TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

JUNE 2026 - VOLUME 20 ISSUE 2

PAGE 04 COOL & EFFICIENT M1 POWER SUPPLY
AMARDEEP CHANA

PAGE 19 RECOVERING 'SCAP'
DICK SMITH'S LOST SYSTEM-80 SOFTWARE
TERRY STEWART

PAGE 30 'BEAST' VOIDSTAR'S NEW GAME

PAGE 31 TA26 NEWS AND REVIEW
IAN MAVRIC

PAGE 38 THE CLOCKWORK PI V2.0

PAGE 49 L.S.D. THE UK'S PRE DECIMAL CURRENCY.
A BASIC ONE-LINER FROM
E.T. FONEHUME

PAGE 52 BREAKING THE 48K BARRIER
MARK PRUDEN

PAGE 57 'HOUSE ON THE EDGE OF THE CRATER'
NICKMAN STUDIO'S NEW GAME

PAGE 61 POWER OF THE ROM PART 6
DRAWING CONCLUSIONS
VINCE OTTEN

PAGE 77 THE WAY WE WERE
WHAT WAS 'COOL' BACK IN THE DAY
'THE PROGRAM THAT WRITES PROGRAMS'

PAGE 79 THE COUNTDOWN QUIZ
NOT QUITE AS EASY AS LAST TIME
CAN YOU DO IT IN 30 SECONDS?

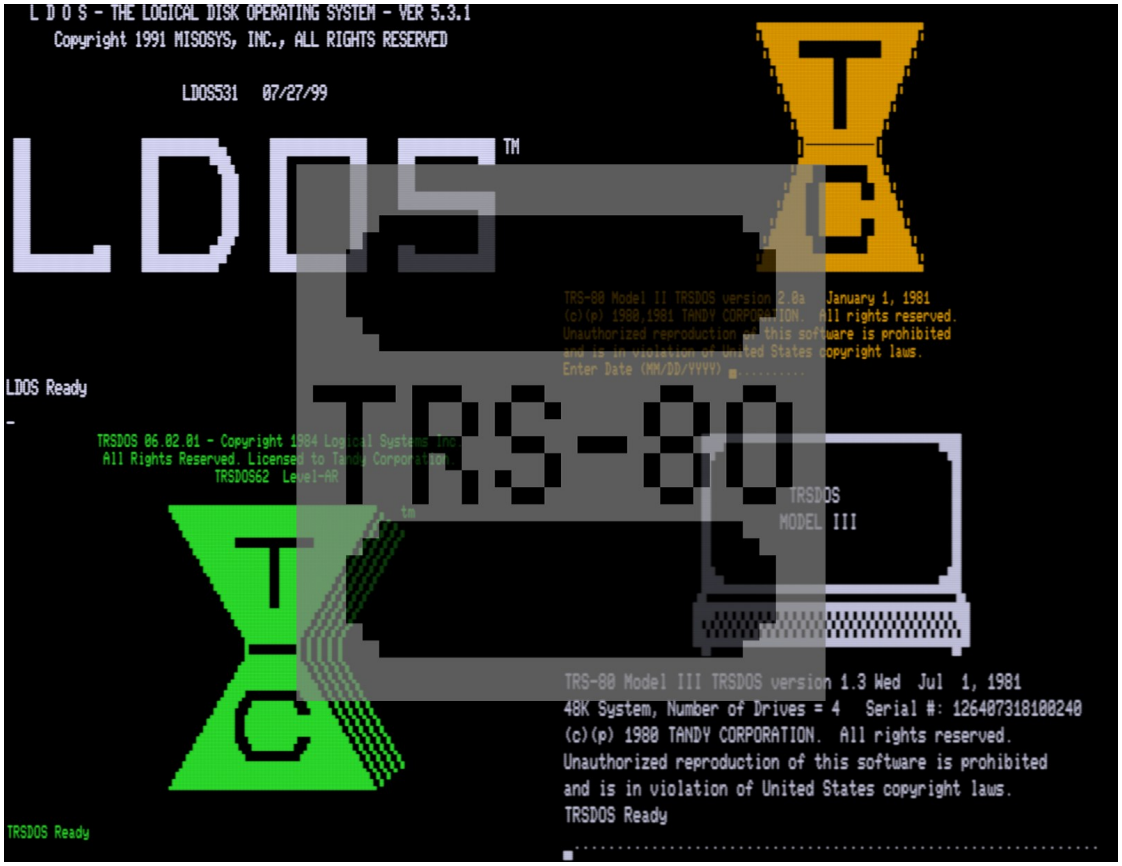
PAGE 80 SURFIN' THE NET
THE '8-BIT BOYZ'

PAGE 83 LDOS 'Y' NEWDOS80 'NO'
IAN MAVRIC

PAGE 92 UK'S EBAY BARGAINS - WERE YOU LUCKY?

PAGE 94 UPGRADE A PAL CO-CO FROM 32K TO 64K
IAN MAVRIC

TRS80GP - LATEST UPGRADE
VERSION 2.5.7





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



RUNS ON

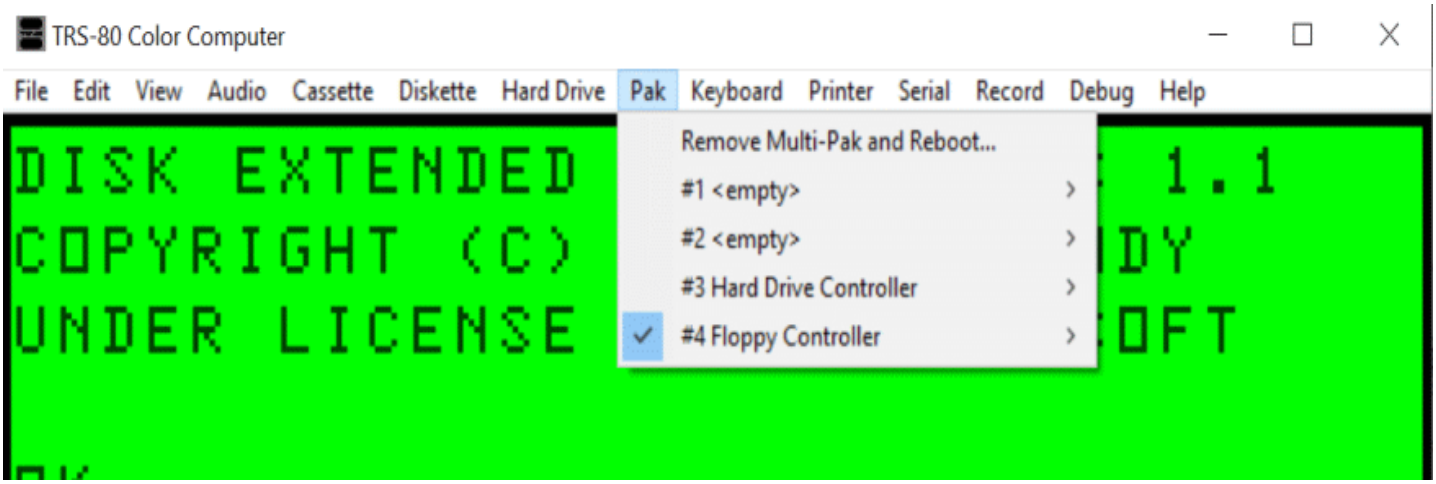
WINDOWS, MACOS, LINUX & RASPBERRY PI.
EMULATES MODELS 1, 2, 3, 4, 12, 16, 6000, MC-10,
COLOR COMPUTER, DT-1, VIDEOTEX, TANDY 10 AND 'GREEN THUMB'.

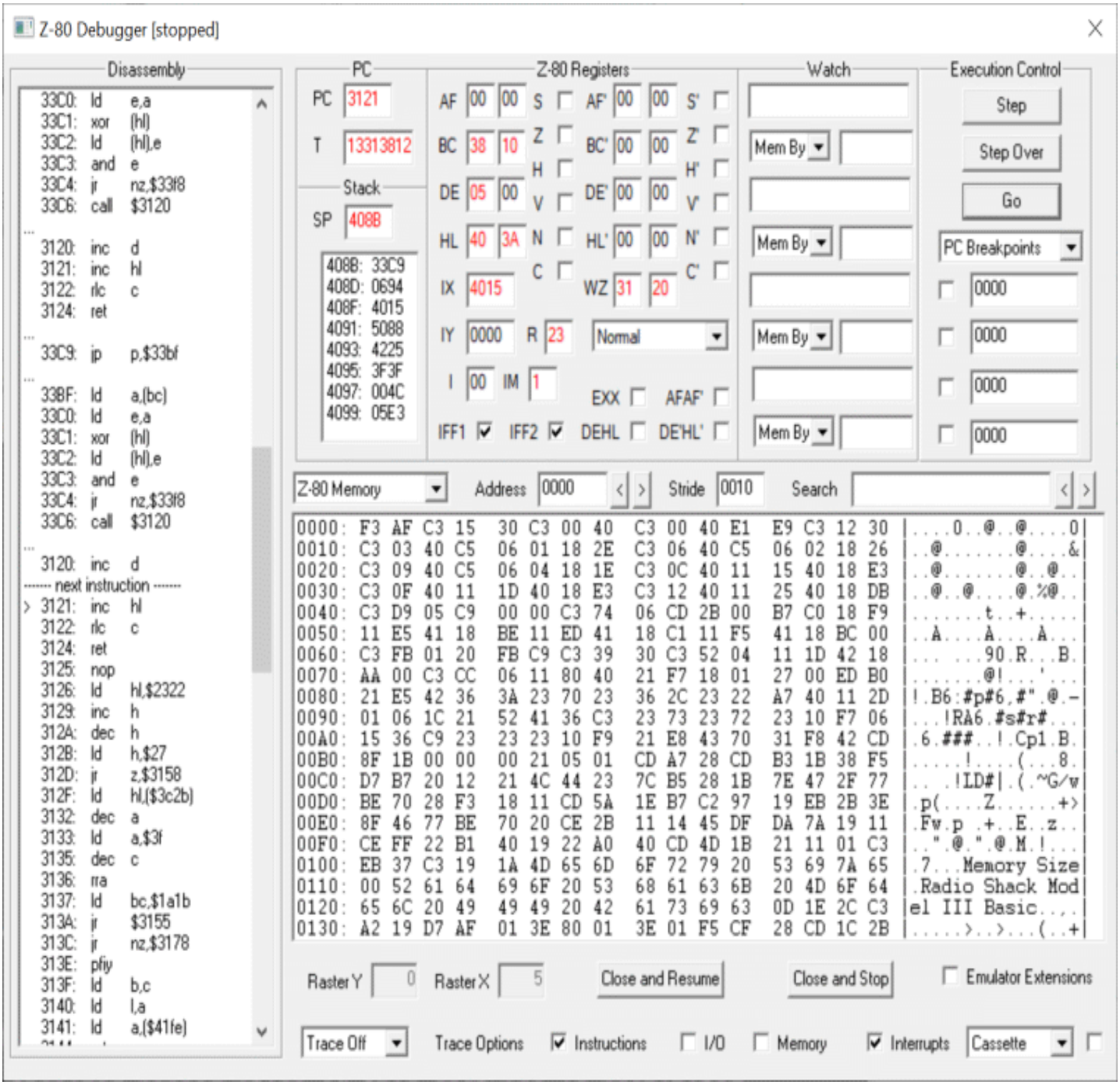
JUST LIKE A REAL TRS-80 WITH NONE OF THE HASSLE.
BUILT-IN DEBUGGER AND OTHER FEATURES, JUST GREAT FOR PROGRAMMERS.

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

UPDATED VERSION, 2.5.7, ADDITIONALLY OFFERS :

MULTI-PAK AND HARD DRIVE EMULATION FOR COLOUR COMPUTER.
CPU BACKTRACE IN DEBUGGER WINDOW.
IMPROVED HARD DRIVE EMULATION.


TRS-80 Color Computer
File Edit View Audio Cassette Diskette Hard Drive Pak Keyboard Printer Serial Record Debug Help
Remove Multi-Pak and Reboot...
#1 <empty>
#2 <empty>
#3 Hard Drive Controller
#4 Floppy Controller
DISK EXTENDED
COPYRIGHT (C)
UNDER LICENSE
1.1
IDY
OFT


Z-80 Debugger [stopped]
Disassembly
PC
Z-80 Registers
Watch
Execution Control
Step
Step Over
Go
PC Breakpoints
Stack
Z-80 Memory
Address
Stride
Search
Raster Y
Raster X
Close and Resume
Close and Stop
Emulator Extensions
Trace Off
Trace Options
Instructions
I/O
Memory
Interrupts
Cassette

Cool and Efficient Model I

Power Supply Mod

2025-11-26 v1.0.3
Amardeep Chana

Revision History

Date	Revision	Notes
2025-11-23	1.0	Initial Release.
2025-11-24	1.0.1	Feedback fixes, general cleanup, skip Q4 removal.
2025-11-24	1.0.2	Added DIN polarity. Major restructuring.
2025-11-26	1.0.3	Minor cleanups and ready to be published.

Credits

Many thanks to Ira Goldklang of | https://www.trs-80.com kiwisincebirth from the Tandy Discord, Pete Cervasio of kjsl.com fame, and John Wilson, founder of the Vintage {Computers | Microprocessors | Microcontrollers} facebook group for their very helpful review and feedback.

A special thank you goes to Matt Boytim for his unwavering enthusiasm about the TRS-80 Model I that inspired me to embrace and improve on the quirks of this beloved computer and for sharing his expertise and invaluable help.

Introduction

This modification replaces the linear power regulation circuitry on the TRS-80 Model I main board with a modern DC to DC switch-mode regulator based on an MP1584. This has the benefit of significantly reducing the power draw and associated heat generation. It still works with the factory power transformer module or any modern reproductions of it. Optionally an inexpensive wall plug-in power transformer ("wall wart") can be used instead.

It is a low cost solution for:

- Missing or failed transformer bricks.
- Failed linear regulator circuits.
- Working machines where reduced heat is desirable to extend service life and reliability.

Summary of the Changes:

1. The onboard linear power supplies are disconnected by cutting some traces. The inputs to the +5V and +12V circuits are cut from the power switch. The -5V and +12V outputs are cut from the DRAM power buses.

2. The bulk filter capacitor C8 is removed. It is used by the existing +12V power supply therefore it is no longer needed. Its space is used to mount the new regulator.

3. The series ballast resistor R4 is removed. This disconnects the +5V linear supply's output from the main board's power bus.

4. A jumper is added from pin 8 to pin 9 of one of the DRAM sockets to reconfigure the sockets to correctly power the 5V-only DRAMs.

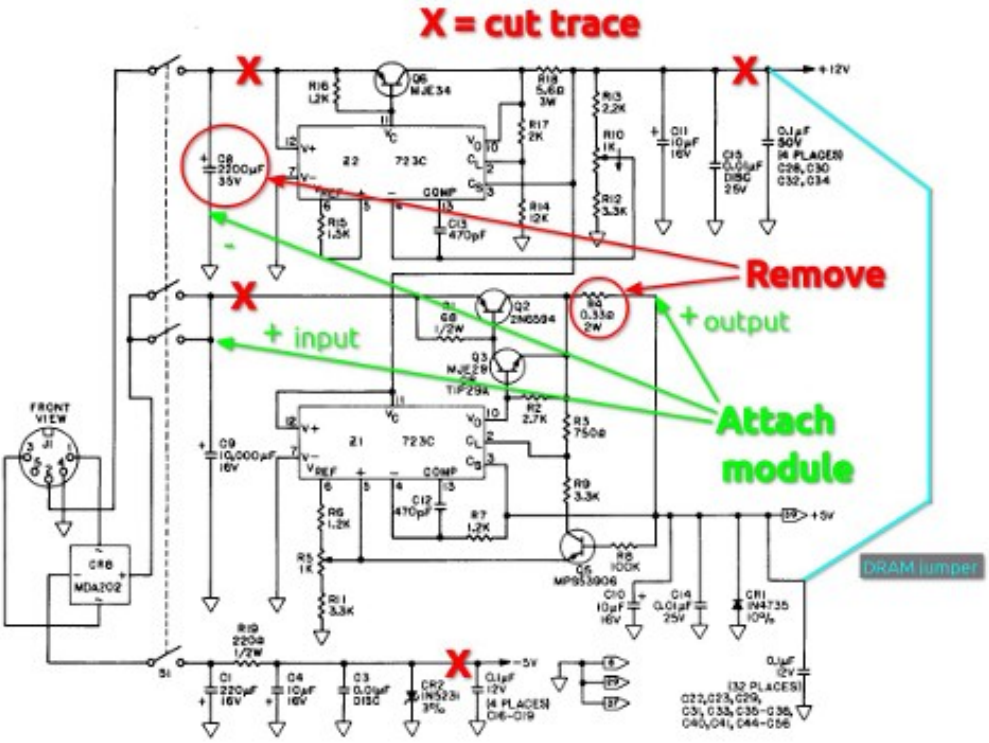
5. The eight 4116 DRAMs are replaced by eight 5V-only memory devices.

7. A 5V switch-mode regulator is installed on the pads vacated by C8 and R4.

8. A wire is installed from the power switch to the input of the new regulator.

These changes produce a cool-running Model I computer with DRAMs that are more reliable. It can be powered from an original Radio Shack transformer brick, one of the modern replacements, or even a 12V AC or DC 500mA wall adapter wired to a DIN-5 connector.

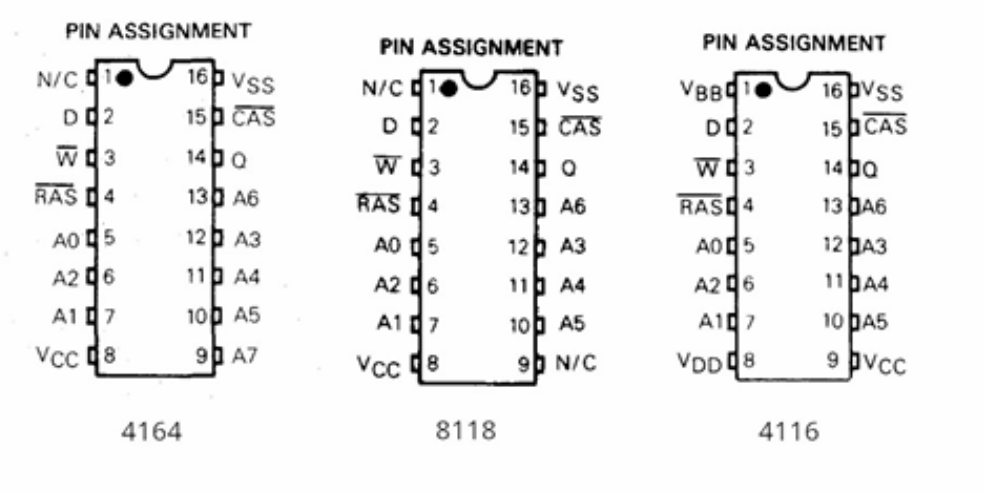
Diagram of Modifications



NOTE: The modifications are applicable to all revisions of the system board.

Explanation of DRAM Change

In order to eliminate the need for -5V and +12V regulators, the original 4116 type DRAM is replaced by cooler running and more reliable single supply +5V parts.



The modification cuts power feed to pins 1 and 8 of the DRAM sockets which normally carry the -5V and +12V required by 4116 memory devices. Then it bridges pin 9 (which carries +5V to a 4116) to pin 8 which is where 8118/4517 and 4164 devices need it.

In the case of 8118/4517 which are 5V only 16K memory devices, that exactly maps to the same functionality of the 4116 it replaces.

In the case of 4164, multiplexed address input A7 ends up tied to +5V (high) state so only ¼ of the memory array will be usable, which means 16k of the 64k will be used. Unlike the Model 4, It is not necessary to use 128 refresh cycle parts since this configuration hard-wires the high address bit. Therefore any 4164 type part will work.

NOTE: Since the appropriate traces are cut and a jumper is installed to make the sockets compatible with the target memory devices, there is no need to bend up any pins or solder jumpers to each memory chip as some other modifications require.

Items Needed

You will need the following components:

- Quantity (8) 5V memory chips (MB8116, MCM4517, or any 4164)
- MP1584 5V 2A switch-mode regulator module (see next page)
- Resistor to test regulator under some load (about 100 Ohms, ¼ Watt)

Optionally, if you intend to use a low-cost wall adapter instead of a TRS-80 transformer unit:

- 12V AC or DC rated minimum 500mA wall power adapter
- Male DIN-5 plug

You will also need some basic tools:

- Screwdrivers, needle nose pliers, diagonal cutters
- Soldering equipment (iron, solder, flux, solder removal tool, etc.)
- Utility knife and/or Dremel tool (to cut traces)
- Digital Voltmeter
- 9VDC to 12VDC power source to test regulator (e.g. 9V battery)

Warning:

Proceed with caution and at your own risk. Modifying your TRS-80 in this manner requires skill and care. If you don't feel comfortable with the steps described in this document, please seek the help of an experienced technician. There is always the risk your computer will become non-functional after making these modifications. Soldering and cutting traces can result in injury. Wear safety goggles and other appropriate protective gear in a well ventilated workspace.

Sourcing the 5V Switch-Mode Regulator Module

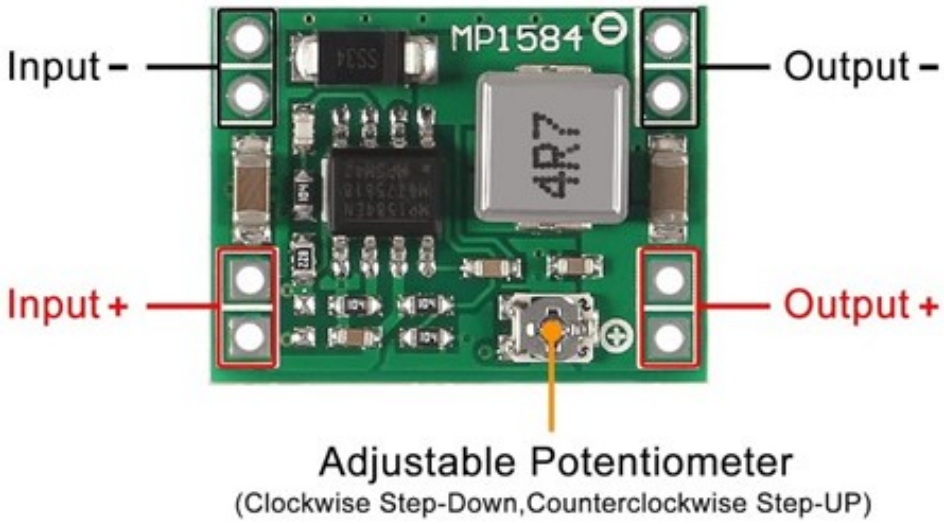
The heart of the modification is an MP1584-based switch-mode regulator which is widely available on Amazon and Ali Express. It comes in either adjustable-output or fixed-output varieties. If possible, source the 5V fixed version since that eliminates the need to adjust it prior to installation.

Use the search term: "mp1584 5v converter module"



Procedure

1. If your regulator module is the type with an adjustable output, start by turning the potentiometer fully clockwise. Power the switching regulator module up with 9V DC to 12V DC on the input terminals and approximately 100 Ohm load on the output terminals. Mind the input polarity to avoid damage. Confirm it is producing from 4.9V to 5.1V output. If it is the model with an adjustable output, turn the potentiometer to read 5.0V.

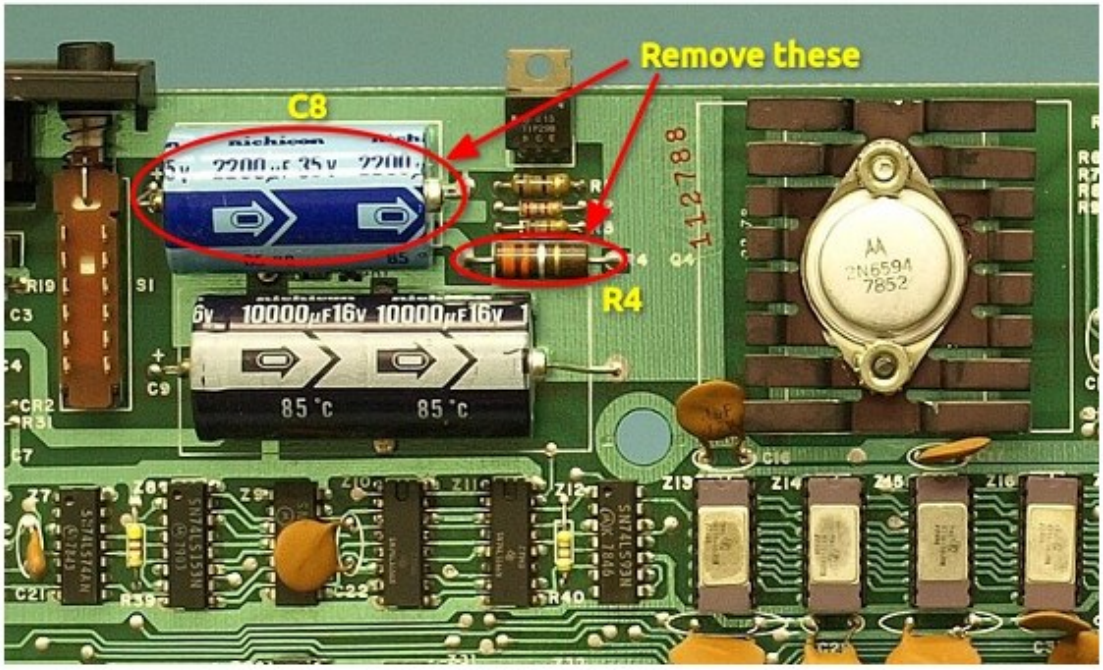


Warning:

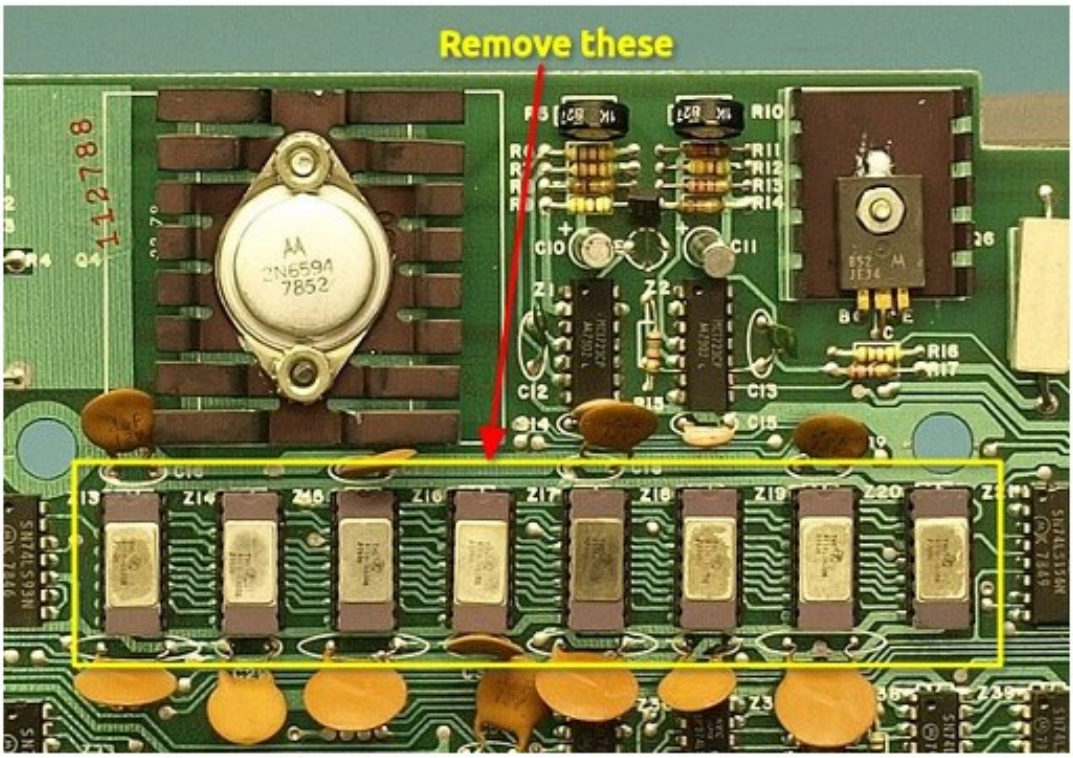
1. Do not reverse the positive and negative terminals to avoid any possible damage
2. Do not use without load or light load (less than 10% of output power)

2. Disassemble the CPU keyboard unit. This would be a good time to remove the aging flex cable and replace it with something better. Preferably one that has a connector so you don't have to deal with an attached keyboard while you are working on the CPU board.

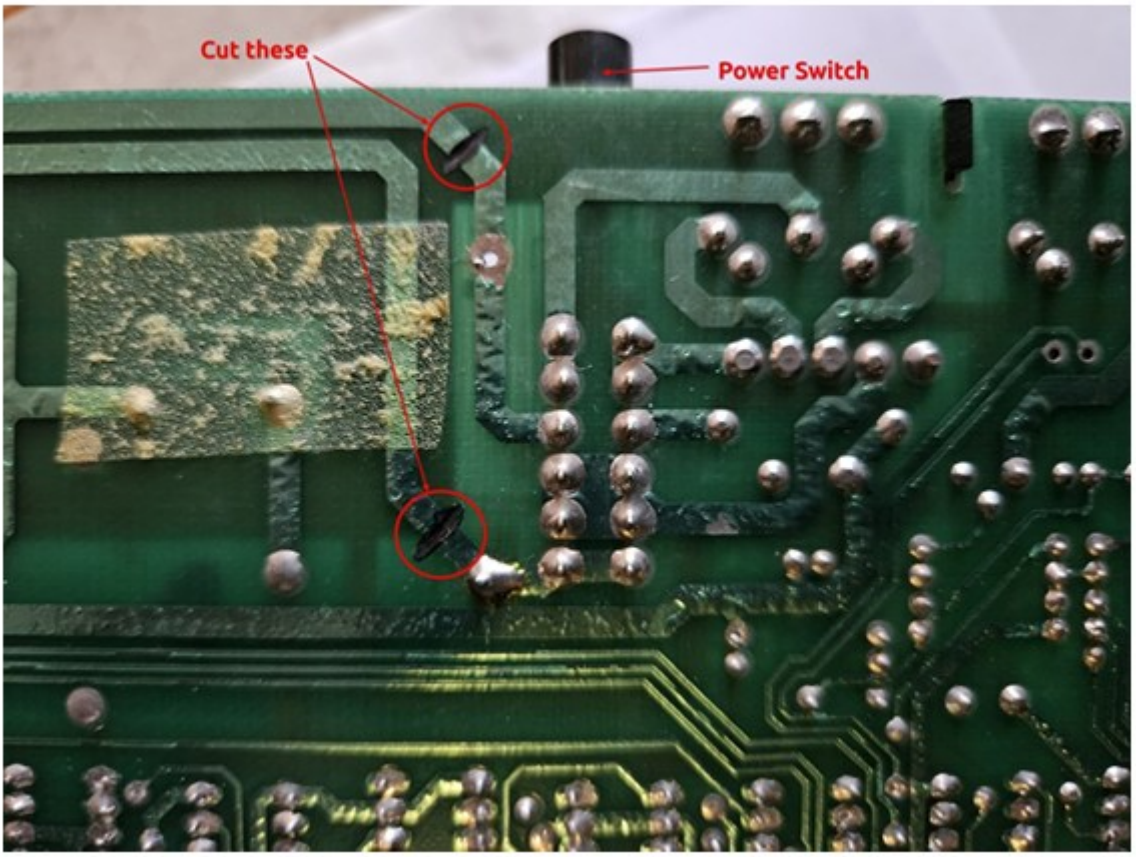
3. Remove filter capacitor C8 and power resistor R4.



4. Remove the eight original DRAM chips from their sockets.

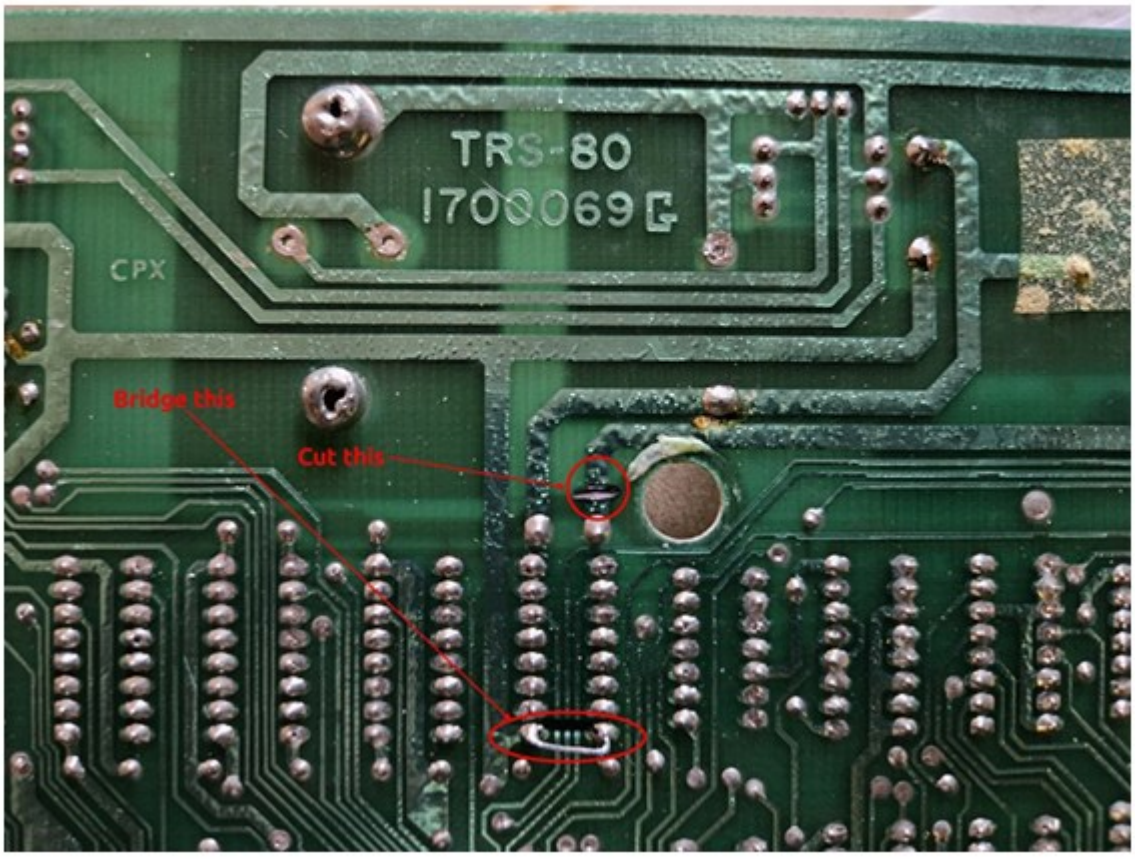


5. Cut the traces that feed the linear regulator circuits from the power switch.

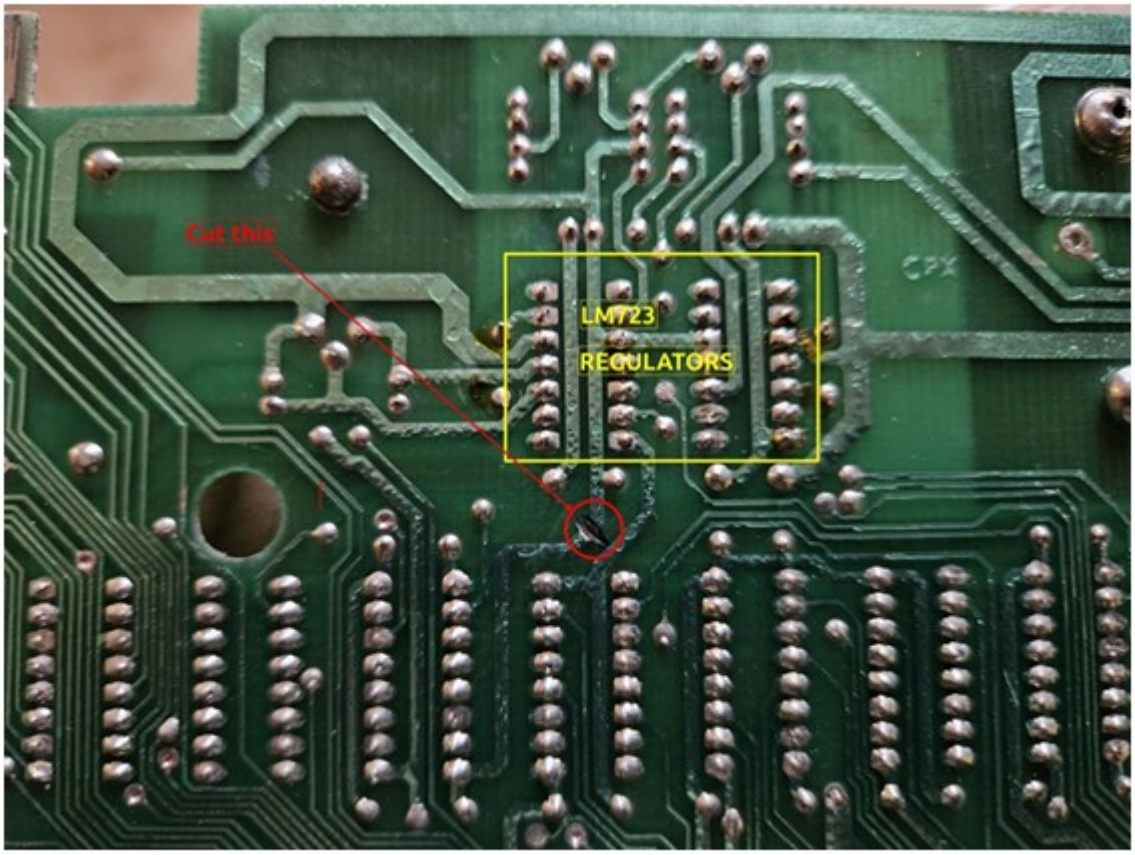


6. Cut the trace that feeds -5V power to the DRAM sockets.

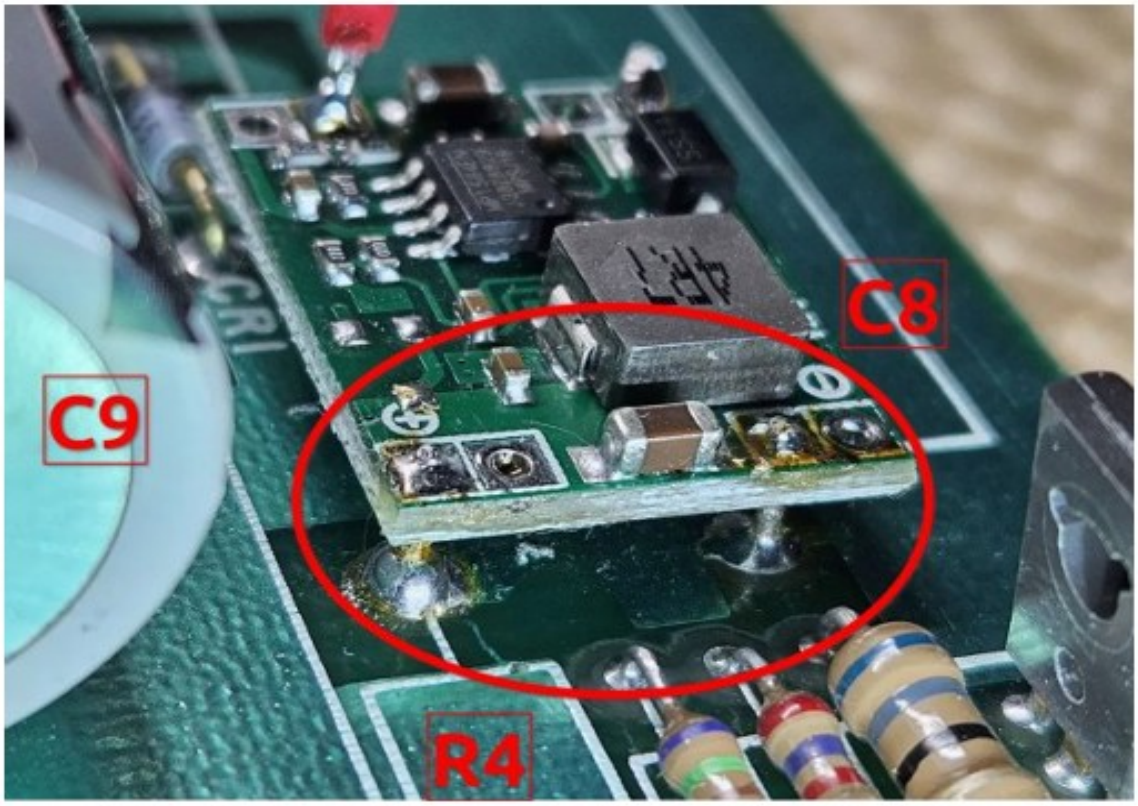
7. Bridge pin 8 to pin 9 on *one* of the DRAM sockets with a wire jumper.



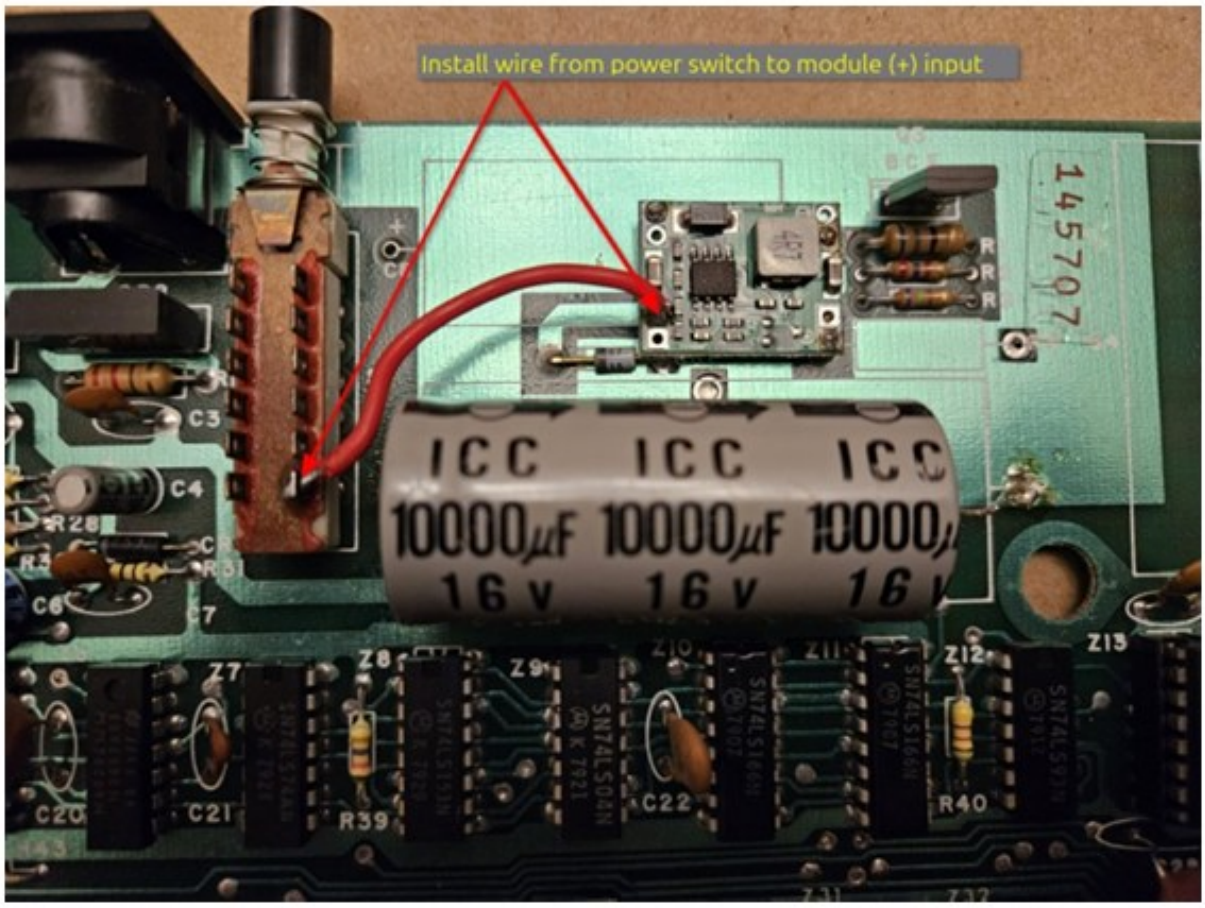
8. Cut the trace that feeds +12V to the DRAM sockets.



9. Install the switching regulator module onto the system board aligning its (-) output to the negative pad of C8 (closest to R4) and aligning its (+) output to the load side pad of R4 (closest to C8). The holes on both boards align almost perfectly. You can use two pins of a breakaway header to mount the module.



10. Install a wire from the power switch (pole that connects to the positive terminal of C9) to the (+) input of the switching module.



11. Install eight 5V-only DRAM chips into the empty memory sockets.

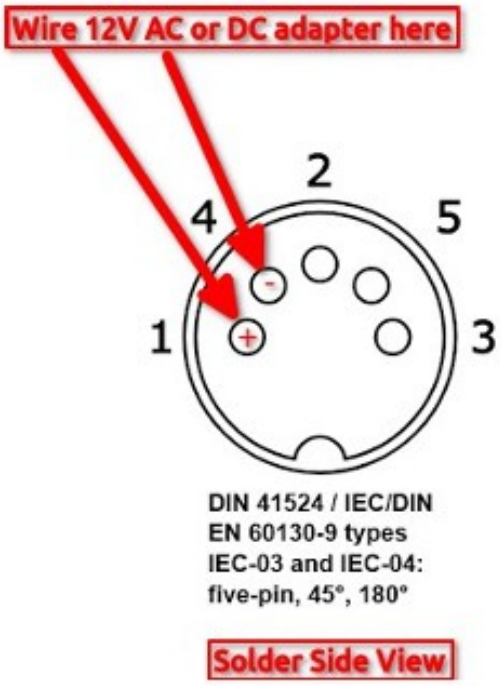
* * * * *

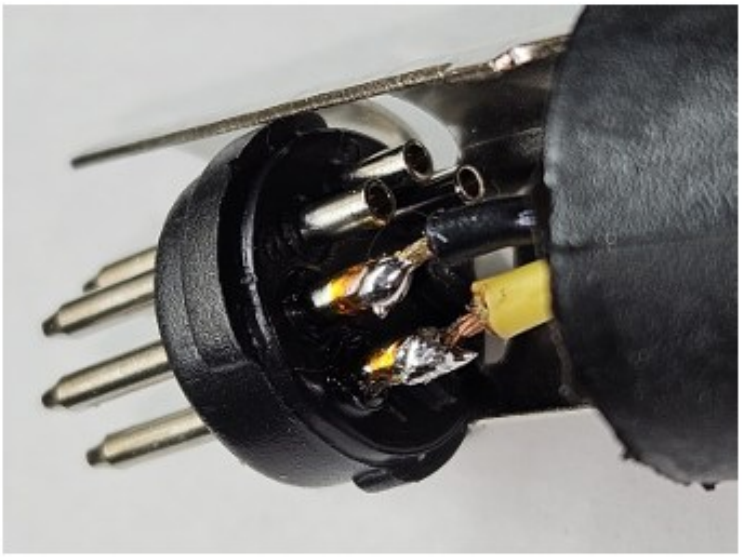
That completes the modifications to the CPU unit. If you have a working TRS-80 power transformer brick, you may power up and test the unit for proper operation. If you don't have a working brick, continue onto the optional steps to create an alternate power source.

Optional Steps To Use An Alternate Transformer

1. Obtain a suitable wall power adapter. It should be rated 12V AC or DC output with a minimum of 500mA current rating. While the switching module can handle higher input voltages, the rectifier filter capacitor C9 is limited to 16VDC.

2. Connect the two wires from the wall power adapter to pins 1 and 4 of a male DIN connector. If it is a DC adapter, wire (+) to pin 1 and (-) to pin 4.



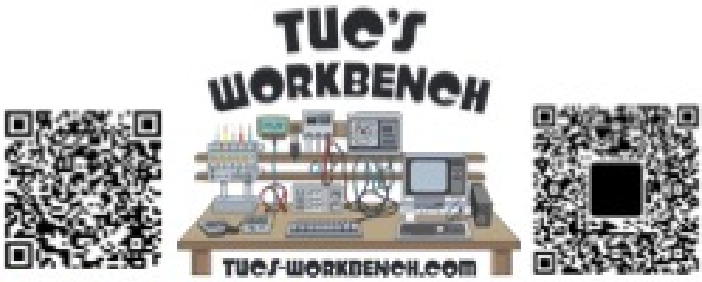




Final Check

After powering on, measure the voltage across C9 to make sure it doesn't exceed 16V DC. Measure the voltage at the module output to make sure it is close to 5.0V DC.

Congratulations! You now have a TRS-80 Model I CPU that draws only 8W from the wall instead of 15W. It runs cooler and that means less stress and deterioration of the components for a longer service life.



https://shop.tucs-workbench.com/ This Flyer online

"Our First TRS8BIT Ad" Featured Items

Enter OFTA in the order notes at checkout for your exclusive reader pricing — applied before your order ships.

One Rom - One Rom to rule them all	
24 Pin	$18.99
28 Pin	$18.99
28 Pin B (Vertical USB)	$18.99
40 Pin	$28.99
Gotek - Includes Rotary Encoder and OLED	$41.99
TRS-IO++	$200.00
Blue-Print TRS-80 Composite Video	$59.99
RGBTOHDMI -	
Composite Adapter	$11.99
TRS-80 Model I Adapter	$12.99
TRS-80 CoCo3 RGB	$12.99
Complete kit	$109.99
X-Mem-80	
Model I	$50.99
Model III	$43.99
Pepper V2 CoCo Salt Replacement	$30.89
TW LNDoubler 5/8	
Without FD1791	$134.99
With FD1791	$170.99
TRS-80 Cassette Cable	$14.99
George Hines Floppy Remapper	$34.99

If you don't see what you're looking for, and it's not on https://shop.tucs-workbench.com/ , feel free to reach out to us at : QUESTIONS@TUCS-WORKBENCH.COM

Tuc's Workbench
PO Box 69
Newburgh, NY 12551
(Most orders received by 1p Eastern are usually shipped the same day!)

Floppy Days
Vintage Computing Podcast
with Randy Kindig

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

Subscribe Today!

Tandy-specific Episodes:

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

https://www.floppydays.com

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



Universal Backpack Drive

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

Choose Options

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

$65.00

Adventures in Digital Archaeology:

Recovering SCAP-Dick Smith's lost System 80 inventory software

Introduction

A passion of mine over the years has been to document the history around the Dick Smith System 80, a TRS-80 Model 1 clone computer popular in Australia and New Zealand from 1980 to 1983. Although much of what was sold for the machine was, in fact, TRS-80 Model 1 software, Dick Smith did commission programmers to write original packages to support the System 80. Over the years I'd managed to collect this software and make it available on my System 80 website here.

That is all except one package…
That package was Stock Control and Pricing, usually abbreviated as SCAP.



Figure 1. The front page of the SCAP brochure

Ok, one could argue that a stock and inventory program doesn't usually get the adrenaline pumping. However, the manual and software for this package had alluded me and I really wanted to complete the Dick Smith Software Collection. I truly thought that SCAP was lost forever until on March 1st, 2026 when, while idly Googling for it (something I did every year or so…just in case!), I came across a post from Alan Laughton, administrator of the Microbee Technology Forum.

It seems about a year ago, Alan had been passed the SCAP manual from a member called kimjohn. The manual had two program listings at the end. This opened the possibility of a working version of the software being recovered from these listings!

Alan scanned the manual at 300 DPI capturing the contents in a PDF file. In the process he converted those program listings to raw text using OCR scanning. It wasn't an easy process. The listings were in a tiny font and had very few spaces. After converting to text, Alan had laboriously gone through correcting any OCR errors he could see. This meant correcting "1"s which had been mistaken for "I"s, "0"s converted as "O"s, "5"s which had been changed to "S"s (and vice versa) and so on. He had also converted some of the <down arrow> line feeds in long single PRINT statements to separate PRINT statements for easier editing. However, even after all this work, he still couldn't get the package to work.

Alan graciously passed both the manual PDF and the program listings as text files onto me, to archive on the System 80 website, and to see if I could get the package running.

And so the adventure began…

SCAP: An overview

As mentioned previously, SCAP was a Stock Control and Pricing package, used to record the comings and goings of inventory in a retail store. The manual is written in plain, relatable English, guiding the user by the hand through setting up and running the software in their own business.

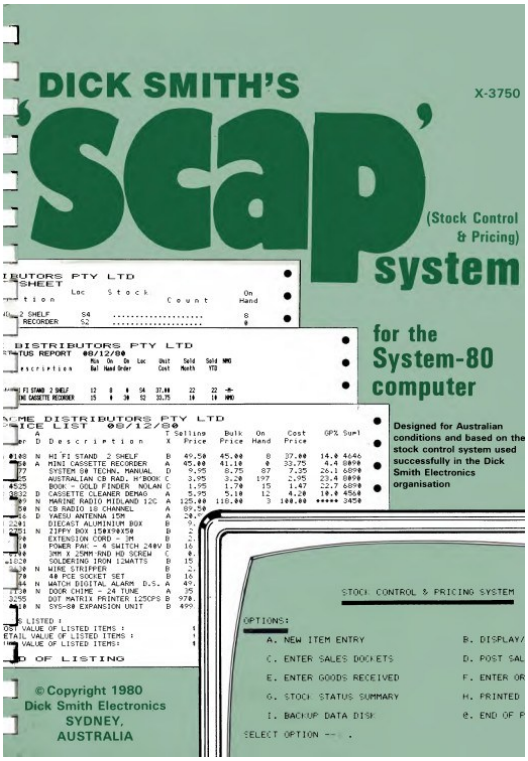


Figure 2 - The front page of the SCAP manual

The manual (and the code) assumes the user has a System 80. Although the software could work just as well on a TRS-80 Model 1, that machine is not mentioned. It is clear that Dick Smith saw SCAP as a useful, "serious" application which could help System 80 sales into the small business market. Figure 1 shows the ideal hardware; A System 80 (preferably the MKII business variant), a dedicated monitor, disk capable with two drives and a printer. The full deal!

SCAP came on a self-booting disk, which contained PERCOM's MicroDOS (rebadged as OS-80 and sold separately by Dick Smith). MicroDOS is tiny, only 7k. It loads into memory on boot and extends Level II BASIC with a few disk handling commands and functionality. SCAP itself consisted of two programs written in BASIC, the larger of which loaded and ran automatically when the computer booted.

Users would carry out all tasks through a series of menus, although users were instructed to use the MicroDOS formatting command directly from BASIC when formatting data disks. All data entered was kept on the data disk in the second drive (Drive 1).

Although the System 80 could be expanded to 48K of RAM, memory was expensive in 1981 so the program, at least according to the manual, was written (or rather squeezed) to be run in 32K. This small memory space, and the fact it was written in BASIC (and mostly Level II BASIC at that!) put many constrains on the software mainly…

- REM statements are few and far between. These statements take up precious RAM space so the program is essentially undocumented

- The code is classic "spaghetti" code with many GOSUBS and GOTOS

- Given that BASIC is an interpreted language (as opposed to machine code), some processes, like data entry can be slow. Fast typists would have had to slow down for the software to keep up.

Code for data entry is split between two programs. Data is loaded in one program, but the second one is called to organise and save data on the disk.

Getting SCAP working

Step 1. Setting up the test platform

I used George Phillip's excellent TRS80gp emulator for most of the testing and debugging. Just for authenticity, I had it operating in Model 1 mode but using the System 80 ROM.

The first problem was getting the code, onto a bootable floppy disk image. A bootable MicroDOS disk image was no problem to produce from the full MicroDOS package. MicroDOS doesn't have a typical DOS directory structure though. BASIC Programs are saved by the user specifying a starting disk sector. The program is then written as a continuous file starting from that sector. MicroDOS tells the user the last sector used for the program. It is up to the user to note this and not overwrite those sectors with any other program being saved.

There were two BASIC programs that constituted the SCAP package, the main program called SCAP and a second one called SCPSST which was a routine which added to and rearranged data on the data disk after data entry. So, where were these SCAP programs supposed to reside on their program disk under MicroDOS? There was no mention of their sector start locations in the user manual. Thankfully the answer was in the code itself. In the main SCAP program there was a single call to LOAD 335, which I figured must load SCPSST from its staring sector at 335. In the latter, once all data disk writing was

finished, there was a call back in turn to the main SCAP program with a LOAD 240 command. This indicates the SCAP program starts at sector 240.

Once I had a bootable MicroDOS disk image, using the emulator I entered in the main SCAP program and saved it sector 240. After a NEW, I then entered the second program and saved it starting at sector 335. Using MicroDOS, I then formatted a data disk in Drive 1. I was now ready to go.

Step 2. Correcting the code from OCR and transcription errors

Even though Alan had done a valiant job in picking up OCR errors, there were many more in the code. I went through the program line by line, comparing the manual's code with Alan's text files. It was laborious work. In the end I found 72 lines with OCR errors in the main SCAP program and one in SCPSST. There were even two lines labeled 4990, existing between a page break! The first of these, which loaded a non-existent program at sector 350 when a backup routine was called, simply didn't make sense!?. I ignored this line.

Once I'd fixed these OCR errors, the program made it to the opening screen and beyond. Hooray!.



*Figure 5 – SCAP Opening Screen***Step 3.**

Two thorny problems revealed during testing

However, as expected, other issues soon surfaced. There were two…

(i) An illegal function error on the first initial data entry

The first issue showed itself when data was entered for the very first time onto a blank disk. During this procedure the program crashed after typing in the identifier for the first data record (Figure 6).

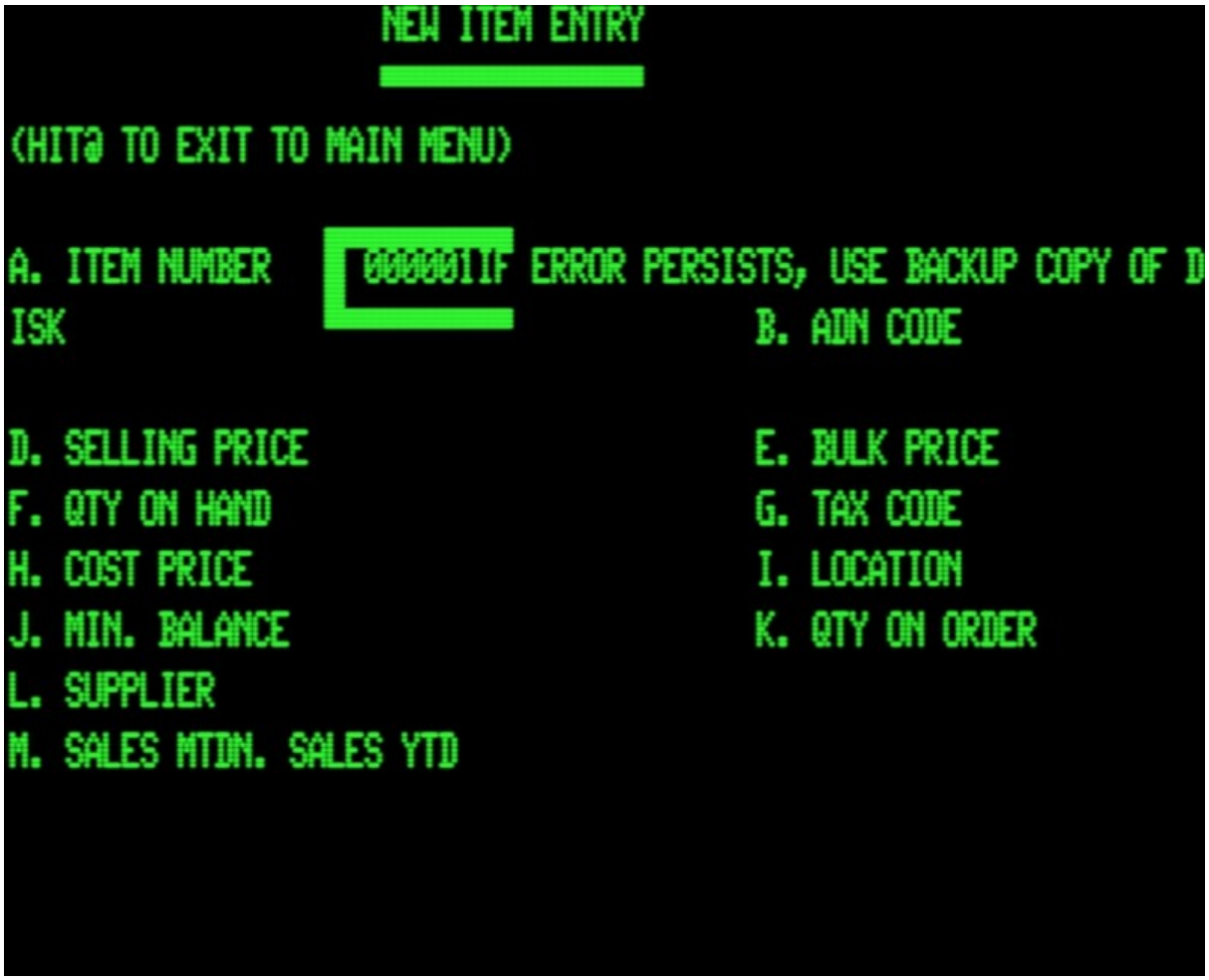


Figure 6. SCAP crashing with an error message on initial data entry

On throwing up the error the user is given the unhelpful but understandable message that an error has occurred and the program terminates. Checking the error code from BASIC once the program terminated pointed to an illegal function call.

Further experimentation showed that after being thrown back to the BASIC prompt, if the program was restarted (re-run), or a warm boot was initiated, or the data disk already had data, then things were fine. The error only happened after a cold boot, with a blank data disk??

Using TRON I tracked down the line where the error was occurring. It was the statement J=ABS(C(N)) in Line 2850. N it seemed was –1. Negative numbers are not allowed to index arrays in this version of BASIC.

I tried to work back to see where the value of N was coming from but, well, you know, many GOTOs and GOSUBS are hard to follow. Uncommented spaghetti code didn't make it easy.

(ii) Error 2 – Out of memory after a reload of SCAP from SCPSST

The second problem occurred once a few records had been entered and SCPSST was called to sort and save the new data on the data disk. This seemed to work, but on reloading the main SCAP program, an out of memory error occurred. (Figure 7).

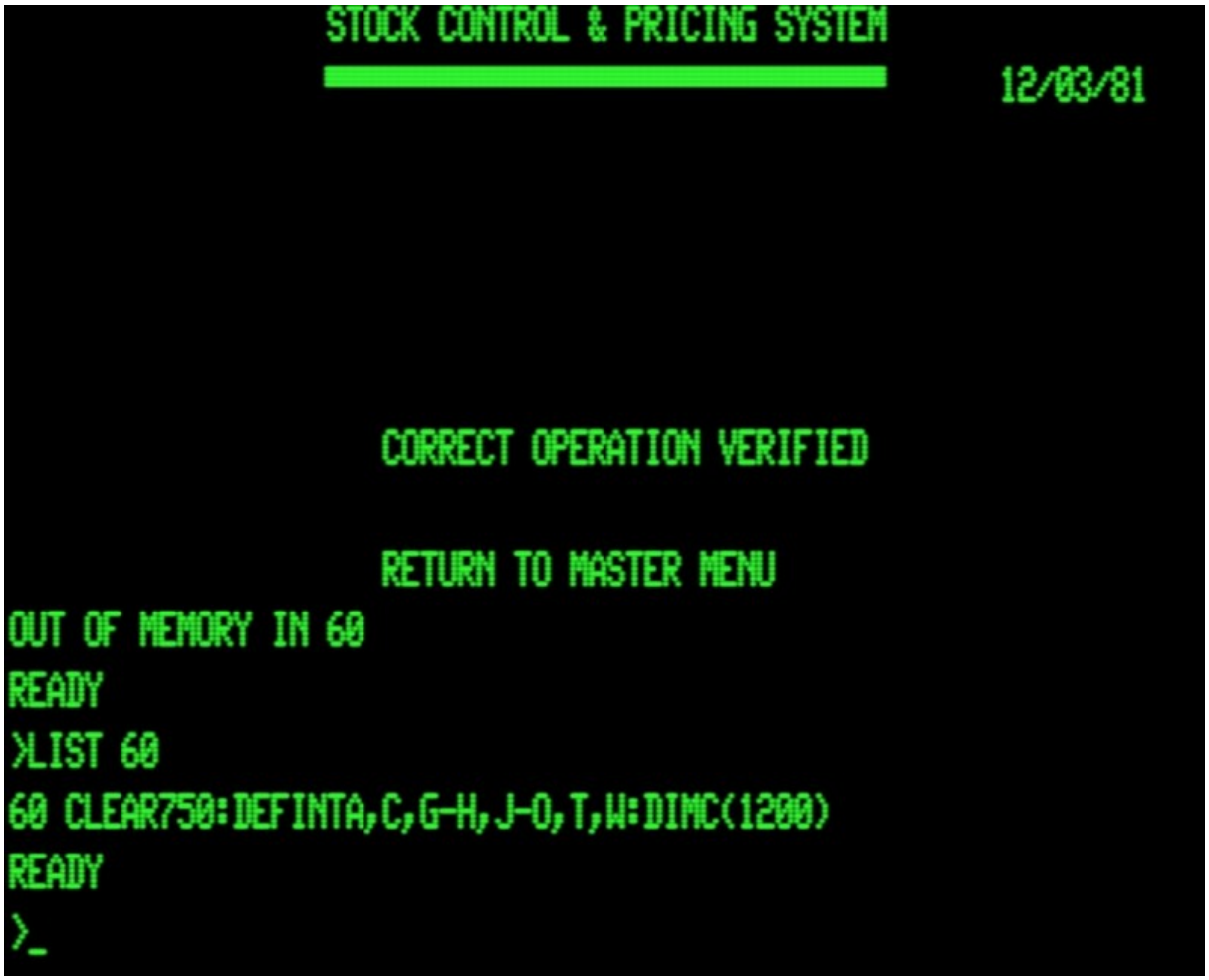


Figure 7. Out of memory error showing the line where the crash occurred

A.I. to the rescue

After trying to find where things were going wrong and failing, I had an idea. A.I. is suppose to be good at coding. Let's see if those modern LLMs can help.

This was a last resort. Indeed, I was extremely sceptical that A.I. would be able to help me at all with these problems. After all, I was dealing with an archaic language (Level 2 BASIC) running under an extremely obscure DOS with code that jumped around to somewhere else at almost every second line.

I didn't subscribe to any of the big three LLM services (Claude, Chat-GPT and Gemini) but, what the hell, I'll just see what the free tiers could do. I outlined the problem and what I knew so far to the models, uploading the two pieces of code and the MicroDOS manual.

Claude mulled on the program then spat the dummy saying I'd used too many tokens, at least for the free service. Chat-GPT congratulated me on my assumptions and error finding so far, elaborated on what I'd already said and was essentially no help whatsoever.

Gemini (Google's Chatbot) on the other hand, nailed it.

Here's what Gemini said regarding the first error …

"Error 1: The "Illegal Function Call"

The culprit is a negative array subscript. In TRS-80 Level II BASIC, passing a negative index to an array (like C(-10)) throws an "Illegal function call" (Error 5).

The Mechanism:

1. *On a cold boot with a fresh disk, GOSUB 3170 executes a GET #1, PT to read the system sector. Because the disk is completely blank, the field buffer I5$ reads as null/garbage.*

2. *Line 3180 converts this garbage using TN=CVI(I5$), which results in a random, often negative, number.*

3. *Line 3190 sets OE = TN, so OE becomes this negative number.*

4. *Because the disk is blank, the program triggers the initialization routine at Line 2110. Line 2110 correctly resets TN=1, TC#=0, etc. However, the programmer forgot to reset OE!*

5. *When you enter your first item, Line 750 sets the binary search upper bound: NH=OE. Because OE is still negative, the search calculates a negative array index N.*

6. *Line 2850 executes J=ABS(C(N)). Since N is negative, Level II BASIC immediately throws an Illegal Function Call.*

7. On a warm boot, the initialization has already saved TN=1 to the disk. The GET command reads it correctly, OE becomes 1, the search evaluates to 0, and Line 2850 is safely bypassed.

The Fix:

Simply add OE=1 to the initialization variables in Line 2110.

Change Line 2110 to:
2110 TC#=0:TP#=0:TB#=0:TM#=0:TY#=0:TR#=0:TI=0:TN=1:HI=0:HN=1:G=0:ER=1:OE=1"

I implemented this suggestion and it worked! The error disappeared and further testing showed everything was now working ok! Well done Gemini!

Here's what Gemini said about the second error…

"Error 2: "Out of Memory" after the sort routine
This one is a classic memory management trap, and it stems from a hardcoded assumption in the Sort program (SCPSST).

The Mechanism:

The Main program, combined with OS-80, the DIM C(1200) array, and string space, takes up nearly 34 KB of RAM. This means it requires a 48K system to run. On your initial boot, your machine (or emulator) provides the full 48K (which goes up to memory address &HFFFF / 65535).

However, look at the Sort program:

- *Line 50 POKEs addresses 16561 and 16562. In Level II BASIC, this is the MEMTOP / string space pointer. It temporarily lowers this limit to &HBE40 (48704) to protect the machine language sorting routine from being overwritten by BASIC.*

Line 240 is supposed to release this protection before reloading the Main program. It does this by executing POKE 16561,255: POKE 16562,191.

*The problem? 191 * 256 + 255 = 49151, which is &HBFFF. The programmer hardcoded the memory reset to the top of a 32K machine.*

When the Sort program finishes, it artificially caps your 48K system at 32K. It then loads the massive Main program. When the Main program hits Line 60 and attempts to run CLEAR 750 and DIM C(1200), the array space violently collides with the string space inside that artificially cramped 32K limit, resulting in an "Out of Memory" (Error 7) crash.

The Fix:

Instead of hardcoding the memory limit to 32K, we can make the Sort program read whatever the memory limit actually is before it changes it, and then restore those exact values.

Add a new Line 45:
45 HM=PEEK(16561):HL=PEEK(16562)

Change Line 240 to:
240 POKE 16561,HM:POKE 16562,HL"

Now in this one, Gemini nailed the cause but not the solution. Any values assigned to variables at 45 were nuked by the CLEAR statement resetting the variable table in line 60 after protecting memory size in line 50. After being made aware that its fix didn't work Gemini suggested a work-around by saving the max memory variables in the cassette buffer before the CLEAR e.g. …

"Change your Line 45 to:
45 POKE 16896, PEEK(16561) : POKE 16897, PEEK(16562)
Then afterwards retrieve them directly from the buffer to restore the memory:

Change Line 240 to:
240 POKE 16561, PEEK(16896) : POKE 16562, PEEK(16897)

This completely bypasses the variable table, making it immune to the CLEAR command on line 60."

This worked but in the end, I didn't implement Gemini's suggestion. I decided to simply hardcode an assumed memory limit of 48k before calling the main SCAP program from SCPSST The following two sections explain why…

The production code vs. the manual code

The code in the manual is simply too large to run in 32k of RAM. Where memory was capped at 32k during testing the main SCAP program ALWAYS ran out of memory! This was even after I deleted any REM comments I added myself to the original program, and resorted back to using line feeds to save space in PRINT statements, as the original code had.

However, I COULD get SCAP to run in 32k of RAM if I stripped out ALL the REM statements (excluding the called one at Line 4310) in the manual code. This is an indication that the manual listing was not exactly the same as the production code, which could also explain how first error mentioned above found its way into the manual listing, and the existence of two lines of 4990 in the same. The code in the manual may have been transcribed and embellished with REMS rather than have been a true listing of the production code.

The reconstructed version vs. the manual code

Anyway, for the reconstructed version of SCAP on the disk image, I elected to use the more verbose version of the code found in the manual together with a few REM additions of my own, Gemini's correction in Line 2110 and extra PRINT statements substituted for the <down arrow> when used in long single PRINT statements. These extra PRINT statements made the code much easier to maintain in text files. Consequently, the restored version of SCAP needs 48k of RAM to work. This is of no concern as 48K is the usual RAM for emulators assuming disk systems and probably any disk-enabled real System 80s in the wild.

Given this (and as alluded to above), I made Line 240 of SCPSST to read POKE16561,255:POKE16562,255 thereby resetting top of memory to 48K, rather than 32k as in the manual code, before LOADing SCAP.

As a final tweak, I used the appropriate MicroDOS commands to alter the SCAP disk image, to load SCAP on boot, with a friendly message telling users that the program was loading and to please wait. I'm sure the original commercial disk ran the program automatically and displayed a similar message.

Final words

As mentioned in the introduction, I'd been looking for SCAP for years. It's now uploaded to the Dick Smith Software Section of the System 80 archive site. At that link you can pick up the manual and a zip file with disk images of the restored SCAP program and a blank data disk. Although this is a restored 48K-only version from code in the manual rather than an image of the commercial disks, it seems to work as described. It's great to have it added it to the repository. What is missing still is the demonstration data disk, referred to in the manual's tutorial. That may be lost forever, but you never know what might be found in someone's dusty garage in the future.



SCAP running on a real System 80 (i.e. my own!)

Debugging was fun. I learned about MicroDOS, was reminded just how convoluted (and indecipherable) some BASIC programs could be AND was very impressed that Google's Gemini could work through this archaic convoluted code, diagnose and suggest work-arounds for both the errors.

Anyway, I hope you enjoyed this example of digital archaeology.

Terry Stewart (Tez)
14th March, 2026

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



Voidstar has release a new game for the TRS-80 Model I and III called BEAST. He writes about its origin:

> This was kind of an accidental project. I never messed with 2x3 semigraphics, and I wanted to demo about it using the POLY-88 (since it's that systems 50th anniversary this year) at VCF next month. Well, its VTI video card is having some issues now. So - I figured how hard could it be to port the 8080 code and 2x3 semigraphics over to the Z80 and the TRS-80's 2x3 semigraphics? Well, it was an adventure - the POLY-88 at least has a 60Hz timer for RNG and tempo-timing. I aimed to support a TRS-80 M1 without an Expansion Unit (since that's all I have in real hardware). But thanks to help with some folks, I went ahead and made the binary/CAS/WAV/CMD compatible with both the M1 and M3 (sorry M2's and M4's!). I do this by probing for the M3's timer clock first. Anyway, it matches the POLY-88 version. There is a README TEXT (and source) in the ZIP for more instructions.

You can watch the game play at https://www.youtube.com/watch?v=k7GsxYuFI3o

The GITHUB is at https://github.com/voidstar78/BEAST/tree/main/TRS80_VERSION

MAV'S TANDY ASSEMBLY QUARTERLY COLUMN

TA26 NEWS

by Ian Mavric



Bringing you the latest news you need to know about the greatest Tandy TRS-80 show on the planet. In this article I mention some of the great speakers and other attractions to be seen at the show.

KEYNOTE SPEAKERS ANNOUNCED

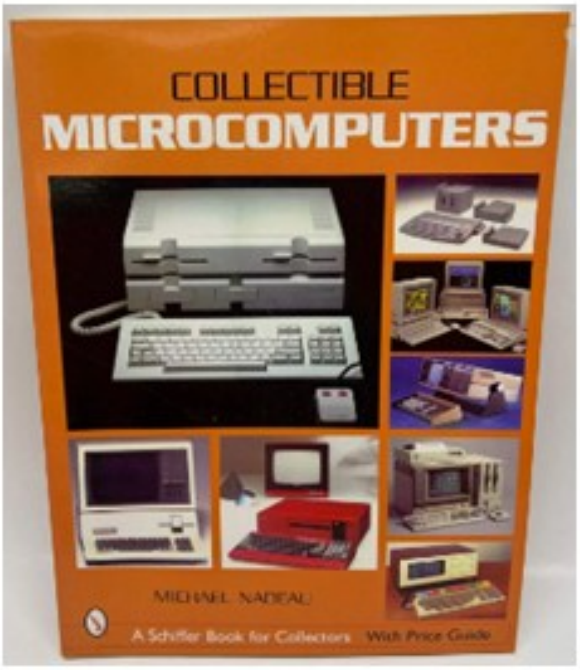
Michael E. Nadeau - former editor at Hot Coco and 80micro magazines

Working at the coalface of TRS-80 when the machines were cutting edge technology, Michael started working at 80micro in 1981. At the time the magazine was getting bigger and bigger and the editor-in-chief was Wayne Green. In 1983 Wayne wanted to spin the Color Computer coverage out of 80micro into it's own magazine, which became Hot CoCo, and Michael was put into the editors seat. After three successful years Hot CoCo re-joined 80micro, and in 1988

Michael took over as editor at 80micro. After 80micro finished up Michael moved to Byte magazine as executive editor and running the Byte testing lab. It will be fascinating to hear what it was like working at a place potentially stressful to get a good publication out monthly. Remember some of those 80micro and Byte issues were more than 300 pages each. In the early 2000s Michael wrote one of the first books on collecting vintage computers, years before the hobby grew to the enormous following it has now.



Pictures: Michael E. Nadeau, now and at 80micro in 1988



Michael's Collectible Computers book

Jerry Heep - Engineer at Tandy Radio Shack 1975-2012

Our second keynote we are very proud to have is Jerry Heep, who worked at Tandy in engineering and design from the 70s until the early 2000s. Jerry has many patents and had a hand in the design of many products sold through Radio Shack stores. Jerry worked with chip designers at Motorola to design integrated circuits for the Green Thumb project, which would eventually be utilised in the Color Computer. His work directly related to the Coco being what is was.

Coco fans will not want to miss either of these great presentations about the early days of those computers.



Picture: Jerry Heep at VCF-SW in 2025 showing off one of his patents and some of the many projects he worked on

Jeff Birt - Hey, Birt! - everything you wanted to know about soldering on vintage computers but were afraid to ask

Tandy Assembly regular Jeff Birt will be back once again this year in the Tandy Tech Track with a lesson about soldering. He will also cover de-soldering and board repair work. So many repairs to vintage computers require unsoldering failed components and replacing them, Jeff will cover the best way to tackle these repairs, without overheating the component or circuit board.



Picture: Jeff Birt working on a Model I motherboard

Arno Puder and Sacha Häberling - forget GameBoy, here's TrashBoy!

Details are few at this point in time but what I can tell you is that Arno and Sacha have been working on a new project called TrashBoy, a tip-of-the-hat to the old GameBoy, we all can't wait to see it. Sacha and Arno are TA regulars, and it will be great to see them back again.



Picture: early photo of Arno and Sacha at a previous TA.

Scott Toenneissen - XLR8er boards

The XLR8er was an accelerator board for the Model 4 and 4P, which increased the clock speed, added more Ram, and upgraded the CPU for faster operation. It's currently available in reproduction form as the 4cellerator. Scott will talk about this history of the product, its capabilities, and why it's still an interesting device to add to your TRS-80.



Picture: Scott will give his first Tech Track talk at TA26

NEXT TIME: Show time!

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 FUTURE IS HERE: THE TRS-80 AND ITS PREMIER NEWSLETTER!



To get your free newsletter, complete the form below.
Return by mail. Offers subject to validation.



Tandy Corporation, TRS-80 Division

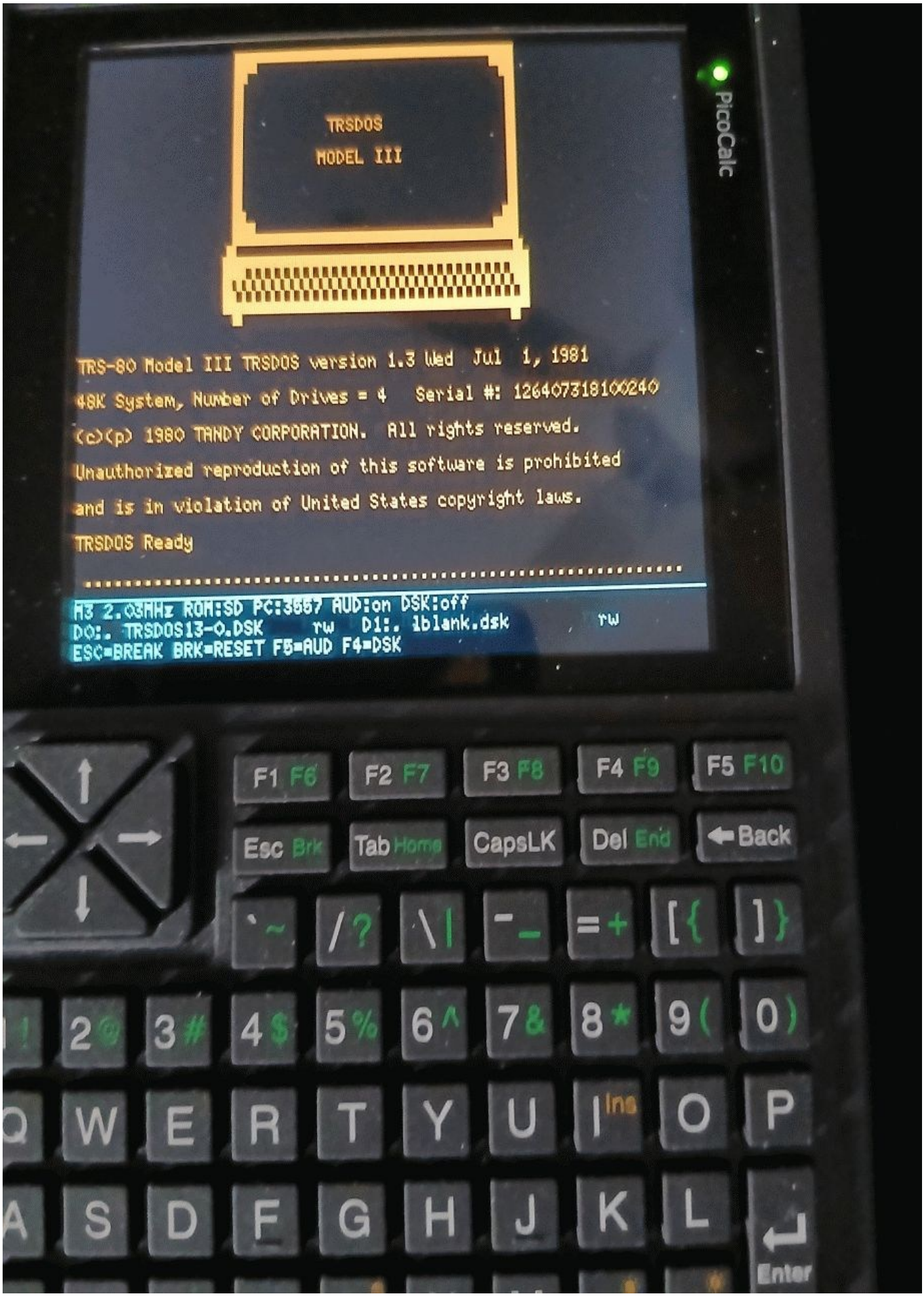
Name Radio Shack _____

Address _____

City _____ State _____

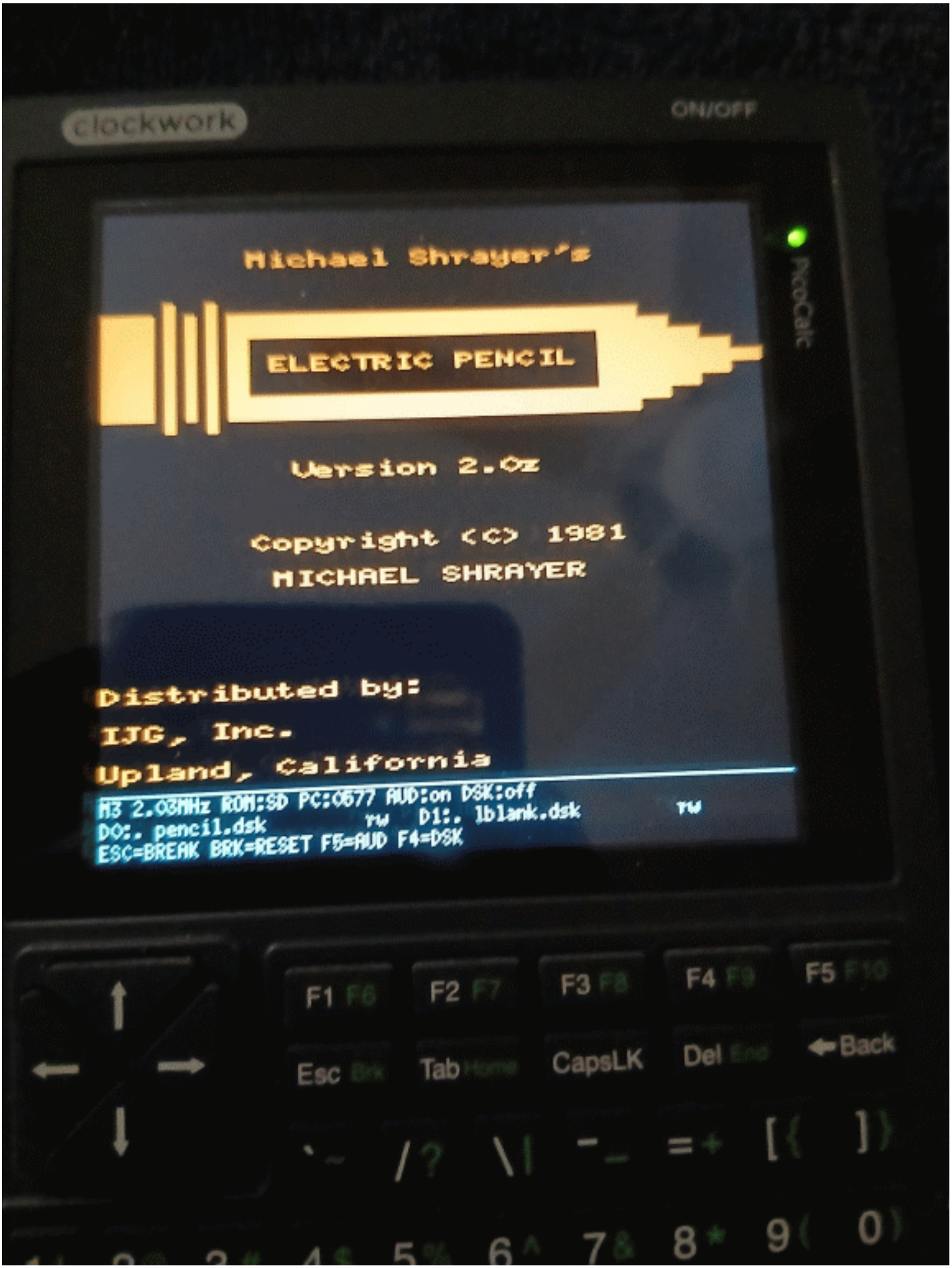
Payment enclosed/Check one ☐

The Tandy TRS-80, the heart of your computing world, and TRS8BIT, your guide to its secrets. From the depths of machine code to the highest peaks of arcade action, we cover it all.



https://forum.clockworkpi.com/t/picocalctrs-trs-80-model-3-emulator/22215

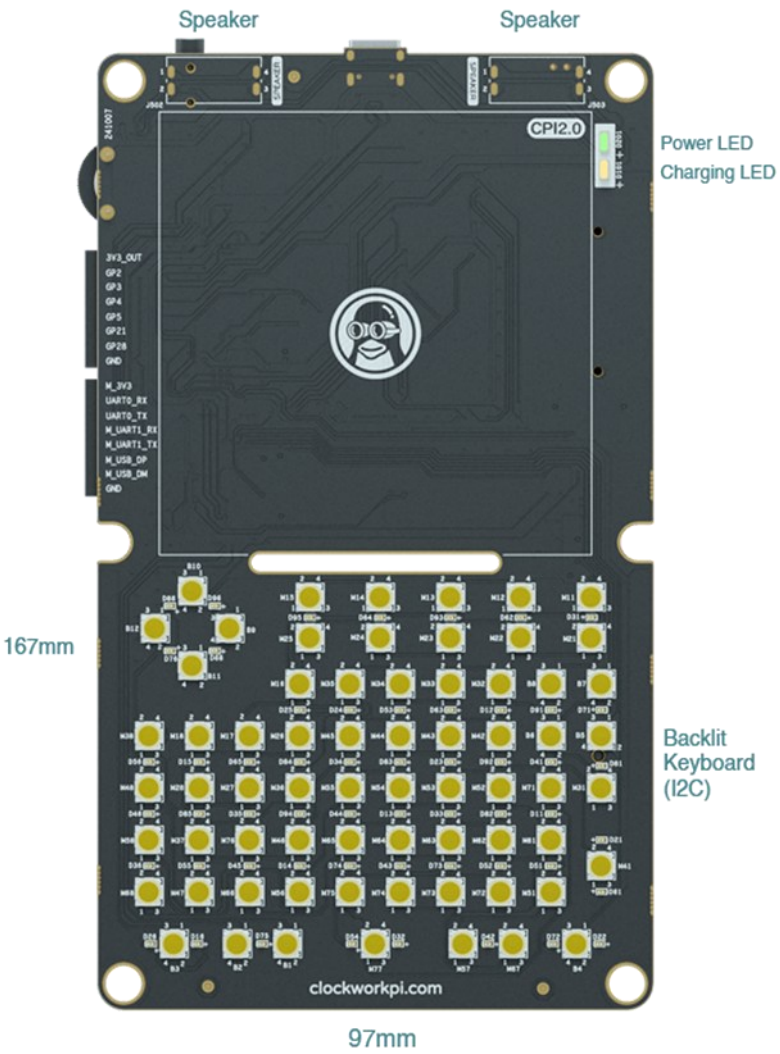

LDOS
Date ?
PicoCalc
F1 F6
F2 F7
F3 F8
F4 F9
F5 F10
Esc
Tab
CapsLK
Del
Back

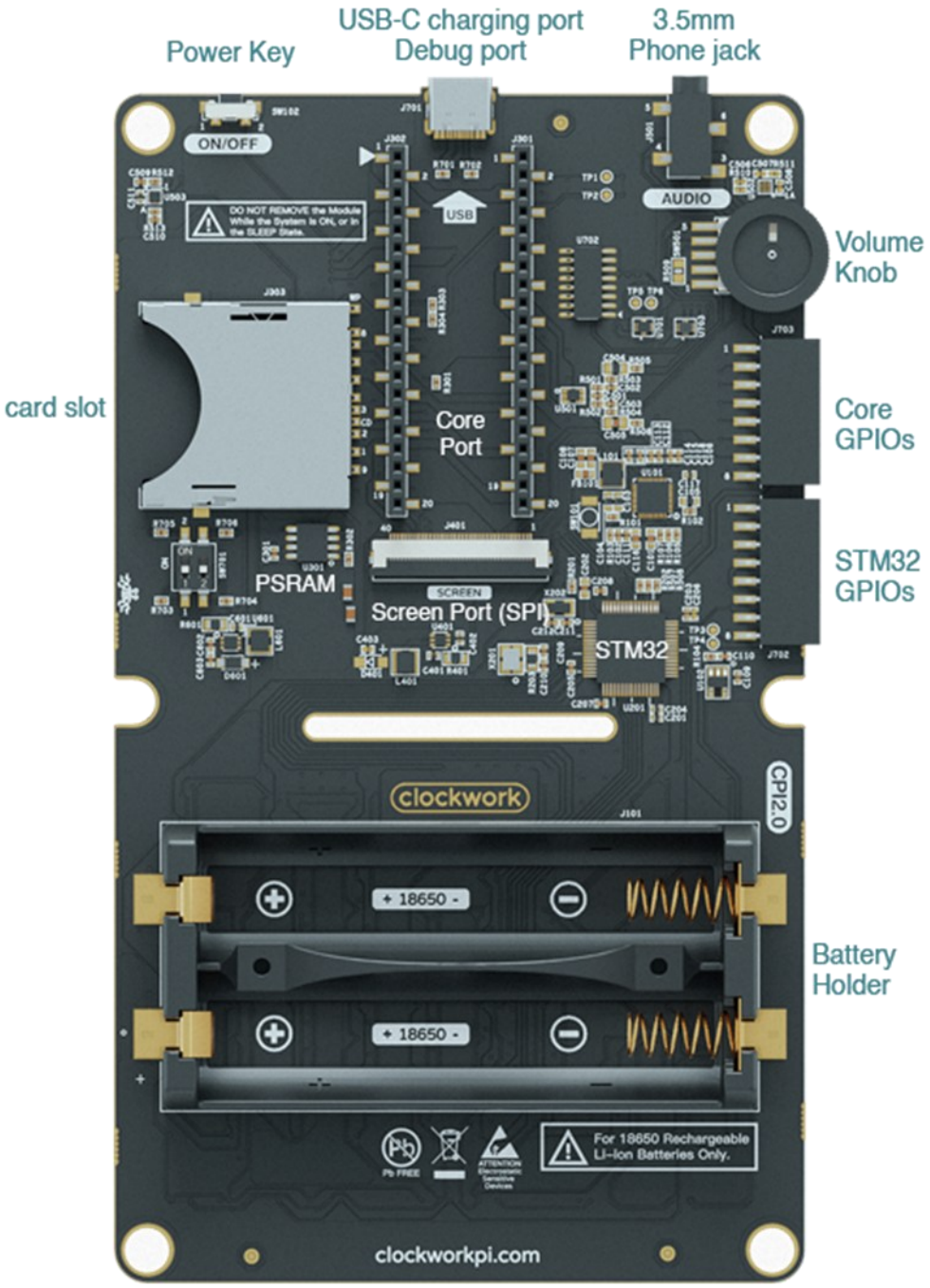

clockwork
ON/OFF
PicoCalc
Michael Shrayer's
ELECTRIC PENCIL
Version 2.0z
Copyright (C) 1981
MICHAEL SHRAYER
Distributed by:
IJG, Inc.
Upland, California
M3 2.03MHz ROM:SD PC:0677 AUD:on DSK:off
D0:. pencil.dsk rw D1:. 1blank.dsk rw
ESC=BREAK BRK=RESET F5=AUD F4=DSK
F1 F2 F3 F4 F5
Esc Tab CapsLK Del Back

Introducing the clockworkPi v2.0

The ClockworkPi v2.0 series offers you an efficient and cost-effective MCU development solution.

ClockworkPi v2.0 supports the Raspberry Pi Pico H/ Pico WH series and various compatible modules. CPI 2.0 features an SPI screen, SD card slot, integrated onboard 8MB PSRAM, dual PWM speakers, 3.5mm phone jack and 18650 lithium battery charge & discharge management, providing the system with ultra-long battery life.





ClockworkPi v2.0 also includes an STM32 chip (similar to the south bridge) to provide a QWERTY keyboard module with I2C interface, and manage the backlight of the screen and keyboard, power management etc., helping you simplify development, focus on application development and save IO resources.

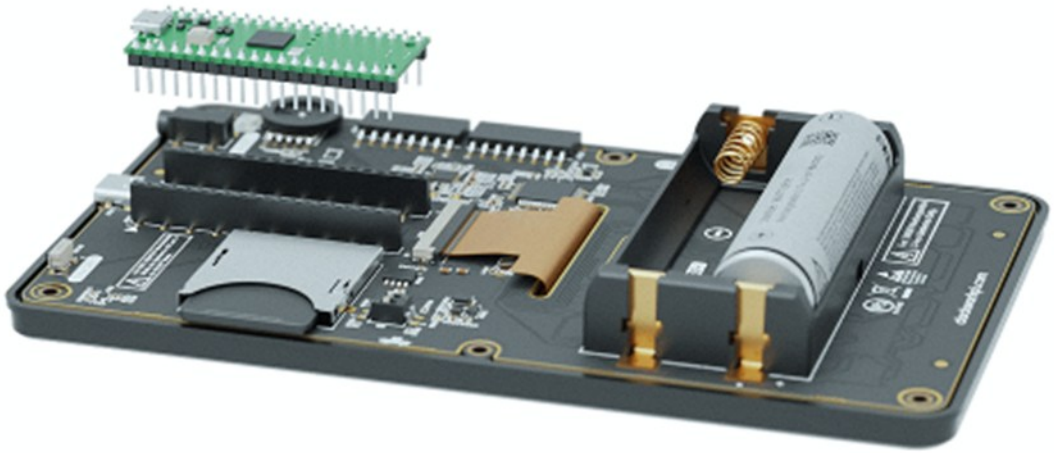
The schematics and related design materials are released under the GPL v3 license, and you can find them on our GitHub.

Pico 1H and Pico 2H are standard microcontroller boards without Wi-Fi, ideal for general embedded development, hardware wallet, independent games and education. Pico 1WH and Pico 2W include Wi-Fi, making them perfect for IoT, wireless communication, and smart home applications.



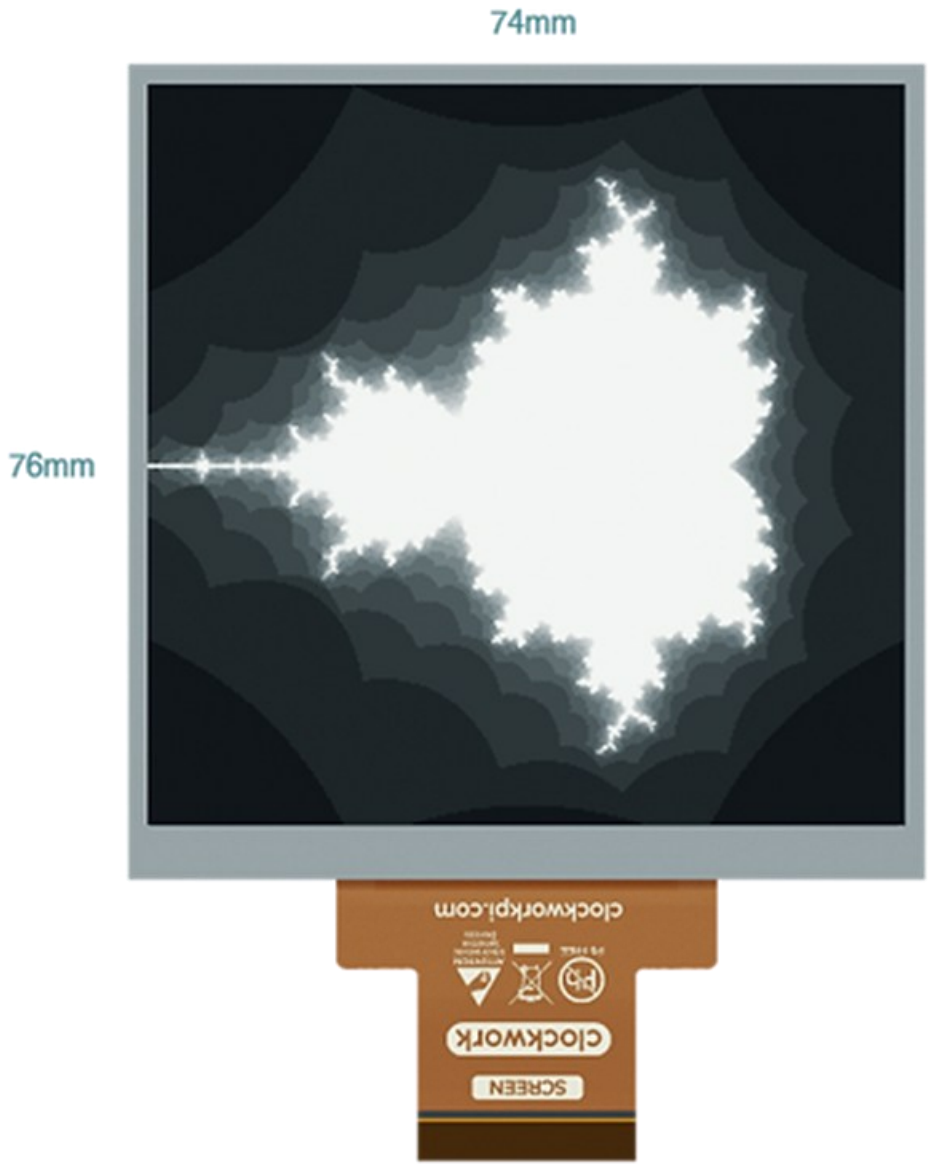
* Pico 1 H and WH come with pre-soldered headers

Compared to the Pico 1 series, the Pico 2 series may have better performance, higher memory and larger storage space, and may be more suitable for RTOS and game emulator development.



** The Pico2 series RP2350 chip also includes a Hazard 3 RISC-V core.

320x320 4-inch IPS screen

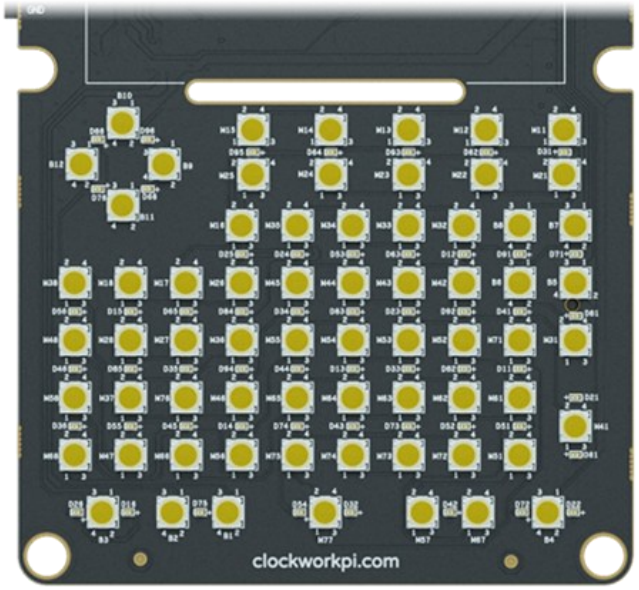


We carefully selected a 4-inch IPS screen, with a 320x320 resolution and SPI interface, making it a perfect match for most MCUs, optimizes resource usage while delivering high-quality visuals for most applications and 80s retro games.

We also designed a tempered glass cover to protect the screen, making it easier for you to carry PicoCalc with you.

Introducing the ultra-portable QWERTY Backlit keyboard

We are excited to introduce an ultra-portable QWERTY keyboard designed for MCU environments. Featuring 67 physical keys and an adjustable backlight, it connects to the core module via a simple I2C interface, optimizing resource usage and streamlining development. Additionally, you can customize its firmware using the Arduino IDE or STM32 official development tools to suit your specific needs.

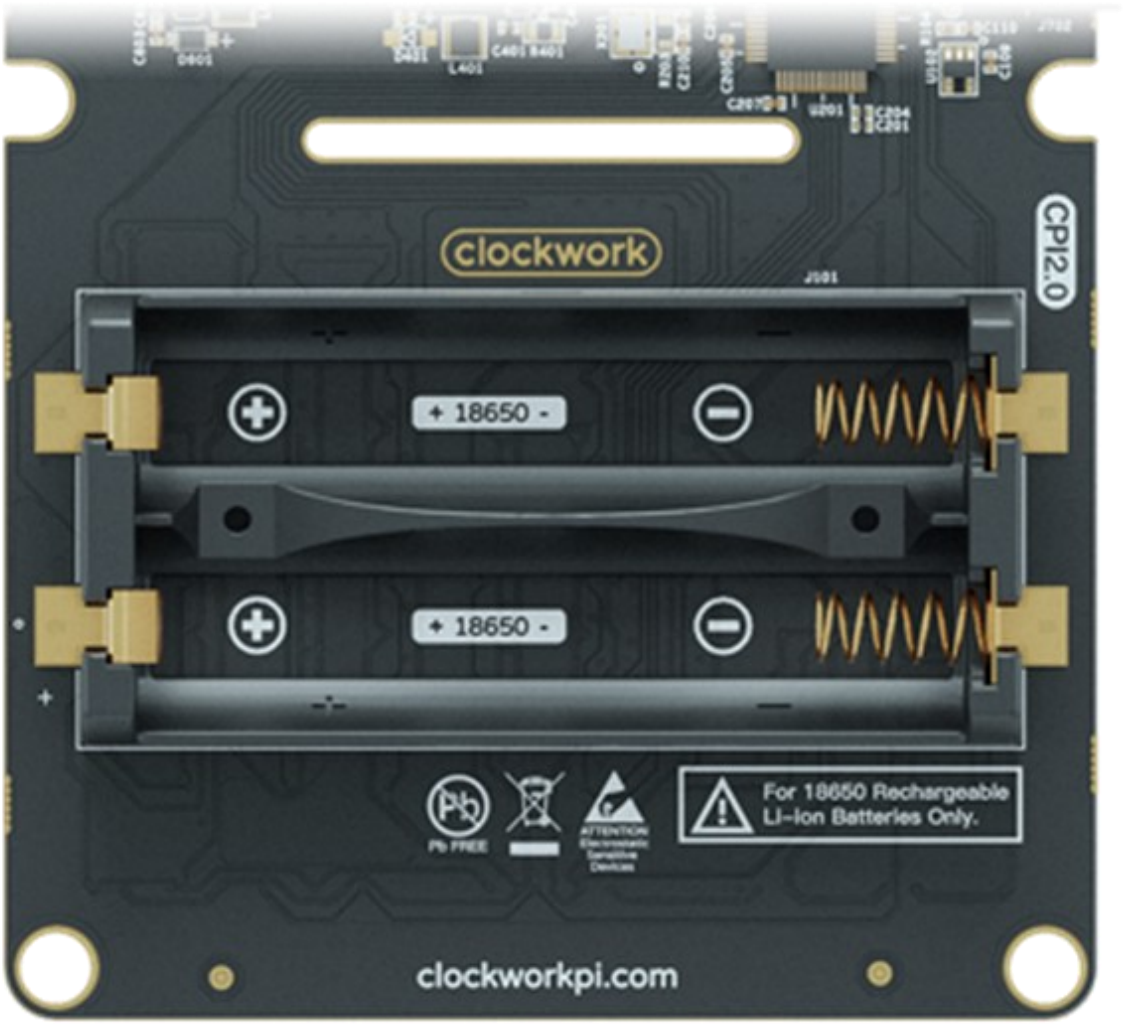




The firmware source code, schematic, and related design materials are released under the GPL v3 license, and you can find them on our GitHub.

Battery module

We carefully designed the battery holder to ensure compatibility with various 18650 battery sizes: Ø18 ± 0.5mm diameter, 65-69mm length. Please check the polarity when installing the 18650 battery. Incorrect installation may damage the device and the battery, or even cause an accident. We also recommend purchasing* the battery from a reliable distributor.



You can use a single 18650 battery to power the system or replace the batteries one by one, even while the system is in use. However, these actions may cause system instability, so we strongly recommend replacing the batteries only after powering down the system. If you do not plan to use the device for an extended period, please remove the battery.
For detailed 18650 battery safety guidelines, consult your battery supplier and strictly follow their instructions.
* Please note: the 18650 batteries are NOT included in the package due to global logistics complexity for lithium batteries.

About DIY*

We chose a team of Mold experts with over 30 years of experience to ensure high-quality results. As with previous ClockworkPi products, we focus on providing an enjoyable DIY experience. No professional tools, skills, or background are required, you only need 2.5mm hex keys** to assemble all the modules and complete the installation in just a few minutes.

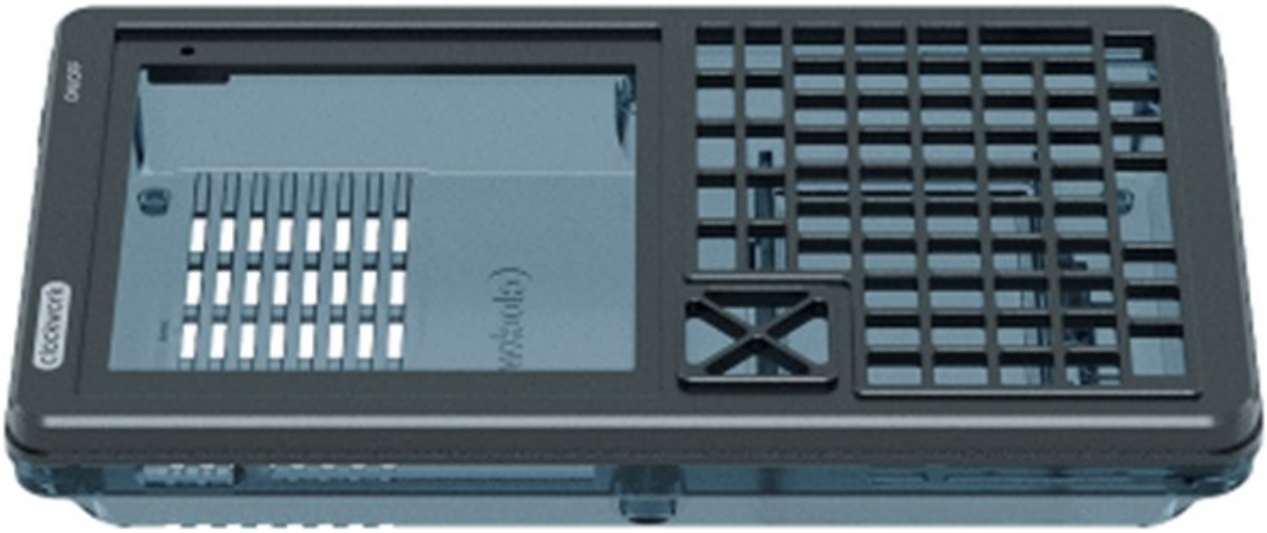
We follow the concept of "Open-Design" and support "Personal Industrial Manufacturing", the 3D files of all structures and related materials are released under the GPL v3 license. You can find them on our online shop and GitHub to customize your own parts through CNC or 3D printing technology.

* Safety reminders:

- The DIY process is ONLY ALLOWED FOR USERS AGED 12 OR ABOVE.

During the DIY process, please READ the instruction carefully.

** The 2.5mm Hex key is included in the package.




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


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



E.T. FONEHUME

HI DUSTY
IT WAS FUNNY YOU REMEMBERING THE POST OFFICE BEING ON STRIKE IN FEBRUARY 1971 AND ALL THE PROBLEMS IT CAUSED WITH DECIMALISATION.

THIS HAD ME THINKING AND WONDERING IF ANYONE (APART FROM US TWO) CAN STILL 'ADD-UP' IN POUNDS, SHILLINGS AND PENCE ANYMORE! I REMEMBER THAT THERE WERE JUST A FEW ADDING MACHINES IN THE U.K. THAT COULD COPE.

(I REMEMBER THEY HAS 2 EXTRA BUTTONS FOR 10 AND 11 PENCE! THEY WERE LIKE 'HENS TEETH' AND ***VERY*** *EXPENSIVE! THERE WAS ONLY ONE IN THE WHOLE GOVERNMENT BUILDING WHERE I WORKED! ED.)*

AS PER USUAL I WONDERED IF IT WAS POSSIBLE TO HAVE A BASIC ONE-LINER JUST TO SEE IF ANYONE ELSE WOULD BE ABLE TO COPE WITH PRE-DECIMAL MONEY!




```
0 CLS:RANDOM:P$="###":INPUT"NO";D:FORX=1TOD:P(X)=RND(9):S(X)=RND
(19):C(X)=RND(11):PRINTUSINGP$;P(X);S(X);C(X):C1=C1+C(X):S1=S1+S
(X):P1=P1+P(X):NEXT:CI=INT(C1/12):TC=C1-(CI*12):S1=S1+CI:SI=INT(
S1/20):TS=S1-(SI*20):INPUTZ$:PRINTP1+SI;"/";TS;"/";TC
```

IT'S QUITE SIMPLE TO RUN. JUST ANSWER HOW MANY AMOUNTS YOU WISH TO ADD (N.B. MAX OF 10 PLEASE, OTHERWISE THE PROGRAM WILL CRASH! - NO ROOM TO 'DIM' ANY VARIABLES!!).

WHEN YOU'VE SCRIBBLED YOUR ANSWER ON A SCRAP BIT OF PAPER, JUST PRESS <ENTER> AT THE QUESTION MARK AND THE ANSWER WILL APPEAR. NO CHEATING THOUGH!

REMEMBER, IT WAS 12 PENNIES TO A SHILLING AND TWENTY SHILLINGS TO A POUND!

SORRY IF THIS ALL SEEM LIKE 'GOBBLE-DE-GOOP' TO NON U.K. READERS, BUT IT WAS EVEN MORE COMPLICATED ORIGINALLY. A PENNY COULD BE MADE UP FROM EITHER FOUR FARTHINGS OR TWO HALF PENNIES AND THERE WAS ALSO A GUINEA WHICH WAS TWENTY ONE SHILLINGS. I WONT MENTION FLORINS, CROWNS AND HALF CROWNS!

```
NO? 5
  5 12 11
  7 12  8
  4 17  6
  2  6  6
  9 11  9
?
 30 / 1 / 4
READY
>_
```





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

Breaking the 48KB Barrier: Hardware Managed Implicit Paging on the TRS-80

The Z80 processor is a masterpiece of 8-bit engineering, but it is famously haunted by a hard ceiling: the 16-bit address bus. For the TRS-80 Model I, this limitation is even tighter, as the memory is shared with ROM, Keyboard, Video, etc.

The original TRS-80 has used disk paging to load code overlays (e.g. TRS-DOS SYS files), while this works, it does have a performance impact. Bank-switching only came latter to the TRS-80 when larger DRAM's became available, however adoption was limited to a few cases. E.g. a RAM disk, printer buffer, multi-tasking.

The Problem

Traditional bank-switching schemes usually divide RAM into large block sizes, (i.e. 32Kb was typical). While simple this made applications harder to build. The entire application had to fit into the low 32Kb (<12Kb accounting for DOS overhead), so data could be accessed banked in upper 32Kb memory address range.

Another problem was utilizing RAM effectively when extending the capability of the system. e.g. writing a custom device driver, would typically require the use of high-memory (top of ram) which would not only take away from application RAM but would make bank switching of the top 32Kb of RAM difficult.

The solution presented tries to make it easier to design applications (and system extensions) that can use make use of banked RAM.

The Solution

I've been working on a new approach using the **RP2350** (Pi Pico 2 microcontroller) to provide 128Kb of banked RAM. It doesn't just do bank switching, but it manages the process automatically, and transparently to the programmer.

The solution uses the RP2350 as a bridge between the Z80 and 128KB of banked RAM. Instead of requiring the Z80 program to manage the banks, the microcontroller monitors the CPU's intent and performs **Implicit Paging**, for the Z80 program.

Very little knowledge (almost none) of the underlying mechanisms is needed by the application or programmer. While a bank register is exposed, it is not typically used. In some respect the programmer sees a flat memory model for application code.

Defining the Memory Space

To understand the solution, we need to look at how we've used memory. First off, I have chosen the mostly unused $3000–$37FF memory area as the location for this solution. It's mainly used by FDC and Printer hardware (still fully supported), and by optional extension ROM's (not supported).

The solution provides **128KB** of extended RAM, this is divided into **256 distinct "Banks,"** each exactly **512 bytes** in size. While small bank sizes may seem counterproductive, the intent is not to store large data sets, but instead use the space for functional code program storage.

This 128KB is accessed through a carefully orchestrated memory map:

- **Trampoline ($3000–$33FF):** This is the control plane. It contains 256 "slots" (4 bytes each), every slot acts as a portal to a specific 512-byte memory bank. Page access is via this Trampoline
- **Window ($3400–$35FF):** This is the active "view" of banked RAM. No matter which of the 256 pages you are currently using, it is mapped to this 512 byte address range.
- **General ($3600–$37DF):** This area is a fixed non paged region of general-purpose RAM. It is the perfect place for sharing data between different pages.
- **Excluded ($37E0-$37EF):** These 16 bytes are ignored by this solution, to provide full compatibility with FDC and Printer hardware.
- **Control ($37F0-$37FF):** Allows manual control of the current page, and some ability to control paging mechanism, **normally** not used.

Implicit Paging

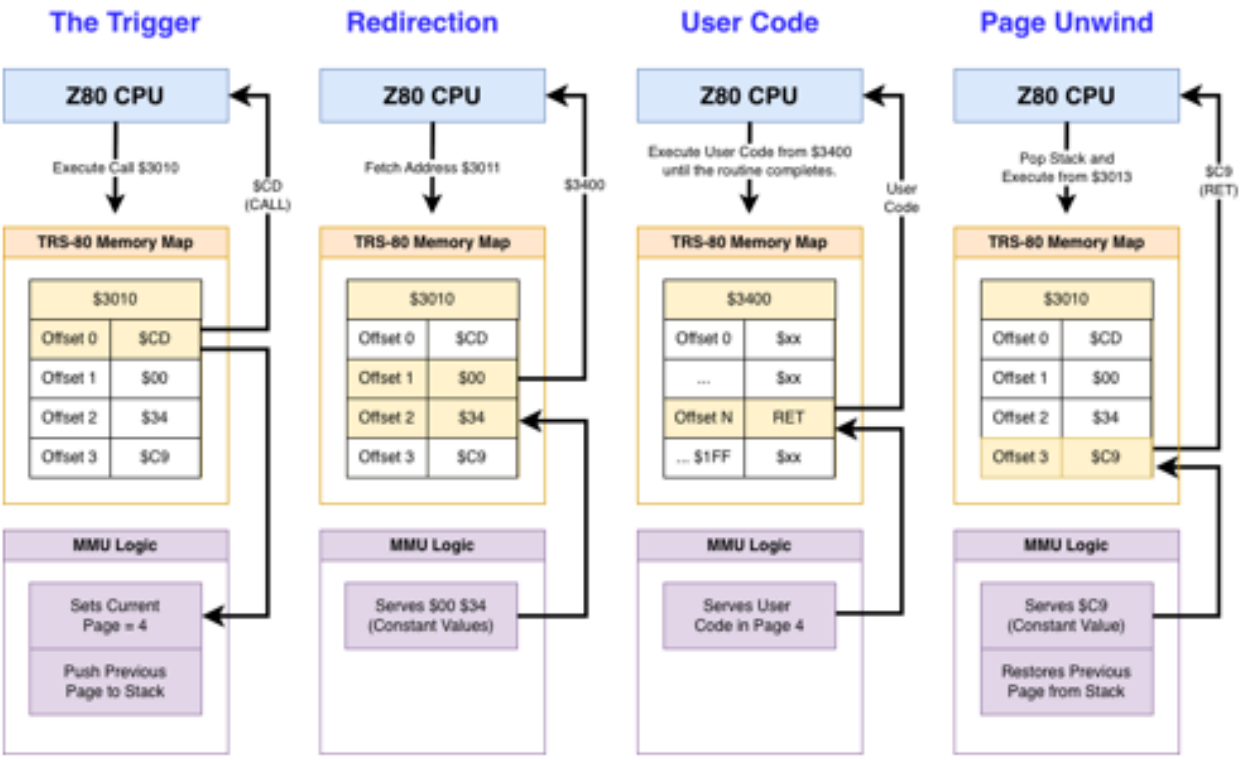
The RP2350 manages bank switching by monitoring the Z80 address bus. When the Z80 reads a slot in the **Trampoline ($3000-$33FF)**, the RP2350 doesn't just treat it as a memory read, it uses its access to manage page state.

Normally each 4-byte slot contains the exact same bytes ($CD, $00, $44, $C9). This translates to 2 instructions (CALL $4400; RET). The process is:

1. **Trigger:** Z80 code calls a **Trampoline** address (e.g. $3010), the RP2350 identifies the page number based on the memory address. (e.g. Page 4)
2. **Page Push:** The RP2350, pushes the *current* page number onto an internal stack, and maps Page 4 into the **Window ($3400)**.
3. **Redirection:** The RP2350 responds to the Z80 with the *CALL $3400* instruction, and execution resumes at $3400
4. **User Code Execution:** The Z80 executes code normally within the 512-byte page bank. Upon execution of a RET statement, the CPU will return control to the 4th byte of the Trampoline slot (e.g. $3013).
5. **Page Unwind:** This forth byte, contains a RET statement. The RP2350 pops the previous page number from its internal stack and restores that Page back to the **Window ($3400)**.

The result is **Transparent Re-entrancy**. The Z80 returns control to its original caller, and the memory environment is perfectly restored without the programmer ever writing a single bank-switching command. It is the CALL and RET statement executed in the Trampoline area (at specific addresses) that perform this function.

This summarises the principles described above.



Zero-Config Development

One of the most powerful features of this architecture is that it requires **no custom assemblers or specialized loaders**. By making the hardware sensitive to **WRITE** operations in the Trampoline area, you essentially turn the standard assembly ORG (Origin) and DB (Define) directive's into configuration commands.

How the Hardware "Learns"

When a standard TRS-80 loader processes your program, it reads data from storage device and performs a series of memory writes. The RP2350 monitors writes to the **Trampoline ($3000–$33FF)** to facilitate loading.

When the loader writes to Offset 0 of a Trampoline slot (e.g. $3010), the RP2350 does 2 things specifically:

- **Page Switching**: The page identified by the address (e.g. Page 4) is immediately switched into the **Window ($3400-$3600)**.
- **Opcode Detection:** If the byte written is $CD (CALL), the hardware enables the Page Stack for that slot as discussed above. If the byte is a $C3 (JP), it marks it as one way and disables the Page Stack push for that slot

Because the page switched when the Trampoline header was written, all subsequent bytes written to the Window $3400, are written to the correct (e.g. Page 4) bank

This effectively turns a 1970s loader into a sophisticated Memory Manager without changing a single line of the loader's code.

The Programmer's Perspective:

From a developer's standpoint, the complexity is entirely encapsulated within a few simple macros. This allows you to write "Page-Agnostic" code that can be loaded into any bank without modification. The following is a simple code example:

```
; MACROS

pageCall macro label, pagenum
       assert pagenum>=0 && pagenum<$100  ; Parameter validation
       ORG    $3000 + (pagenum * 4)       ; Trampoline address
label  EQU    $              ; Define a program label for the address
       DEFB   $CD            ; Write CALL opcode to Offset $0
       ORG    $3400          ; All code sits in the 512-byte window
endm

pageEnd macro
       assert $ <= $3600
endm

; CODE STARTS HERE

pageCall routine1, 4         ; Define "routine1" in Virtual Page 4
       ld     a,$5A          ; Do some Work
       ret
pageEnd                      ; End of Paged Routine

pageCall routine2, 8         ; Define "routine2" in Virtual Page 8
       ld     a,$A5          ; Do some Work
       call   routine1       ; Call routine in Page 4
       ret
pageEnd                      ; End of Paged Routines

       ORG    $5200          ; main Program Load/entry point
mainRoutine:
       call   routine1       ; Call routines in different pages
       call   routine2       ; without concern for page mechanisms.
       jp     $402D          ; exit

; CODE ENDS HERE

END    mainRoutine
```

The code just defines routines that are assigned to paged RAM, but doesn't have to deal with any of the underlying mechanisms. Code can be written, while observing some simple constraints. i.e. A single page entry point, and no shared page data.

The assembler just creates binary code blocks, and the loader thinks it is loading into flat memory. Behind the scenes, the microcontroller uses those exact bytes to allow you to "fold" your code into 128Kb of banked RAM

Consequences

One challenge of making memory "intelligent" is that it breaks some fundamental assumptions, i.e. reading memory shouldn't have side-effects. i.e. If a debugger were to dump's the Trampoline area, it will trigger unintended page flips, with likely unexpected outcomes.

To solve this, I've implemented a Command Port ($37F8). This allows a developer to "freeze" the smart logic, ensuring that the system remains stable

Another slight downside is additional cycles for the double CALL/RET, plus a slightly larger stack, but considering the alternatives this should not really be a concern.

Conclusion

While initially the page size of 512 bytes seemed initially small, it should now make more sense. Given each page only has a single "automatically managed" entry point, each page should be "enough" to provide a reasonably sized discrete function.

To avoid wastage (remembering 128Kb total space) parameters can be passed in and related functions grouped into a single page, it's all about how an application is partitioned. For comparison most TRS-DOS's partitioned Overlays into 1.25Kb

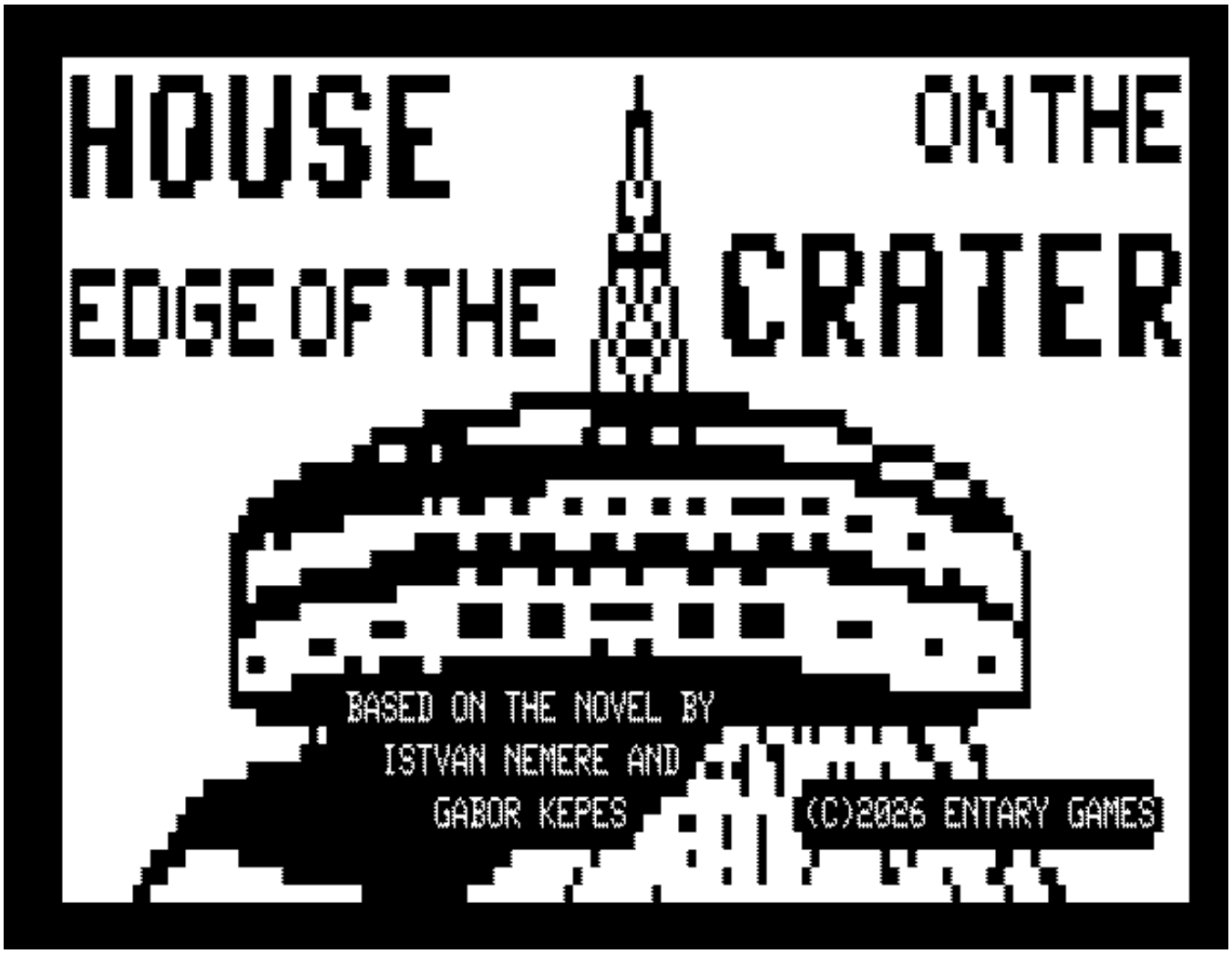
So what is this useful for:

- An application could be loaded into and execute primarily from banked memory, leaving main memory (48Kb) available for pure data storage.
- A driver (e.g. keyboard) or system extension (e.g. device driver) could be loaded (resident), without the use of HI Memory (below $FFFF)

This approach brings a level of sophistication to 8-bit programming usually reserved for more advanced MMUs, all while maintaining the charm and simplicity of the Z80 programming model.

Breaking the 48KB Barrier © 2026 by Mark Pruden is licensed under CC BY-NC-SA 4.0

To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-sa/4.0/



Nickmann Studios has developed a new game for the TRS-80 - "House on the Edge of the Crater"

The authorities of the age of normality have exiled the most creative people to the moon: artists, scientists, dangerous eccentrics, and keepers of great secrets.
You are Yordi, the reporter.
Conduct interviews with the residents, explore the house on the edge of the crater and bring back to earth the knowledge that could transform the cold world of the age of normality.
Based on the novel house on the edge of the crater by István Nemere and Gábor Képes.
Requirements for a successful mission: Conduct 9 interviews and obtain all 3 keys.

It is a 'name your price' download - You can find all the information and download links at https://nickmann-studio.itch.io/crater

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

The amazing

"FreHD"

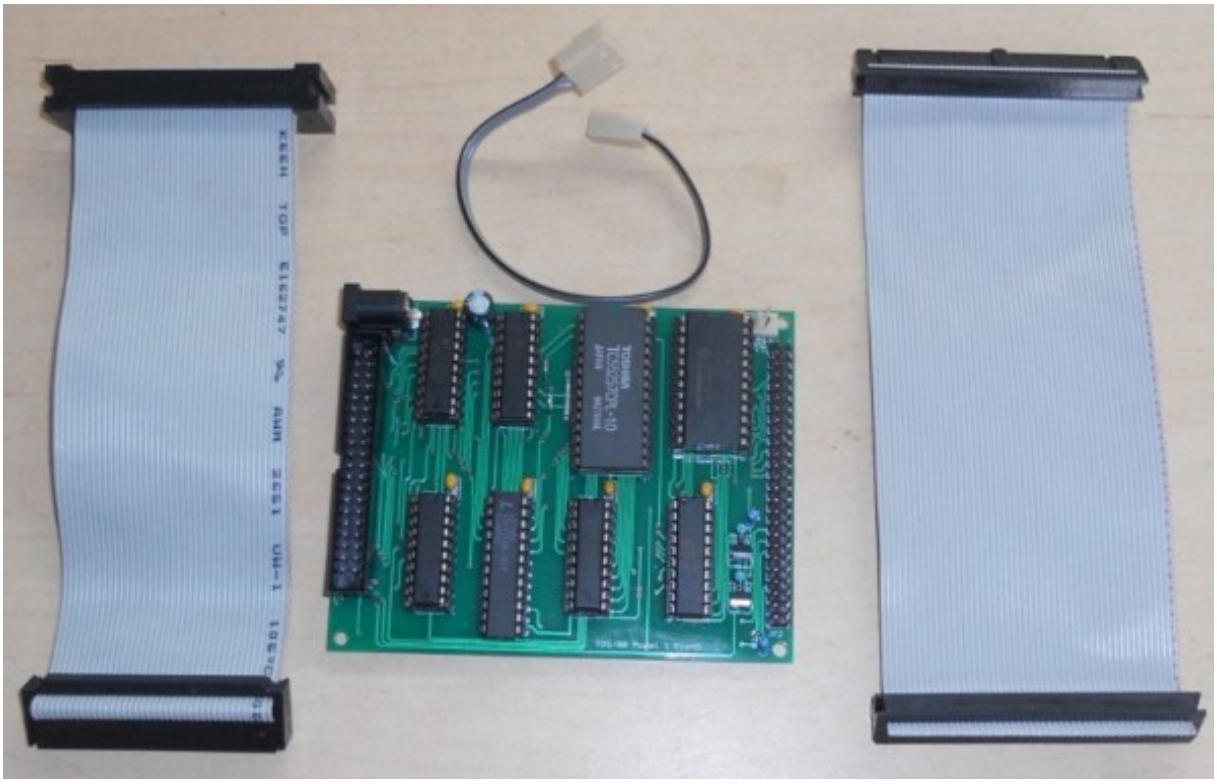


-Emulates a TRS-80 hard drive, but faster than any hard drive!
-Works with your favourite DOS (LS-DOS, LDOS, CP/M, Newdos/80 2.5)
-Uses SD card for storage medium
-Bonus free Real Time Clock function!
-Designed in Belgium and proudly built and shipped from Australia
-Kit form or fully assembled

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”


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

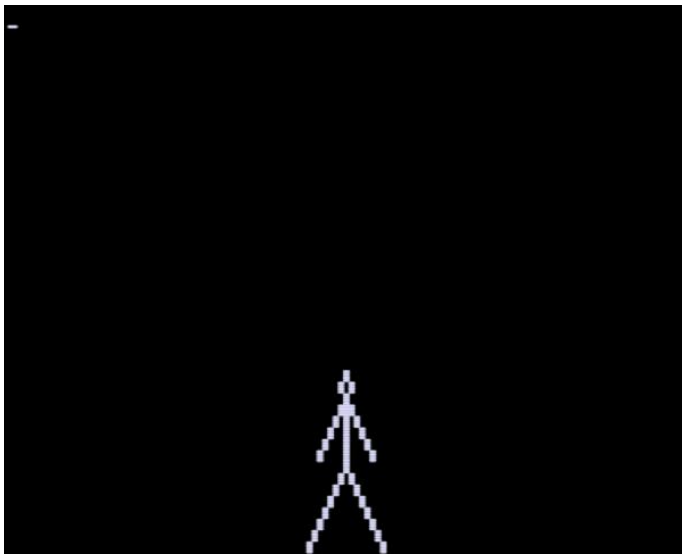
Part 6 - DRAWing Conclusions

Vince Otten

How many of us have faced this real-life challenge?



For those of you struggling with manual artwork, might I suggest using a "Computer-aided Drawing"? Here's a sample of what you can achieve with your TRS-80 Home Computer.



In the "BASIC Programming Language" book we've been using as an outline, Dr. David Lien teaches Level I users how to draw and animate this little guy. Here we'll discuss ROM calls to SET or RESET graphics points, or use the POINT routine to discover if a point is SET, analogous to the LEVEL II BASIC statements. Then we'll make Stick Man glide across the screen, too. This is what some sample calls would look like in our assembly-language version:

```
;
; "SET( 10, 20)"
;
            LD      B,10            ; X CO-ORDINATE
            LD      C,20            ; Y CO-ORDINATE
            LD      A,80H           ; "SET"
            CALL GRFXSB             ; "GRAPHICS SUB"
;
; "RESET( 11, 21)"
;
            LD      B,11            ; X CO-ORDINATE
            LD      C,21            ; Y CO-ORDINATE
            LD      A,01H           ; "RESET"
            CALL GRFXSB             ; "GRAPHICS SUB"
```

and here's "GRFXSB", the subroutine we use to call the ROM SET/RESET/POINT routine.

```
;
;    GRFXSB -- "GRAPHICS SUBROUTINE"
;                SET, RESET, OR POINT
; ARGS:
;        A       "SET"           80H
;                "RESET"         01H
;                "POINT"         00H
;        B       X CO-ORD        (0..127)
;        C       Y CO-ORD        (0..47)
;    RETURNS (IF CALLED WITH "POINT" FUNCTION):
;        WRA1: 0                 0000H RESET
;                -1              FFFFH SET
;        NTF:    2 (INTEGER)
;
GRFXSB      E Q U   $
            PUSH AF                 ; "MODE" ARGUMENT
            LD      A,B
            PUSH AF                 ; "X CO-ORD" ARGUMENT
            LD      A,C             ; "Y CO-ORD" ARGUMENT
            JP      0150H           ; ROM GRAPHICS SUB
```

So we can jump right in? Not so fast. There's a challenge with the common SET/RESET/POINT ROM routine at 0150H. It uses a couple of "RST 08H" instructions to "parse" the BASIC statement. RST 08H forces a jump to address 08H in ROM, which then quickly jumps out to the vector table at 4000H that we've discussed earlier in this series. What's going on in BASIC is that when the interpreter encountered a SET (or RESET or POINT) statement, it arrives at the ROM routine at 013AH. This is where we find the code to convert SET/RESET/POINT's X and Y arguments into a Screen address and a byte code to combine with whatever graphics may already be stored there. Using SET as our example, the X and Y values can be simple "literals", like

```
SET( 3, 17)
```

or quite complex "expressions", like

```
SET( A+ SQRT( B), INT( AR( N + 2, X * 12) / VAL( T$)))
```

Microsoft provided a method for evaluating these expressions using RST vectors. The graphics code uses RST 08H to scan the BASIC source code for the comma following the X co-ordinate, such as "3" in our example of "SET(3, 17)". The trick that Gates and Allen incorporated in their original Altair BASIC code was to *follow* the RST 08H instruction with the "token" to search for— in this case, a comma. Here's what that looks like in the ROM (with comments based on Ira Goldklang's trs-80.com disassembly notes):

```
0145    PUSH    AF          ; Save X co-ordinate on stack
0146    RST     08H         ; Call "Compare Symbol" routine with next character
                            ;    --in this case, a ","
0147    DEFB    02CH        ; ","
0148    CALL    02B1CH      ; Evaluate expression at the location in BASIC
                            ; program pointed to by HL
```

The problem with the RST 08Hs embedded in the ROM code we want to use is two-fold:

1. It sends us into the interpreter's "Mirkwood". The TRS-80 is expecting some BASIC program code, but we don't have any for it. Can't we just install a RET instruction at the 4000H vector that RST 08H jumps to?

2. The problem with that idea is that the return address stored on the stack is 1 less than it should be. In the example above, a RET would bring us back to address 0147H, which is where the 02CH representation of a comma is stored; the 02CH will then be interpreted as an INC L instruction, probably not what we want. We need the Program Counter to be restored to address 0148H instead.

Here's what we need to do then, before we begin using SET, RESET, or POINT via the GRFXSB code above:

1. Provide a short subroutine that adjusts the return address at the "top of stack" by 1;
2. Save the original vector at 4000H; and
3. Install a Jump to our new subroutine at 4000H.

This also means that we need to restore the vector at 4000H to its original instructions after we're done using graphics. (Side note: some programs like Editor Assembler Plus "borrow" the vector at 4000H like we're doing here, and carefully restore it when done; others, like University of Oregon's UO -LISP, "hijack" it and fail to put everything back, causing the next invocation of BASIC to crash!) Here's the three subroutines:

```
;
;    ADJRET -- INCREMENTS A RETURN ADDRESS
;          ON THE STACK BY 1
;          (TO SKIP RST 08H CODE IN ROM)
;
ADJRET EQU       $
          LD       HL,0000          ; *HL= RET ADDR TO
          ADD      HL,SP            ;      BE ADJUSTED
          LD       E,(HL)
          INC      HL
          LD       D,(HL)           ; DE= RET ADDR
          INC      DE               ; DE= RET ADDR++
          LD       (HL),D
          DEC HL
          LD       (HL),E           ; RET ADDR INCREMENTED
          RET

;
;    SVSYNX -- CODE TO SAVE RST 08H VECTOR
;
SVSYND DEFW 0000                   ; SAVE RST 08 / 4000h
          DEFB 0                   ; VECTOR HERE

SVSYNX EQU $
          LD       HL,(SVSYND)      ; 1ST CHECK IF WE'VE
          LD       A,H              ; ALREADY SAVED
          OR       L                ; SOMETHING HERE
          RET      NZ               ; IF (FIRST TIME):
          LD       HL,4000H              ; COPY VECTOR
          LD       DE,SVSYND             ; TO OUR
          LD       BC,3                  ; SAVE LOCATION
          LDIR
          LD       A,0C3H                ; STORE A "JP"
          LD       (4000H),A
```

```
        LD      HL,ADJRET           ; "ADJRET" IN
        LD      (4001H),HL          ; 4000H..4002H
        RET                     ; :ENDIF
;
;   RESYNX -- RESTORE RST 08 / 4000H VECTOR
;
RESYNX EQU $
        LD      HL,(SVSYND)     ; CHECK IF
        LD      A,H             ;    "SYVSND" IS
        OR L                    ; EMPTY
        RET     Z               ; IF (NOT EMPTY):
        LD      HL,SVSYND           ; COPY SAVED VECTOR
        LD      DE,4000H            ; TO ORIGINAL
        LD      BC,3                ; 4000H VECTOR LOCATION
        XOR     A                   ; NOW
        LD      (SVSYND),A          ; ZERO OUT
        LD      (SVSYND-i-1),A      ; "SVSYND"
        LD      (SVSYND-i-2),A
        RET                     ; :ENDIF
```

"SVSYND" is where we temporarily store the vector we found residing at 4000H..4002H. Notice that in the SVSYNX ("Save Syntax") code we *only* save that vector's 3 bytes *if* we haven't already done so. This prevents SVSYNX from accidentally overwriting a previously saved vector—say, if we'd already called SVSYNX without using RESYNX to restore the saved vector back to 4000H. It also makes a call to SVSYNX bail out a little more quickly, although I doubt that my neophyte Z80 assembly-language programming skills are producing the fastest possible code. Also, RESYNX ("Restore Syntax") will only update 4000H..4002H if SVSYNX *was* previously used, and will then zero out SVSYND so that SVSYNX will actually do its job if called upon again.

It's time to test this out. Let's implement the Level **I BASIC** code that Dr. Lien provided to animate the stick man—that is, to slide him from the center of the screen to the right, and then loop back around to the left edge and continue. Here's the BASIC code:

```
BREAK AT 105
>LIST
 10  CLS
 20  X= 64
 30  FOR K= 0 TO 7
 40  SET( X+K, 40+K)
 45  RESET( X+K-1, 40+K)
 50  SET( X-K, 40+K)
 55  RESET( X-K-1, 40+K)
 60  NEXT K
 70  FOR K= 0 TO 5
 80  SET( X+K, 34+K)
 85  RESET( X+K-1, 34+K)
 90  SET( X, 34+K)
```

```
 85  RESET( X+K-1, 34+K)
 90  SET( X, 34+K)
 95  RESET( X-1, 34+K)
 100  SET( X-K, 34+K)
 105  RESET( X-K-1,34+K)
 110  NEXT K
 120  SET( X, 32)
 125  RESET( X-1, 32)
 130  SET( X+1,33)
 135  RESET( X, 33)
 140  SET( X-1,33)
 145  RESET( X-2, 33)
 150  X= X + 1
 160  GOTO 30
 999  GOTO 999
>_
```

The BASIC program first draws the stick man's legs, RESETing the pixel to the left of each element. Then it does the same for his torso and arms, and finally for his head.
He definitely doesn't walk. Instead, he slowly does that dance move where you sidle across like a sidewinder rattlesnake. In Level I BASIC, it takes about 58 seconds for him to totally disappear from the right side of the screen, immediately beginning to re-appear on the left, point by point. (Disk/ Level II BASIC takes a similar amount of time, but it stops as soon as the first X co-ordinate exceeds 127 with an "ILLEGAL FUNCTION CALL" error. It runs perhaps a little faster if we add line 5: DEFINT A-Z .)
Our assembly-language version takes approximately 3.5 seconds to do the same thing. Interestingly, Stick Man does not appear on the left side of the screen right away as he did in Level I BASIC: it takes another 3.5 seconds for our byte-sized version of the X variable to cycle from 128 to 255 before he shows up again. Nevertheless, our A/L graphics seem to run about 16x faster than Level I BASIC. (If you scan the code you'll see that I included four intermediate temporarily variables to avoid having to constantly recalculate "X + **K**", "**X** – **K**", "**K** + 40", and "K + 34", so that speeds the assembly-language version up as well.) Here's the code:

```
;
;    *** MAIN() ***
;
 MAIN EQU        $
        CALL     INIT

        CALL     SVSYNX     ; "USE GRAPHICS"

        CALL     CLSCRN     ; CLEAR SCREEN
```

Before we actually start implementing the BASIC code, we have to set up our ROM routines with CALL INIT, and our graphics routines with CALL SVSYNX. "CALL INIT" is covered in the TRS8bit June 2025 "Power of the ROM" article. A note for those who have been following along and using the Math calls I've detailed in previous TRS8bit "Power of the ROM" articles: you'll still need to CALL I00CRM to set up the BASIC vectors at 4000H..4012H, but you should do so *outside or before* any CALLs to SVSYNX, since I00CRM adjusts the vector at 4000H..4002H. (Since my implementation of Stick Man just uses integer/byte arithmetic, I don't CALL I00CRM at all.)
"CLSCRN" looks like this:

```
CLSCRN EQU      $
        CALL    01C9H          ; "CLS"
CURSOF EQU      $
        LD      HL,CLSCOF      ; "CURSOR OFF" CODE
        CALL    PRINT
        RET
```

```
CLSCOF  DEFB    0FH
        DEFB    00
```

Here's the code emulating the actual BASIC program:

```
;
; "10 CLS"
;
LN0010  EQU     $
        CALL    CLSCRN
;
; "20 X= 64"
;
LN0020  EQU     $
        LD      A,64
        LD      (XVAR),A
;
; "30 FOR K= 0 TO 7"
;
LN0030  EQU     $
        XOR     A
        LD      (KVAR),A        ; "FOR K= 0"
LN030A  EQU     $               ; "TO" (7, SEE BELOW)



; "40 SET(X+K, 40-K)":

LN0040  EQU     $
        LD      A,(XVAR)        ; CALCULATE
        LD      C,A             ; "X +"
        LD      A,(KVAR)        ;     "K"
        ADD     A,C             ; AND STORE
        LD      (XPLK),A        ; "X-PLUS-K"
LN040A  EQU     $
        LD      A,(KVAR)        ; CALCULATE
        ADD     A,40            ; "K + 40"
        LD      (KPL40),A       ; & STORE IN "K-PLUS-
LN040B  EQU     $               ; NOW GRAPH



        LD      C,A             ; Y CO-ORD
        LD      A,(XPLK)
        LD      B,A             ; X CO-ORD
        LD      A,80H           ; "SET"
        CALL    GRFXSB
```

```
;
; "45 RESET( X+K-1, 40+K)"
;
LN0045  EQU     $
        LD      A,(KPL40)       ; "K + 40"
        LD      C,A             ; Y CO-ORD
        LD      A,(XPLK)        ; "X + K
        DEC     A               ;      - 1"
        LD      B,A             ; X CO-ORD
        LD      A,01H           ; "RESET"
        CALL    GRFXSB
;
; "50 SET( X-K, 40+K)"
;
LN0050  EQU     $
        LD      A,(KVAR)        ; CALCULATE
        LD      C,A             ;    "X -"
        LD      A,(XVAR)        ;         "K"
        SUB     C               ; & STORE IN
        LD      (XMIK),A        ;    "X-MINUS-K"
LN050A  EQU     $
        LD      A,(KVAR)        ; CALCULATE "40+K"
        ADD     A,40            ; & STORE IN
        LD      (KPL40),A       ;    "K-PLUS-40"
LN050B  EQU     $
        LD      C,A             ; Y CO-ORD
        LD      A,(XMIK)
        LD      B,A             ; X CO-ORD
        LD      A,80H           ; "SET"
        CALL    GRFXSB
;
; "55 RESET( X-K-1, 40+K)"
;
LN0055  EQU     $
        LD      A,(XMIK)        ; "X - K
        DEC     A               ;      - 1"
        LD      B,A             ; X CO-ORD
        LD      A,(KPL40)       ;    "K + 40"
        LD      C,A             ; Y CO-ORD
        LD      A,01H           ; "RESET"
        CALL    GRFXSB
```

```
;
; "60 NEXT K"
;
LN0060  EQU     $
        LD      A,(KVAR)        ; IF
        LD      C,A             ;    K
        LD      A,7             ;     <=
        CP      C               ;        7
        JP      C,LN060A        ;    CONTINUE TO...
        JP      Z,LN060A
        LD      A,C
        INC     A               ;         ( K++)
        LD      (KVAR),A
        JP      LN030A          ;    ...LOOP
;
;
; "70 FOR K= 0 TO 5"
;
LN0070  EQU     $
        XOR     A
        LD      (KVAR),A        ; "FOR K= 0"
LN070A  EQU     $               ; "TO" (5, SEE BELOW)
;
; "80 SET( X+K, 34+K)"
;
LN0080  EQU     $
        LD      A,(XVAR)        ; CALCULATE
        LD      C,A             ;   "X +"
        LD      A,(KVAR)        ;        "K"
        ADD     A,C             ; AND STORE IN
        LD      (XPLK),A        ;   "X-PLUS-K"
LN080A  EQU     $
        LD      A,(KVAR)        ; CALCULATE
        ADD     A,34            ;   "K+34"
        LD      (KPL34),A       ; & STORE IN "K-PLUS-34"
LN0808  EQU     $               ; NOW GRAPH THOSE
        LD      C,A             ; Y CO-ORD
        LD      A,(XPLK)
        LD      B,A             ; X CO-ORD
        LD      A,80H           ; "SET"
        CALL    GRFX5B
;
; "85 RESET( X+K-1, 34+K)"
;
LN0085  EQU     $
        LD      A,(KPL34)       ; "34 + K"
```

```
        LD      C,A             ; Y CO-ORD
        LD      A,(XPLK)        ; "X + K
        DEC     A               ;      - 1"
        LD      B,A             ; X CO-ORD
        LD      A,01H           ; "RESET"
        CALL    GRFXSB
;
; "90 SET( X, 34+K)"
;
LN090A  EQU     $
        LD      A,(KPL34)       ; "34 + K" IS
        LD      C,A             ; Y CO-ORD
        LD      A,(XVAR)        ; "X" IS
        LD      B,A             ; X CO-ORD
        LD      A,80H           ; "SET"
        CALL    GRFXSB
;
; "95 RESET( X-1, 34+K)"
;
LN0095  EQU     $
        LD      A,(KPL34)       ; "34 + K"
        LD      C,A             ; Y CO-ORD
        LD      A,(XVAR)        ; "X
        DEC     A               ;   - 1"
        LD      B,A             ; X CO-ORD
        LD      A,01H           ; "RESET"
        CALL    GRFXSB
;
; "100 SET( X-K, 34+K)"
;
LN0100  EQU     $
        LD      A,(KVAR)        ; CALCULATE
        LD      C,A             ;    "X -"
        LD      A,(XVAR)        ;         "K"
        SUB     C               ; & STORE IN
        LD      (XMIK),A        ;    "X-MINUS-K"
LN100A  EQU     $
        LD      A,(KPL34)       ; "34 + K" IS
        LD      C,A             ; Y CO-ORD
        LD      A,(XMIK)        ; "X - K" IS
        LD      B,A             ; X CO-ORD
        LD      A,80H           ; "SET"
        CALL    GRFXSB
```

```
;
; "105 RESET( X-K-1, 34+K)"
;
LN0105  EQU         $
        LD          A,(KPL34)           ; "K + 34"
        LD          C,A                 ; Y CO-ORD
        LD          A,(XMIK)            ; "X - K
        DEC         A                   ;      - 1"
        LD          B,A                 ; X CO-ORD
        LD          A,01H               ; "RESET"
        CALL        GRFXSB
;
; "110 NEXT K"
;
LN0110  EQU         $
        LD          A,(KVAR)            ; IF
        LD          C,A                 ;     K
        LD          A,5                 ;       <=
        CP          C                   ;         5
        JP          C,LN110A            ;     CONTINUE TO...
        JP          Z,LN110A
        LD          A,C
        INC         A                   ;           (K++)
        LD          (KVAR),A
        JP          LN070A              ;     ...LOOP
;
; "120 SET( X, 32)"
;
LN0120  EQU         $
        LD          C,32            ; Y CO-ORD
        LD          A,(XVAR)
        LD          B,A             ; X CO-ORD
        LD          A,80H           ; "SET"
        CALL        GRFXSB
;
; "125 RESET( X-1, 32)"
;
LN0125  EQU         $
        LD          C,32            ; Y CO-ORD
        LD          A,(XVAR)        ; "X
        DEC         A               ;   - 1"
        LD          B,A             ; X CO-ORD
        LD          A,01H           ; "RESET"
        CALL        GRFXSB
```

```
;
; "130 SET( X+1, 33)"
;
LN0130  EQU         $
        LD          C,33            ; Y CO-ORD
        LD          A,(XVAR)
        LD          B,A
        INC         B               ; X CO-ORD
        LD          A,80H           ; "SET"
        CALL        GRFX5B
;
; "135 RESET( X, 33)"
;
LN0135  EQU         $
        LD          C,33            ; Y CO-ORD
        LD          A,(XVAR)
        LD          B,A             ; X CO-ORD
        LD          A,01H           ; "RESET"
        CALL        GRFX5B
;
; "140 SET( X-1, 33)"
;
LN0140  EQU         $
        LD          C,33            ; Y CO-ORD
        LD          A,(XVAR)
        LD          B,A
        DEC         B               ; X CO-ORD
        LD          A,80H           ; "SET"
        CALL        GRFX5B
;
; "145 RESET( X-2, 33)"
;
        LD          C,33            ; Y CO-ORD
        LD          A,(XVAR)
        DEC         A
        DEC         A               ; "X - 2"
        LD          B,A             ; X CO-ORD
        LD          A,01H           ; "RESET"
        CALL        GRFX5B
;
; "150 X= X+1"
;
LN0150  EQU         $
        LD          HL,XVAR         ; "X=
        INC         (HL)            ;     X + 1"
```

```
;
; "160 GOTO 30"
;
LN0160  EQU         $
        JP          LN0030          ; "GOTO 30"
```

Of course, this is a continuous loop, which is how the original BASIC code was written. I actually altered LN0160 a bit:

```
LN0160      EQU     $
            CALL    002BH           ; SCAN KEYBOARD
            OR      A               ; A == 0 IF NO KEY PRESSED
            JP      Z,LN0030        ; IF NONE, "GOTO 30"

            CALL    RESYNX          ; RESTORE 4000H VECTOR

            CALL    CURSON          ; TURN CURSOR BACK ON

            JP      FINISH          ; EXIT
```

We defined "FINISH" in TRS8bit March 2025. Your program can exit to TRSDOS or (as is typical in my case) to Editor Assembler Plus's Z-BUG. Notice that I've included CALL CURSON, which simply restores the hidden screen cursor:

```
;
;       CURSON – TURN CURSOR ON
;           (EQUIVALENT TO "CURSOF" WHICH TURNS
;           CURSOR OFF, EMBEDDED IN "CLSCRN" CODE)
;
CURSON      EQU     $
            LD      HL,CURONC       ; "CURSOR ON" CODE
            CALL    PRINT
            RET
CURONC      DEFB    0EH
            DEFB    00
```

Perhaps this is a good time to list what routines and objects have been covered in which TRS8bit issues so far.

Name	***Description***	***TRS8Bit Issue***
	I/O Routines:	
CALL SCPRTB	"PRINT TAB()" emulation: TABs to position on Screen	V20 #1 Mar 2026
CALL PRINTB	"LPRINT TAB()" emulation: TABs to position on Line Printer	V20 #1 Mar 2026
CALL TAB	General-purpose subroutine supporting PRINT TAB() or LPRINT TAB() operations	V20 #1 Mar 2026
CALL PREPSC	Prepare TAB() argument for Screen before CALLing TAB; used by SCPRTB	V20 #1 Mar 2026
CALL PREPRT	Prepare TAB() argument for Line Printer before CALLing TAB; used by PRINTB	V20 #1 Mar 2026
CALL TABEXC	Subroutine to correct conditions when TAB() position exceeds the device's Width setting	V20 #1 Mar 2026
CALL SCOMMA	Emulate BASIC's "trailing comma" when PRINTing; see "COMMA"	V20 #1 Mar 2026
CALL PCOMMA	Emulate BASIC's "trailing comma" when LPRINTing; see "COMMA"	V20 #1 Mar 2026
CALL COMMA	Emulate BASIC's "trailing comma" in a PRINT or LPRINT statement, as in "PRINT X, Y, Z"	V20 #1 Mar 2026
CALL VPR1CH	Write 1 character to Screen, updating cursor position. Suitable for use with trailing comma and TAB(n).	V20 #1 Mar 2026
CALL PRINT	Print 0-terminated string to screen (version 1)	V19 #1 March 2025
CALL LPR1CH	Write 1 character to the Line Printer (final). Suitable for use with trailing comma and TAB(n). Checks head position against carriage width. Waits for Printer Ready. Uses CALL LPRINC.	V20 #1 Mar 2026
CALL LPRINC	Write 1 character to the Line Printer ("raw" version: does not check for Printer Ready; see CALL LPR1CH, above). Suitable for use with trailing comma and TAB(n). Checks head position against carriage width.	V20 #1 Mar 2026
CALL LPRT1C	Write 1 character to the Line Printer (version 1)	V20 #1 Mar 2026
CALL LPRWTR	Wait for Line Printer to be Ready	V20 #1 Mar 2026
CALL PRINSP	Print Single-Precision variable pointed to by HL to the Screen	V20 #1 Mar 2026
CALL CLSCRN	"CLS" Clear Screen, turn off cursor	This Issue
CALL CURSOF	Turn off cursor	This Issue
CALL CURSON	Turn cursor on	This Issue
CALL LINPUT	Line Input String	V19 #3 Sept 2025
40A6H	BASIC: stores current column # of Screen cursor. Used by CALL VPR1CH.	V20 #1 Mar 2026
4098H	BASIC: stores current head position for Line Printer. Used by CALL LPRINC.	V20 #1 Mar 2026
	Math:	
	Single-Precision Add	V19 #3 Sept 2025
	Single-Precision Multiply	V19 #2 June 2025
	Single-Precision Divide	V19 #3 Sept 2025
CALL SPCMP	Single-Precision Compare Operation	V19 #3 Sept 2025
	Convert Single-Precision...	V19 #2 June 2025,
CALL SP2STR	...to String	V20 #1 Mar 2026
CALL STR2SP	Convert String to Single-Precision	V19 #3 Sept 2025
CALL CONVRT	Convert Signed Integer to Single-Precision	V19 #2 June 2025
CALL SUI2SP	Convert Unsigned 24-bit Integer to Single-Precision	V19 #4 Dec 2025
	Single-Precision Constants and Constant Expressions	V19 #3 Sept 2025

	Floating Point Representation of Integers	V19 #4 Dec 2025
	BASIC Program to display all 4 bytes in a Single-Precision value	V19 #3 Sept 2025
	Graphics:	
CALL GRFXSB	SET, RESET, or POINT x, y co-ordinates	This Issue
	Setup and Cleanup (Start and Finish) Routines:	
JP FINISH	Exit to TRSDOS, DEBUG, or Editor Assembler Plus' Z-BUG	V19 #1 March 2025
CALL INIT	Clear "BERROR" and turn off Tracing ("TROFF") Also clear Video cursor and Line Printer head position to 0	V19 #2 June 2025
CALL I00CRM	"USES MATH": Initialize Math Support routines, vectors, error-trapping	V19 #2 June 2025
CALL V4000R	"FORGET MATH": restores vectors at 4000JH..4012H to original state	V19 #2 June 2025
CALL SVSYNX	Save RST 08H vector prior to using graphics. Installs ADJRET vector there in its place.	This Issue
CALL RESYNX	Restore RST 08H vector saved previously by SVSYNX	This Issue
	Error-trapping:	
CALL GETSPC	Place this instruction directly before calling a ROM routine that may result in errors (like "Divide by Zero")	V19 #2 June 2025
CALL INITER	Sets up ROM routine error-trapping, using "ERTRAP"	V19 #2 June 2025
CALL ERTRAP	Error-trapping routine set up by CALL INITER. Catches ROM BASIC errors like "Divide by Zero", allowing user Assembly-Language programs to resume	V19 #2 June 2025
BERROR	Address of BASIC error code produced by a ROM call which fails (like "Divide by Zero")	V19 #2 June 2025
CALL BERTST	Checks BERROR and sets Z flag accordingly.	V20 #1 Mar 2026
	Data Structures:	
BARGS, BRETS	Byte (8-bit) Argument List, Return List	V20 #1 Mar 2026
WARGS, WRETS	Word (16-bit) Argument List, Return List	V20 #1 Mar 2026
RETURN	Carriage Return, aka "Newline", as a DEFW	V19 #1 March 2025
	Miscellaneous:	
4049H	Highest RAM Address as reported by TRSDOS 2.3	V19 #4 Dec 2025
	"FREE" Memory calculation for Assembly-Language Programs	V19 #4 Dec 2025
CALL HLSUB	"CALL HL": CALL subroutine whose address is stored in HL	V20 #1 Mar 2026
CALL ADJRET	Increments return address on stack by 1	This Issue

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)



4116 16K DRAM

250nS cermaic package.

16K . . . £14.00
32K . . . £27.00

LARGE STOCKS!!

2114-4 . . . £2.50
2102-4 . . . £1.30
Z80 . . . £7.00
Z80 PIO . £6.00

Many other ICs available at competitive prices, SAE for list.

SPECIAL OFFER
Apple keyboard £28.00

All prices are FULLY INCLUSIVE CWO please. ICs sent 1st class return post. Mail order only.

Ground Control

Alfreda Ave,
Hullbridge Essex, SS5 6LT

NATGUG

Users of Tandy & Genie microcomputers supported

★ Meetings on Basic, M.Code, W.P. Model 16, Model 100, DOS
★ Vast range of expertise
★ Free software library
★ Local groups
★ Regularly manned 'phone
★ Back issues of newsletters and specialist journals

Write: NATGUG, 24 Oxford Street, Stony Stratford, Milton Keynes
Tel: Brian Pain on 0908 564271

80 - NET

ELECTRONIC NETWORK

** FOUNDED NOVEMBER 1980 **

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

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

Tel: Milton Keynes 0908 - 566660

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

WOULD YOU LIKE TO HAVE A PROGRAM TO DEVELOP ALL OF THE APPLICATIONS SOFTWARE YOU WILL EVER NEED ?.....

THEN YOU NEED –

THE PROGRAM WRITER/REPORTER®

A PROGRAM THAT WRITES PROGRAMS



THE PROGRAM WRITER/REPORTER®

Enables ANYONE to write complete, running, debugged BASIC LANGUAGE Programs in 35 to 40 minutes with NO PRIOR PROGRAMMING KNOWLEDGE OR ABILITY.

IF you are one of the many who bought a microcomputer in the belief that with just a little studying you could write your own programs, you know that you can't.

IF you, as a businessman, thought you could have stock software modified at a reasonable cost with reasonable results, you know that's not possible either.

IF you are a hobbyist getting tired of the untold hours it takes to write a program, only to find it takes more hours to debug than to write.

IF you are a skilled programmer you don't have to be reminded of the repetitious time spent on each new application.

IF you have left your microcomputer sitting somewhere gathering dust...meet THE PROGRAM WRITER/REPORTER®.

THE PROGRAM WRITER/REPORTER® is not just another data base generator.

THE PROGRAM WRITER/REPORTER®, at your direction, makes complete running programs that are thoroughly documented, easy to modify at any time by YOU!

THE PROGRAM WRITER/REPORTER® cuts programming time up to 90% for a skilled programmer.

THE PROGRAM WRITER/REPORTER® will make anyone a skilled programmer in 30 to 35 minutes!

THE PROGRAM WRITER/REPORTER® does the work! You can answer the simple direct questions and THE PROGRAM WRITER/REPORTER® CREATES...AND ALL IN BASIC LANGUAGE.

Q. After THE PROGRAM WRITER/REPORTER® has produced a program, can it be modified ?
A. Yes, the resulting program is modular, fully documented and readily accessible for alterations or deletions.

Q. Does the program created use so much disc space that there is very little space left for the record storage ?
A. No, the code produced is extremely compact despite complete documentation. If requested THE PROGRAM WRITER/REPORTER® will even 'pack' or compress information. You may even delete the 'remarks' making it even more space efficient.

Q. Must I be expert or even conversant with Basic Language ?
A. No, all questions to and answers from the operator require no computer language knowledge, simple, every day English will do.

Q. What about maths ability ?
A. If you can count your fingers and toes, you'll have no problems.

Q. Will the programs which I produce with THE PROGRAM WRITER/REPORTER® be bulky, slow or amateurish ?
A. No, the resulting programs will be sophisticated and extremely fast operating. For example, should you create a mailing list or inventory program, the time for any record to be retrieved and displayed from a full disc would take a maximum of 1 second.

Q. Must the programs produced conform to a pre-determined format and file length ?
A. No, you determine format and file size to fit your requirements. You may have as many as 500 fields or as few as 1.

Q. Can I develop my own business programs ?
A. For the most part, yes.

Q. What are the limitations ? What programs can I produce with THE PROGRAM WRITER/REPORTER® ?
A. Your own ingenuity and hardware limitations. 100's of different programs.

TECHNICAL ASPECTS

Writes stand alone data base storage/retrieval/update programs to the user's specifications and generates source code in basic (you do not need to use PWR to run generated programs) .

Generated programs are easily modified.

User-defined prompts, edits, error messages and video attributes are standard, fully supported and easily modified.

You may have 500 fields per record with CP/M; 47 fields with APPLE II+; or 200 fields with APPLE III.

You may have up to 32,000 characters per record with CP/M; 3,000 characters per record with APPLE II+ and APPLE III.

Any field may be a key field. New keys may be added at any time. Duplicate keys permitted.

Unlimited records per file (disk limited) .

Unlimited disks per file.

Record deletion automatically supported.

Record access and file maintenance is user transparent.

Record access by a hashing algorithm guaranteeing fast record retrieval.

Minimal disc overhead since there is no special assembly language routine called. No 'basic' overhead.

Programs produced can be transported between 6800, 6502, 8080, Z80, 8085, 8086 and Z8000 based systems.

All packed fields (5 types) may be updated computationally, (2 types – APPLE II) . You may add (or subtract) an amount to a field, or multiply or devide a field by a constant.

Multiple file access and update allowed.

Can be used with Microsoft basic on CP/M systems.

Programs are compilable, using Microsoft's BASCOM (CP/M only) .

Complex selection/exclusion criteria are possible and easily understood when using the reporter.

With the reporter columns may be the result of computations on fields or other columns or intermediate results limited only by the complexity permitted by Microsoft basic. All basic computational functions are permitted, as are all logical and string functions.

AVAILABLE FOR :–

CP/M 8", CP/M 5¼", NEC, OSBORNE, NORTH STAR, VECTOR GRAPHIC, APPLE III & APPLE CP/M AT – £325 + V.A.T. (£373.75)

APPLE II AT – £275 + V.A.T. (£316.25)

PLEASE SUPPLY _____ COPIES OF THE _____ VERSION
FOR MY _____ COMPUTER, MODEL _____

PLEASE SEND FURTHER INFORMATION ☐ tick box

NAME .....

ADDRESS .....

.....

.....

TELEPHONE No. .....

DEALER ENQUIRIES INVITED

ASET

Computer Systems and Services

ASET LIMITED, 24 Princes Street
Southport, Merseyside PR8 1EQ
Telephone : 0704 43008

● Circle No. 173



2,1,3,6,10,9

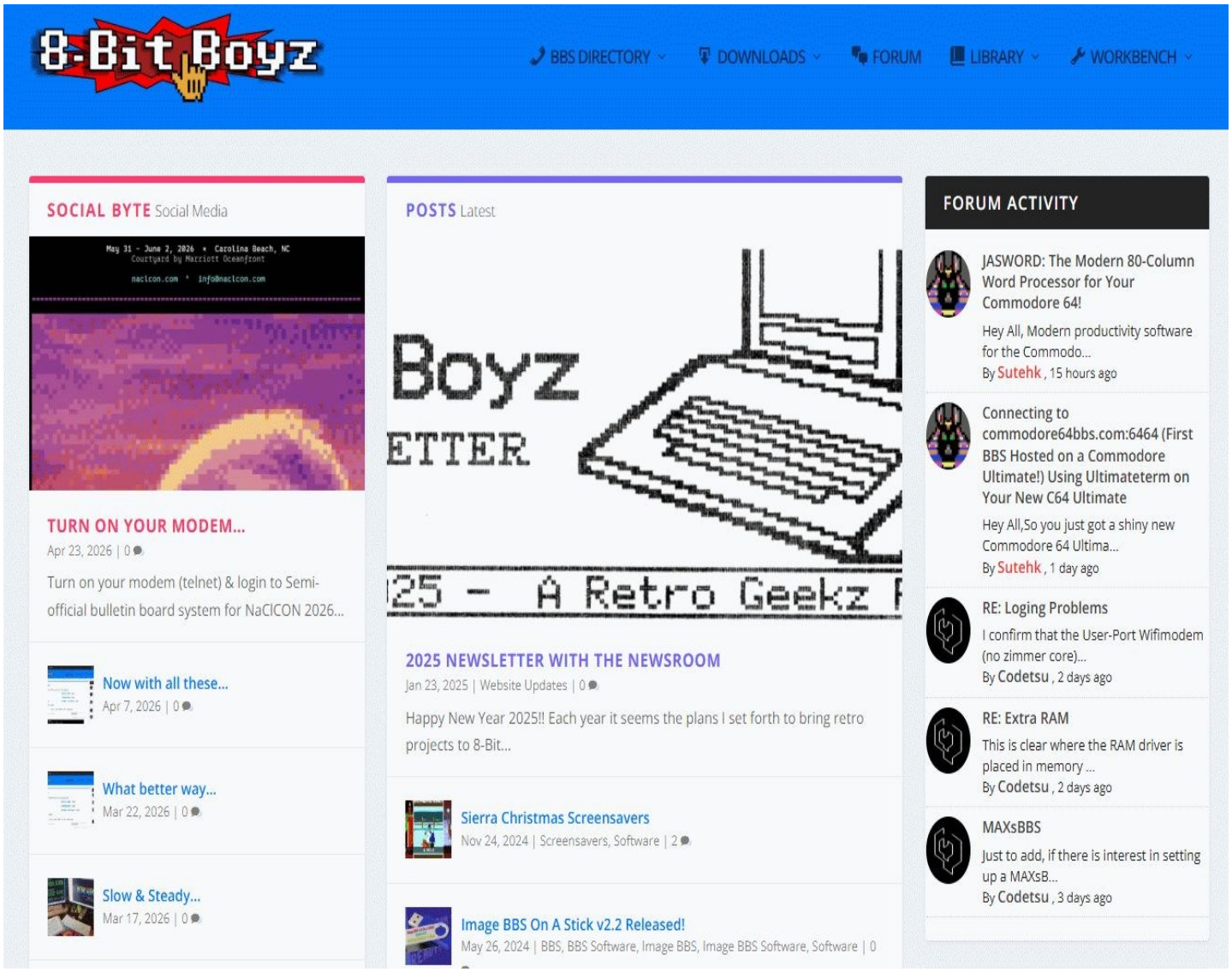
919

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

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

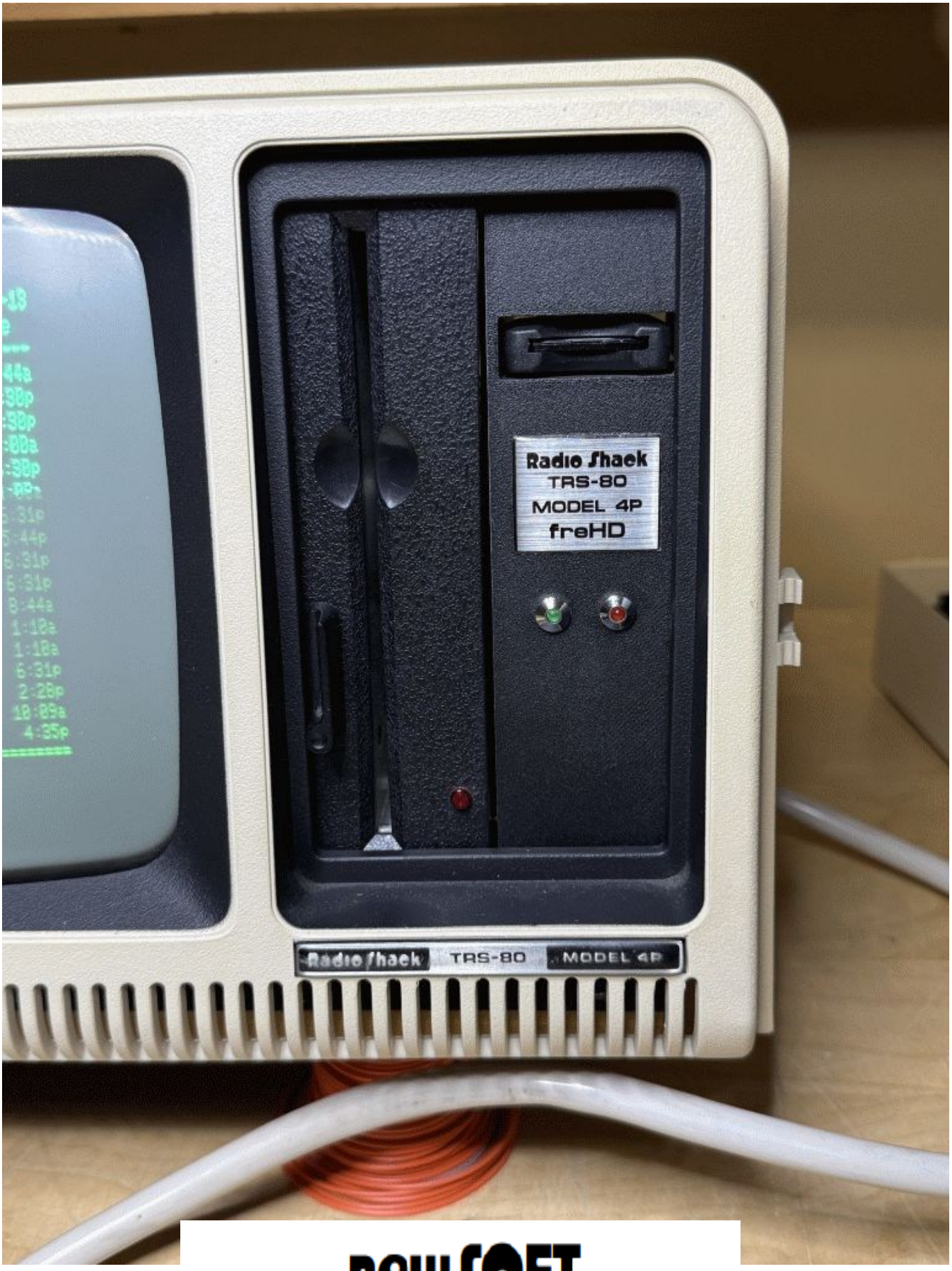
NO CHEATING, BUT THE ANSWER IS ON PAGE 100

Surfin' The Net



https://8bitboyz.com


RS-80
Radio Shack
TRS-80



newSOFT

https://jaynewirth.wixsite.com/newsoft

jaynewirth@verizon.net (email)

IN MAV'S WORKSHOP

LDOS Yes! NewDOS/80 No!

...how did this happen?



by Ian Mavric

OG or former TRS-80 users from the 1970s and 80s sometimes ask me why NewDOS/80 didn't become the de-facto standard hard disk operating system for FreHD hard drive emulation systems. It's an interesting question because NewDOS/80 (referred to as ND80 for the rest of this article) was the gold standard operating system for TRS-80 Model I and III users, as well as Model 4 users running software in Model III mode. In this article I'll explain how FreHD came to be, and why LDOS and LS-DOS became the operating system of choice by FreHD users.

BACK IN THE DAY THERE WAS ONLY FLOPPY DRIVES

Model I and early Model III users had two storage mediums to choose from. Cassette was slow and at times unreliable, but it was cheap. Disk drives cost a lot more but were faster and if proper disk handling precautions taken, almost 100% reliable. A single disk drive was nice but most users opted for a pair of them. That way a system disk could reside in drive 0 and programs and data in drive 1. Making backups of disks was simple with two drives, and a chore if you only had one. TRSDOS, the disk operating system for the Model I and III worked well enough but was lacking customisation that hackers liked. If you ran software that only needed a simple operating system environment to run in and backup disks, TRSDOS did a splendid job. Both TRSDOS 2.3 for the Model I and TRSDOS 1.3 for the Model III were dependable workhorses.

Before TRSDOS 2.3 and even before the Model III was released Radio Shack released version 2.1 which was buggy, and 2.2 which fixed a few of the bugs, but not all of them. Talented programmers released a series of patches, bug fixes and other improvements to TRSDOS that became NEWDOS+ in the late 1970s. The Model I world loved it, it didn't crash and was a big leap forward over TRSDOS. The same programmers later formed Apparat, and in the early 80s released their greatest and most influential product, NewDOS/80 V2.0 and it was an immediate success. While the improvements in ND80 2.0 were numerous, by far the most useful was PDRIVE, which allowed you to set up floppy drives in any configuration you liked. Say you could afford an 80-track double-sided drive, you could tell the PDRIVE table those parameters, and that drive was set. Once set,

access on that drive was incredibly fast, because ND80 didn't do a series of checks to see how many heads, tracks or sides the drive had before proceeding. Properly set up, ND80 made disk operation great.

```
pdrive,0
0*  TI=CK,TD=E,TC=40,SPT=18,TSR=3,SPG=5,GPL=2,DDSL=31,DDGA=3
1*  TI=C,TD=E,TC=40,SPT=18,TSR=3,SPG=5,GPL=2,DDSL=31,DDGA=3
2   TI=C,TD=E,TC=42,SPT=18,TSR=3,SPG=3,GPL=2,DDSL=17,DDGA=2
3   TI=A,TD=A,TC=40,SPT=10,TSR=3,SPG=5,GPL=2,DDSL=17,DDGA=2
4   TI=CK,TD=E,TC=40,SPT=18,TSR=3,SPG=5,GPL=2,DDSL=17,DDGA=3
5   TI=C,TD=A,TC=35,SPT=10,TSR=3,SPG=5,GPL=2,DDSL=17,DDGA=2
6   TI=C,TD=E,TC=40,SPT=18,TSR=3,SPG=5,GPL=2,DDSL=17,DDGA=4
7   TI=C,TD=E,TC=40,SPT=18,TSR=3,SPG=3,GPL=2,DDSL=17,DDGA=2
8   TI=C,TD=G,TC=80,SPT=36,TSR=3,SPG=6,GPL=8,DDSL=35,DDGA=6
9   TI=CK,TD=E,TC=40,SPT=18,TSR=3,SPG=5,GPL=2,DDSL=31,DDGA=3

NewDos90 Ready
```

Picture: a typical PDRIVE floppy configuration screen in ND80 V2.0

WHERE DID LDOS EVEN COME FROM?

Most people first recall seeing LDOS offered as an alternative operating system in Radio Shack catalogs around 1983. Prior to that it was available to be purchased from Logical Systems Inc., and it was an evolution of an operating system called VTOS 4.0 but was also up against many other operating systems released around the same time. LDOS treats disk drives differently to ND80, in that when you format a disk, the drive specs are requested, in the case above it would ask how many cylinders (80) and how many sides (2) then format the disk. When you replaced the diskette in the drive LDOS didn't assume it was the same format, it would examine the disk,

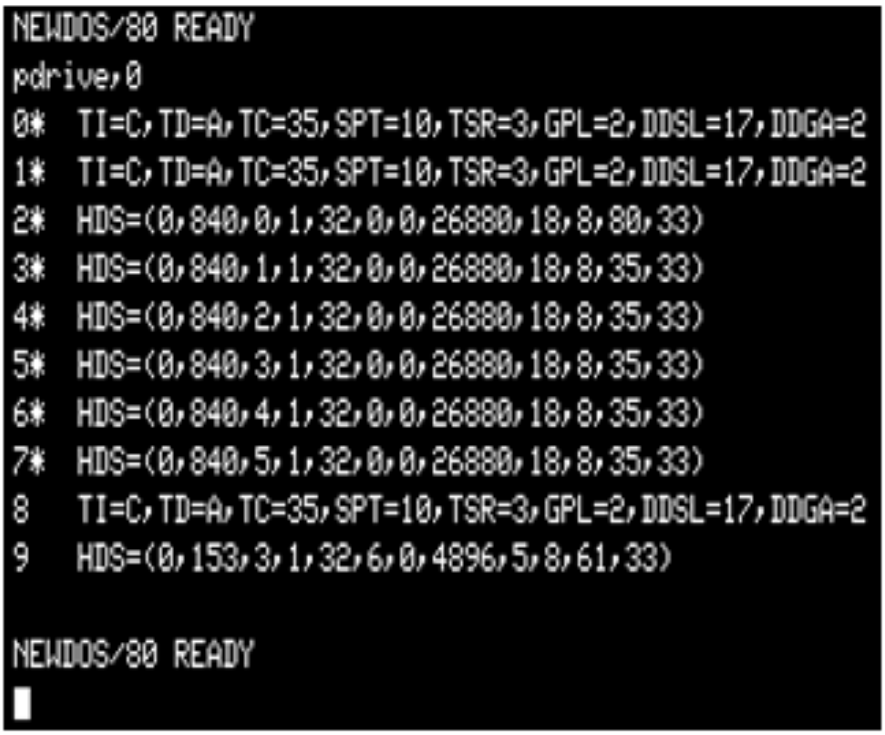
deduce how many tracks and sides, and density, then allow access to it once these were figured out. The process was s-l-o-w compared to ND80. Once LDOS figured out the drive parameters, and assuming you didn't change the disk, its operation was quite acceptably fast. However ND80 remained king of the operating systems.

In 1982 Radio Shack needed an operating system for their new 5Mb hard drive which was available for the Model III, and via an adapter, for the Model I. TRSDOS 2.3 and 1.3 did not lend themselves to hard disk operation, so after a little bit of research and discussions with Logical Systems Inc., LDOS was settled on to be the official hard disk operating system when you bought a 5Mb hard drive from RS. It didn't state it as such in the RS computer catalog, but buying a Radio Shack hard drive meant you were automatically moved to LDOS. RS thusly made LDOS available to be purchased as a stand-alone operating system for people who had both hard disk and non-hard disk based TRS-80s, to allow floppy disk exchange between systems. Another interesting thing happened around this time; the Model 4 was in development and RS decided a whole new TRSDOS version was needed to utilize the Model 4s advanced features. Rather than design their own, RS contracted Logical Systems to produce TRSDOS 6, which is really a version of LDOS for the Model 4. TRSDOS 6 is far removed from TRSDOS 1.3 on the Model III. By choosing TRSDOS 6 for the Model 4 it can be rationalised that this would facilitate future hard disk operators, as RS would have predicted more Model 4 users would have hard drives. LDOS on the Model III (and I with double-density adapter) used the same format as TRSDOS 6, so you can move disks easily between each of those three computers.

HARD DISK DRIVERS - FOR CONTROLLING HARD DRIVES

Driver software back then, as it still does now, is required to add additional hardware and integrate it into an operating system. A hard drive plugged into a TRS-80 with no hard disk drivers installed is useless. Hard disk drivers for the TRS-80 takes up a small amount of high memory, most software doesn't mind them being there. The way LDOS handled this was custom drivers were supplied with the hard disk drive, and were configured using the SYSTEM command, the drives were formatted using a hard disk formatting utility, then the configuration was saved with the SYSGEN command. Drivers supplied by Radio Shack were simplistic but did the job.

Around the same time Apparat started selling hard drives, you may remember their full page advertisements in 80micro. They needed a way to make a hard drive run under ND80 V2.0 and eventually settled on a re-write of relevant parts of V2.0. So much of the operating system was updated a new version, V2.5, was released for the Model I and III. In V2.5 the hard disk drivers are integrated into the operating system with extensions to the PDRIVE command. It makes the PDRIVE command much more complicated and less forgiving if you mess things up, but it worked for both Apparat hard drives as well as Radio Shack hard drives.



Picture: PDRIVE example, HDS are Hard Disk System parameters

In the late 80s Misosys decided a better hard disk driver was needed as interest in hard drives was still high in the TRS-80 world. Used drives were relatively plentiful, and changing the MFM hard drive was easy. If your 5Mb MFM hard drive was dead it was easy, thanks to articles in CN80, to replace it with a 20 or 40Mb MFM unit. RSHARD5/6 was a easy to use driver package which allowed you to specify the number of heads and cylinders required to get the maximum use out of the hard drive. RSHARD5 was for LDOS on the Model III (or 4 in III mode), and RSHARD6 was for TRSDOS 6 and LS-DOS on the Model 4. Patches appeared in CN80 in the early 90s to allow RSHARD to run on a Model I with a hard drive. Naturally it became called RSHARD1. So we now have a suite of drivers for all TRS-80s and all hard drives with MFM capacities up to 70Mb.



FRED VECOVEN HAS AN IDEA - THE BIGGEST IDEA IN YEARS

Though we didn't know it at the time. For those who haven't heard the story, basically Fred had restored a 4P and wanted a hard drive for it. Being in Belgium a TRS-80 hard drive is an extremely rare find. Fred, however, had the ability to design a device which worked exactly like a hard drive and stored the information on an SD card. And thus the TRS-80 Hard Drive Emulator was born. Beta test units were sent to the USA, New Zealand, and to me in Australia. My experience with real TRS-80 hard drives helped debug the emulator, ensuring it's speed was comparable with that of a real TRS-80 hard drive, and in most instances faster. Fred turned it over to public domain and in a gentleman's agreement, I was allowed to sell fully

assembled units. A shorter name was needed, and I came up with FreHD. All units from me bear the name Genuine FreHD on the underside of the PCB.

The first machine it was tested on was a Model 4, and so the question of hard disk drivers arose, and it was decided to design the FreHD in a way that the TRS-80 thinks it's talking to a real Radio Shack hard drive, then the existing drivers can be used instead of writing new ones. This saves development time, but it also means that as hard disk drivers for other operating systems are obtained, we are not limited to just LDOS/LS-DOS, but can set up any operating system. To date that has included CP/M 2.2, Dosplus, MultiDOS, and of course ND80 V2.5.

So, since I am a long time user of TRS-80 hard drives and used LDOS and LS-DOS back as far as the 80s, I am the one to thank/blame for FreHD being released with LDOS as it's preferred operating system choice. Back in the 80s my TRS-80 workhorses included a Model I with 5Mb hard drive, a Model III with dual floppy drives, a Model 4 with 15Mb hard drive, and a Model 4P I kept in the boot (trunk for American readers) of my car. While I was aware ND80 was able to be put on my hard drives, I stuck with what I knew and maintained compatibility between all my computers. You may think that I don't like ND80 or have some other problem with it but in actual fact I just mostly ignored it. When I did look into the way hard drives in ND80 were set up I did, however, find it difficult to understand. To me, if I found it hard to understand, so would many other end users. LDOS and RSHARD, by contrast, I understood perfectly and could help other users get them running with ease.

BUT... THERE *ARE* NEWDOS/80 V2.5 IMAGES FOR FreHD

Brett Paulin, a lifelong ND80 V2.0 enthusiast in my home town of Melbourne, never owned a hard drive, but he did know enough about ND80 that given a FreHD and all the information on V2.5 available, would be able to sift through it all and make sense of it. A few weeks after starting he did have working copies for both the Model I and III. With a few tweaks, they are still in use today when you buy a FreHD and look on the SD card, at least one image for each the Model I and III are ND80 V2.5. Do I recommend using them? On the one hand they do work perfectly well, on the other hand the incompatible floppy disk format makes copying disks made with TRSDOS 1.3, or LDOS or LS-DOS rather tricky. If you stick with LDOS/LS-DOS then copying files from disks you may acquire from places like eBay or other TRS-80 enthusiasts will be easier to load and use.

CONCLUDING

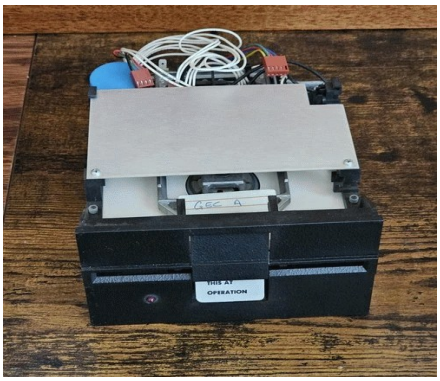
I like to give people options which is why I like the FreHD so much compared to other similar devices out there. Operating system choice is always a personal one and with the FreHD you can use your favourite. Having different operating systems like ND80 and CP/M allow you to run programs which may have been written specifically for those OSs, and won't run under LDOS/LS-DOS. The most recent addition to FreHD OS availability is MultiDOS, and those who have been testing it have found it works very well and is very compatible.

Ian Mavric

ianm@trs-80.com

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?



Tandon TM 100-3 - 5.25" - Floppy Disk Drive - 100 Series 5 1/4" - used untested

£15



Radio Shack TRS-80 Model 16b Microcomputer Floppy Disk Untested Spares or Repair

£1,700



Radio Shack TRS-80 Model 1 Micro Computer Tandy Vintage Computer (Untested)

£148



Tandy Radio Shack TRS-80 Model 4P - Model 26-1080A Untested Spares or Repair

£250

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?



Vintage RadioShack TRS-80 Line Printer VII

£37



Tandy trs-80 model 102 24K Working with Damaged display

£53



Working!! UK Tandy 102 Portable Computer 32k RAM REX Option ROM included!

£183



1980 Micro Computer EACA Video Genie System EG3003 aka Tandy TRS-80 Zilog Z80

£170

Upgrade the PAL Color Computer 1 to 32K or 64K RAM

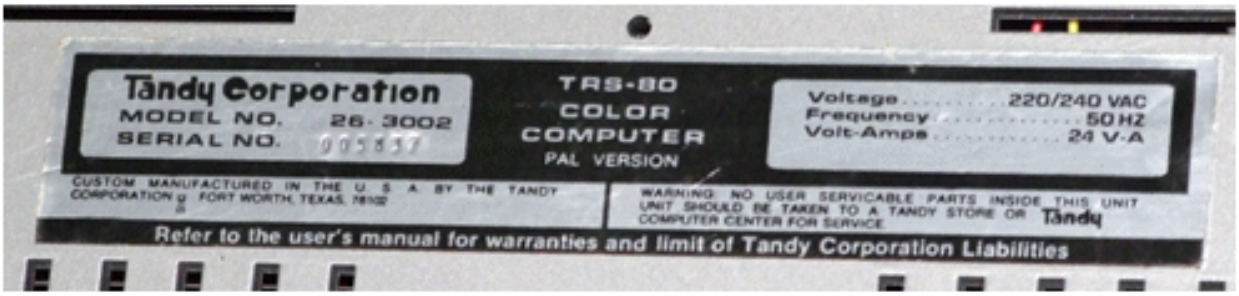


by Ian Mavric

PAL Color Computers are interesting as they differ from their NTSC counterparts. The first version were sold as 240V versions and made in the USA to PAL television standard. In this short article I talk about the differences and how to upgrade the memory to make them more useful. This article concerns itself with the first version, pictured above.

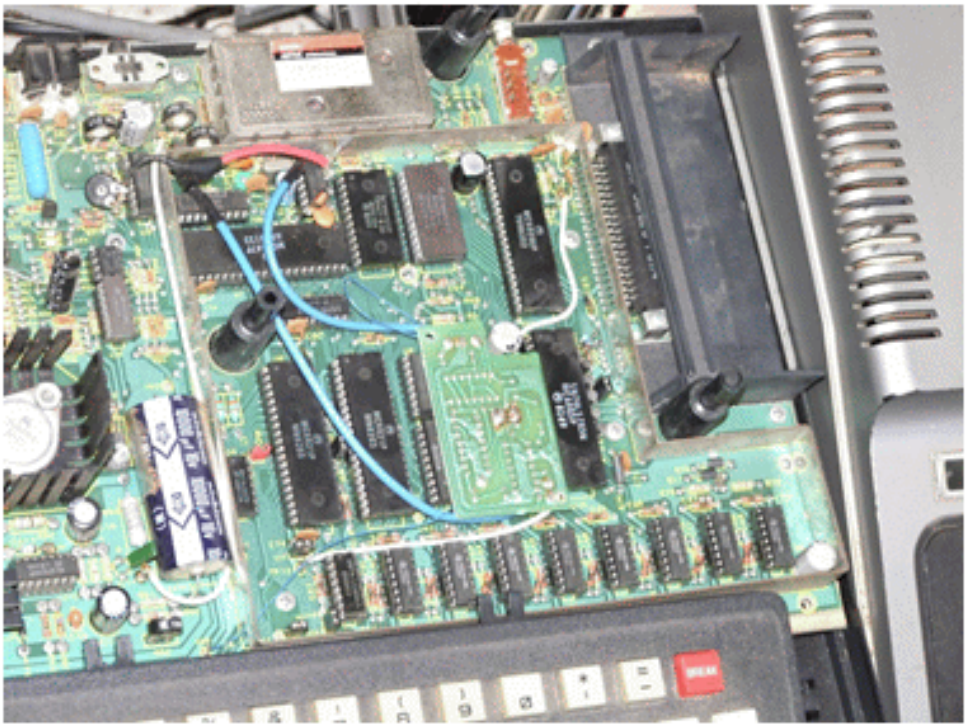
IDENTIFYING AN EARLY PAL COCO 1

In 1980 the Coco 1 was released in the USA as either a 4K Color Basic system, or a 16K Extended Basic. Their catalog numbers were 26-3001 and 26-3002 respectively. In the USA the RF output was to NTSC television sets. These were sold to NTSC countries including Canada. When the computer was deemed a success it was decided to take it worldwide but the differing mains voltage and television standards meant some re-work was is order. Little known outside of the USA, is Radio Shack tweaked the Coco for use on PAL TVs elsewhere in the world.



While the US Coco 1 was initially released with Rev D motherboard (with a few development Rev Cs released really early on) the PAL Coco 1 motherboard is actually based on the Rev E US board. You can tell this because the Rev D board has no jumpers for 32K or HI/LO jumper for selecting which half of the 4164 Ram chips are used in a 32K system. Rev E came out in the USA in 1981 and at the time a 32K version of the Coco 1 became available for purchase and it carried the new catalog number 26-3003. When you look at the PAL motherboard the first think you see is that it's Rev A, but this does not pre-date the US Rev D (or C) version. It has the jumpers for 32K and the aforementioned HI/LO jumper.

The big change is that right in the middle of the motherboard is a satellite board which handles the PAL conversion. It is also accompanied by a lot of bodge wires and passive components to make it all work. When you first see the satellite board it can be quite concerning.



Picture: PAL Decoder daughterboard in a 32K Rev A motherboard Coco 1

UPGRADING PAL REV A MOTHERBOARDS TO 32K

If your PAL Rev A has 4 or 16K, replace the RAMs with 4164s and set all four jumpers to the 32K position. One jumper is to the right of U10 (the SAM chip) and the other three are just above the keyboard connector. Remove capacitors C48, C70, C45, C67, C35, C64, C61 , and C31.

If it is not already installed, add a jumper connecting the LOW and the unmarked centre pin to the left of U8 (a PIA). The other pin,

marked "HIGH", is not used. Locate the two pins Without anything connecting them labelled R83 to the right of U10. Solder in place a 33Ω, 1/4-watt resistor connecting the two pins. You now have what Radio Shack called a 32K machine.

ROM-WRITE DISABLING - TURNS A 32K COCO INTO A 64K

You must disable the ROMs during a write cycle. RAM will occupy the entire address space from $C000 to $FDFF. The interrupt vectors will not be affected since they reside in the protected area from $FF00 to $FFFF. Follow these steps:

• On the motherboard, locate and remove the two ICs (integrated circuits) labelled U11 and U29. They are a 74LS138 and a 74LS02.

• Prepare the chips as follows:

• Carefully bend pins 4 , 5, and 6 of the 74LS02 straight up.

• Carefully bend pin 5 of the 74LS138 straight up.

• Solder a short piece of wire to pin 8 on the 74LS02 at the point where the pin enters the chip, as you must plug this pin back into the socket.

• Solder the other end of the wire to pin 6 of the same chip. Replace the chips in their sockets.

• Solder a piece of wire from pin 5 of the 74LS138 to pin 4 of the 74SL02.

• Locate TP1 and cut a length or wire that will reach from pin 5 of the 74LS02 to TP1. Solder one end of the wire to pin 5 and wire-wrap the other end to TP1.

• Recheck your wiring and look for any shorts, especially between the 74LS02 and the shield.

At this point you need to remove the Level II Eproms out of Z33 and Z34 on the motherboard, or if you have an earlier Level II upgrade connected via a ribbon cable remove that, flip over the board and remove the Level II ROM board and the 4 (or 6) wires which run to the motherboard. One of them, which is usually Yellow, is address line 11 (A11), take note of that.

4. Insert the 2732 into Z33 and solder the wire from pin 21 to where the Yellow A11 wire used to be, this time on the component side.

AN INTERESTING THING ABOUT THE 32K UPGRADE

From time to time I see Rev A machines and I have never been 100% happy with the quality of the picture on the TV. It's Ok but it could be better... In fact the later Coco 1, the 2nd version (or cost reduced version) has a much better picture on the screen. Notice that in the 32K upgrade section above a 33Ω resistor is placed between two pins? I have seen more times than I care to remember that a jumper has just been placed between the pins. One day when I was diagnosing a computer which had the resistor not the jumper, I bridged the two pins and the picture got markedly worse. Also the

sound from the speaker had a more annoying 'trebly' character to it. Since that time whenever I see a jumper in place I remove it and solder in the 33Ω resistor as required. I am not entirely sure what the resistor does but it must quiet down an otherwise noisy line on the motherboard. In a way, this is the most important part of the 32K upgrade, lest you be stuck a crappy picture even though you have more memory.

NEXT TIME

I'll look at the next PAL Coco, which was sold from 1982. This time construction moved to South Korea, and it was markedly different to the US 'F' (or '285) motherboard sold in NTSC countries.

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





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

WE LOOK FORWARD TO HEARING ANY OF YOUR RECENT TANDY ADVENTURES.

WHERE 8 BITS, 16K WITH AN ACULAB STILL RULE OK

COMPETITION ANSWER ((2*10)-3)*(6*9)+1 = 919