numeric keypad and 24 function keys
- An 80 by 24 text display, supporting upper and lower case, blinking, underlining, reverse video, and half intensity
- Two double-sided single-density 8 inch floppy drives
- Extended "Dartmouth BASIC"
- ADOS (ADDS Disk Operating System)
- And it was built into its own metal desk, described as "Desk Packaging"

Looking a little more closely, the design of the computer seems to pre-date the TRS-80 Model I. The 8080 CPU has a clock speed of 2MHz but is less sophisticated than the Z80 which came out two years after the 8080 (Z-80: 1976, 8080: 1974). Having said that the 8080 was a good workhorse for early CP/M systems. 48K Ram was also a pretty standard memory size for the time, remember that the Model I and Apple][both had maximum memory configurations up to 48K Ram. 48K was common on early computers because since 8-bit CPUs can address up to 65535 memory locations, at the time it

was normal that the ROM took some of that space away (eg. in the case of the Model I the ROM was 12K and on the Model III the ROM was 14K). It wasn't until the Model II the idea of running in all RAM mode without the need for space set aside for ROM was implemented and further refined in the Model 4/4P.

The keyboard and screen are interesting and surprisingly better than most computers of the time. Most people don't know it but ADDS used to be into the business of making terminals for connection to mini and mainframe computers. In fact the Tandy 10 is really an ADDS terminal mounted to a table, with a microcomputer in it's own box under the desk on the left-hand side. True to ADDS skill in building smart terminals for mainframes the video display is totally deluxe with lowercase characters, reverse video, and other nice features like underline, blinking, and half-intensity characters, in an 80x24 screen. In it's normal booted up mode the screen is white with black characters. Compare this to any popular microcomputer available at the time: None of the trinity of the C= Pet, Apple][, or the Model I had lowercase, and the first two only had 40x24 character screen. The Tandy 10 display and keyboard are all class.

The disk drives were 8in floppy disk drives which were standard for 'serious' computers. Smaller and home microcomputers were using 5.25in disk drives though they did eventually make their way out of the 'hobbyist' world into the mainstream which made 8in drives obsolete. The Tandy 10 drives were double-sided single density so they hold about 970K of information each. (By contrast the TRS-80 Model II which came out at the end of '79 had single-sided single density drives holding around 486K each.) Another hint to the age of

the design of the Tandy 10/ADDS 70 is that the disk drive controller is made up of discrete TTL chips, not a floppy disk controller IC like you find in the Model I and II. This, along with the 8080 CPU makes me think the design of the ADDS 70 goes back to maybe late 1975 or early 1976.

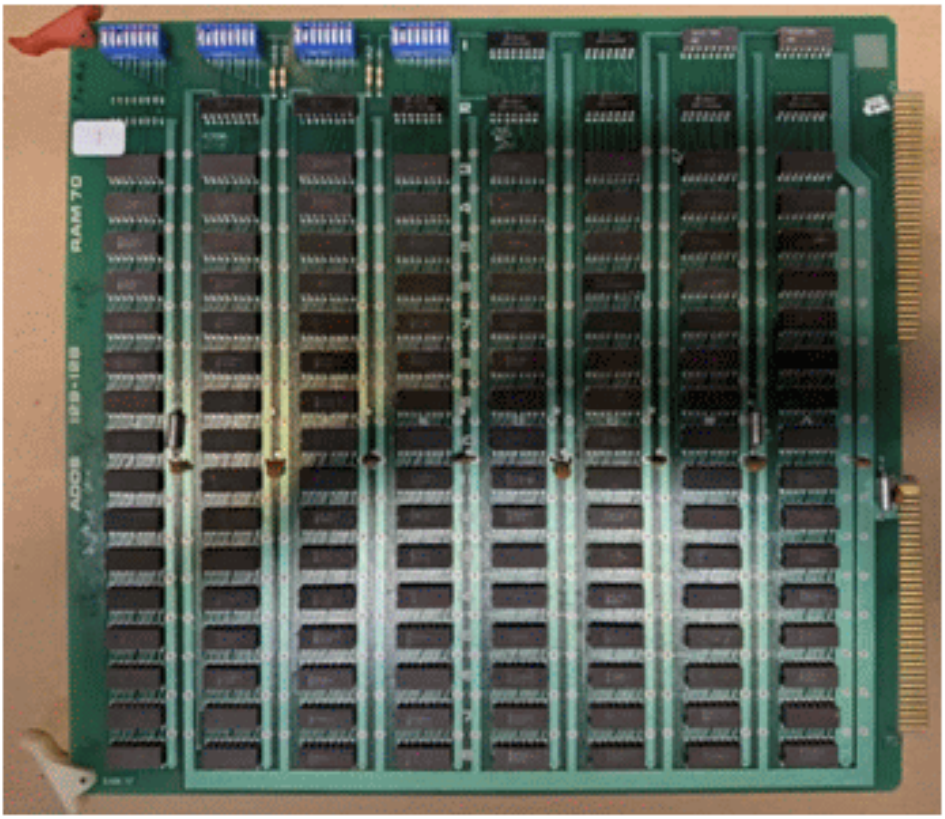


Photo: 16K Ram card consists of 8 rows of 16 1K SRAM chips = 128 chips!

The Ram cards are both interesting and scary. Each 16K Ram card (there are three of them in the system) has eight tows of 16 1Kx1Bit SRAM chips, ALL SOLDERED IN. Thank goodness not to far in the future dynamic ram in chip sizes of 4K, 16K and 64Kx1Bit were invented.

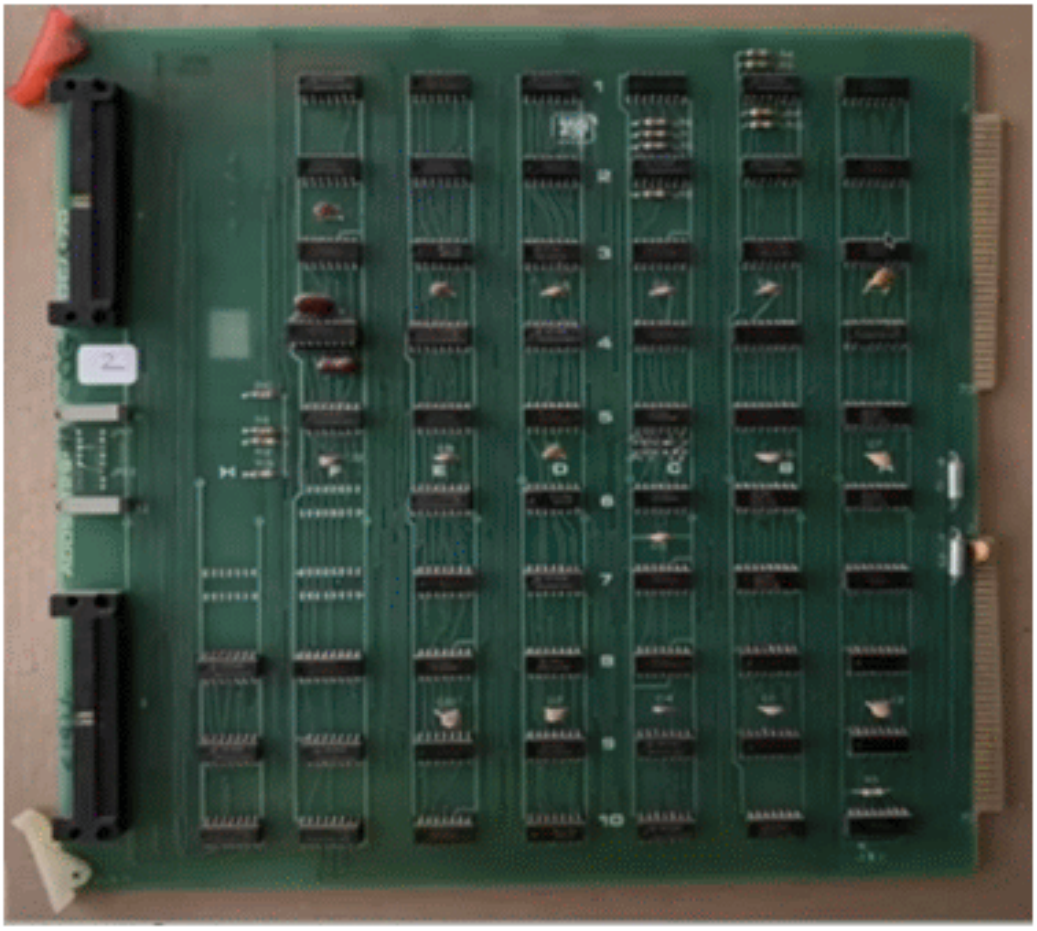


Photo: Floppy disk controller made of discrete logic gates

PETER CETINSKI'S TANDY 10 SYSTEM

Until the middle of 2024 the whereabouts of any Tandy 10 systems was a complete unknown. In the previous 5 years a beaten up Tandy 10 VDU/keyboard section was on eBay in very poor condition and the asking price was insane. Without the rest of the computer it was hard to put a price on it however around 5% of it's asking price of $US30K was the agreed value of the item:



Photo: eBay Tandy 10 beaten-up console was the butt of jokes for years

During 2024 Pete was alerted to the existence of an almost complete Tandy 10 system in California, including manuals and software. A deal was reached and the system packed into wooden shipping crate and made it's way across the USA to New York. This Tandy 10 will eventually be restored (at the time of writing it has yet to be powered up) so that a proper study of Tandy 10 can be undertaken.



Photo: Pete's Tandy 10 system including manuals and disks

FACEBOOK TANDY 10 GROUP - PLEASE JOIN!

Peter has also set up a Facebook group for the Tandy 10. The idea is a place for people who like to share their experiences about the T10 and any knowledge or media related to the system, and document his restoration of his own system. It also is a place marker starting point where hopefully anyone who find another Tandy 10 and does a Google or FB search can quickly find the group and advise of their existence. With 50 systems made and sold there is every likelihood that at least a few more are stored away in warehouses or junk rooms around the country. To find the FB here is the link:

https://www.facebook.com/groups/1101598494873167

EMULATION ALREADY? WHY NOT....

The amazing thing about Peter's collection is that there was enough information included to allow an emulation module to be written quite quickly by George and Peter Phillips, and will be included in the next release of TRS80gp. With an emulator, the contents of the boot ROM decoded, and images of the floppy disks at hand, you can actually drive the Tandy 10, even though no working ones are known to exist. The speed at which the guys were able to get to the point of a working emulation was both amazing and rapid. Here is what a working Tandy 10 would look like. It starts with a boot screen with black text on a white background:



The screen can be inverted back to a more common white text on a black background and used like a normal CP/M type computer:

```
48K TANDY VERS 2.0

A)DIR
A: COPYDBL.COM  TED.COM      DEBUG.COM    UNLOCK.COM   SUBMIT.COM   F80.COM
A: L80.COM      M80.COM      CREF80.COM   STAT.COM     BASIC.COM    VERIFY.COM
A: PIP.COM
A)DIR B:
B: COPYDBL.COM  TED.COM      DEBUG.COM    UNLOCK.COM   SUBMIT.COM   F80.COM
B: L80.COM      M80.COM      CREF80.COM   STAT.COM     BASIC.COM    VERIFY.COM
B: PIP.COM      VID.BAS
A)BASIC
BASIC Rev. 5.0
[ADDS*BASIC SYSTEM/70]
Copyright 1977 (C) by Microsoft
22618 Bytes free
Ok


10 ?"HELLO"
RUN
HELLO

HELLO
```

Some of the features of the ADDS terminal which weren't common in microcomputers for many years includes underline, blinking and half-intensity characters, as well as reverse video:



At the time of writing the Tandy 10 implementation in TRS80gp is very new and as more of the characteristics of the T10 make themselves known and understood, the emulation will be updated.

For those who want to try it out, the place to find TRS80gp is, as always: http://48k.ca/trs80gp.html

WHAT IS RARE?

Rarity is a word often overused these days to described everything that is hard-to-get. Some rare cars are easy to get, for example the DeTomaso Pantera (an Italian super-car powered by an Australian-made US design Ford 351ci "Cleveland" engine) only sold 7260 units but if you want one and have enough money you can buy a used one tomorrow. Rarity, to my mind, implies something that was sold to the public, in small numbers, and was expensive. The Tandy 10, until we find information to the contrary, sold around 50 units. By contrast some things we consider extremely rare by today's standards sold many more units than the Tandy 10:

1967-72 Mazda Cosmo 110S wankel engine sports car: 1176 units

1969-70 Ford Mustang Boss 429 muscle car: 1350 units

Yamaha GX-1 analogue polyphonic synthesiser: around 100 units

Tandy 5000MC Micro-Channel 386 computer: around 100 units

1976 Apple 1 micro computer: around 200 units

CONCLUSION

This is where I would normally ask the questions of do you need one in your collection? The point is a little moot until (or if) some others

show up. Some would say it would be easier to find an ADDS System 50 or 70.

Some of the reasons for the failure of the Tandy 10 to pique the interest with buyers were no doubt it's price and relatively dated specs, but also the extremely heavy desktop design made servicing and repairs a nightmare. The specifications for the Model II had been laid down by the end of 1978 which included a faster 4MHz Z80A CPU, 64K memory, and a design which lent itself to being easily picked up (if you ate a whole box of Weetabix for breakfast) and transported to a Radio Shack store or Computer Centre for repairs. By the time the Model II was released in late 1979 a 64K Model II ($US3899) and a triple drive expansion unit ($US2350) you got more CPU speed, RAM, and the same disk storage capacity as the Tandy 10 for only $US6249. If you added the nice Model II system desk ($US369) to put it all on you still came out at roughly 2/3rds the cost of the Tandy 10.

Stay tuned I might write a future article when the restoration of Pete's Tandy 10 is complete and up and running, to chronicle that process and get some thoughts on what it's like to operate.

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.

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)

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

The amazing

"FreHD"

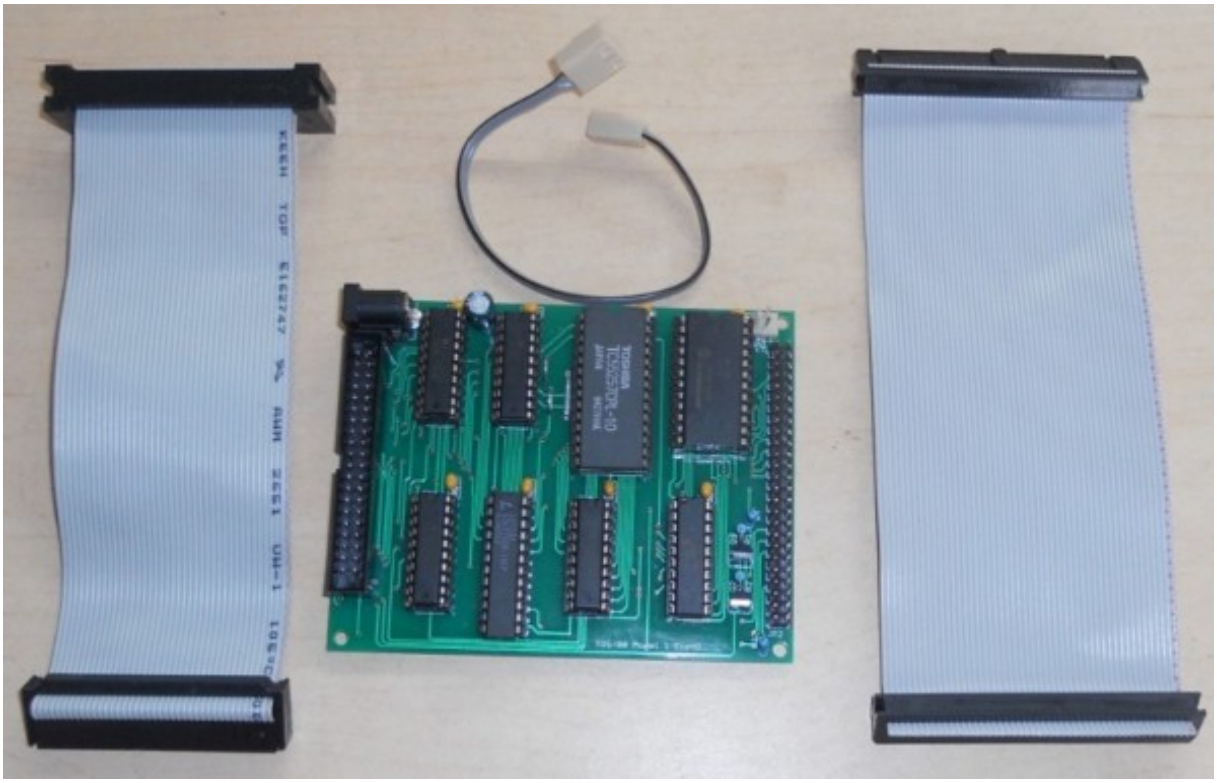


-Emulates a TRS-80 hard drive, but faster than any hard drive!
-Works with your favourite DOS (LS-DOS, LDOS, CP/M, Newdos/80 2.5)
-Uses SD card for storage medium
-Bonus free Real Time Clock function!
-Designed in Belgium and proudly built and shipped from Australia
-Kit form or fully assembled

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”

EBAY.CO.UK 'BARGAINS'

HERE ARE JUST SOME OF THE MORE INTERESTING/UNUSUAL TANDY EBAY LISTINGS, SINCE OUR LAST ISSUE.
WERE YOU ONE OF LUCKY ONES?



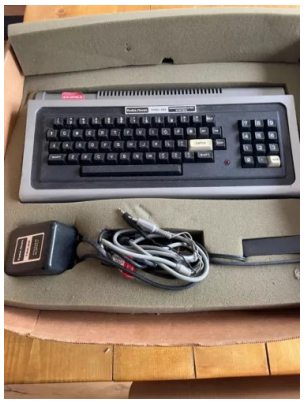
£1



£130



£230



£124



£139



£10
NS



£142



£387



£100



£25



£50

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



E.T. FONEHUME

HI DUSTY. JUST TO WISH YOU & DEE A VERY MERRY XMAS AND A HAPPY, TRS80BIT FILLED, NEW YEAR (PLEASE)! HERE'S SOMETHING I'VE HAD IN MIND FOR A WHILE WHICH FITS NEATLY DOWN TO A ONE-LINER RATHER WELL. I'VE RESTRICTED IT TO INTEGERS JUST FOR SPEED BUT WITH THE TURBO OPTION IN 'TRS80GP' EVEN DOUBLES PERFORM REASONABLY WELL!

```
0 DEFINTA-Z:INPUTN:I=0:T=2:FORX=0TO1STEP0:J=INT(N/T):K=J*T:IFN=K
THENPRINT N;"IS NOT PRIME.";T;"IS THE LOWEST FACTOR":ENDELSEI=I+
1:L=T*T:IFL>N THENPRINT" PRIME":END:ELSET=(I*2+1):NEXTX
```

```
READY
>RUN
? 121
 121 IS NOT PRIME. 11 IS THE LOWEST FACTOR
READY
>RUN
? 127
 PRIME
READY
>_
```





As always, of course, I'm on the lookout for any software for the U.K.'s **Aculab 'Floppy Tape'**. There were versions specifically 'zapped', (by Laurie Shields), for use on the AFT, of Electric Pencil, Scripsit, Zen, Level III BASIC and VisiCalc. There was also the **enhanced BASIC, XBAS** (from Aculab). It's so disappointing to think that they have all been lost!

Please email me, at my usual address, if you can help.

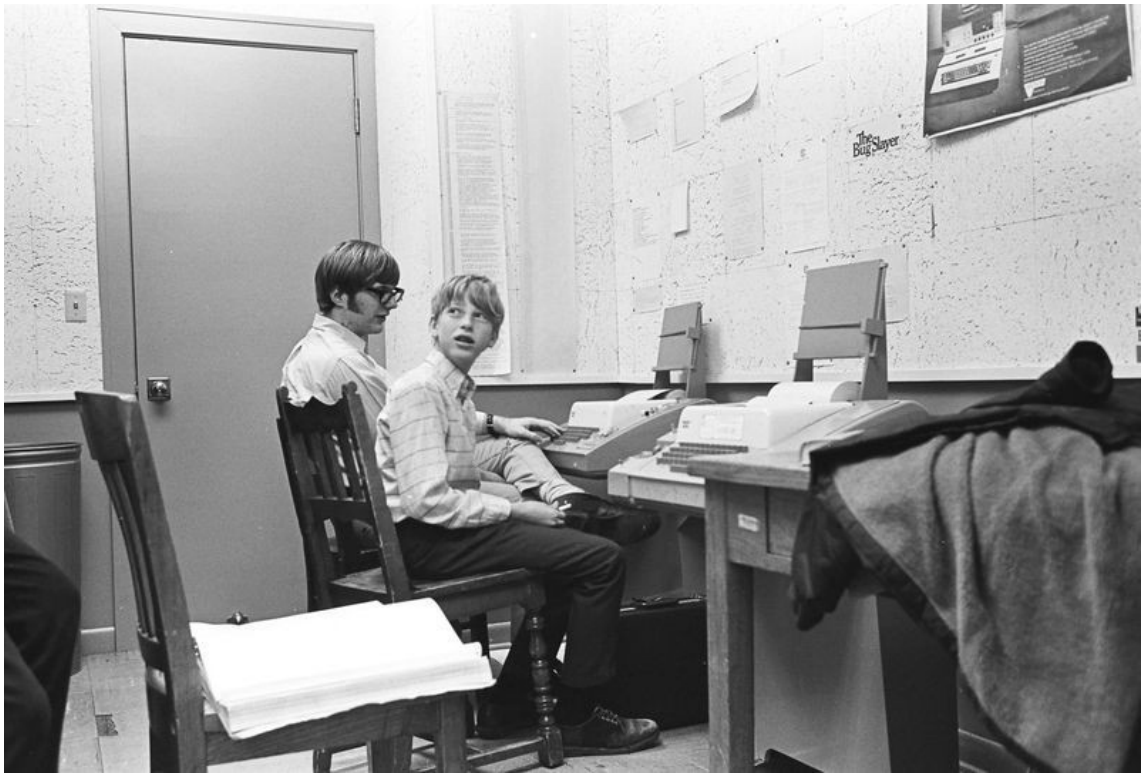
It started here!
Michel Talbot

13-year-old Bill Gates[1] with classmate and Microsoft co-founder Paul Allen[2] writing programs on a Teletype Model 33 ASR terminal[3] connected to a remote time-sharing General Electric (GE) mainframe computer in the computer room at Seattle's private Lakeside School (1969).

Gates wrote his first computer program on this machine, an implementation of tic-tac-toe that allowed users to play games against the computer, it was written using an interactive interpreter developed at Dartmouth College called BASIC (Beginners' All-purpose Symbolic Instruction Code[4]).

In 1975, the language inspired Gates and Allen to write in Intel 8080 assembler code[5] the first microcomputer version of BASIC using a modified version of Allen's Intel 8008 emulator running on Harvard's PDP -10 time-sharing mainframe, selling the code under their newly formed company name Micro-Soft. The 4K Altair BASIC[6] interpreter was released by MITS in punch tape format for the Altair[7] 8800 shortly after the machine itself, immediately cementing BASIC as the primary language of early microcomputers.

1) https://en.wikipedia.org/wiki/Bill_Gates
2) https://en.wikipedia.org/wiki/Paul_Allen
3) https://en.wikipedia.org/wiki/Teletype_Model_33
4) https://en.wikipedia.org/wiki/BASIC
5) https://en.wikipedia.org/wiki/Intel_8080
6) https://en.wikipedia.org/wiki/Altair_BASIC





DOES THIS BRING BACK MEMORIES? *ED*

The Men Behind the TRS-80s

by Jonathan Erickson

It had all the ingredients of a top-secret project—which, industrially speaking, it was. There were labs that moved from place to place; a less-than-adequate number of people involved (all sworn to secrecy); nights, weekends, and holidays spent in laboratories poring over schematics and programs; and deadlines that seemed impossible to meet.

Steve Leininger



When Steven Leininger joined Radio Shack in the fall of 1975, the company was still trying to decide whether to sell a microcomputer. No one at Radio Shack seemed to know what kind of computer it should be or even what it should look like, much less what it should do.

Today, six years and countless TRS-80s later, Radio Shack's Model I has found a comfortable niche in computer history and Steve is director of product development for Radio Shack's computer division.

What happened in the interim? Simply, Steve designed and built a personal computer. With, of course, accompanying software. In less than a year. And it has been the leader of the microcomputer pack from the start.

Steve might disavow the comparison, but based on the popularity of the Model I TRS-80, you might argue that he's the "Calvin Klein" of microcomputers. Along the way, he opened the door to the second generation of popular-computer engineers—to people like Dale Chatham, who developed the TRS-80 Color Computer. But that comes later.

In the Beginning

It began with an interest in electronics.

"When I was ten years old and growing up in South Bend, Indiana, my mom bought an electronics kit that didn't have an instruction manual," Steve explains. "I had to figure out everything—what the parts did and how to put them together. That's really when I got into electronics."

By the time he entered high school, his interest in electronics was primarily devoted to audio. During the day he studied electronics, and after school he experimented with practical applications (electric guitars and homemade amplifiers). The decisions that plague most college-bound students—where to go and what to major in—posed no problem. "Where" was hometown Purdue University. "What" was electronics.

Then his focus shifted. Leininger recalls, "I was interested in audio, so all of my preparation was in analog electronics. It wasn't until the last semester of my junior year that I took my first digital course. I was at the Engineering Library and picked up a magazine that had an advertisement for an Intel computer. I looked at that ad for a while and said to myself 'that's going to be a big thing'; then I signed up for some more digital courses. You might say it was love at first byte."

In 1974, with bachelor's and master's degrees in electronic engineering, Steve headed west to the heart of the emerging microcomputer industry—California's Santa Clara Valley. There he divided his time between designing SC/MP (National Semiconductor's early microprocessor) and communicating long distance with Susanne, the girl he left behind. "Six months of calls back to Indiana just about broke me. It was either get married and have her move to the West Coast, or move back to Indiana." She joined him on the West Coast.

Discovered at Last

To earn a little extra cash, Steve began moonlighting at a computer store. One fateful evening a couple of Texans from Radio Shack walked in, and—as luck would have it—Steve was behind the counter.

"Steve's computer-store experience was extremely valuable to him and to us," says Jack Sellers, one of the Texans who walked into the computer store

that night. "He knew what the hobbyist market wanted and yet he was very knowledgeable in the technical areas of computers. It was an exposure that computer people didn't have then."

The Leiningers relocated to Texas, Steve as project engineer for Radio Shack.

The Early Stages

Before Steve came on the scene, Radio Shack's early idea was to market a kit that the customer could assemble. The decision to scrap the kit idea and sell a "ready-to-go" computer set the course of personal computing to what it is today.

"That really was an important decision," Leininger notes. "The main reason we decided to forget about the kit was we felt it was too difficult a project. Can you imagine the instructions that would go along with a self-assembled Model II? Some customers were having trouble assembling '100-in-1' kits, let alone something as complex as a computer."

Dale Chatham



But the hardest part was deciding what the computer should be, Leininger remembers. "For instance, there needed to be some way to input information. We chose a standard typewriter keyboard because we felt the customer would be less intimidated. Then we needed some way to output information. We chose a TV-like video because the customer would be familiar with it and it was cheaper than a printer. Next, we needed a way to store information. Cassette tapes would work, they were cheap, and the customer knew what they were. We wanted BASIC built into the ROM and we wanted plenty of RAM for user programs. When we established these criteria, the actual design of the machine seemed relatively simple."

> It's rare to find someone who has that almost total grasp on both hardware and software.

Final Push

But the battle wasn't over. In February 1976, Leininger and Sellers (now general manager of Tandy Business Products) decided to go with a Z80 microprocessor and dynamic RAM. The changes would have to be incorporated into the design before April—when they would meet with Charles Tandy, chairman of the board of Tandy Corporation, and try to convince him that the microcomputer could sell.

"At that stage in the development the hardware was never a problem," Steve says. "It was the software that gave us fits. We hired some guy to work on the software, but after a few months he disappeared and we never heard from him again. I was supposed to do the hardware, but now I had to write the software, too."

Days before Tandy's first look at the computer, the BASIC interpreter still wasn't working. They decided not to leave the factory until the computer ran.

"It was after 2 a.m.; both of us were so tired we couldn't see straight because we'd been at the factory about two days," says Sellers. "We knew we were down to the wire and had just about given up. Steve was trying to debug a program and couldn't get the answer. Finally, he took a listing into the other room. I was working on something else and all of a sudden I heard him yell 'Eureka!' He came running into the lab, made the change, and it worked beautifully."

When Tandy came to the factory a couple of days later, Leininger and Sellers had a simple BASIC game for him to try. Tandy apparently liked what he saw and ordered 3000 TRS-80s built with the understanding that if they didn't sell, Radio Shack stores could use them for inventory control.

That was the good news. The bad news was that the first TRS-80 had to be delivered to customers no later than August of the same year—barely three months away. But with Charles Tandy backing them 100 percent, Leininger and Sellers met the deadline—with hours to spare.

"I don't think anyone can really appreciate what Steve did in the early days," says Sellers. "It's rare to find someone who has that almost total grasp of both hardware and software. I don't think there was anyone else that could have done the job under the conditions he worked."

An "Idea Man"

All of that's changed now. Carpeted offices and modern labs have replaced converted closets, and while he still tests a few circuits, Leininger is basically an "idea man" who develops and evaluates new products. Since the Model I, Steve has directed the development of the Model II, the Model III, and the Color Computer, although he didn't work them from start to finish as he did the Model I.

"I think I'm still proudest of the Model I because I did most of that by myself," he says. "None of what we're doing today would ever have taken place if it hadn't been for that computer. And it was great doing a job that nobody else had done before—that no one thought possible. It was a real challenge, one of those once-in-a-

lifetime happenings that I hope I only do once."

A Fellow Engineer

If Steve is a first-generation popular-computer engineer, Dale Chatham is characteristic of the second generation. Since joining Radio Shack a little over two years ago, Dale's major effort has been the TRS-80 Color Computer, a computer that Leininger believes will sell as well as the original Model I.

"Dale was the right person to develop the Color Computer," Steve says. "He's a very good engineer. He's careful and does his planning the first time around."

And, like Leininger, Dale doesn't forget what the customer, the average user, wants. It's easy for him to remember what the computer hobbyist needs: that's how Dale got hooked on personal computing.

"Technically speaking, my background isn't computers," he explains. "When I graduated from UTA (University of Texas-Arlington) in 1975, computers weren't emphasized the way they are today. But I knew a little about computers and was interested in them so I bought a kit and built one in my spare time. Then I bought a Model I. . .now I have a Color Computer too – naturally."

He dropped out of college for a year when he couldn't decide between electrical engineering and genetic engineering and went to work on the assembly line for Texas Instruments. In retrospect, that experience paid off in three ways: (1) he was able to see how design problems influence production; (2) he decided that it would be easier to get a job as an electrical engineer than as a genetic engineer; and (3) assembly-line work wasn't all it was cracked up to be, so he'd better get back to school.

After he earned his degree, Dale returned to Texas Instruments – this time as an engineer designing military products. "The products were very limited," he says. "They definitely weren't consumer-oriented, unless, of course, you like to drop bombs on people."

When Dale answered Radio Shack's ad, he knew he wanted the position.

"The job Steve described was exactly what I had decided I wanted to do," Dale says. "I like working with consumer products. I like being able to go into a store and see something I designed. I feel lucky because every day I'm doing exactly what I want to be doing."

Two designers. Two men who are shaping the future. Steve and Dale are obviously not the sole members of Radio Shack's computer-design group. But they're representative of the "forward-looking" thinkers who have changed Radio Shack's company tag line from "the supermarket of sound" to "the biggest name in little computers." They're the men behind the 8-bit, and they're having a ball. ■

Editor's note: After this article went to press, Steve Leininger resigned from Tandy Corporation to start his own consulting business.

Jonathan Erickson lives in Forth Worth, Texas, and is a technical writer for a major electronics manufacturing and retail firm.

EXCITING MAIL ORDER DISCOUNTS

apple II 16K computer

Disk II with Controller $535. . .without $446



$1045

APPLE II PLUS 16K $1045
APPLE II PLUS 48K $1095
APPLE II PLUS 64K $1249

SOFTWARE

Adventure by Apple 29
Alien Rain by Broderbund 22
Apple Bug Debugger 29
Apple Graph & Plot 59
Apple Soft Utility Prog. Cass. by Hayden 29
Applewriter 65
Asteroid Field by Cavalier 19
Budgen Space Game Album 39
Budgen 3-D Graphics 39
CCA Data Mgmt. by Personal Software 84
Cyber Strike by Sirius 34
DAKIN 5 Programming Aid 3.3 75
Data Factory by Microlab 129
DB Master II by Stoneware 199
Desktop Plan II by Personal Software 169
DOS Tool Kit 65
Dow Jones Portfolio Evaluator 45
Flight Simulator by Sub-Logic 34
Forth II by Softape 45
Fortran 159
Hi-Res Football by On-Line 33
PASCAL Language System 425
Personal Filing System 84
Raster Blaster by Budgeco 24
Sargon II Chess by Hayden 29
Space Eggs by Sirius 25
Space Invader/Cosmos Mission 19
Supertext II by Muse 129
Visicalc II by Personal Software 169
ZORK by Personal Software 34

ACCESSORIES FOR THE APPLE II

ABT Numonic Keypad model A or B 114
ALS Smartarm 80 column board 299
Andromeda 16K Ramcard 169
Apple Clock/Calendar by Mtn. Comp. 225
Applesoft Basic Firmware Card 149
Arithmetic Processor #7811 by CCS 339

Large Selection Of Software Just CALL

WE SHIP FAST!

Asynchronous Serial #7710A by CCS 129
CPS Multi Function Card by Mtn. Comp. CALL
Centronics Card by Apple 179
Communication & DB25 Cable 179
DOS 3.3 Upgrade Kit 49
Graphics Input Tablet by Apple 639
Hayes Microcomputer II 299
Integer Basic Firmware Card 149
Intec II-10 System by Mtn. Comp. 339
Joystick for Apple 47
M&R Sup-R-Term 80 column board 319
Microsoft 16K Ramcard 189
Parallel Interface by Apple 139
Parallel Interface #7720A by CCS 99
Peymar Lower Case Kit model 1 or 2 49
Programmable Timer Module by CCS 159
Romplus w/o Keyboard Filter 159
Serial Interface by Apple 139
SSM A/D Serial/Parallel Interface 149
Symtec Light Pen 214
Versawriter Digitizer 239
Videx Videoterm 80 column card 319
Z-80 Softcard CP/M by Microsoft 319

MONITORS & PRINTERS

Amdek-Leedex 12" Color 369
Amdek-Leedex 12" B&W Video 100 129
Amdek-Leedex 12" Green Screen 169
NEC 12" Green Screen 239
Sanyo 12" B&W 255
Sanyo 9" B&W 189
Epson MX-100 FT CALL
Epson MX-80 FT CALL
Epson MX-80 CALL
Epson MX-70 CALL
NEC Spinwriter #5510 2590
Paper Tiger 445G 725
Paper Tiger 460G 1194
Paper Tiger 560G 1495
Qume Sprint 5/45 2499
Silentype Thermal w/Interface 349
Starwriter Daisywheel by C. Itoh 1750

TOLL FREE ORDER
1-800-854-2833

TO ORDER: Phone orders invited using Visa, Mastercard or bank wire transfers. Visa and MC service charge of 2%. Mail orders may send charge card number (include expiration date), cashiers check, money order or personal check (allow ten business days for personal or company checks to clear). Include 3% ($5 minimum) for UPS shipping, handling, and insurance. Calif. residents add 6% sales tax. Please include phone number. No COD or PO accepted. APO and FPO include 5% for postage. FOREIGN ORDERS include 15% handling — shipped air FREIGHT COLLECT only. Credit cards not accepted on foreign orders. All equipment is in factory cartons with manufacturers warranty. Opened products not returnable. Restocking fee for returned merchandise. Equipment subject to price change and availability. Retail store prices differ from mail order prices. WE SHIP THE SAME DAY ON MOST ORDERS!

VISA

master charge

MAIL TO: 1251 BROADWAY EL CAJON CA., 92021 (714) 579-0330

AUTHORIZED APPLE SALES & SERVICE

COMPUTER SPECIALTIES

DIV. OF COMPUTER METRICS INC.

Radio Shack's $399 TRS-80 Color Computer—The Colorful Way to say "Merry Christmas"

Make this Christmas really memorable with a gift the whole family can enjoy—a Radio Shack TRS-80 Color Computer. It's the exciting way to enter the age of computers—because you can pop in a Program Pak ™ for instant fun and games, or let the excellent 308-page manual teach you how to write your own programs.

Ready to Plug In and Use.

Just attach to your TV set, or add our high-resolution TRS-80 Color Video Receiver ($399). Then plug in an instant-loading Program Pak and you're ready to enjoy exciting pinball action, pit your skills against the computer in chess or checkers, command a starship through outer space or get down to earth and quarterback your own pro football team. (Optional $24.95 Joysticks required for some games.)

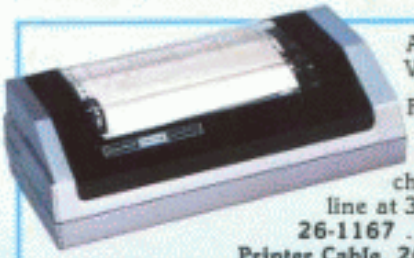
And since the Color Computer features brilliant color *and* sound, you can even compose music! There are also Program Paks to help you manage the family budget and tutor your youngsters in math.

Start With Our Standard Color Computer.

For $399 you get our TRS-80 Color Computer with BASIC language, 4000 characters (4K) of internal memory and an RS-232 interface for communications or a printer.

Expand With Lots of Exciting Options.

Extended Color BASIC and our 16K memory option makes complex graphics simple to program. Memory is expandable to a big 32K. Our new 156K Disk Drives allow lots of very fast "on-line" storage. We also have printers, plotters and telephone communications options, and much more. Find out all about Radio Shack's exciting TRS-80 Color Computer—just follow the reindeer tracks to your nearby Radio Shack store, dealer or Computer Center.



Add a Line Printer VII to Your System

Prints high-density graphic information, 80 or 40 characters on an 8" line at 30 cps. U.L. listed.
26-1167 . . . . . $399.00
Printer Cable. 26-3020. . . . $4.95

Radio Shack®

The biggest name in little computers ™

A DIVISION OF TANDY CORPORATION

Retail prices may vary at individual stores and dealers. Special order may be required.

Circle 61 on inquiry card.

Put a Color Computer Under Your Tree This Christmas!

VISA MasterCard

These Cards Honored at Most Stores

Mr. P. Minch B.A.,

Offset problems with a circular solution

If you own a micro that is programmed in hexadecimal but do not have access to an assembler then you have to calculate offsets by hand. Not many people know their hex addition tables by heart (for example A + B = 5 carry 1), so they use look up tables. The circular slide rule described in this article is much quicker to use.

Making the Rule

Cut out two circular discs of stiff card 10 cms. and 14 cms. in diameter. Divide each scale into 16 equal segments and mark the divisions as shown in Fig. 1, writing the appropriate numbers into each segment. Both scales have the hex digits 0 to F on them as their main inscription. They both also have the decimal equivalent of A to F where the scales meet as a reminder of the hex times 1 table. The outer edge of the outer scale has the 16 times table on it. The inner edge of the inner scale has the 15's complement on it marked as 0, 1, etc.

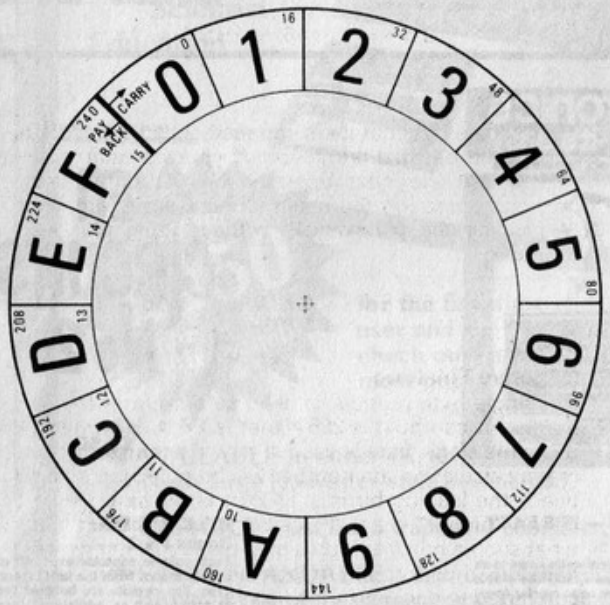
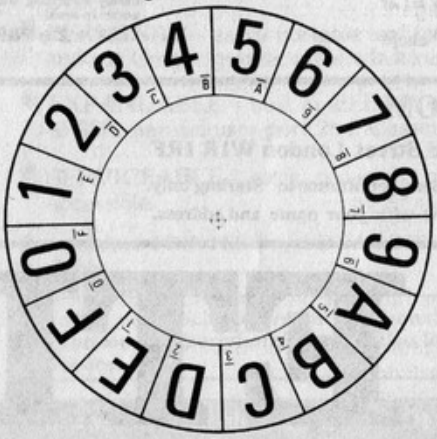


Fig. 1. The two scales must be fastened together with a small nut and bolt through the centre.



How to Use a Slide Rule

Now that the only useful scale on a normal slide rule is the one on the edge for drawing straight lines it might be worth reminding the very young or forgetful of us what a slide rule did. As the scales are rotated and the inscribed numbers counted off the distances, or angles in our case, are added or subtracted depending on the direction. Originally the scales were marked logarithmically so the effect was to multiply or divide the numbers, but our scales are linear to give addition and subtraction.

Now For How To Do It

TO ADD e.g. X + Y = ANSWER

Rotate the inner scale until its zero is under 1 on the outer scale. Clamp the scales together with your fingers and trace round **clockwise** until you find Y on the inner scale. The ANSWER is above it on the outer scale. If you pass the outer heavy black line carry 1 to the next column.

TO SUBTRACT e.g. X − Y = ANSWER

Find X on the outer scale. Rotate the inner scale until Y on it is under X. Clamp and trace round **anticlockwise** to zero on the inner scale. The ANSWER appears above it on the outer scale. If you pass the outer heavy black line pay back one to the next column.

Practice adding and subtracting with low decimal numbers so that you can easily check the result. Now check through the following examples with your slide rule. They are based on 8 bit bytes and assume 2's complement arithmetic, that is, if the most significant bit is a 1 the number is negative.

Addition

2 D
3 E+
6 B

D + E = B carry 1. The slide rule in Fig. 1 is shown set

TRS8BIT - VOLUME 18 ISSUE 04 - DECEMBER 2024 - PAGE 62 - TRS8BIT

HEX RULE OK!

for this addition. Remember to add clockwise or you will lose the carry 1. 2 + 3 + 1 = 6 you can do in your head of course.

Subtraction

```
  73
  56-
  1D
```

3 — 6 = D pay back 1. Fig. 1 shows the slide rule set for this calculation also. The pay back 1 makes the 5 into 6 giving 7 — 6 = 1.

Decimal to Hex Conversion

This is achieved by successive division by 16, the hexadecimal remainders forming the result. It is hear that you need the 16 times table on the outer ring of the rule. The following example shows the conversion of a double byte (double precision) number, $10{,}815_{10}$ to $2A3F_{16}$.

```
16) 10815
16)   675 r F
16)    42 r 3
16(     2 r A        2A3F16
         0 r 2
```

A common mistake is to forget the last line, that is, 16 into 2 goes nought remainder two.

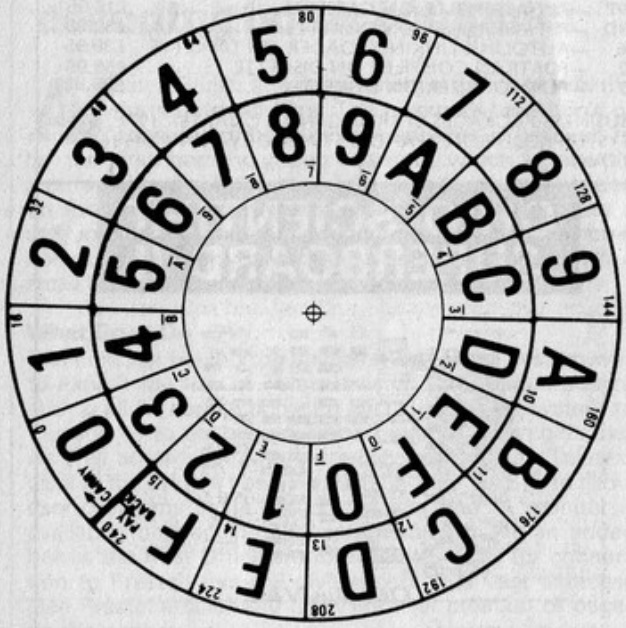


Fig 2. The rule is shown set to add D + E as in the addition example.

Hex to Decimal Conversion

For single byte numbers use the 16 times and 1 times tables on the rule to look up the value of each nibble and add them together. For example,

$$3b_{16} = (3 \times 16) + (B \times 1)$$
$$= 48 + 11$$
$$= 59_{10}$$

For multiple byte numbers you really need a calculator to handle the decimal arithmetic. If your calculator is the sort where the next operator (+ X) completes the previous operation the result is most easily obtained by "expanding the polynomial" without using brackets, for example,

$$6BE4_{16} = 6 \times 16 + 11 \times 16 + 14 \times 16 + 4$$
$$= 27620_{10}$$

Negative Thinking

If a single byte number is used as a counter the largest number it can hold is 255_{10} that is FF_{16}. However when the same single byte is used in a two's complement arithmetic calculation the number will be considered as negative if the leading bit is a 1. That is, all single byte Hex codes "higher" than 7F are negative numbers. To find what such a number represents in decimal it must first be negated, thus forming its positive hex version, and then converted to decimal in the normal way.

Negating Hex Numbers

To negate a number write down its 15's complement and add 1. The complements of the hex numbers are given as 0 etc. on the inner scale of the rule. Here the example in hex and binary for comparison.

```
negative number   = C5  = 11000101
complement        = 3A  = 00111010
add 1             = 3B  = 00111011
decimal value     = 59  = 59
```

So $C5_{16}$ represents -59_{10} in 2's complement form.

Similary to load an address with a negative number simply repeat the above procedure, for example.

```
negative decimal number  = -12
positive hex version     =  0C
complement               =  F3
add 1                    =  F4
```

So $F4_{16}$ represents -12_{10}.

TRS8BIT - VOLUME 18 ISSUE 04 - DECEMBER 2024 - PAGE 63 - TRS8BIT

SEND FOR OUR SPRING '79 CATALOGUE - 0·60p

SORCERER COMPUTER SPEAKS YOUR LANGUAGE



Are you looking for a PERSONAL or SMALL BUSINESS Computer? Look at these features for only £850 + VAT

- 256 Character Graphics Set includes 128 that you can define
- Z80 Based computer
- 32K of RAM on board
- Power-on 4K Monitor
- Excellent bit mapped graphics (512 x 240 point resolution)
- Full 79 key (3) keyboard including 16 key numeric pad (Full ASCII)
- Plug in ROM facility (e.g. 8K BASIC supplied). Assembler/editor, word processing etc. (so that you can change your language or utility by simply plugging in the one you want)
- Serial I/O ports for cassette/RS232 etc. (300 or 1200 Baud)
- Parallel I/O port for printer (Centronics Compatible)
- Z80 bus available at the rear for connection to the optional 8 slot S100 expansion box (e.g. S100 Floppy disk drives, graphics boards, extra memory etc.)

EXIDY SORCERER 32K RAM computer	£850
Add £5 for UHF output (16K RAM computer)	£751
12" Video monitor	£69
S100 expansion box includes 6 Slot Motherboard and power supply	£210

DISK SYSTEM as shown
MICROPOLIS S100 disks — 315 K bytes per drive
Drive 1 includes S100 controller
Micropolis M DOS and BASIC software £645
Additional 315 K bytes drives (up to 4 per controller) £345
Cables and regulators for 2 disk system £41

PRINTER as shown
ANADEX DP8000
80 columns, 112 characters 5 x 7 dot matrix head
Connects to parallel or serial ports £575

ADDITIONAL SOFTWARE FOR SORCERER PLUS DISK SYSTEM
CP/M operating system £145
CP/M with C BASIC (Compiler/Interpreter) £235
FORTRAN 80 £350
COBOL 80 £490
MICROPOLIS BASIC PACKAGES FROM COMPUMAX: STOCK CONTROL (INVENTORY), PAYROLL, ACCOUNTS RECEIVABLE, LEDGER, PERSONNEL — £99 each

BOOK SHELF

LOW PRICES FAST SERVICE!

Vol 0 The Beginner's Book	£5.40
Vol 1 Basic Concepts +	£5.90
Vol 2 Some Real Products (June 1977)	£9.90
Microprocessor series by Rodney Zaks C201 From Chips to Systems	£7.50
C207 Interfacing Techniques	£7.50
Microprocessor Systems Design by Edwin Klingman (ISBN 0-135-61413-8 (1977) 480 pages hard cover	16.40
CMOS Cookbook by Don Lancaster	£6.90
6800 Assembly Language Programming	£6.50
6800 Software Guide & Cookbook from Scelbi	£7.90
8080 Software Guide & Cookbook from Scelbi	£7.90
8080 A/8085 Assembly Language Programming	£6.50
8080A Bugbook Interfacing & Programming by Rony, Larsen & Titus (1977) 416 pages	£6.90
Z80 Programming for Logic Design by Adam Osborne	£5.90
Z80 Microcomputer Handbook by William Barden	£7.20
Structured Programming & Problem Solving with Pascal ISBN 0 138 54869 2 (1978) 365 pages by Kiebuntz "When will your microcomputer speak Pascal"	£9.50
BASIC Computer Games — Microcomputer edition	£5.40
Best of BYTE Vol 1 (1977) 376 pages	£8.90
Best of CREATIVE COMPUTING Vol 1 (1977) 326 pages	£6.90
Vol 2 (1977) 323 pages	£6.90
MOI OTHELLO Game for 2 players on NASCOM 1 cassette	£2.00
MOI MIND PILOT powerful text editor and interpreter in ½KB!	£5.00

ATARI VIDEO SYSTEM

The Best 200 Games plus REDUCED TO £139 — SAVE £30 — Selection of Cartridges £13.90 each

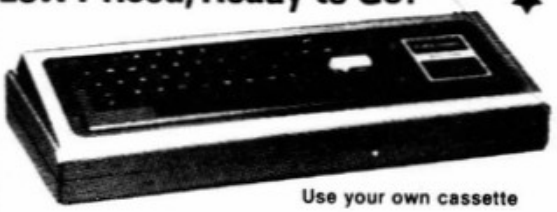
PET COSTS LESS AT COMP and it's a pedigree

RRP £550

The No. 1 Personal Computer in the U.K.
Affordable £499 + VAT — SAVE £50
for the first time user and the professional check out the PET, the world's most popular personal computer.



THE TRS-80 (SPECIAL SCOOP) Low Priced, Ready to Go!



Use your own cassette

Level-II with 4K RAM
Improved graphics, print formatting, and a faster cassette transfer rate are features of Level-II BASIC.

£399

PLUGS INTO YOUR OWN TV

Level-II with 16K RAM
A combination of 16K RAM and the powerful Level-II BASIC produces a system capable of handling most demands.

£499

KEY BOARD ONLY | COMPLETE WITH UHF MODULATOR

UK POWER SUPPLY — £9.90 + VAT

ATTENTION! TRS 80 & APPLE II USERS

SAVE £90

Simple to fit — only a screwdriver is required.

16K UP GRADE KIT — HALF NORMAL PRICE £99 + VAT — LIFETIME GUARANTEE

ITHACA AUDIO

S-100 bus products from the experts.

Z-80 cpu board. Featuring
- ★ On board 2708 EPROM.
- ★ Power-on jump. ★ Selectable wait states.
- ★ 8080 or Z-80 I/O addressing modes.
- ★ 2.0 MHz or 4MHz. ★ DMA.
- ★ Power: 8v 1 Amp, + 16v 100mA, -16v 50mA.

Bare printed circuit board £26.25
Assembled and tested 2 MHz £130.00
Assembled and tested 4 MHz £150.00



8K Static RAM board
Featuring: ★ Memory protect. ★ Power on clear.
★ Selectable wait states. ★ Two independent 4K halves. ★ Fully buffered.
★ Uses 2102L-1. ★ Power requirements: +8volts 1.4 Amps.
Bare printed circuit board £17.90
Assembled and tested (450nS) £110.00
Assembled and tested (250Ns) £130.00

NASCOM IMMEDIATE EXPANSION S100 from COMP —strongly recommended

The only available S100 motherboard kit (fully buffered) that plugs directly into your Nascom. Designed for the insertion of S100 boards (e.g. Static RAM, EPROM and discs etc.).

S100 Motherboard/Buffer (Complete kit + documentation) £47.50
Suitable 8K Static RAM Memory (fully assembled tested and guaranteed) ~~£125~~ £110

Motherboard comes complete with 2K Tiny Basic On cassette.

COMP COMPUTER COMPONENTS
(PART OF THE COMPSHOP LTD. GROUP)

For technical and sales literature send SAE to:
COMP COMPUTER COMPONENTS (Exidy UK Sales).
Freepost, 14 Station Road, New Barnet, Herts.
Tel: 01-441 2922 (Sales) 01-449 6596 Telex: 298755

TAKE ADVANTAGE OF OUR SHORT DELIVERY TIME

All prices exclusive of VAT

ADD £6.00 FOR SPECIAL DELIVERY ON LARGE ITEMS

TRS8BIT - VOLUME 18 ISSUE 04 - DECEMBER 2024 - PAGE 64 - TRS8BIT



https://www.youtube.com/watch?v=ZE3nMxTLgjk

IN MAV'S WORKSHOP - MODEL I INFORMATION COMPENDIUM - PART 1

(NOTE: this is an excerpt from a possible future book on the TRS-80 Model I)

CHAPTER: TBA

Working chapter title: Catalog numbers, keyboards, and board revisions

Author: Ian Mavric, Mentioned: Mike Yetsko, Steve Leininger

After the debut of the TRS-80 on the 3rd of February 1977 at the Warwick Hotel in NYC, Radio Shack was inundated with calls requesting information on the system and many wished to place orders on the spot. There were so many orders, even though a deposit had to be placed in order to secure a TRS-80 system, that it would take months for the backlog catch up. During this time you could hardly even find one in Radio Shack stores, and those who got them sold them just as quickly. Those last three months of 1977 must have been both an amazing and crazy time to work at RS. While continuing to work on the backlog of TRS-80 orders, designers and engineers were working on the Expansion Interface, Mini-disk drives, and printers, and programmers were working on the operating system which would become known as TRSDOS 2.x, and Microsoft was putting the finishing touches on Level II BASIC, all of which were to become available in the first quarter of 1978.

As most people know, Radio Shack uses a catalogue numbering system to put products into categories, and since the TRS-80 was a new product unlike anything else RS had sold before a new series was created starting with 26-1001. This was the 4K Level I TRS-80 keyboard unit.

The complete series set aside for the TRS-80 hardware line is as follows:

26-1001: TRS-80 keyboard/CPU 4K Level I BASIC

26-1003: TRS-80 keyboard/CPU 16K Level I BASIC

26-1004: TRS-80 keyboard/CPU 4K Level II BASIC

26-1006: TRS-80 keyboard/CPU 16K Level II BASIC

Notice how 1002 and 1005 are missing? Those catalog numbers were set aside for the 8K versions of the TRS-80. At the time of design an 8K version was considered, along with the 4K and 16K versions. Mike Yetsko explains that 1002 would have been the Level I version and 1005 the Level II

version, and the 8K would have been achieved by using half-bad 4116 16Kx1 bit Dram chips. (This isn't as much of a silly idea as it sounds and was implemented successfully a few years later on the 32K Color Computer 1 (26-3003) which used half-bad 4164 64Kx1 bit Dram chips, hence the HI/LO jumper on the mainboard.) 4116 Dram chips were expensive in 1977 so the cost to upgrade a 4K TRS-80 to 16K was an eye-watering $US299.00, whereas the proposed 4K to 8K upgrade was only going to be only $US99.00

It was rationalised some time after the TRS-80 was released that half-bad 4116 chips might be ticking time-bombs before the other good half of the chip would fail, creating a warranty nightmare that might tarnish the reputation of the computer and so the 8K version was quietly dropped. Those few 8K machines which were sold were recalled back and upgraded to 16K. By 1978 all references to the 8K TRS-80 had been removed from RS literature.

The other accessories released along with the TRS-80 in 1977 were:

26-1201: Radio Shack TRS-80 Video Display

14-841: Realistic CTR-41 Portable Battery/AC Cassette Recorder

I'll discuss these later on in the book. While it was possible to buy a TRS-80 without the monitor and tape recorder, and certainly some people did, it was far more common for people to buy their TRS-80 as a package with the video display and tape deck. These combined packages carried their own catalog numbers which were also logical the only way Radio shack knew how.

26-1051: TRS-80 4K Level I System

26-1053: TRS-80 16K Level I System

26-1054: TRS-80 4K Level II System

26-1056: TRS-80 16K Level II System

Keen eyed numerical spotters will have noticed straight away there is no 1052 or 1055, which would have been 8K Level I and Level II systems respectively.



Photo: The original carton TRS-80 complete systems came in, proudly stating on the outside it's catalog number, ROM Level, and memory size.

KEYBOARD

The first interaction people have with the TRS-80, after plugging it all together switching it on is via the keyboard, and it is worthwhile looking at it in detail. Two keyboards were used in the TRS-80, firstly by Hi-Tek, and then by ALPS Japan. In TRS-80 circles they are often referred to at the 'keybounce' keyboard and the 'good one'. From the beginning up until the end of 1978, and possibly into early 1979 the Hi-Tek keyboard was used so let's talk about it. Steve chose the keyboard

because he liked the keyboard on the DecWriter from Digital Equipment Corp. His decisions that as well as normal alpha-and-numerical keys, a full set of arrows were desirable, and a Break key to stop a program running. Clear was also handy when clearing the screen. Steve also decided something better than a slightly enlarged key with 'Return' (as in Carriage Return, a carry-over from the electric typewriter era) so specified a more applicable name 'Enter' (which is obvious when you think about it, you are Entering information to the computer, not returning a typewriter carriage), and while at it, make it big and white, so no one can miss it. Yes, in once instance of clear thinking Steve invented the Enter key, which has been a feature on practically every computer and terminal keyboard since.

Hi-Tek keyboards use an interesting mechanism called "Hi-Tek High Profile", which is mechanical in nature and are of a design which gives a good service life. The switch contacts are a pair of vertical leaf springs placed side by side, that are held apart by a horizontal bar in the slider. When the slider is depressed, the bar is lowered and the contacts close. As a key is pressed down the two leaf springs touch and by closing a circuit on the keyboard matrix the key-press is registered. Everything going to plan you press the 'A' key once and you 'A' once on the screen. The trouble with they Hi-Tek design is that if the key is pressed even moderately hard the leaf-springs can hit each other, momentarily separate (or bounce off each other) before coming together again. The computer will put 'AA' on the screen and you have just discovered the infamous keybounce of the TRS-80 Model I.



Photo: Hi-Tek High-Profile keyswitches

Many keyswitch designs are inherently bouncy by design and the usual way this is dealt with is a keyboard debounce routine in software, and this is where things get interesting. A debounce routine simply ignores any extra contacts of the same key for a small amount of time after the first depression of the key.

The Hi-Tek keyboard on the TRS-80 was arguably the best keyboard offered of the 1977 trinity of computers. The Commodore PET keyboard was derived from Commodore's previous involvement in building calculators and so that's why it looks and feels like an oversized desktop calculator. The Apple][had a better keyboard than the Commodore but it lacked UP and DOWN arrow keys. The price difference between the Apple][and the Commodore and TRS-80 meant it should have had the best keyboard of all three.

The TRS-80 keyboard encourages fast typing. What was it like in normal use? In Level I BASIC there is no n-key rollover so you need to release the last key before you type the next. For fast typists this can be annoying. You soon get used to it but for high speed touch typists there is definite speed limit you reach on Level I systems. The lack of n-key rollover in the Level I does a good job of masking keybounce. It was not uncommon back when owners upgraded their systems from Level I to Level II to notice keyboard was much more apparent. Microsoft designed Level II Basic for the TRS-80 and included n-key rollover for fast typing, the polling of the keyboard was so fast that bouncy keyswitches were the norm. To rectify this problem Radio Shack released a small program on cassette called KBFIX to properly debounce the Level II keyboard. You received the tape if you went into Radio Shack and complained the keybounce was intolerable. It did a pretty good job but was not perfect. It's interesting to note that most sophisticated disk operating systems that were popular, namely LDOS, NewDOS/80 V2.0, DOSPlus, and MultiDOS have a keyboard debounce routine incorporated that loads at boot up. It's why a TRS-80 which has so much keybounce as to be unusable as a cassette-based system becomes much less bouncy as a disk system. TRSDOS 2.x does not have a keyboard debounce routine.

One thing to look for with the Hi-Tek keyboard is dry and cracked soldering on the under-side of the keyboard where the leaf springs are soldered. The solder is thin and not of a very high quality so it cracks after hundreds of thousands of presses. The is exacerbated by vigorous game playing where the arrow keys and space bar are often hammered by the operator. If a key refuses to work I always check under the keyboard and most of the time there is a dry solder joint. Since there is one and you already have the computer open I re-solder each keyswitch with good quality solder. It looks like a big job but only takes a few minutes, and you don't have to worry about dead keys on the Hi-Tek again.

Around the time of the TRS-80s second birthday in the middle of 1979 the keyboard supplier was changed from Hi-Tek to ALPS. ALPS is a Japanese company which were in the business of making quality keyboards and mainly did custom work for large computer manufacturers. This started a long association between Radio Shack and ALPS, as variations of the keyboard could be found in the future in the Model III, 4, and 4D. They type of keyswitch used in the TRS-80 is known as the SKCC Cream, which is a switchplate design which gives a much sturdier feel while typing (the ALPS key switches are mounted to a metal plate which also contributes to the sturdier feel), and extremely low chance for keybounce. The keybounce problem was finally solved. ALPS keyboards found their way into my more high-end computers like the Apple /// and early Macintosh computers. They have

a very calming and assured sound and Radio Shack would have been rightly proud of their inclusion in their computers.



Photo: ALPS SKCC key switches

MOTHERBOARD REVISIONS: A,D,G

Naturally Revision A was the first motherboard the TRS-80 was released with and it's also likely the rarest one you will find. These were included include in the first few thousand machines in the early days of TRS-80 production hand built. They are easily identified by having white sockets for the CPU and ROM chips. None of these were exported outside of the USA and by the end of 1977 the backlog of TRS-80 orders had been completed. Typically these were sold with Level I Basic in the two memory configurations of 4K and 16K. Although these could be upgraded to Level II Basic, it was less common.

Rev D was the first major version which sold in enormous numbers. These arrived in early 1978 around the same time that the Level II upgrade became available. Level II was achieved by the infamous daughterboard connected via a ribbon cable. Level II was so large (12K) and the two ROM sockets on the TRS-80 motherboard were designed around 2K or 4K ROM chips that when Level II Basic was completed a way was needed to fit three ROMs in a place designed for two. The daughterboard was a novel approach which worked quite well. The ribbon cable carried the address and data, as well as power for the chips on the daughterboard, and 4 wires connected additional needed address lines to various points on the motherboard for A11,12 and 13. Later on this was updates to 6 wires which carried power to the daughterboard. Around this time the 6 wires started to be placed inside a line clear hose so that the wires couldn't short out on the motherboard; an odd thing which I have seen happen at least once.

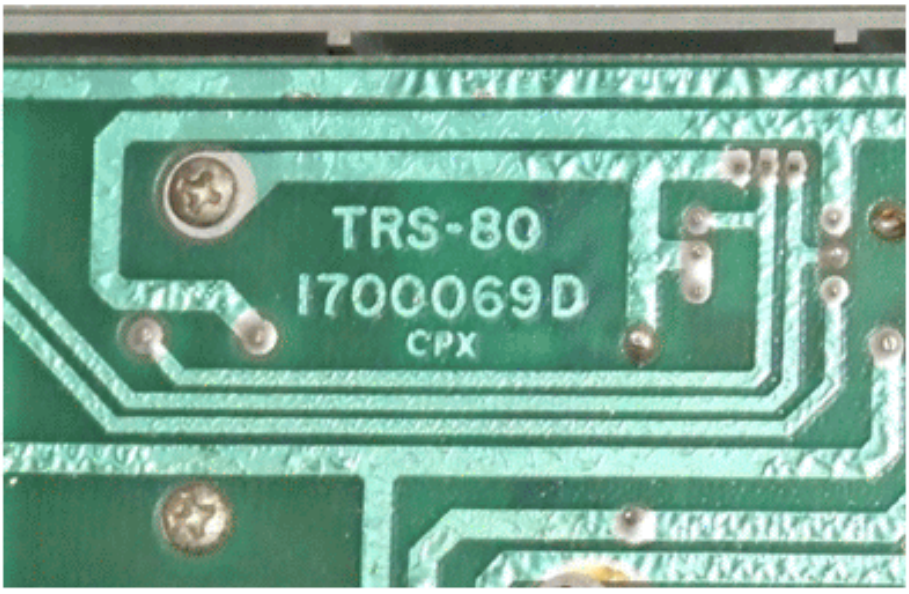


Photo: How to identify the Rev D motberboard

Underneath the computer the catalog numbers changed with a 'D' after them, eg. 26-1006D, indicating a Rev D board was installed. This practice didn't continue however because Radio Shack reverted back to the plain catalog number 26-100x with no letter when the next revision of the motherboard was put into production. It's interesting to note that Mike Yetsko mentioned it was not unknown for RS computer service centres to replace a Rev A board with a Rev D if the computer was broken or when upgrading and not advise the customer. The reasoning behind this is not exactly clear but it's possible the Rev A board had a bad reputation around the 'Shack.

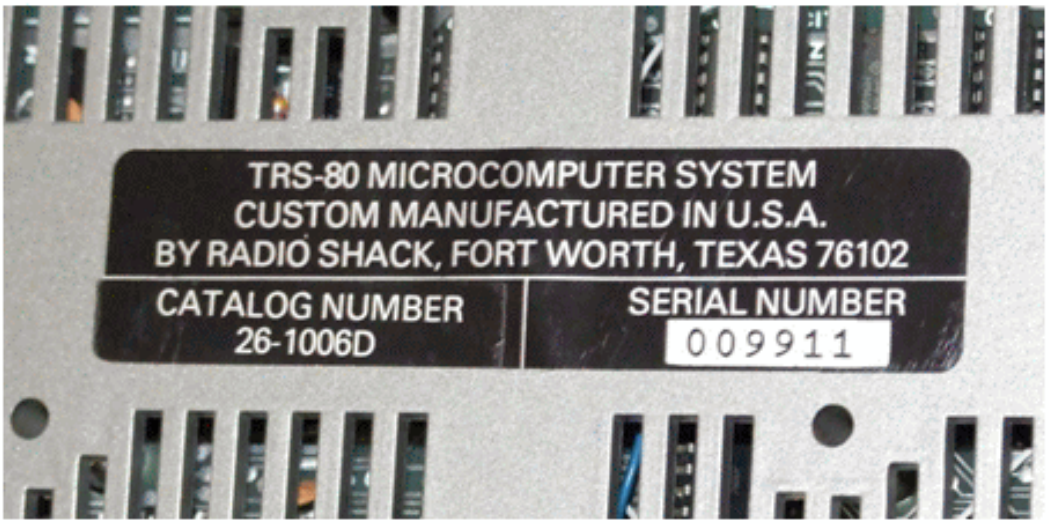


Photo: TRS-80 catalog number with D-suffix

A fully maxed-out Rev D 16K Level II TRS-80 was something behold. There was a lot going on inside the familiar grey case. A problem with cassette loading on Level II systems being extremely sensitive to volume settings had become a major concern during the production of the Rev D board, as a small board called the XRX-III which is a synchronous 500-baud wave shaper circuit. It helped by making the wave shape coming into the TRS-80 from the cassette port more palatable and mostly worked. It was a free upgrade though you did have to pay a Radio Shack technician to install it. Once the XRX-III was released all Rev D systems henceforth came with it already installed. Also around this time a numerical keyboard upgrade became available which was literally a pad of numbers and symbols connected in series via wires to their corresponding keys on the main keyboard. Add to that the Level II daughterboard and there was lots going on inside the TRS-80.

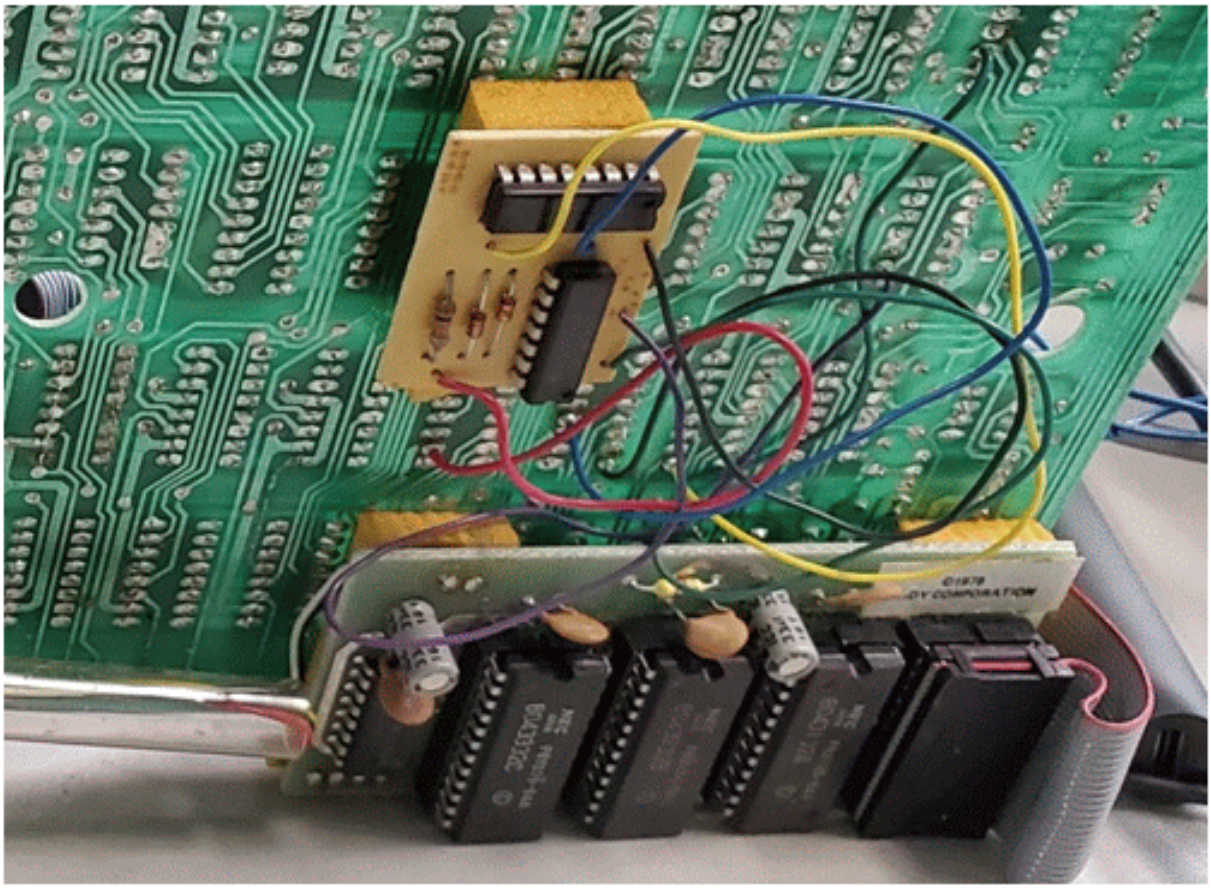


Photo: Level II Basic plus the XRX-III modification inside a Rev D TRS-80.

By mid-1979 the first and only major update to the TRS-80 arrived and these are clearly distinguished by the new ALPS keyboard. Inside the computer was the latest Rev G motherboard which would see the TRS-80 our until production of the machine ended in late 1980, due to the FCC's concerns about RF interference (noise). XRX-III was gone, and the Level II daughterboard was gone. The design on the ALPS keyboard meant it could be fitted with or without a numerical keypad but was now one long unit that stretched the width of the TRS-80. Newer ROM chip designs meant that now Level II could be fit into two chips if a slight re-working of the motherboard in the ROM are was

made. Microsoft did a final upgrade of Level II BASIC including fixing a bug which made loading cassette tapes very sensitive necessitating XRX-III, so that was removed. Also added was a better keyboard de-bouncing routine which in conjunction with the ALPS keyboard effectively eradicated keybounce. Due to a few extra bytes needed for these updates in the ROM they were taken from space formerly where the labels "MEMORY SIZE?" and "RADIO SHACK LEVEL II BASIC" resided so these were now shortened to "MEM SIZE?" and "R/S L2 BASIC". When you see these you know almost 99% of the time you have a Rev G board.

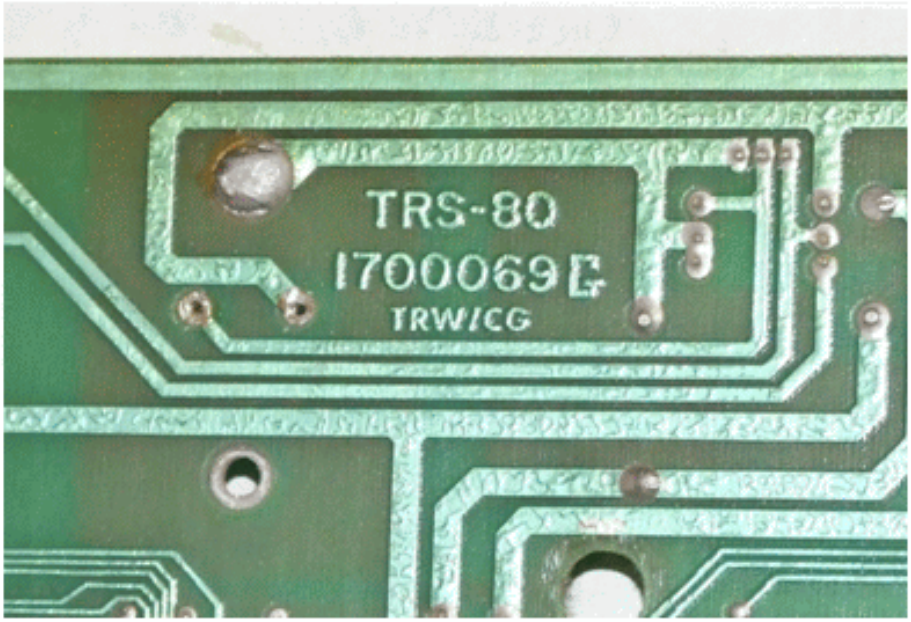


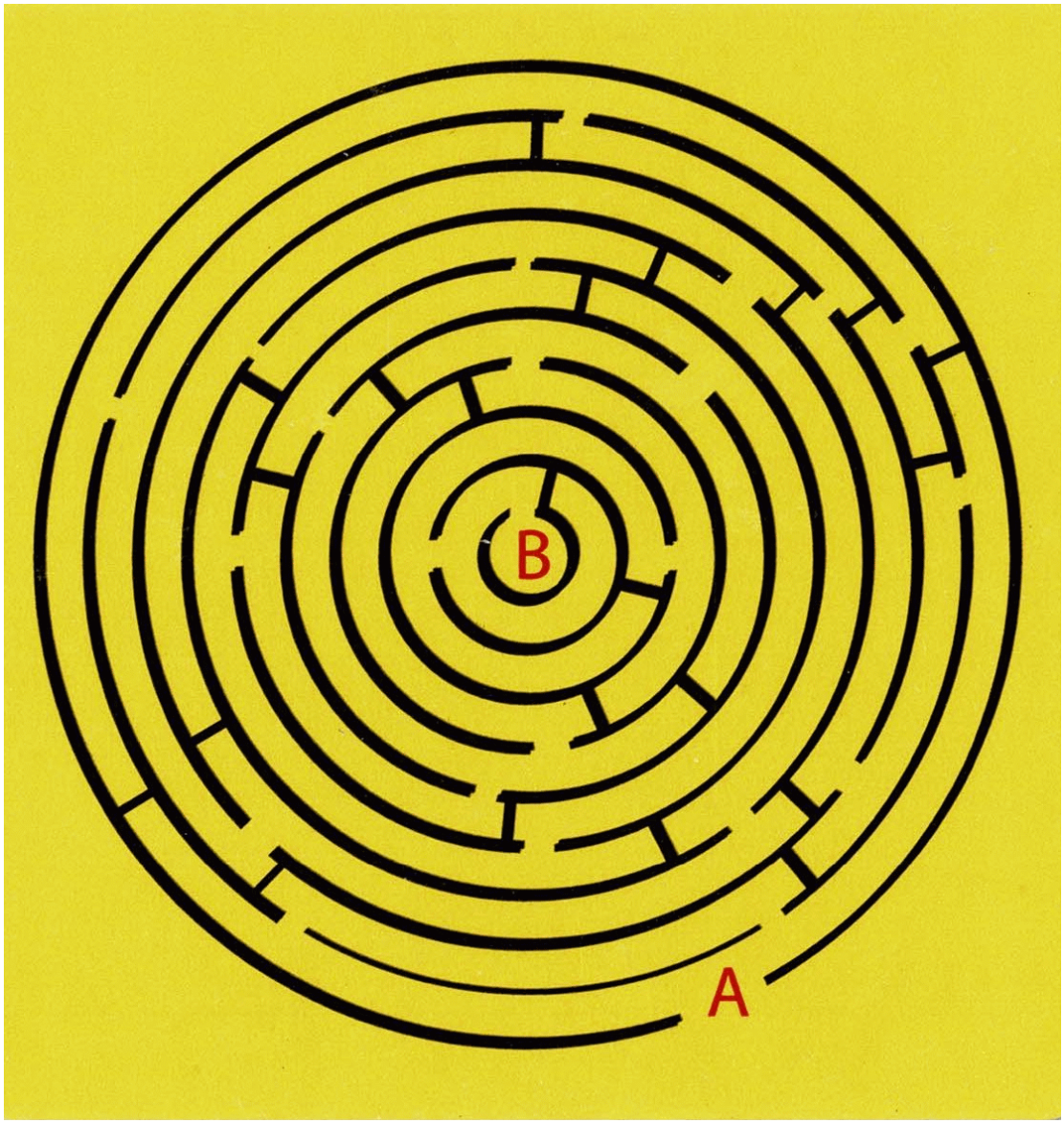
Photo: How to identify the Rev G TRS-80 motherboard

The Rev G board had a very clean appearance to it. Another change which was not immediately obvious was that sockets were added for Z29 (the character generator ROM) and Z45 (one of the 2102 SRAM chips). By '79 the lowercase modification was both popular a profitable for Radio Shack and with those two chips socketed could be performed by a competent technician in under 10 minutes.



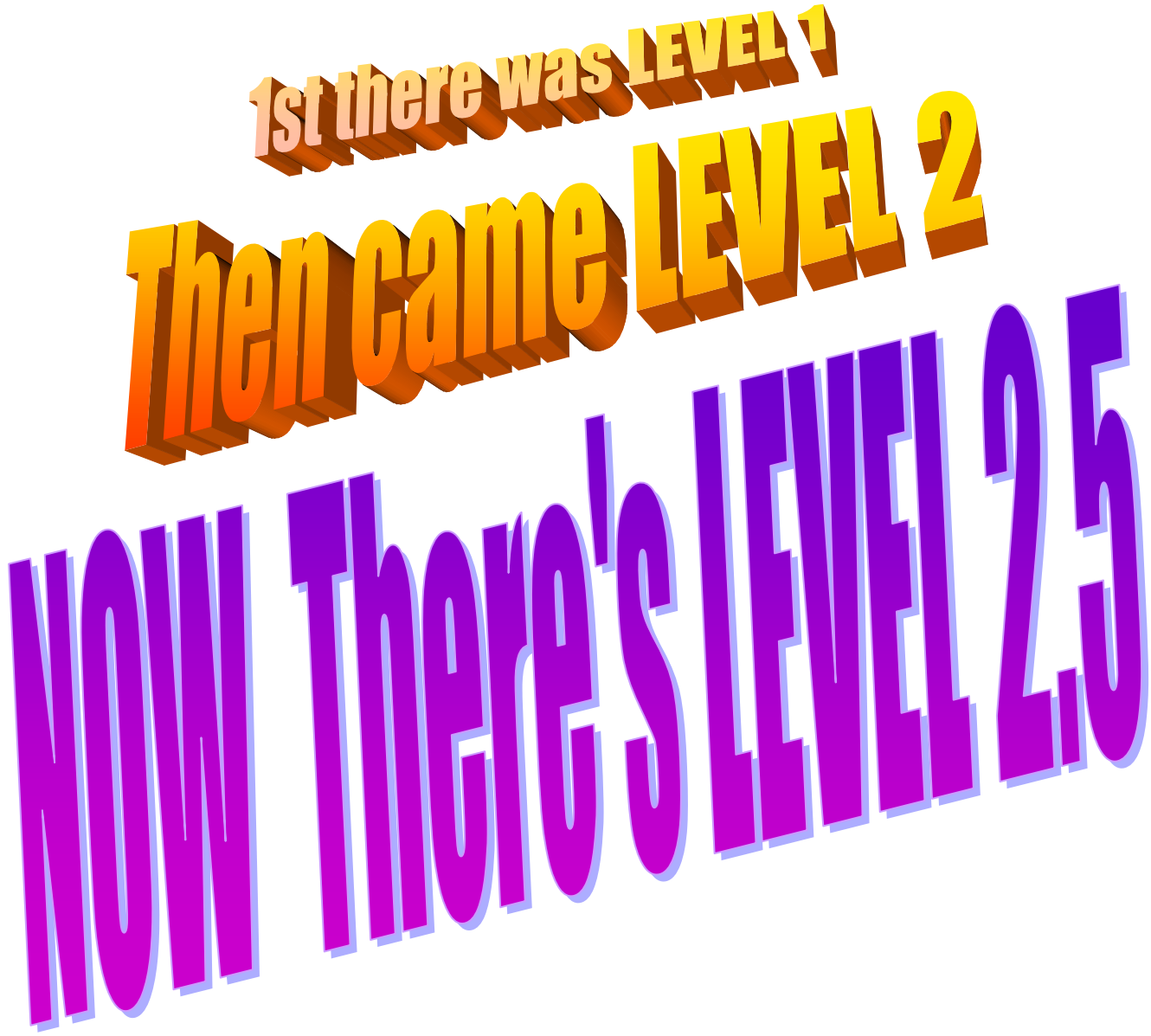
Ian Mavric is an IT Specialist who also restores and collects TRS-80's and classic cars. He lives with his wife and kids in Melbourne, Australia.

AMAZED



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 HOW LONG IT TAKES TO GET FROM A TO B, FOLLOWING THE YELLOW ROAD.
TRY FOR UNDER 10 SECONDS!



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

Surfin' The Net

Soigeneris
Your Resource for Hi-Tech Hobbies

Search store SEARCH

Tools | Vintage Computing | Clearance | Bits | Collets & Toolholding | Work holding & Lathe Chucks | Carvewright Accessories | Cleaners & Lubricants | Drives & Controls | Machine Tools | Parts & Accessories | Software and Books | Documentation and Downloads | Welcome to our new website!

Home / Vintage Computing / 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

☆☆☆☆☆

Be the first to review this product

Manufacturer: StarDust

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

1 ADD TO CART

♡ Add to wishlist | ⇄ Add to compare list | ✉ Email a friend





STILL WORKING ON SOME MORE SERIOUS APPLICATIONS OF THE ORCHESTRA IN THE STUDIO, INCLUDING A PLANNED 6-MACHINE SYNTH RACK WITH THE ORCHESTRA 90S AND MIDI80S AT THE CORE. STILL LEARNING THE DRUM TRACKER AND DESIGNING THE SOUND ROUTING/NOTE ROUTING. MORE TO COME.

JOEL HILLIARD

TRS-80

Repairs & Upgrades

HNC qualified electronics engineer
with 40 years TRS-80 experience.
Enquiries to: trs80@shanefoster.co.uk

OBIT.



Thomas Eugene Kurtz
(February 22, 1928 – November 12, 2024)

Thomas was an American computer scientist and educator.

A Dartmouth professor of mathematics, he and colleague John G. Kemeny set in motion the then revolutionary concept of making computers as freely available to college students as library books were, by implementing the concept of time-sharing at Dartmouth College.

In his mission to allow non-expert users to interact with the computer, he co-developed the BASIC programming language (Beginners All-purpose Symbolic Instruction Code) and the Dartmouth Time Sharing System during 1963 to 1964.



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

http://members.iinet.net.au/~ianmav/trs80/





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

WHERE 8 BITS AND 16K STILL RULE OK