TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK



HI EVERYONE AND A WARM WELCOME TO THE MARCH 2023 EDITION, WHICH STARTS OUR 17TH YEAR!

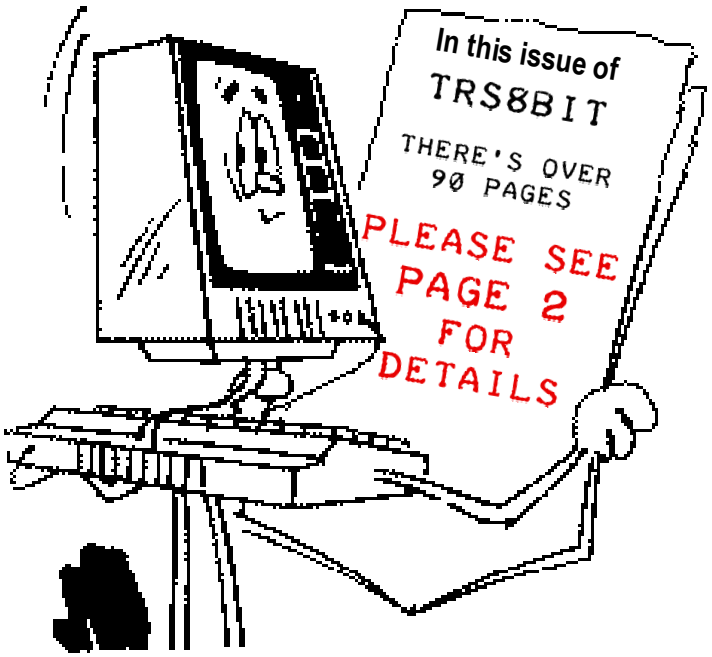
FIRST, LET ME THANK EVERYONE WHO HAVE CONTRIBUTED ARTICLES TO THIS BUMPER EDITION! THANK YOU. WITHOUT YOUR SUPPORT AND INTEREST THIS NEWSLETTER JUST WOULDN'T HAPPEN!

SECONDLY, GREAT NEWS FOR ALL TANDY FANS....

Tandy Assembly 2023

IS HAPPENING IN
SPRINGFIELD, OHIO
29SEP23 - 01OCT23
HAVE YOU BOOKED YOUR TICKET?

NOW, HERE'S A BIT OF SAD NEWS. I'M SORRY TO REPORT THAT MAV IS UNWELL AND RECOVERING FROM A SERIOUS ILLNESS SO THAT'S WHY THIS ISSUE LACKS AN ARTICLE FROM HIM. A STELLAR RUN OF ARTICLES, AT LEAST ONE PER ISSUE SINCE SEPTEMBER 2011, PERHAPS NOT EQUALLED BY ANY OTHER CONTRIBUTOR, WILL RETURN IN JUNE. I'M SURE WE ALL WISH MAV A SPEEDY RECOVERY AND LOOK FORWARD TO HEARING FROM HIM IN JUNE.



GEORGE AND PETER PHILLIPS ARE RELEASING THE LATEST UPGRADE OF THEIR TRS80GP EMULATOR. VERSION 2.4.11 AVAILABLE FROM THEIR WEBSITE, SHOULD BE UP AND RUNNING BY THE TIME YOU READ THIS.

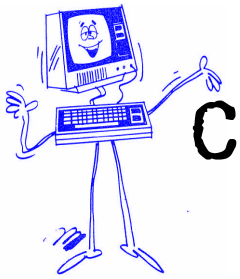
SO, DOWN TO THIS ISSUE AND AS I'M SURE YOU CAN SEE, IS YET ANOTHER BUMPER ISSUE. IT STARTS OFF WITH AN ARTICLE ABOUT RUNNING THE TRS-80 ON THE 'NABU' AND, UNBELIEVABLY JUST GOES ON GETTING BETTER!. RUN A MODEL 1 FROM A USB PORT, STRING SCANNING, RE ENGINEERED HARDWARE, A GENIE III RELIVES, A NEW SERIES, 'HANDY HINTS' (JUST FOR FUN). DID YOU REALISE THE MASSIVE DIFFERENCE BETWEEN THE UK'S AND USA'S M100? WELL, THAT'S SORTED! PASCAL HOLDRY IS IN NEED OF THE ORIGINAL CASSETTE SOFTWARE FOR THE TC-8. PLEASE CAN YOU HELP?

AND FINALLY, THERE'S A FULL REPORT OF THE 2022 COMPETITION AND DETAILS OF THE 2023 COMPETITION, WITH, ONCE AGAIN, A FABULOUS PRIZE.



ENJOY THE READ!
SEE YOU ALL IN JUNE, UNTIL THEN -

TAKE CARE - DUSTY



CONTENTS

PAGE 04 — MODEL 1 ON THE NABU
TED FRIED

PAGE 08 — TIME TUNNEL
E.T. FONEHUME

PAGE 10 — MODEL 1 USB 'A' POWER MOD
TIM HALLORAN

PAGE 27 — HELP, FREE, FOR SALE & WANTED

PAGE 28 — AT THE READY PROMPT
PETER PHILLIPS, ML ARNAUTOV, CLIVE DAVIDSON

PAGE 31 — STRING SCANNING
VINCE OTTEN

PAGE 48 — GENIE III LIVES AGAIN
KRIS GARREIN

PAGE 49 — NEWS FROM FRANCE - RE-ENGINEERED TC-8
SEE PAGE 27 - CAN YOU HELP PLEASE?
PASCAL HOLDRY

PAGE 57 — TRS-80 JOKE
GEORGE PHILLIPS

PAGE 60 — THE WAY WE WERE
SERIAL PRINT DRIVER & MOLIMERX £250 REWARD COMP.

PAGE 66 — THE RUNNING CAT
GEORGE PHILLIPS

PAGE 68 — * NEW SERIES * HANDY-HINTS
I'D NEVER THOUGHT OF THAT!

PAGE 69 — SDLPT SD-CARD THROUGH THE PRINTER PORT
JUST FOR THE TANDY-PC BOYS

PAGE 70 — U.K. EBAY BARGAINS
WERE YOU ONE OF THE LUCKY ONES? - LET US ALL KNOW!

PAGE 72 — M100 U.K. AND U.S.A. ROM DIFFERENCES SORTED
SARAH LIBMAN

PAGE 78 — 2022 COMPETITION - WRAP-UP DETAILS
GEORGE PHILLIPS

PAGE 90 — **THE 2023 COMPETITION**
SUPER PRIZE ON OFFER AGAIN!



NEW VERSION 2.4.11

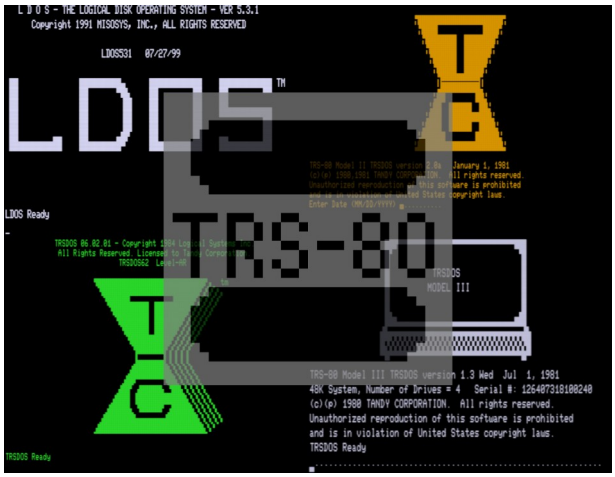
DON'T HAVE A TRS-80?

CRAZY PRICES ON EBAY?

TIRED OF RIFA CAPS BLOWING?

OF FLOPPIES SHEDDING OXIDE?

OR JUST PLAIN DEAD CHIPS?



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

RUNS ON WINDOWS, MACOS, LINUX & RASPBERRY PI.
EMULATES MODELS 1, 2, 3, 4, 12, 16, 6000,
MC-10, DT-1, VIDEOTEX AND MICRO CO-CO.

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

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

NEW VERSION 2.4.11, AVAILABLE NOW, FEATURING :-

PRELIMINARY DEVICE VIEWER.
SOUND EFFECTS WITH STEREO SOUND ON MACOS AND LINUX
DISK VIEWER SHOWS SECTORS IN ID ORDER INSTEAD OF POSITION



Model 1 L2 on the NABU

Ted Fried

The NABU Personal Computer has been getting a lot of attention recently thanks to the video by Adrian Black which re-kindled interest in this vintage computer and pointed out that someone was selling about a thousand of them which were new in the box, so I purchased one!

Unfortunately there isn't much one can easily do with this machine, however there are a few projects underway which port CP/M to it and also creating a system which can serve over the network the original games that this computer ran.

I thought I would make a contribution by implementing something totally different which took advantage of the NABU's motherboard components...

I thought it would be an entertaining project to use my MCLZ8 to emulate the Zilog Z80, the TRS-80 Model I ROM BIOS and DRAM, and remap the keyboard and video to the motherboard resources of the NABU Personal Computer. Now the NABU is effectively acting like a Tandy TRS-80 Model 1!

This project was built quickly thanks to already having most pieces from existing projects and only took about an afternoon to complete.

The first step was to confirm that the MCLZ8 would work in the NABU as a replacement CPU for the Z80. The MCLZ8 was previously tested to work on the TRS-80 Model III which has a slower bus speed than the NABU, but it worked the first time without an issue.

Next I added some storage arrays in the MCLZ8 to store all of the NABU's configuration IO accesses to the video chip and keyboard. This saved me a lot of time so I would not need to learn how these chips worked from the datasheets!

I then added the TRS-80 BIOS ROM and RAM emulation into the MCLZ8 so that all accesses to these memory ranges were handled in the Teensy using an array in C.

The NABU and TRS-80 have different video sizes with the NABU being 40×24 and 64×16 for the TRS-80. Since the NABU had more vertical space I was able to start the TRS-80 video downwards 8 lines which left room to keep the NABU splash screen which I thought is a neat effect! The TRS-80 has more columns so the NABU can only display 40 of them. The characters are still there in video memory, they are just not displayed. I used an array to convert the TRS-80 row/column mapping to the NABU which is faster than performing the math…

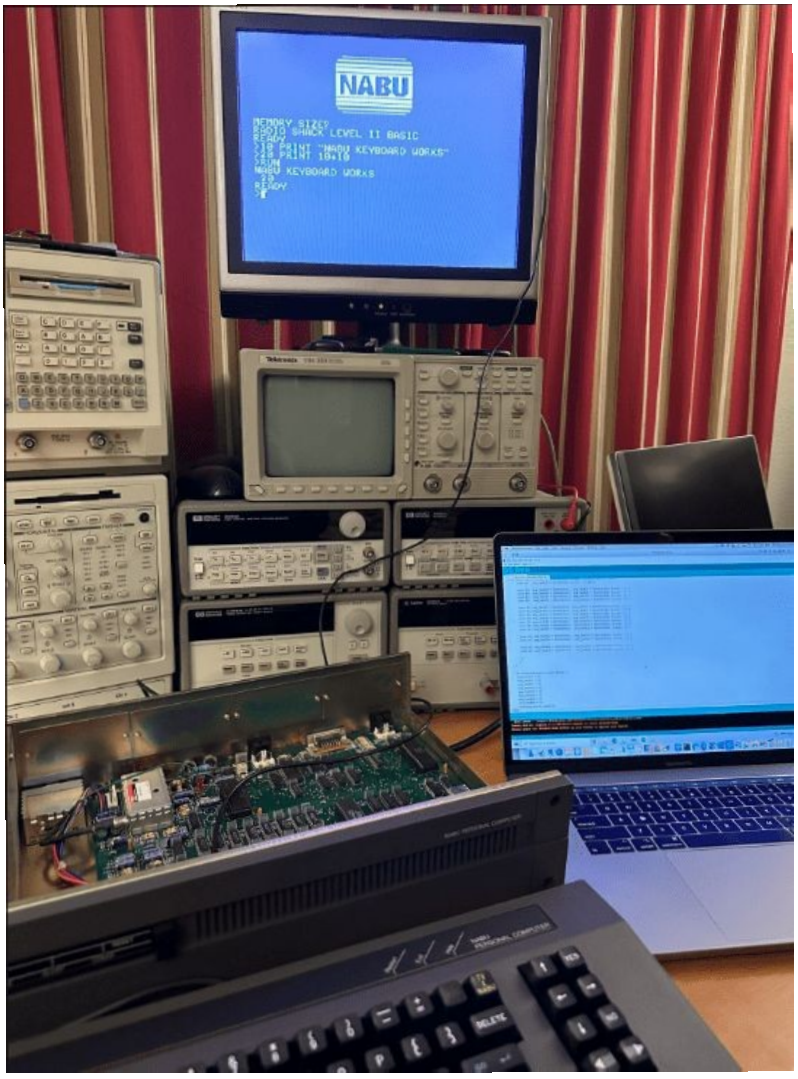
The NABU keyboard sends keystrokes as ASCII characters to an Intel 8251A on the motherboard, so I only needed to poll the UART's status bit and read characters when they arrive. The TRS-80 performs parallel keyboard polling instead of using a UART so I needed to convert from the NABU's ASCII to the address bit map for each TRS-80 keyboard polling address. This work was done on one of my previous projects which I was able to reuse.

So, that's about it! The NABU now boots to the splash screen and then runs the TRS-80 Model I Level II BASIC. Below are a few pictures of the machine in action.

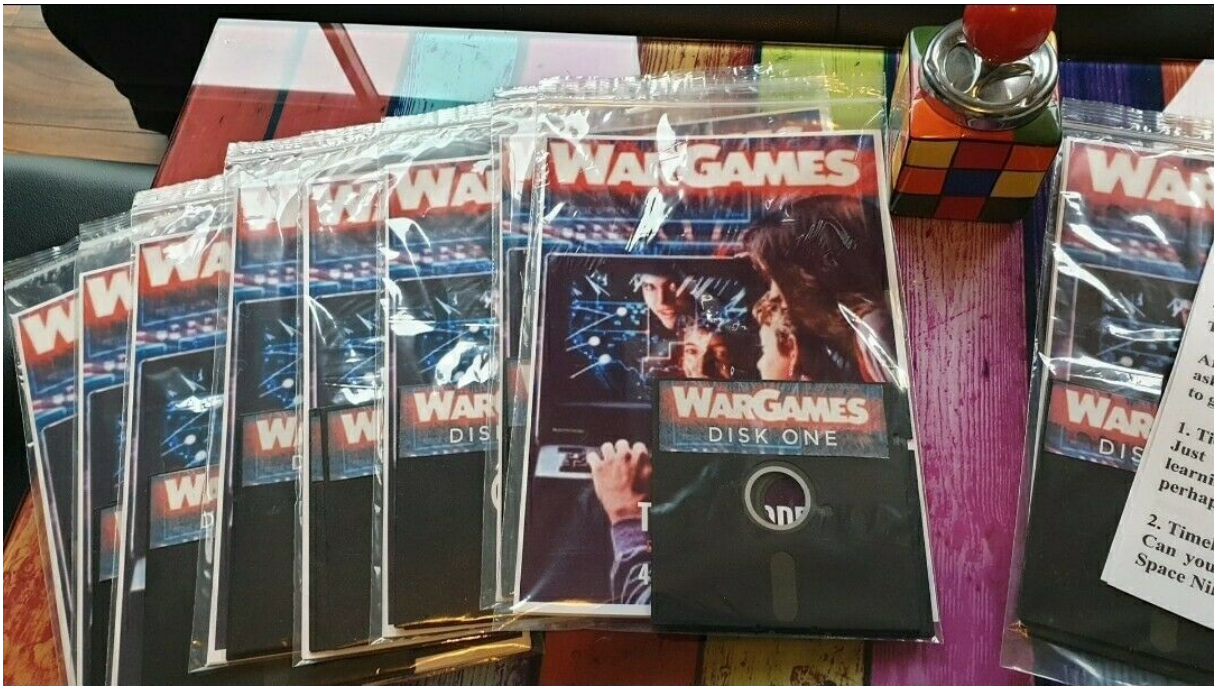
All of the source code is uploaded to GitHub.

Ted Fried




NABU
MEMORY SIZE?
32610 BYTES FREE
NABU LEVEL II BASIC
READY
>10 PRINT "NABU KEYBOARD WORKS"
>20 PRINT 10^10
>RUN
NABU KEYBOARD WORKS
 2.5
READY
>
SHARP


NABU
SHARP



All new from Dreamland Software.

This two disk package comprises what is in effect the FULL Dreamland Software collection.

Based on the classic 80's film 'Wargames', your entire experience is controlled by the W.O.P.R computer from the aforementioned movie.

Greetings Professor Falken.

New titles including (of course) Global Thermonuclear War. (Ouch.)
Retro Football Manager. (Based on the ORIGINAL 80's classic, a full soccer management simulation.)
The Battle of the Mutara Nebula. (If you are a Star Trek fan, then you know FULL WELL what this is.)
And the Kobayashi Maru Scenario. (Another one for the Trekkers.)

Also included, are **ALL** games released by Dreamland Software since inception four years ago.

The hugely successful Actual Reality. It's sequel Actual Reality II - Battlestar Raven And RockStar.

14 Programs in total (Including the W.O.P.R Master Program.)
Both disks full to capacity whilst remaining bootable.

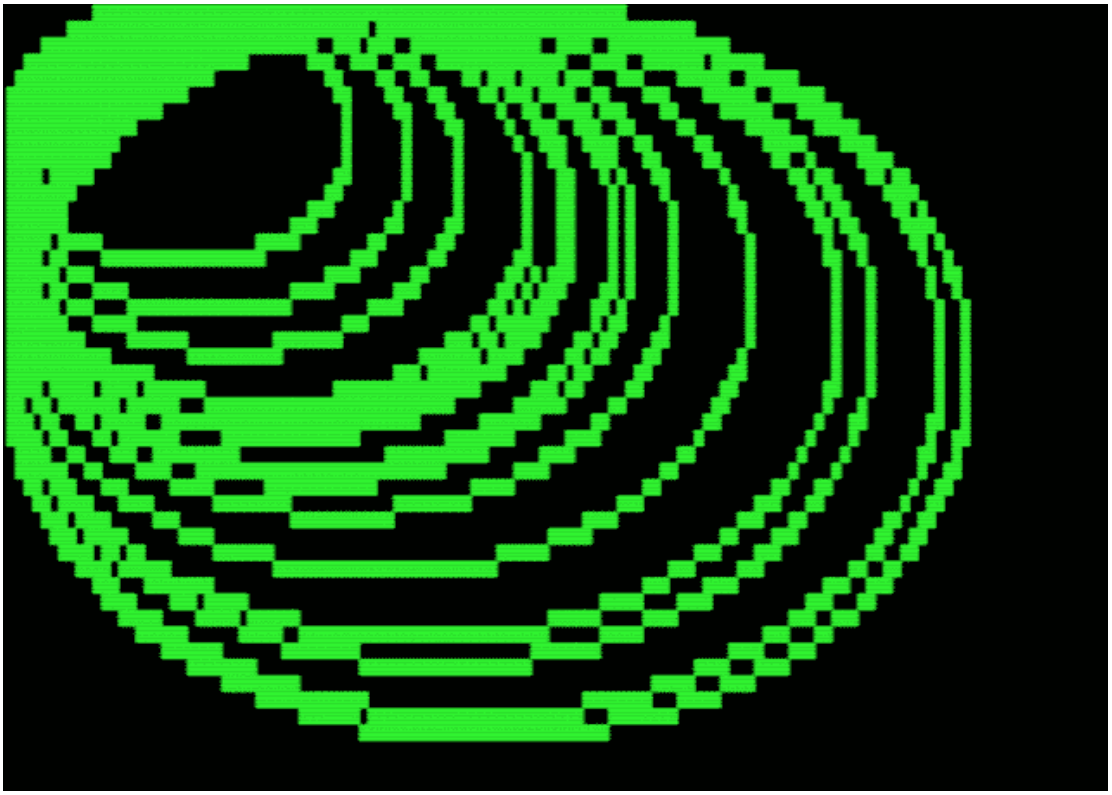
Full video preview available here:
https://youtu.be/XXD1IG8-CO0

Please note that this item is currently available for Model One ONLY.
Model Three/Four will be available soon. (Watch for update.)



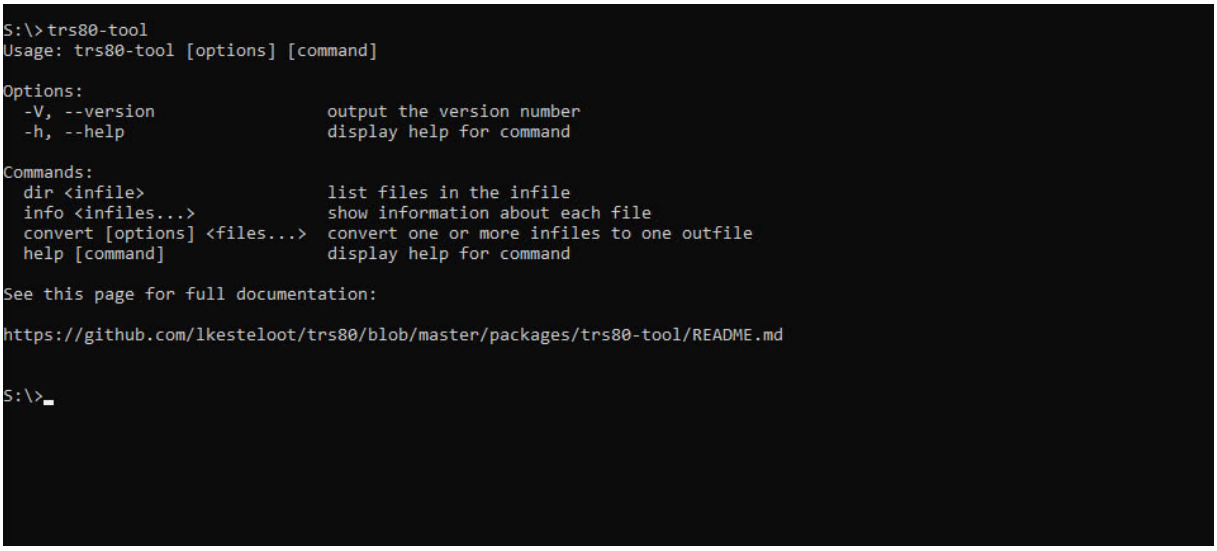
E.T. FONEHUME

HAPPY NEW YEAR DUSTY. I WAS PLAYING ON MY TANDY TO MAKE A ONE-LINER CIRCLE CREATION PROGRAM AND WAS CARRIED AWAY ON A 'NEW YEAR' EXCITEMENT 'BURST'! HERE'S A 'TIME TUNNEL' IMAGE CREATOR WHICH IS BEST RUN ON, AS ALWAYS, TRS80GP BUT WITH THE TURBO OPTION SWITCHED ON, UNLESS, OF COURSE, TIME'S NOT A PROBLEM.



```
0 RANDOM:CLS:FORT=0TO1STEP0:D=RND(90)+30:FORX=.02TO6.5STEP2/D:SE
T(((SIN(X)+1)*.5*D),((COS(X)+1)*.2*D)):NEXT:NEXTT
```

Lawrence Kesteloot has rewritten, his TRS-80 Cassette command line tool to support many new functions. Called "TRS80-TOOL" it is a command-line tool for manipulating TRS-80 files.



Lawrence Kesteloot has updated his "TRS80-TOOL" to v2.3.1

It runs under NODEJ, Linux, macOS, and Windows. It can be downloaded from Lawrence's site or locally from the The Virtual Tape Utilities page on this site.

The new version adds support for multi-file CAS files (such as CLOAD Magazine or Temple of Apshai) and will try to guess (and set, if you want) the starting address of a SYSTEM file.

The main page for TRS80-TOOL can be found at

https://github.com/lkesteloot/trs80/tree/master/packages/trs80-tool

along with instructions.

It runs under NODEJ, Linux, macOS, and Windows. It can be downloaded from Lawrence's site, the The Virtual Tape Utilities page on this site or right here for Windows / Linux / Mac.

TRS-80 Model 1 USB-A Power Mod (5V DC conversion)

Tim Halloran

This modification can be useful if your Radio Shack power brick is bad or noticeably hums. You might not have one at all. The modification lets your Model 1 run significantly cooler inside. I've tested this modification on two Model 1 keyboards (one with the Rosser 64K in-the-keyboard modification select *Upgrading the Model I to 64K of DRAMs (pdf)* from the menu that the link brings up on Ira's site) and run them for multiple days.

I strongly suggest you do not do this modification on your only working Model 1 unless you are quite skilled with electronics work. The modification is pretty simple but does require one trace cut and the removal of several components from the motherboard. Several wires are then soldered in place. Finally, a 5-pin DIN to USB-A cable is constructed.



This image shows the resulting system.

Key features of this modification are

The outside of the computer is unmodified and is indistinguishable from a stock Model 1. As you will see below, I use a sticker and orange paint on the 5-pin DIN socket to distinguish my prototype units so I use the right power cable.

It won't work, but is safe, to connect a

real Radio Shack model 1 power brick. It is very easy to make this mistake. I have several times.
Power still comes in via the 5-pin DIN connector.
The power switch still works (so does the reset).

MEAN WELL power supply is recommended

The modification creates the ability to use USB power, but I recommend you setup a MEAN WELL RS-15-5 AC to DC power supply. This has very low ripple and noise (on par with the original power brick). It also works on AC power worldwide. One of these can power a keyboard and an expansion interface in my tests.

My experience leads to this rule of thumb. If you changed the memory and have a 5V only keyboard a USB charger tends to work fine as power. If you want to use an expansion interface, or you kept the 4116 memory (using a DC-to-DC converter) then you need to use the MEAN WELL.

Two modification options

There are two options, one I consider GOOD and the other BAD:

(OPTION GOOD) This replaces the 4116 memory with either 4517 (16K) or 6665 (64K used as 16K). This simplifies the modification and allows the entire computer to run off 5V DC power. I've had far less trouble with this modification.

(OPTION BAD) keeps the 4116 memory in the computer but requires a small DC-to-DC converter. I've found this is less successful, and more trouble. It seems to only work with the MEAN WELL linear power supply (where you have clean power and can dial in the voltage to about 5.1V. This also tends to bring out weaknesses in the video sync logic. Be prepared to swap out Z5, Z6, and Z57 with new chips.

A word of caution

I strongly suggest starting with a working Model 1. Do not do this modification unless you are comfortable working inside a Model 1. If you have never opened one up odds are you will destroy your computer. I can almost guarantee that the flimsy stock keyboard connector will be damaged. Consider removing the original keyboard connector and installing headers or another more robust solution of your choice. You'll see my "quick connect" approach in the pictures.

This modification is harder than installing a lowercase modification but easier than doing the Rosser 64K in-the-keyboard modification.

What you need

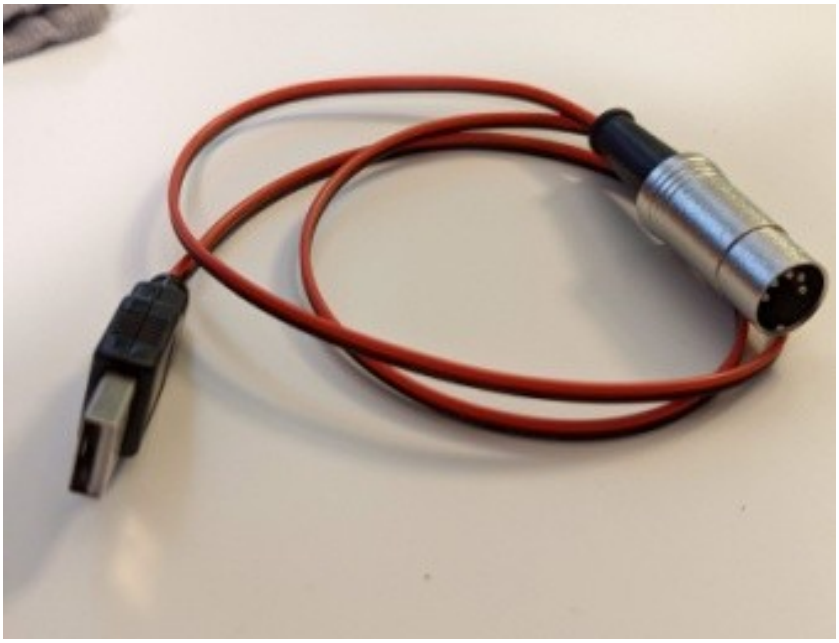
Here's the list of parts you'll need

- One male 5-pin DIN solder connector
- One male USB-A solder connector
- Wire at least 22 gauge (to carry DC power). Silicone is recommended. For the cable I use parallel red-black wire. Inside the computer it doesn't matter. Do not use wire-wrap gauge wire for any part of this modification!
- **(OPTION GOOD)** Eight MCM4517P15 (16K) or MCM6665BP20 (64K used as 16K) memory chips. Search AliExpress or eBay for deals (or reach out to sellers like Mav). Memory is socketed (thank goodness) so this is easy to install. Ensure what you buy is in a 16-pin DIP package. Also if you upgraded a CoCo 2 from 16K to 64K, the 16K memory you took out is 4517 memory (and will work). Further, if you have done the Rosser 64K in-the-keyboard you are all set. Rosser's modification already converted your memory to 6665 chips which only require 5V.

(OPTION GOOD) A 14-pin DIP socket and a 14-pin DIP 7 position switch. I'll show this below, however, you can just use a short wire if you wish.
(OPTION BAD) A DC to DC converter 12V/-12V. I used this one, pick +-12V no pins.
What tools do you need
• A good desoldering tool that you are comfortable using. I use a Hakko 301.
• A good soldering iron. I use an (older) Weller WES51.
Tools to take the Model 1 apart. I assume you can take apart a Model 1. (If not please see YouTube as there are lots of videos showing this.)

Building the power cable

Below is a picture of the cable you need to construct. I might have one, please reach out to me on Facebook or Discord and I'll try to send you a cable. Constructing the cable is a difficult task to accomplish (DIN connectors are a pain to solder).

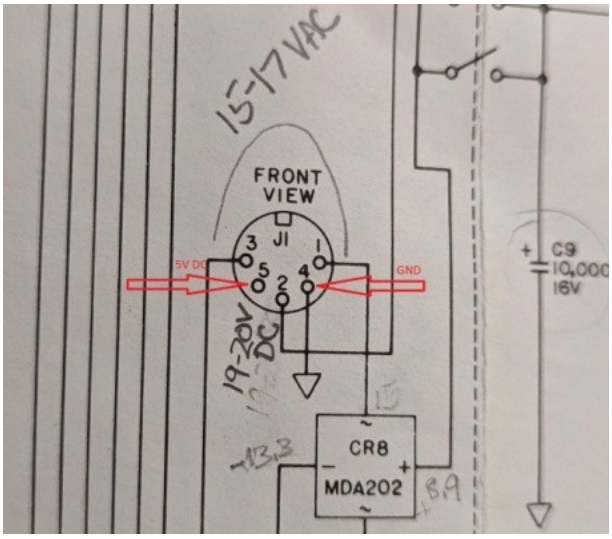


Caution

Do not cut and "reuse" a USB-A cable to save time. This is a bad idea. We want to use 22 gauge wire (or larger) to ensure good current flow. The Model 1 keyboard uses ~0.9 amps. A lot of commercial cables are intended for data and use very thin wire. Soldering a USB-A plug is easy, it's soldering the DIN that is difficult.

Some background and theory behind our cable

To understand this cable, we need to look at what we are doing and why the Model 1 is somewhat complex to convert to DC power. The image below shows the pinout of the Model 1 power connector from the *TRS-80 micro computer technical reference handbook*.



This image explains why it is not easy to just create a replacement power brick and plug it in. AC and DC power is sent into the keyboard. Pins 1 and 3 send in 15 -17V AC (I measured this at 15V into the bridge rectifier CR8 on my machine). ~20V DC comes in on pin 2 with Ground on pin 4. So 15V AC and 20V DC come into the computer.

What the computer actually needs to operate (inside) is three DC voltages: 5V, -5V, and 12V. However only the 4116 memory uses all three voltages. The remainder of the motherboard only uses 5V.

This modification keeps ground unchanged on pin 4. We'll send 5V DC in on the unused pin 5.

Solder the 5-pin DIN first

Take apart the DIN and solder the black wire to pin 4 and the red wire to pin 5. Place a small piece of heat shrink on the wire to slide down after soldering. You should get something like the next picture.



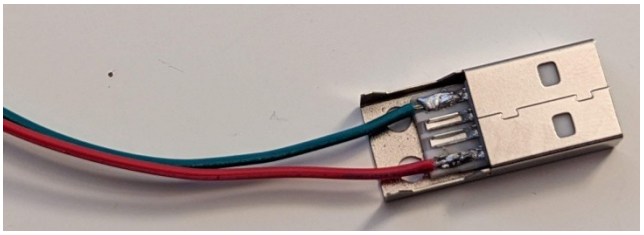
One suggestion is to tape the DIN (pins down) to the edge of the table and bring the wire down while soldering to the back. I do not recommend that you clamp a pin to an alligator clip holder and solder, I've found that this approach often melts the plastic holding the pins. Resulting in a bent pin.

Ensure you give the connections a good tug. If they pull apart you did a bad solder job and need to try again.

Next put the entire DIN connector together. We started with this connector because you have to slide the cover over the wire. This is easier before the USB-A connector is attached.

Solder the USB-A connector second

Ensure the entire 5-pin DIN is soldered and put back together. Half the cable is done. The male USB-A connector gets soldered as shown in the image below.



Note in the image above red is 5V and green is ground. This is a connector I keep around to check which side is which. Your cable will probably use black for ground.

Ensure you give the connections a good tug. If they pull apart you did a bad solder job and need to try again.

You then snap the USB plug casing around the metal. I tend to use

some CA glue and clamp the two sides together to ensure the casing doesn't come apart over time. This glue is not required, but I suggest you keep it in mind if plug casing keeps coming apart as you use it.

Your cable is complete. It is a good idea to test it with the continuity tester on your multimeter.

Modifying the Model 1 computer

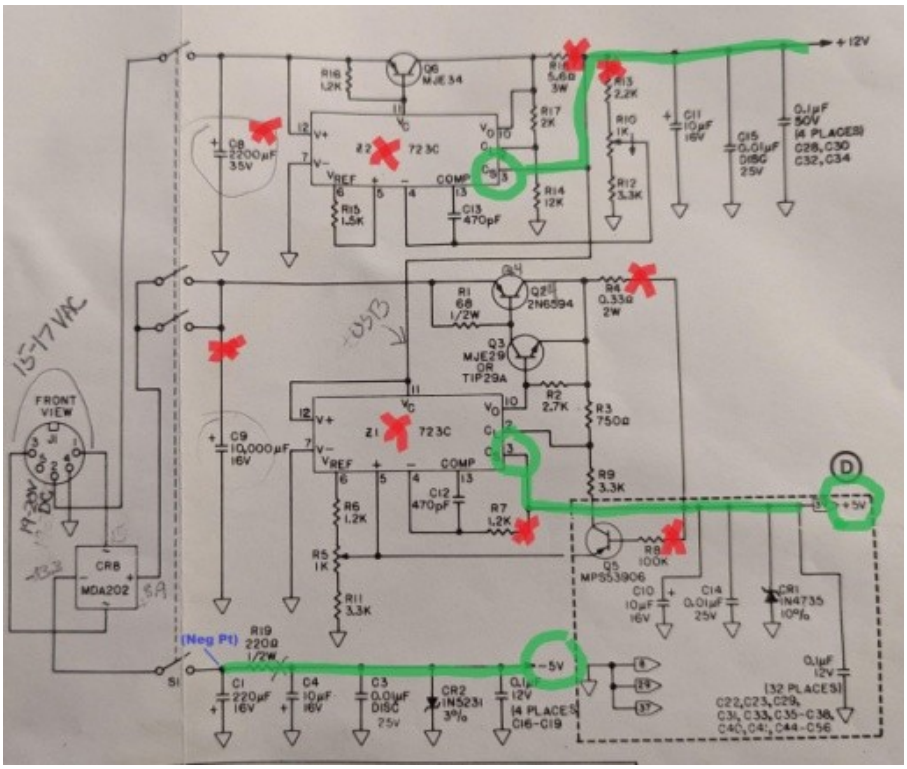
In this section I'll describe how to modify your Model 1 motherboard. We'll start by explaining what we are doing and then go into the step-by-step.

Some background and theory behind our changes

Changing a Model 1 to use DC (even batteries) has been done in the past. One instance that I'm aware of is discussed in TRS-80 Trash Talk Episode 18. Mike Yetsko created a battery powered TRS80. In this episode Mike describes using batteries to supply all three voltages 5V, -5V, and 12V. If you really know the computer anyone can do this. It is more complex to keep the modification simple. My first prototype was a tangle of wires and cuts. Its gotten better.

What we want to do is bypass the power regulation circuits on the motherboard. The trick is to channel the power around the active power regulation circuits (that create the heat in a stock Model 1). Also we do not want to remove the big transistors (this was required in the CoCo but is not in the Model 1). What we want is a clean path to the DC power out to the voltage rails. My pencil marks give some

insight into earlier attempts and can be ignored. The red-X's show where we will remove components and the green show the clear path out of the power regulation circuits to the motherboard logic.



(OPTION BAD) KEEP 4116 MEMORY: To use the original 4116 memory we have to supply each power path with the voltage that it needs. However, there is a subtle shortcut that we can do for the -5V. DC-to-DC transformers are cheap, however we will get 12V and -12V not 12V and -5V. What can we do? The bottom path of the schematic above is a circuit to regulate -13.3V (my measurement yours will vary) out of the bridge rectifier (CR8) to -5V. What we will do is connect our -12V DC to the *(Neg Pt)* (the negative or right side of C1) to use part of the Model 1's power regulation to get us -5V. This works very well in practice and does not generate excessive heat because the regulation here is not active, its just a diode (CR2). In fact, CR2 is a 5% 5.1V Zener diode. Perfect.

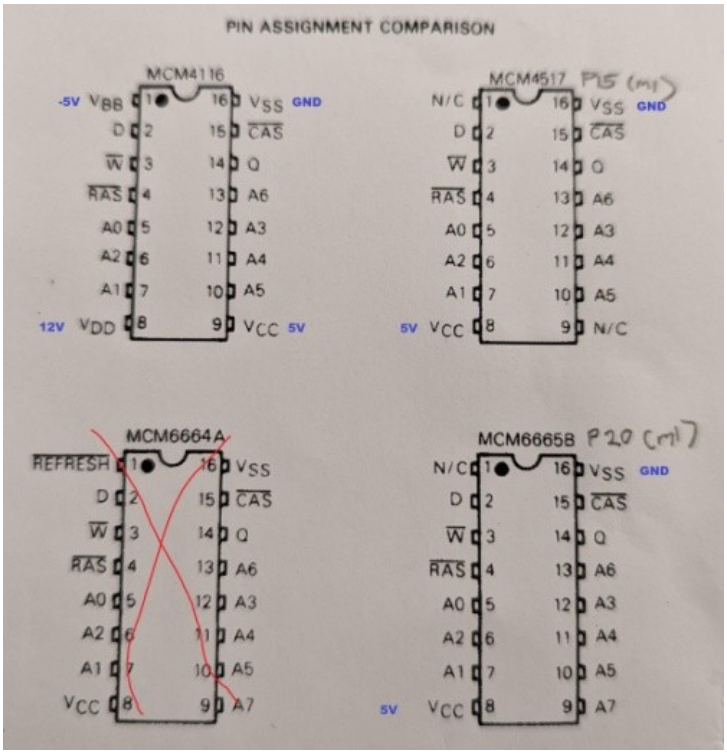
Given the DC-to-DC converter the rest is easy. 5V is hooked up to Z1 pin 3 and 12V is hooked up to Z2 pin 3. This makes all the power rails exactly like the original machine.

To summarize this option we

- Connect 5V DC power into the 5V power grid (the left hand side of R4 as presented below) and into the input of the DC-to-DC converter.
- Connect -12V out of the DC-to-DC converter to *(Neg Pt)* to get -5V. Connect 12V to Z2 pin 3.

The drawback of this approach is we have to include the DC-to-DC converter. I've also noted it brings out problems in the video sync circuit. The benefit is that we do not need to replace the memory chips.

(OPTION GOOD) USE 4517 OR 6665 MEMORY: If we swap out the memory we can make a 5V only system. This removes the need for the DC-to-DC converter, however, there is snag. If you look at the memory chip pinouts for the 4517 (16K) and the 6665 (64K using 16K) in the diagram below, in both cases, we have to get 5V on the lines where previously there was 12V. We can safely ignore the -5V lines.



To get 5V where the stock computer had 12V is easy. Simply connect Z1 pin 3 to Z1 pin 12 (connected to Z2 pin 3). This will channel 5V over to the (old) 12V lines and make them 5V as well.

To summarize this option we
• Change all the memory chips (Z13 through Z20) from 4116 to either 4517 (16K) and the 6665 (64K using 16K).
• Ignore the -5V path (bottom of the diagram above) it will not be used.
• Connect 5V DC power into the 5V power grid (the left hand side of R4 as presented below). Bridge Z1 pin 3 to Z1 pin 12 (connected to Z2 pin 3) to change the 12V path to 5V.

Note: If you already did the Rosser 64K in-the-keyboard modification some of the above is not need. That modification disconnected the 12V lines and bridged them to 5V. So this change will be easier as we discuss below.

Remove the DRAM chips

Start off by removing the socketed Z13-Z20 memory chips.
Regardless of option we want to test power before we put memory chips back into the machine.

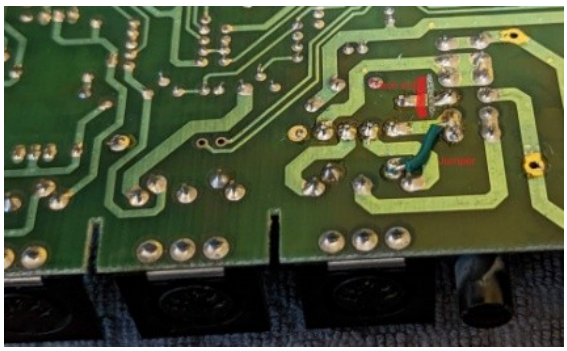
Removing components

We'll start by removing several components. Required for both options.
• Remove two DIP chips: Z1 and Z2
• Remove resistors: R4, R7, R8, R13, and R18
• Remove capacitor C8 entirely.
Unsolder and disconnect only the + side of C9 (near the power switch).
Remove things carefully and save them. Worst comes to worst you can reverse the modification. Below is a picture of what this should look like.



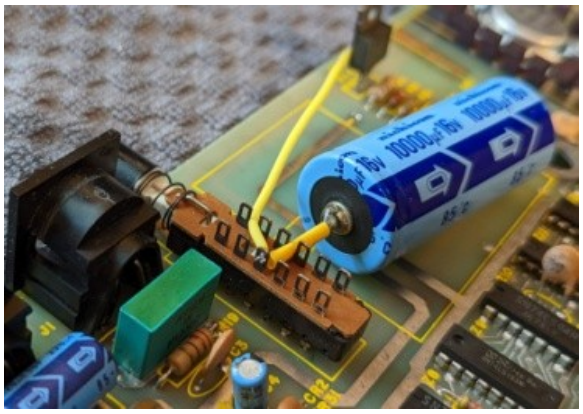
Setup the power switch

Flip the computer over to the back and focus near the power connector and power switch. We need to do a trace cut and solder a wire from pin 5 of the power connector to the right position on the switch. None of this is labelled so use the image below to guide you.



Use the continuity tester on your multimeter to ensure this is all correct. Check that the connection on pin 4 of the power connector is not shorted to the pin close to it. Also be sure your trace cut is not conducting. I used a Dremel tool to do the trace cut (just to be sure) but great care is required (practice this a few times if you try it).

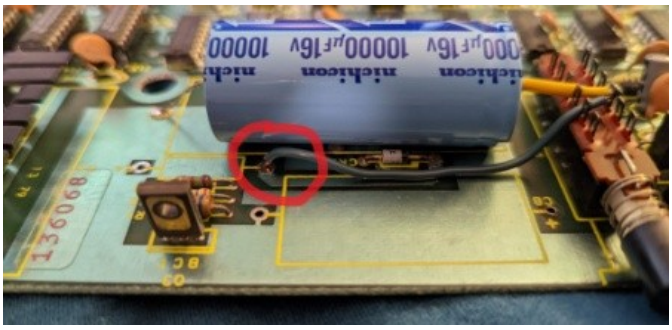
The trace cut isolates the power switch lead we will use to connect power to the motherboard. To do this turn the board around to show the chips. You need to position the positive lead of C9 to reach the third connection back on the left hand side of the power switch. Add a bit of shrink wrap to ensure the C9 lead doesn't connect with any other power switch terminals. We'll also insert a wire of about 4 inches that will connect to the motherboard. The picture below shows what this should look like prior to soldering.



I suggest you look over the leads on the switch. I've had to use some sandpaper to clean some of these off. If it looks black or dirty do this. We want a good solder connection. Then solder the two wires to the power switch.

At this point the procedure differs depending on which option you choose.

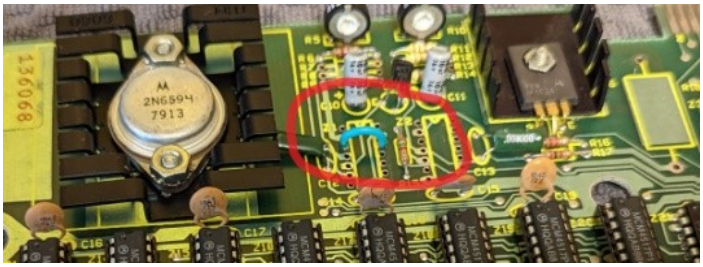
Completing (OPTION GOOD) Using 4517 or 6665 Memory



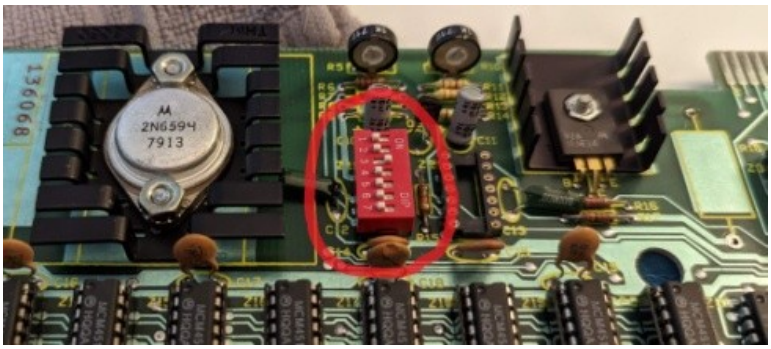
Trim and solder the wire connected to the power switch to the R4 hole closest to the power switch. This should look like the image below.

Note: Do not do the next step if you have Rosser 64K in-the-keyboard modification. The modification already did what the next step asks you to accomplish. If you have Rosser's modification then you are done.

Finally, we need to bridge the old 12V grid to the old 5V grid so everything is 5V DC. You can do this one of two ways. The first option is to simply put a short wire between Z1 pin 3 and Z1 pin 12 (straight across). This is shown in the image below.

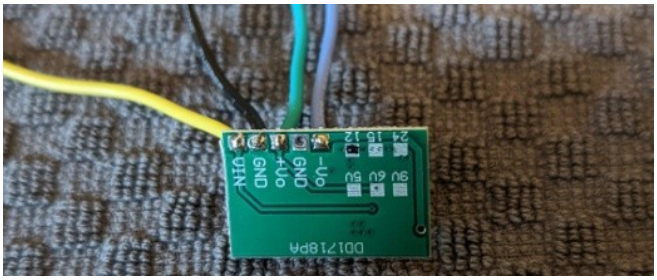


The second option is to solder sockets into Z1 (and optionally Z2) and use a 7 switch DIP. This is shown in the image below. Be sure to turn off all the switches except 3 (which is on).



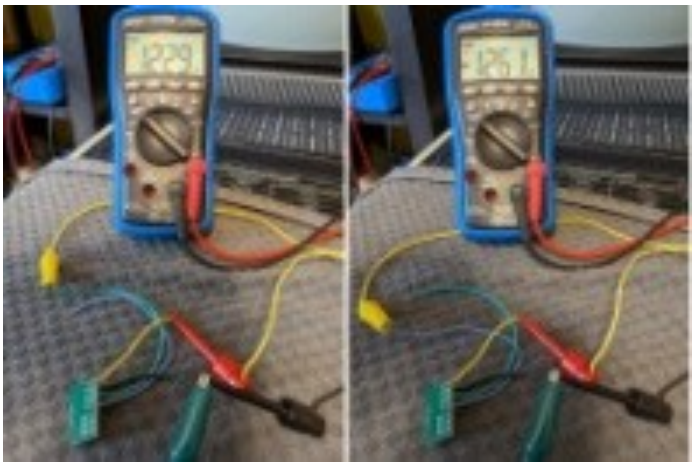
It really doesn't matter which you choose (I used the second when I was testing). The modification is complete (you can skip the next section).

Completing (OPTION BAD) Keep 4116 Memory

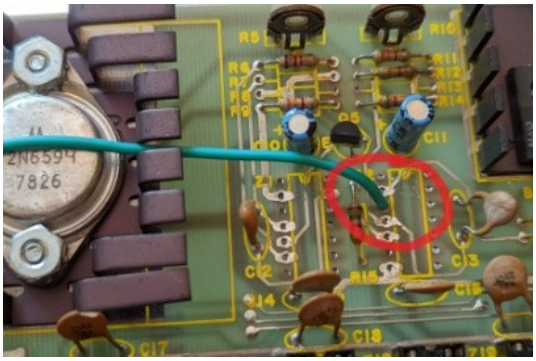


First we need to wire up the [+-12V DC-to-DC converter]. Check that ground in and out are common (be sure!).

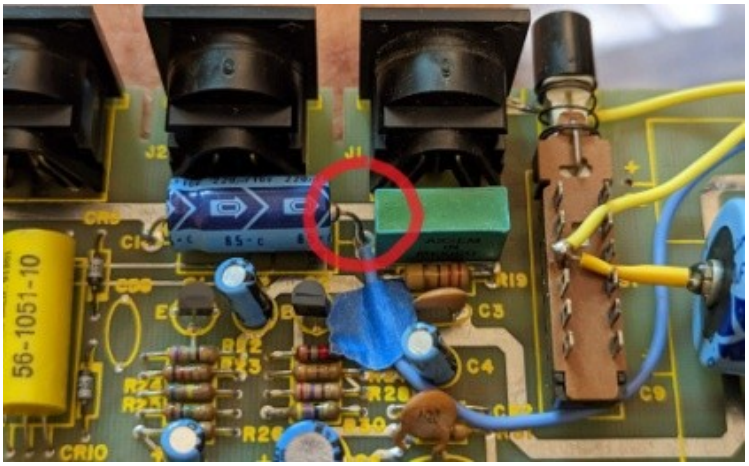
The output wires need to be longer than the VIN and GND wires. I strongly suggest you test this converter before installing (with a bench power supply or a custom USB cable). I've had these be mislabelled and give me, for example, +-9V rather than +-12V. You should get voltages like those shown in this image.



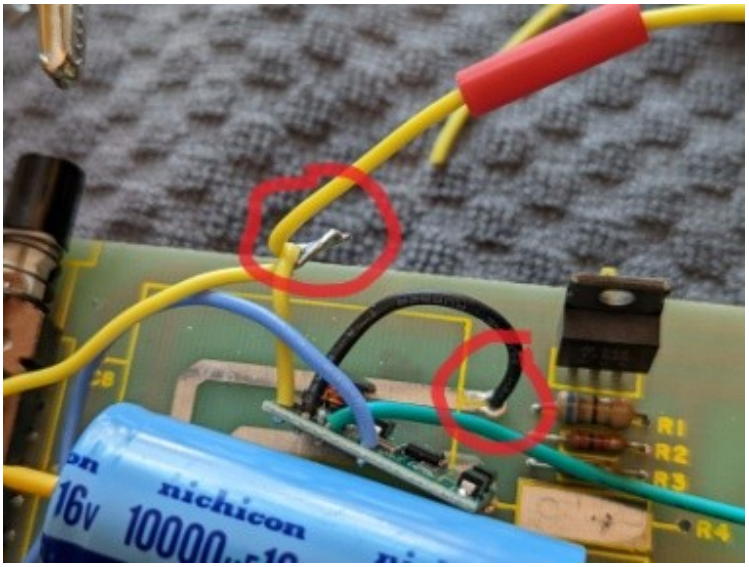
First solder the 12V wire from the converter to Z2 pin 3 as shown in the image below.



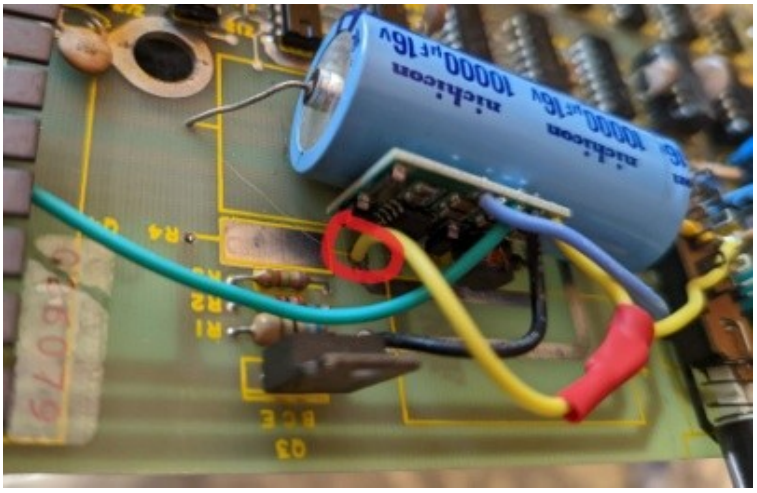
Next solder the -12V wire to the side of C1 closest to the power switch as shown in the image below. This is tricky to do. Tape down the wire so it stays in place.



Next solder the GND wire from the converter to the hole for C8 farthest from the power switch (see the image below). We also need to connect the wire from the power switch with the input wire from the converter to a new third wire. The third wire will connect to the motherboard. Use some shrink cover on this connection.



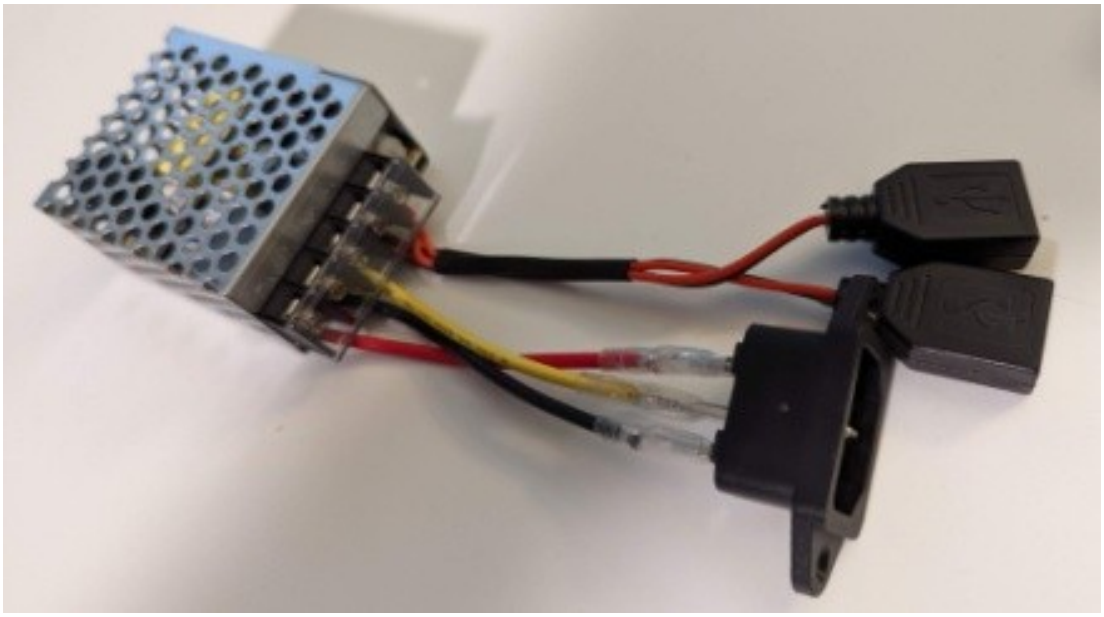
Trim and solder the new wire you just added (and soldered into a bundle) to the R4 hole closest to the power switch. This should look like the image below.



The modification is complete.

Setting up a MEAN WELL power supply

Before we test I want to show how to use a MEAN WELL RS-15-5 AC to DC power supply. This has very low ripple and is recommended. This is simple to setup. But you need a plug adapter and a female USB -A connector. My prototype looks like the image below.



You can create one as you wish. A few notes

- Mine uses USB-A female connectors because I want to be able to quick connect to either USB-A power or the MEAN WELL. This seems to be nice to have. But you could just connect the GND and 5V DC to the DIN power cable if you wish.

- I have two USB-A female connectors because I use this to power both a Model 1 keyboard unit as well as a prototype Expansion Interface (EI). This has the advantage that both have a common GND (helps stability). I'm not yet done testing the EI setup (but it is quite similar to OPTION BAD).
- This could clearly be packaged better but I haven't figured out how to do so.

If I haven't stressed this enough already. This is highly recommended, especially if you choose OPTION BAD.

I suggest adjusting the MEAN WELL output voltage to about 5.1V.

Testing your modification and putting the memory back

Ensure your power switch is off on the Model 1 computer. Hook up the power cable and connect it to a USB charger or a MEAN WELL. For all this testing I suggest you use a clip to connect the negative lead of your multimeter to the side of C9 not connected to the power switch (as we do on a stock Model 1).

(TEST 1: Any Power?) Before powering this on check the pin on the power switch one up toward where you press the switch. This should read around 5V DC.

(TEST 2: Motherboard Grid Okay?) Power on the machine. For OPTION GOOD test that pin 8 of any of the DRAM sockets (any of Z13 through Z20) is 5V DC. For OPTION BAD we need to test several pins of a socket (any of Z13 through Z20). Pin 1 should be -5V DC. Pin 8 should be 12V DC. Pin 9 should be 5V DC.

If any of these tests fails. Stop, something is wrong. Double-check the instructions and see if you shorted something with a solder connection.

If these tests pass shut off the computer and disconnect the power cable. Carefully insert your memory. On the bench connect the power cable, you monitor cable, and the keyboard. Turn on the monitor and then the computer. It should work as a Model 1 is expected to work. If you computer doesn't go. Some suggestions are

- Check voltages when the machine is turned on. Do they look okay? Troubleshoot to get power correct.
- If you did OPTION BAD and the screen is scrolling? Try adjusting the MEAN WELL voltage a bit. If this doesn't help you likely need to swap out Z5, Z6, and Z57 with new chips (as I noted up above). If you are using USB power (not a MEAN WELL) you may never get OPTION BAD to work. Last resort, convert to OPTION GOOD.

Use the *TRS-80 micro computer technical reference handbook* to isolate the problem.

You can reverse the modification (if you were careful with Z1 and Z2). Just socket Z1 and Z2, undo all the new stuff, and put all the old parts where they belong. Move the wire under the power switch to reconnect the trace cut. I've done this several times. The only real danger is that you destroyed Z1 and Z2 when you took them out.

(Optional) Marking the power modification more obvious

I have plugged in modified Model 1 computers with the wrong power source. To avoid this, the solution I came up with is to paint the power socket orange and put a *5V DC* sticker below the power socket.

My approach is in the image below.



Footer
© 2023 GitHub, Inc.
Footer navigation





PASCAL HOLDRY, IN FRANCE, IS LOOKING FOR THE CASSETTE SOFTWARE TO GO WITH THE RE-ENGINEERED TC-8

CAN YOU HELP? PLEASE EMAIL ME WITH DETAILS AND I'LL PASS THEM ON TO PASCAL.

I'M TRYING TO LOCATE A COPY OF THE PROGRAM "ENHANCED BASIC". THIS WAS AVAILABLE, IN THE U.K. FROM HARDINGS (MOLYMERX). IT NOT THE 'EXTENDED BASIC' OR M'SOFT'S LEVEL III BUT MUCH SMALLER AND NEATER, GIVING ALL LEVEL 2 PROGRAMERS MANY OF THE DISK BASIC COMMANDS. I'VE PUT OUT REQUESTS IN VARIOUS SITES, BUT, SO FAR, TO NO AVAIL! (*SEE ADVERT ON PAGE 91 ED*)

AS ALWAYS, OF COURSE, I'M ON THE LOOKOUT FOR ANY SOFTWARE FOR THE U.K.'S **ACULAB 'FLOPPY TAPE'**. THERE WERE VERSIONS SPECIFICALLY 'ZAPPED', FOR USE ON THE AFT, OF ELECTRIC PENCIL, SCRIPSIT, ZEN, LEVEL III BASIC AND VISICALC. THERE WAS ALSO THE ENHANCED BASIC, XBAS. IT'S SO DISAPPOINTING TO

AT THE READY PROMPT



THERE ISN'T A 'MOD' RESERVED WORD IN LEVEL 2 BASIC, SO PETER SHOWS US 2 WAYS TO FIND THE REMAINDER OF A DIVISION.

A MOD M IS A-M*INT(A/M) FOR REALS, ALTHOUGH IT _MIGHT_ BE WRONG BECAUSE OF MATH ROUNDING IN SOME CASES.
IF YOUR NUMBERS ARE SMALL AND THE MOD IS A POWER OF 2 MINUS 1 (LIKE 7 OR 255) YOU CAN USE AND: A AND M

PETER PHILLIPS

```
READY
>LIST
10 A=34:M=8
20 PRINTA-M*INT(A/M);
30 PRINT A AND (M-1)
READY
>RUN
 2  2
READY
>_
```

UNKNOWN LOADING A SYSTEM TAPE
ML ARNAUTOV

LOADING A SYSTEM TAPE, OF UNKNOWN NAME, CAN BE A PROBLEM. HAVING LOOKED AT THE LEVEL 2 ROM, I WORKED OUT THIS SOLUTION.

TO LOAD A SYSTEM TAPE OF UNKNOWN NAME, ON A MODEL 1 LEVEL 2, OR A VIDEO GENIE, RUN THE FOLLOWING PROGRAM.

```
10 FOR I=16924 TO 16932:READ J:POKE I,J: NEXT: END
20 DATA 49,136,66,205,147,2,195,231,2
```

NOW PREPARE THE TAPE AS YOU WOULD FOR A NORMAL LOAD. TYPE 'SYSTEM' AND REPLY TO THE PROMPT *? WITH /16924 INSTEAD OF THE PROGRAM NAME. THEN SIT BACK AND WATCH YOUR PROGRAM LOAD.

THIS SIMPLE SOLUTION RELIES ON THE FACT THAT THE STANDARD LOAD PROCEDURE ALLOWS PROGRAM NAMES TO BE ABBREVIATED DOWN TO A SINGLE CHARACTER. WHILE THE NUMBER OF POSSIBLE 6 CHARACTER NAMES IS LARGE, THE NUMBER OF CHARACTERS WITH WHICH THEY START IS NOT!

EXTRACTING A CASSETTE FILE NAME
CLIVE DAVIDSON

HERE'S A SIMPLE SOLUTION TO FIND THE NAME OF A CASSETTE PROGRAM. THIS SMALL BASIC PROGRAM WILL REVEAL ALL.

```
5 CLS
10 INPUT -1,A$
20 REM EXTRACT THE FILE NAME
30 B$=MID$(A$,2,(LEN(A$)-3))
40 PRINT"FILE NAME IS";B$
```

NOTE THE FILE NAME IS ALWAYS PRECEDED BUY 'U' AND '<'. THIS INFORMATION IS REMOVED BY LINE 30, TO GIVE THE ACTUAL FILE NAME.


Joe Kay
Director
NORTH COMPUTER WEST MUSEUM
Create Experience Learn
joe@nwcomputermuseum.org.uk
01942 582826
Information Technology facilities & training
Workshops for programming, Electronics & VR
Community classes, clubs & retro games
IT Skills for the young & old
City & Guilds Creditation
Supporting education of local people
Creating opportunities & careers
Disable friendly
Volunteer-led training & museum operations
NORTH WEST COMPUTER MUSEUM
Create Experience Learn
www.nwcomputermuseum.org.uk
Leigh Spinners Mills, Park Lane, Leigh, WN7 2LB

String Scanning Routines
Vince Otten

In this article, I show how I developed a general set of string scanning routines which look for not just a single character, like INSTR(CH$, SR$), but any member of a character set (like whitespace, alphanumerics, "+-*/", etc.)

I needed to be able to scan a line like "program -option1 filename" and pull out the individual "tokens", storing them in an array named AV$(),--like C's argv[]--and keeping a count in AC%--like C's argc :

```
AC%         2
AV$(0)      program
AV$(1)      -option1
AV$(2)      filename
```

("program" has 2 arguments, "-option1" and "filename".) In the process of creating that, I developed a nice little general-purpose string-scanning routine which
1. tests for or against a character set (like all digits, or whitespace, or "() +-"; and
2. remembers where in the string it left off scanning.
Assuming the string to be scanned was stored in SR$, my program was able to

```
50200 GOSUB 9000
```

and the first token would be returned in TK$. The entire loop is:

```
50200 GOSUB 9000
50210       IF SC% THEN
          AC%= AC% + 1
          AV$(AC%)= TK$
50220       IF MR% THEN
          GOTO 50200
```

We get the next token (GOSUB 9000). But what if TK$ doesn't contain a token? The "GetToken()" routine at line 9000 also makes sure to only set SC% ("Success") if TK$ isn't empty. So only IF GetToken() is Successful do we increment AC% in line 50210 and fill the corresponding AV$() with the token (AC% is initialized to -1 in line 50060, not shown). GetToken() also keeps track of where it's scanning, ready to look for the next token. If there's still "more string to scan", MR% will be set, so in line 50220 we loop back to ask for another token.

How does GetToken() know where to begin when it first starts scanning SR$? Whenever we assign something new to SR$, we'll need to call InitScanString() in line 6000, which sets up a number of "scan state variables". Here's a typical application of that:

```
50099 '
      PROMPT FOR TEST STRING:
            REJECT ANY EMPTY ""

50100 PRINT: PRINT "ENTER STRING, IE 'COMMAND LINE (PROGRAM
ARGS... )'"
50110      LINE INPUT SR$
50120      IF SR$ = "" THEN GOTO 50100

50149 '
      SET UP SCANNING VARIABLES FROM NEW SR$

50150 GOSUB 6000

50199 '
      SCAN & RETRIEVE TOKENS LOOP:

50200 GOSUB 9000
50210     IF SC% THEN
                AC%= AC% + 1:
               AV$(AC%)= TK$
               ' SUCCESS: SAVE TOKEN IN AV$()
50220     IF MR% THEN
               GOTO 50200
               ' MORE TOKENS TO SCAN
```

My scanning routines don't do a lot of error-checking, so here I make sure SR$ isn't empty before I begin to use them. Then I call line 6000 to initialize the "Scan State Variables", which keep track of where we are currently scanning from in SR$, what the length of SR$ is, etc. We'll look at InitScanString() / 6000 later on.
Here's what GetToken() at line 9000 looks like.

```
7999 '
      2. NEXT TOKEN -- SKIP WS, RETURN TOKEN AFTER TOKEN
                        UNTIL MR% IS F
9000 '
9010 TK$= """' START WITH EMPTY TOKEN$
9020 SC%= 0' "RESULT" (IE, SUCCESS/FAILURE)
9100 GOSUB 7000' SKIP WS
9110 IF NOT MR% THEN GOTO 9500
9120   GOSUB 5500' "GATHER UNTIL" CS$ (WS)
```

```
9130   IF SC% THEN
           TK$= MID$( SR$, CC%, SL%)
9500 GOSUB 5900' UPDATE CC%= NC%
9990 RETURN
```

We start with a blank TK$ (line 9010), and assume we're "not successful" (9020). Then we call SkipWS() / line 7000 to move our search start point in SR$ (the "current cursor", CC%) over past any "whitespace" (blanks, carriage returns, linefeeds, stored in the scan's "character set", CS$). SkipWS() is kind enough to let us know whether there's something other than whitespace remaining in SR$ by setting the "more" flag, MR% (9110).

If there's more to scan, we GatherUntil() / 5500, ie, until we run into more whitespace. This routine reports the "selection length" SL% of the substring starting at CC% that does not match the whitespace in CS$, as well as providing a "succeed/fail" flag in SC%. Only if it's succeeded in identifying a non-whitespace token do we actually return that as TK$ (9130).

In any case, we UpdateCursor() / line 5900 to move the current cursor CC% to where the "new cursor" NC% is, just past the end of the token (9500).

SkipWS() is implemented in two steps: specify the character set to search for (space, linefeed, or carriage return); and SkipWhile() (line 5200) that set is matched.

```
6999 '
     1. SKIP WS

7000 CS$= " " + CHR$( 10) + CHR$( 13) ' WHITESPACE CHARACTERS
7050 GOSUB 5200' "SKIP WHILE" OR "IGNORE" WS
7090 RETURN
```

By now we've seen two examples of a set of four generalized scanning options:

SkipWhile()	line	5200	scan while matching a member of the character set
SkipUntil()		5300	scan until a member of the set is matched
GatherWhile()		5400	identify substring that matches the set
GatherUntil()		5500	identify substring until a member of the set is matched

"Skips" simply "move the cursor", whereas "Gathers" identify a span of 1 or more characters in SR$. Examples of how each could be used:

SkipWhile()
moving past all whitespace, as in our program here.
SkipUntil()
in Pascal, a comment begins with a { . We can skip comment contents by using SkipWhile() to look for the matching } .
GatherWhile()
Get the next variable name by setting CS$ to "alphanumerics" (a collection of all uppercase & lowercase letter + all digits)
GatherUntil()
Collecting a substring until whitespace, as in our program here.

SkipWhile() is coded :

```
5199 '
      "SKIP WHILE" ("IGNORE") --
            SKIP UNTIL NOT IN CS$, UPDATE CC%, MR%

5200 NF%= 0' "MATCH" CHAR SET
5210 GOSUB 5000' SCAN
5220 GOSUB 5900' UPDATE CC%
5230 GOSUB 6900' DETERMINE IF THERE'S "MORE" TO SCAN
5290 RETURN
```

NF% is the "Not Flag". If it's set (to -1), then the main GeneralScan() (5000) routine understands to not match what's in CS$; otherwise, we match at least 1 member of CS$, as in this demo. Both "Skips" will call UpdateCC() (5900) to "move" the current cursor CC% to where GeneralScan() has placed the new cursor NC% (even if NC% hasn't changed). Finally, we call IsThereMoreToScan() (6900) to signal whoever's called SkipWhile() that there may or may not be more information in SR$.

SkipUntil() isalmost exactly the same: it does set NF%, and then simply jumps to line 5210, above.

```
5299 '
      "SKIP UNTIL" ("FIND") --
            FIND 1ST MATCH W/ CS$, UPDATE CC%, MR%

5300 NF%= -1' "DON'T MATCH" CHAR SET
5310 GOTO 5210' SCAN; UPDATE CC%; DETERMINE IF "MORE" TO SCAN
```

Before addressing GeneralScan(), here's UpdateCC():

```
5899 '
      UPDATE CC%= NC%

5900 CC%= NC%: RETURN
```

Why not simply type CC%= NC% whenever we need to "update CC%"? It's not as though using GOSUB 5900 saves any memory—always a consideration on a TRS-80. But it saves having to memorize the steps, and it avoids misprints, like accidentally typing CC= NC%.
We also need to

```
6899 '
       DETERMINE IF THERE'S "MORE" TO SCAN

6900 MR%= CC% < LN%' IF WE'RE NOT AT END OF SR$, THEN "MORE"
6920 IF NOT MR% THEN
           MR%= NOT FN IC%( MID$( SR$, CC%, 1), CS$)
           ' IF CHAR AT CC% NOT IN CS$, THEN "MORE"
6990 RETURN
```

This is a bit more complicated. First, we check if the current cursor CC% has been moved all the way to the position of the last character in SR$, ie the "length" LN% of SR$. If it's not, then obviously we have more to scan, so MR% will be set to -1, "true". On the other hand, say SR$ was " A" (5 spaces and an 'A'). Once we'd skipped all the whitespace and updated CC%, it would point to the last character in SR$, in this case the 'A', which is not in CS$, and so may need to be scanned later. This is what line 6920 tests for.
The defined function
FN IC%(MID$(SR$, CC%, 1), CS$)
returns "true" (-1) if the character in SR$ at position CC% is in CS$. In our example here, CC% is the last character position, and its value is 'A', which is not in CS$ (currently containing whitespace). Line 6920 inverts this result (NOT FN IC%...) so that it returns "true", and sets MR% to -1. (The definition of FN IC% is given down below.)

There's also GatherWhile() and GatherUntil.

```
5399 '
       "GATHER WHILE" -- SR$(CC%..NC%) MATCHES SET CS$

5400 NF%= 0' "MATCH" CHAR SET
5410 GOSUB 5000' SCAN
5440 SC%= SL% > 0' IF SELECTION LENGTH SL% > 0, "SUCCESS"!
5490 RETURN
```

The two Gather()s are like simpler versions of the Skip()s. Like those, they set NF% (lines 5400 / 5500) and call GeneralScan() (5000). But they don't update CC% or MR%, because they're not "moving the cursor".

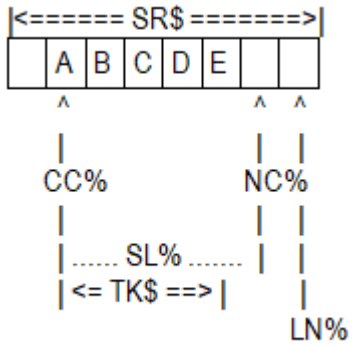
(Any code which uses a Gather() must manually update these after making use of the results: see line 9500 in GetToken() above, for example.) Instead, they report success/failure in SC% based on what the length of the "selection" SL% is. SL% gets set in GeneralScan() and is the difference between CC% and the new cursor position NC%. (If nothing was "gathered", NC% is the same as CC%, and SL% will be "false", 0.)

```
5499 '
      "GATHER UNTIL" -- SR$(CC%..NC%) UP TO CHAR IN SET CS$

5500 NF%= -1
5510 GOTO 5410' SCAN, REPORT "SUCCESS/FAILURE"
```

By this point, we have up to 2 "results":
MR% true (-1) if there's "more to scan"
SC% true (-1) if we "successfully" gathered a substring
and 4 "Scan State Variables" which tell us about the scan in progress:
LN% the length of SR$
CC% the "current cursor", where our scan currently begins within SR$
NC% the "new cursor" position after a scan
SL% the "selection length", or length of the substring that's been identified; SL%= NC% - CC%

As an example, GetToken() might be in the midst of processing the string " ABCDE ". It's already called SkipWS() and GatherUntil(). Here's what the Scan State Variables look like:



Now that we can picture them all, it`s time to revisit InitScanString() (6000), which is called whenever a new value is assigned to SR$ (for example, line 50150 in our Demo Program, above).

```
5999 '
      SET UP SCAN STATE VARIABLES AFTER NEW SOURCE$

6000 LN%= LEN( SR$): UT%= LN%: MR%= LN% > 0
6010 NC%= 0
6020 IF MR% THEN
        NC%= 1
6030 GOSUB 5900' CC%= NC%
6090 RETURN
```

(Note that these routines do not do a lot of bounds-checking: if SR$ is empty (ie, ""), line 6000 will crash. But then, if you write a program line that says

100 PRINT 5 / 0

it will crash, too: tradeoffs are always being made.)
A new Scan State Variable is introduced: "Up To", UT% defines how far along in SR$ the scan is allowed to progress, and it defaults to the end of the string. (It's possible to do scans on subsections of SR$ by resetting CC% and UT% to different values after InitScanString() is called.) Why do we even test for LN% > 0? This makes it easy to add bounds-checking on the length of SR$ if needed, and it doesn't cost much in processor time because it's only called once for every new string assignment. So long as SR$ is not empty, MR% will be "true" / -1 (there's "more to scan"), NC% will be set to the first character position, and by calling UpdateCC() (5900), we set CC% to character position 1 as well.

Each of the Skips and Gathers call a common GeneralScan() routine:

```
4999 '
     SET UP, CALL SCAN, REPORT RESULTS

5000 GOSUB 5100 ' SCAN SR$
5020 NC%= CC% + SL% ' UPDATE "NEW CURSOR"
5090 RETURN
```

A BasicScan() (5100) is performed, and we update the "new cursor", NC%.

Here's the engine of the entire string-scanning operation, BasicScan():

```
5099 '
     BASIC SCANNING ROUTINE

5100 SL%= 0
5105 FOR C%= CC% TO UT%:
     IF NOT FN XR%(  NF%,
               FN IC%( MID$( SR$, C%, 1), CS$)) THEN
                    RETURN
5110    SL%= SL% + 1: NEXT
5190 RETURN
```

First, the "selection length" SL% is initialized to 0. Next, index C% loops from the "current cursor" CC% all the way to the "up to" marker UT%, by default the end of SR$ (but BasicScan() or GeneralScan() can be called with different value for UT%).

Line 5105 should be deciphered from the "inside out". First of all, we're going to examine one character in SR$, at position C%.

MID$(SR$, C%, 1)

That single character is tested by FN IC%() to see if it's in the character set CS$ (in our demo here, the set of whitespace characters we set up in SkipWS() earlier).

FN IC%(MID$(SR$, C%, 1), CS$))

If it is in that set, FN IC%() will return "true" (-1), and our scan is "satisfied". But there's another condition to consider: the "do NOT match" Flag NF%. If NF% is set (-1), and FN IC%() reports that the single character was found in the set CS$ (-1), ie a "match", then the scan at this point is "not satisfied": "you told me not to find whitespace, but FN IC%() says this character is whitespace, so I'm not satisfied." In other words, NF% should invert the result of FN IC%().
Here's a "truth table" that illustrates what we want to see happen, with the first line depicting the example above. In line 2, NF% still requires that we "don't match" the character set, and FN IC%() replies, "Right: we didn't find it in CS$", so the scan is satisfied. Line 3: NF% now requires a match, and FN IC%() replies that it did indeed find one, so the scan is satisfied again. Finally, if NF% is not set and FN IC%() didn't find a match, the scan is also not satisfied.

"Do Not Match" flag NF%	Single Character is found in set CS$: Result of FN IC%()	Scan Satisfied?
-1	-1	
-1	0	
0	-1	
-1	0	
0	-1	
-1	0	
-1	-1	
0	0	
0	0	
0	0	

Another way to describe this is that if NF% is set (-1), the scan result is the opposite or the "inversion" of FN IC%(); but if NF% is not set (0), the result is just the whatever FN IC%() reports.
We could have coded this as:

```
5107    IF NF% THEN
        IF FN IC%(...) THEN
            RETURN
5108    IF NOT FN IC%() THEN
        RETURN
```

Instead, we're using XOR ("exclusive OR"), which only results in true (-1) if its two arguments don't match.

Argument 1	Argument 2	XOR result
-1	-1	0
-1	0	-1
0	-1	-1
0	0	0

This is the effect of the defined function FN XR%():

```
FN XR%(     NF%,
        FN IC%( MID$( SR$, C%, 1), CS$))
    )
```

The result of FN XR%() here tells us if the scan is "satisfied", which is not the same as "matching". For example, if we want to SkipWS() (see above), we leave "argument 1", NF%, reset (0). If FN IC%() finds that the character at position C% matches a member of the set CS$, then "argument 2" is true (-1), and the XOR result is "satisfied" (-1).
However, if we want to GatherUntil() (see above), we set argument 1, NF%, as -1, "don't match", and if the character at position C% does not match a member of the set CS$ (whitespace in our demo program), then argument 2 is 0, and the XOR result is "true" (-1), again "satisfied".
If FN XR%() is satisfied, then the "selection length" SL% gets incremented (line 5110), and we go on to test the next character (NEXT). If it's not, we immediately leave this subroutine: it's done its work. Therefore, the full test to RETURN in the second part of line 5105 is:

```
IF NOT FN XR%(  NF%,
            FN IC%( MID$( SR$, C%, 1), CS$)) THEN
                RETURN
```

All that BasicScan() does, then, is to process SR$ one character at a time, from position CC% to UT%, incrementing SL% each time the XOR test is satisfied, or exiting as soon as it's not.
Here's the function definitions for FN IC%() and FN XR%():

```
40010 DEF FN XR%( A%, B%)= (A% OR B%) - (A% AND B%)' XOR
( A%, B%)
40030 DEF FN IC%( CH$, ST$)= INSTR( ST$, CH$) > 0' T IF CH$ IN SET
ST$
```

The demo program "Main()" in line 50000 calls the subroutines and functions described here like this:

```
Main() / 50000
      DefineFunctions() / 40000
      InitScanString() / 6000
      GetToken() / 9000
            SkipWS() / 7000
                  SkipWhile() / 5200
                        GeneralScan() / 5000
                              BasicScan() / 5100
                                    FN IC%()
                                    FN XR%()
                              UpdateCursor() / 5900
                              IsThereMoreToScan() / 6900
            GatherUntil() / 5500
                  GeneralScan() / 5000
                              BasicScan() / 5100
                                    FN IC%()
                                    FN XR%()
                              UpdateCursor() / 5900
                              IsThereMoreToScan() / 6900
            UpdateCursor() / 5900
```

and here's the entire demo program listing:

```
10 GOTO 50000'

      SCANARTC/BAS -- STRING SCANNING DEMO
                  JAN 31, 2023

4999 '
      GENERAL SCAN

5000 GOSUB 5100' SCAN SR$
5020 NC%= CC% + SL% ' UPDATE "NEW CURSOR"
```

```
5090 RETURN
5099 '
     BASIC SCANNING ROUTINE

5100 SL%= 0
5105 FOR C%= CC% TO UT%:
     IF NOT FN XR%( NF%, FN IC%( MID$( SR$, C%, 1), CS$)) THEN
          RETURN
5110   SL%= SL% + 1: NEXT
5190 RETURN
5199 '
     "SKIP WHILE" ("IGNORE") --
          SKIP UNTIL NOT IN CS$, UPDATE CC%, MR%

5200 NF%= 0' "MATCH" CHAR SET
5210 GOSUB 5000' SCAN
5220 GOSUB 5900' UPDATE CC%
5230 GOSUB 6900' DETERMINE IF THERE'S "MORE" TO SCAN
5290 RETURN
5299 '
     "SKIP UNTIL" ("FIND") --
          FIND 1ST MATCH W/ CS$, UPDATE CC%, MR%

5300 NF%= -1' "DON'T MATCH" CHAR SET
5310 GOTO 5210' SCAN; UPDATE CC%; DETERMINE IF "MORE" TO SCAN
5399 '
     "GATHER WHILE" -- SR$(CC%..NC%) MATCHES SET CS$

5400 NF%= 0' "MATCH" CHAR SET
5410 GOSUB 5000' SCAN
5440 SC%= SL% > 0' IF SELECTION LENGTH SL% > 0, "SUCCESS"!
5490 RETURN
5499 '
     "GATHER UNTIL" -- SR$(CC%..NC%) UP TO CHAR IN SET CS$

5500 NF%= -1
5510 GOTO 5410' SCAN, REPORT "SUCCESS/FAILURE"
5899 '
     UPDATE CC%= NC%

5900 CC%= NC%: RETURN
5999 '
     SET UP SCAN STATE VARIABLES AFTER NEW SOURCE$

6000 LN%= LEN( SR$): UT%= LN%: MR%= LN% > 0
6010 NC%= 0
6020 IF MR% THEN
```

```
      NC%= 1
6030 GOSUB 5900' CC%= NC%
6090 RETURN
6899 '
      DETERMINE IF THERE'S "MORE" TO SCAN

6900 MR%= CC% < LN%' IF WE'RE NOT AT END OF SR$, THEN "MORE"
6920 IF NOT MR% THEN
      MR%= NOT FN IC%( MID$( SR$, CC%, 1), CS$)
      ' IF CHAR AT CC% NOT IN CS$, THEN "MORE"
6990 RETURN
6998 '
      SOME USEFUL SCANNING ROUTINES:

6999 '
      1. SKIP WS

7000 CS$= " " + CHR$( 10) + CHR$( 13) ' WHITESPACE CHARACTERS
7050 GOSUB 5200' "SKIP WHILE" OR "IGNORE" WS
7090 RETURN
7999 '
      2. NEXT TOKEN -- SKIP WS, RETURN TOKEN AFTER TOKEN
                        UNTIL MR% IS F

9000 '
9010 TK$= ""' START WITH EMPTY TOKEN$
9020 SC%= 0' "RESULT" (IE, SUCCESS/FAILURE)
9100 GOSUB 7000' SKIP WS
9110 IF NOT MR% THEN GOTO 9500
9120   GOSUB 5500' "GATHER UNTIL" CS$ (WS)
9130   IF SC% THEN
            TK$= MID$( SR$, CC%, SL%)
9500 GOSUB 5900' UPDATE CC%= NC%
9990 RETURN
40000 '
      INITIALIZATION

40010 DEF FN XR%( A%, B%)= (A% OR B%) - (A% AND B%)' XOR
( A%, B%)
40030 DEF FN IC%( CH$, ST$)= INSTR( ST$, CH$) > 0' T IF CH$ IN SET
ST$
40090 RETURN
49999 '
      *** STRING SCANNING DEMO
      ***     MAIN PROGRAM

50000 CLEAR 500
```

```
50010 CLS: PRINT "STRING SCANNING DEMO:"
50020 PRINT "PARSE 'COMMAND LINE' INTO AV$(),"
50030 PRINT "TOTALLING ARGUMENTS IN AC%"
50049 '
      INITIALIZE:

50050 GOSUB 40000
50059 '
      INITIALIZE "ARGC" / AC% TO INDICATE
            "NO TOKENS YET" / -1

50060 AC%= -1
50070 SR$= ""
50099 '
      PROMPT FOR TEST STRING:
            REJECT ANY EMPTY ""

50100 PRINT: PRINT "ENTER STRING, IE 'COMMAND LINE (PROGRAM
ARGS... )'"
50110     LINE INPUT SR$
50120     IF SR$ = "" THEN GOTO 50100
50149 '
      SET UP SCANNING VARIABLES FROM NEW SR$

50150 GOSUB 6000
50199 '
      SCAN & RETRIEVE TOKENS LOOP:

50200 GOSUB 9000
50210     IF SC% THEN
            AC%= AC% + 1:
            AV$(AC%)= TK$
            ' SUCCESS: SAVE TOKEN IN AV$()
50220     IF MR% THEN
            GOTO 50200
            ' MORE TOKENS TO SCAN
50299 '
      IF ANY TOKENS GATHERED, PRINT THEM OUT:

50300 IF AC% < 0 THEN
            PRINT "NO TOKENS SCANNED":
            GOTO 51000
50309 '
        HEADER

50310     PRINT
50320     PRINT "AC%:"; AC%
```

```
50330     PRINT "TOKEN #", "TOKEN"
50340     PRINT "-------", "-----"
50399 '
          PRINT OUT ALL TOKENS

50400     FOR T%= 0 TO AC%
50410          PRINT T%, AV$(T%)
50420          NEXT
50999 '
          DONE

51000 PRINT
51010 GOTO 50060
```

Some notes on the listing (for the curious):

Why immediately GOTO 50000? This is my usual BASIC program layout. It's kind of like a Pascal program (remember those?)--declarations, functions and procedures first, then the Main() program. In the case of TRS-80 BASIC, I like to list the most-used subroutines first, because the interpreter will search for and find those lines the quickest.

Having said that, is this the quickest implementation? Hardly. Here's a quicker version of GeneralScan() and BasicScan() (lines 5000..5190) combined:

```
5000 SL%=0:FORC%=CC%TOUT%:IFFNXR%(NF%,INSTR
(CS$,MID$(SR$,C%,1))>0)THENSL%=SL%+1:NEXT
5002 NC%=CC%+SL%:MR%=NC%<=LEN(SR$):RETURN
```

Why did I develop these routines? Firstly, for the fun of it. Secondly, to prototype routines I hope eventually to write in Z-80 Assembler.

Jens Guenther has updated his fork of the SDLTRS Emulator to version 1.2.26. Changelog includes:

- Added Sector Size Support for WD1000/1010
- Added Hard Disks (WD) for EACA EG 3200 Genie III and TCS Genie IIIs
- Added Floppy Disk controller (Expansion Interface) option
- Added debug command to modify the ROM
- Fixed Aster CT-80 charset in TRS-80 mode
- Fixed TRS-80 Model 4/4P Video Page

You can download the most recent SDLTRS II from the Gitlab page: https://gitlab.com/jengun/sdltrs.

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



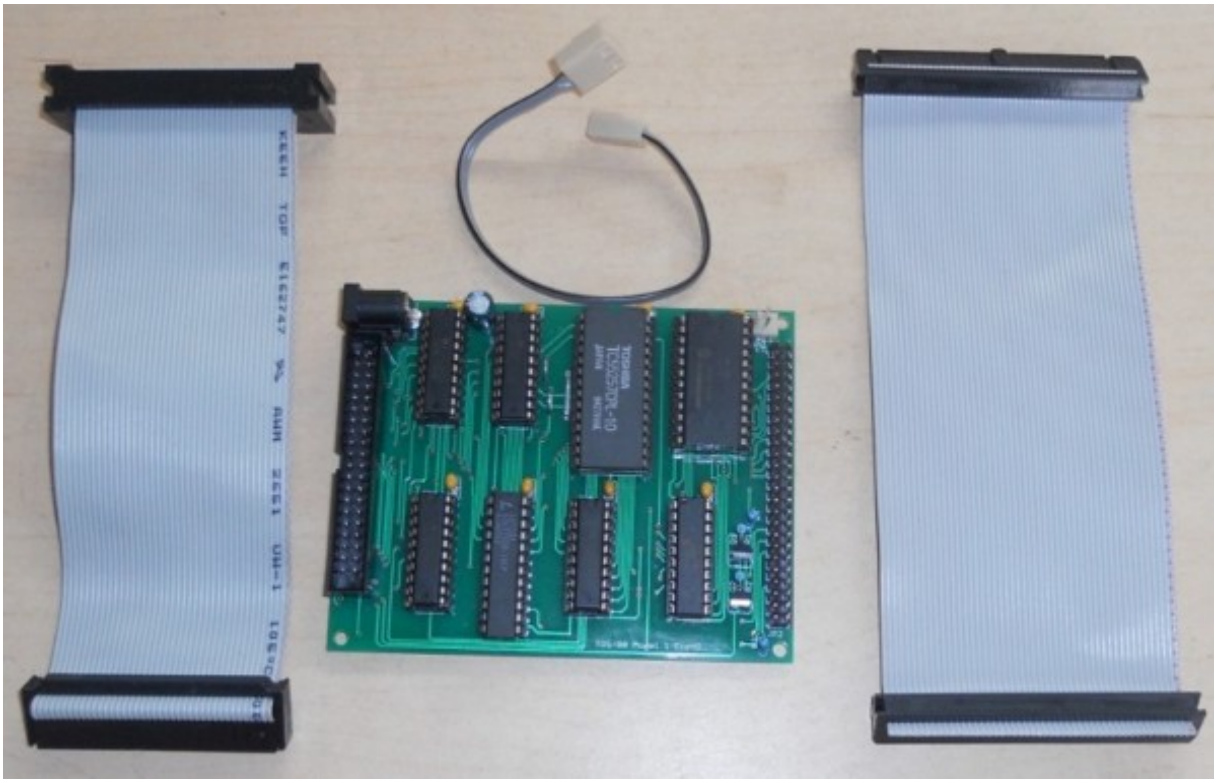
* Uses modern day MEAN WELL power supply units
* 50w for Model 3 version and 65w for the Model 4 version
* The Model 4 version uses a SEPARATE +12V POWER SUPPLY (prevents video "spasm")
* No more RIFA CAP explosions
* No more COLD solder joints
* Units are DIRECT REPLACEMENTS for the original power supplies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

newSOFT

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

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

- COMPLETLY ASSEMBLED AND TESTED -



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

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

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

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

The amazing

"FreHD"



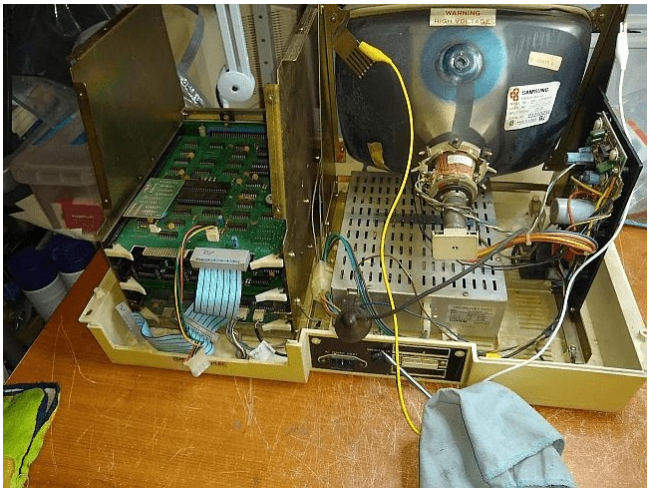
-Emulates a TRS-80 hard drive, but faster than any hard drive!
-Works with your favourite DOS (LS-DOS, LDOS, CP/M, Newdos/80 2.5)
-Uses SD card for storage medium
-Bonus free Real Time Clock function!
-Designed in Belgium and proudly built and shipped from Australia
-Kit form or fully assembled

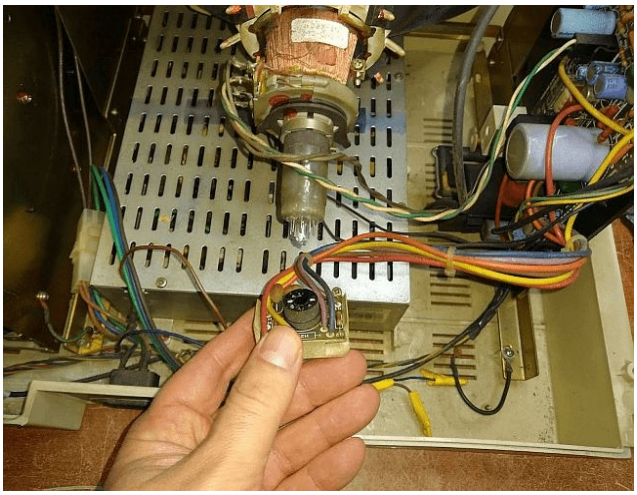
Order yours today
http://members.iinet.net.au/~ianmav/trs80/

HERE'S A SIGHT FOR SORE EYES!
THIS IS **KRIS GARREIN'S** VIDEO GENIE III
AFTER A LOT OF WORK













NEWS FROM FRANCE

Liberté • Égalité • Fraternité

PASCAL HOLDRY

Bonjour Dusty, Bonjour à tous,

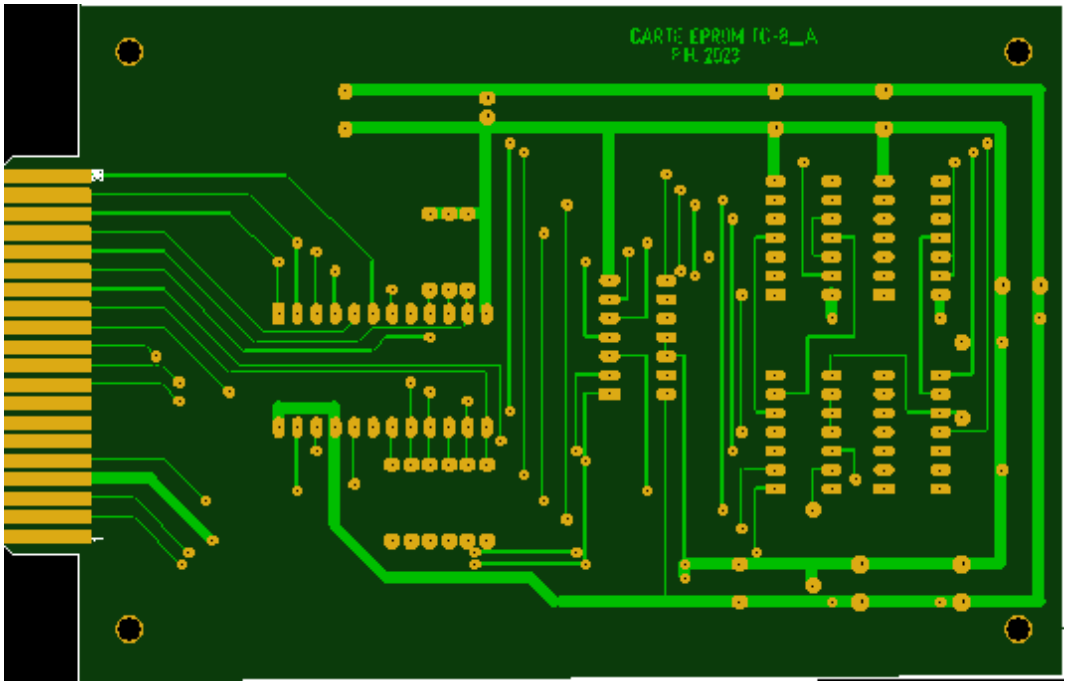
I just made the clone card of the TC-8 card which allows to read the cassettes with X5 speed. The TC-8 has almost the same functions as the "EXATRON", but with traditional audio cassettes.

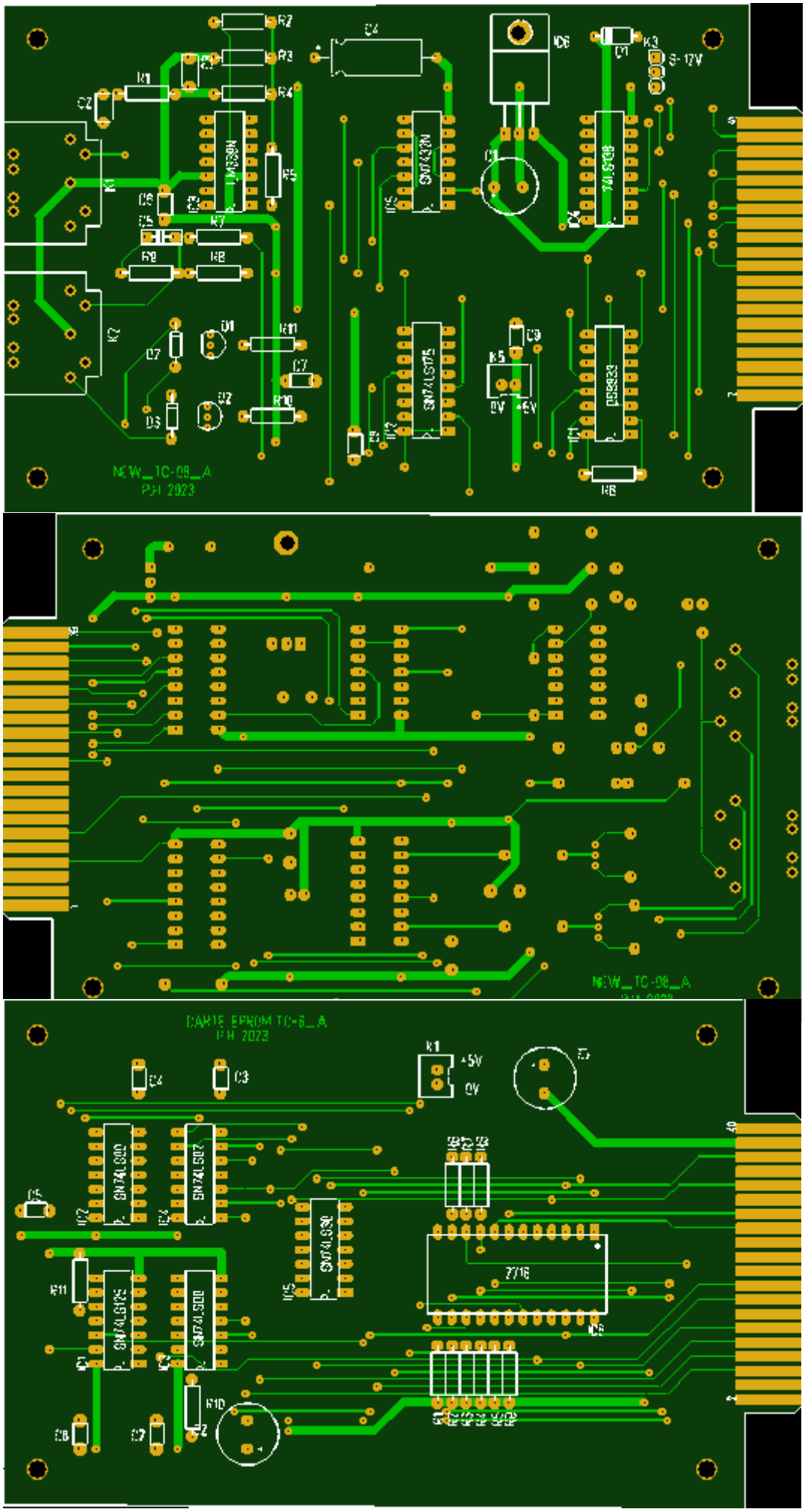
This TC-8 was lent to me by Stephen. The TC-8 seems to have been made in Great Britain. Hans had sent me a TC-8 clone a few years ago This TC-8 has been tinkered with an additional card with an Eprom.

Unfortunately I don't have the cassette sold with the TC-8 box, I'm looking for this cassette.
Maybe the Eprom allows to use part of the TC-8, but I would like to have a copy of the original program

Dusty can you help to find a copy of the cassette?

Sure can Pascal, if anyone has a copy of the necessary TC-8 software, please contact me and I'll pass on your details to Pascal.




NEW_TC-08_A
PH 2023
NEW_TC-08_A
CARTE EPROM TC-8_A
PH 2023
+5V
0V
Z716

"VOXBOX"

Voice Recognition Unit for TRS-80 Model 1

Pascal Holdry

The TRS-80 VOXBOX voice recognition unit allows you to take advantage of this new technology on your Level II TRS-80 computer. You can now use words and phrases to control and instruct your computer and to enter data. Focus on the work or game at your fingertips, like watching video, without the distraction of typing on the keyboard.

- Simple to use, connects directly to TRS-80 or TRS-80 Bus Card Edge expansion interface.
- Includes a dynamic push-to-talk microphone with coiled cord for communication with the VOXBOX.
- Includes a machine language "driver" program and three application programs.
- Using the examples and instructions in this manual, you can write your own programs for custom applications.
- Use the learning mode to teach the VOXBOX up to 32 words or phrases, each word/phrase lasting up to 1.2 seconds.
-

Re-train it at any time to recognize up to 32 different words in any language!

Note: Speech recognition is a new technology. In fact, your Radio Shack VOXBOX is one of the first devices of its kind to be both available and affordable to the general public.

For this reason, Radio Shack recommends that the device be used primarily for entertainment and experimentation. Proceed with caution before engaging the unit in serious application.

Equipment required

TRS-80 with 16K RAM
Tape recorder
Optional expansion interface

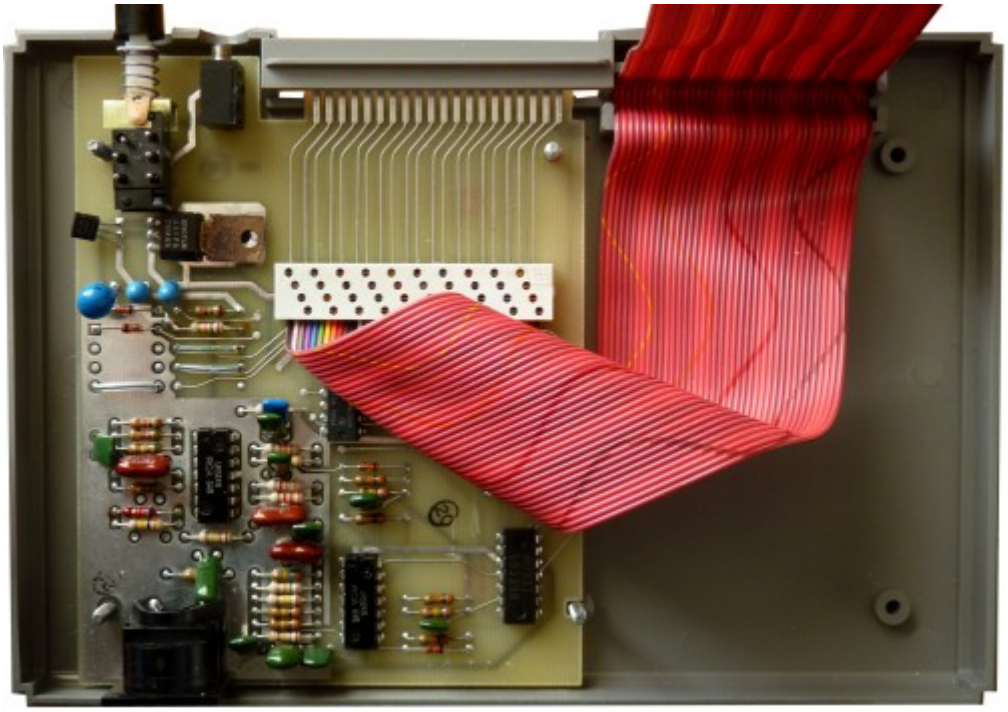
This box was given to me by Ian Mavric, whom I sincerely thank.

View with the microphone



As you can see the case does not completely close, Tandy had forgotten the thickness of the "PCB" when creating the molds to build its case

View of the inside of the box



You can see that the bottom right screw is not included in its entirety in the "PCB", a notch in the "PCB" replaces a drill hole.

View of the back of the box



On this photo you can better see the design error of where Tandy had forgotten the thickness of the "PCB" when creating the molds to build this case.

RE-VOX_BOX

by Pascal Holdry

Aim:

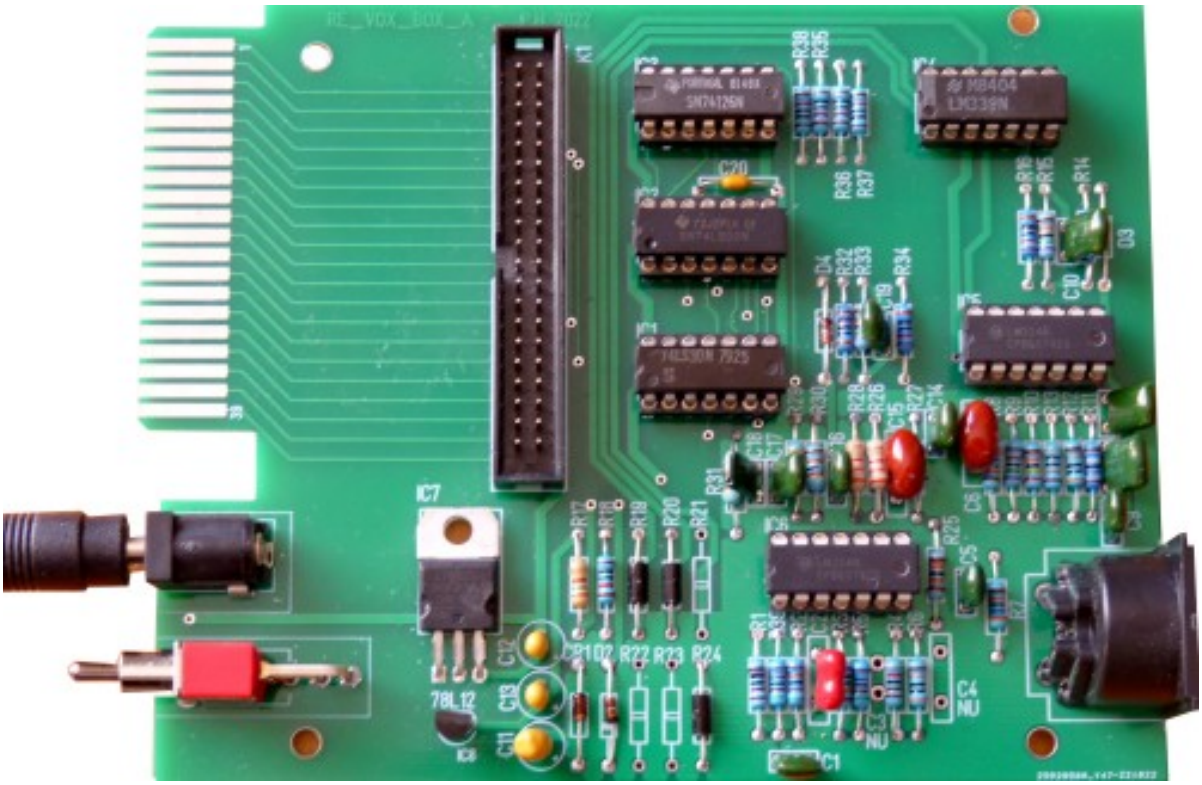
1. Being able to recover existing software
2. Keep track of our digital heritage
3. Provide a schematic for troubleshooting and understanding how the board works.

Specifications:

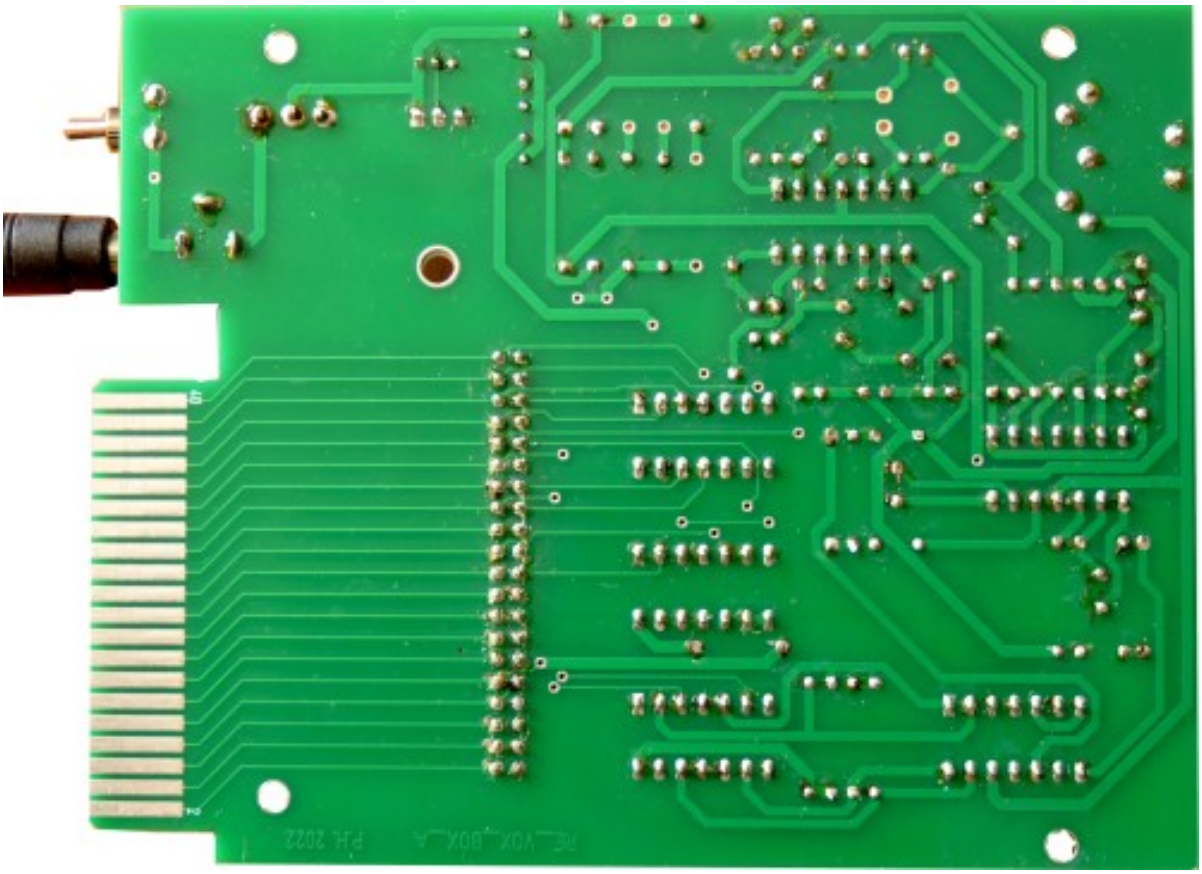
1. Be as compatible as possible with the VoxBox box
2.

Warning: This card is version A and contains an error for the installation of the DIN socket (holes too small and footprint only allowing the installation of a single type of DIN socket), a version B is in production correcting these errors and allowing the installation of two types of DIN socket

Top view



Bottom View



The RE-VOX_BOX voice recognition unit allows you to take advantage of old technology on your TRS-80 M1. You can now use words and phrases to control and instruct your computer and to enter data. Focus on handy work or play, like viewing video, without the distraction of typing on the keyboard. Simple to use, connects directly to the TRS-80 or USB interface TRS-80 Bus Card Edge expansion.

- 1) Simple to use, connects directly to TRS-80 or TRS-80 Bus Card Edge expansion interface.
- 2) Using the examples and instructions in this manual, you can write your own programs for custom applications.
- 3) Use learning mode to teach the RE-VOX_BOX up to 32 words or phrases, each word/phrase can last up to 1.2 seconds.
- 4) Re-train it anytime to recognize 32 different word vocabulary in any language!

Note: This voice recognition is old technology. In fact, your RE-VOX_BOX is one of the first devices of its kind for the general public. For this reason, I recommend that the device be used primarily for entertainment and experimentation. Proceed with caution before engaging the unit in serious application.

Equipment required

1. TRS-80 with 16K RAM
2. Sound Recorder
3. Optional expansion interface

Please note the RE-VOX_BOX and the VOXBOX were designed to operate mainly on magnetic tapes, but there is a program that I discovered on the WEB "VXDRIVER/CMD" which allows these cards to operate under LDOS 5.3.1..

This program replaces the original programs SP16, SP32, SP48.

To make this card work under floppy and 48K of memory

1. You run: LDOS 5.3.1
2. Then: VXDRIVER/CMD
3. Then: BASIC (MEM=61439)
4. Then your program



YOU choose which emulation system you want.
Out of:

BBC B
TRS-80 Model I/III/IV
ZX81

State your choice (and choice of keyboard) in "message to seller" upon purchasing.

Whilst we keep all parts in stock we build to order, so allow up to five days for dispatch.
(It's usually done in three.)

ALL systems are preloaded with hundreds if not thousands of games from back in THE day.
And load instantly. (No cassettes!)

Some games support any standard USB controller which you can plug straight into your RX-1000. Keyboard controls are identical to the original now vintage computers of our childhoods.

FREE postage to UK buyers.

Place your order and get ready for the adventure.
Your childhood/youth in a box.
Plug and play.

All software included is in the public domain aside from RX-1000 operating systems which are (C) 2022 Retro Shack.

YouTube link below is to a short promo video made when the RX-1000 was developed.

https://youtu.be/eRB8T4bNv6Q

I THOUGHT UP A SILLY LITTLE TRS-80 JOKE WHICH I DECIDED TO PUT IN THE FORM OF A BASIC PROGRAM. THE PUNCH LINE IS SCRAMBLED JUST ENOUGH TO KEEP FROM GIVING IT AWAY.

GIVEN THE QUALITY OF THE JOKE I DEEMED IT NECESSARY FOR THE PROGRAM TO DO ITS OWN "LAUGH TRACK" IN SUPPORT.

-- GEORGE --




```
READY
>@LOAD"GEOSJOKE/BAS"
>LIST
10 CLS:PRINT"WHY ARE TRS-80 GRAPHICS SO COOL?"
20 IFINKEY$=""THEN20
30 A$="CFIDBVTF!JUT!QJYFMT!BSF!OPU!TRVBSF!"
40 PRINT:PRINT" ";
60 FORX=1TOLEN(A$):PRINTCHR$(ASC(MID$(A$,X,1))-1);
70 SET(2*X,4):SET(2*X,9):NEXT
80 FORY=4TO6STEP2:SET(0,Y):SET(2*X,Y):SET(0,Y+3):SET(2*X,Y+3):NE
XT
90 P=256+RND(764):PRINT@P,"HA!";:
100 FORX=0TO50:NEXT:PRINT@P,"   ";:GOTO 90
READY
>_
```



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

Thirsting for adventure?
Brass lantern getting dusty?



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

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



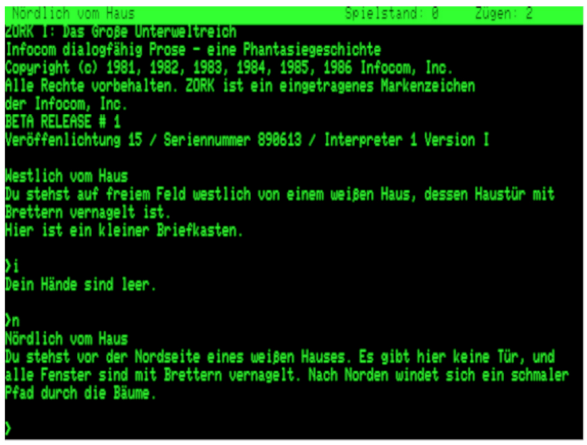
M4ZVM is available for download from https://gitlab.com/sijnstral/m4zvm (includes feature details and source)
Enhanced versions ready for download!

*128K recommended

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



Beyond Zork



Zork 1 (German)

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



One of the most infuriating features of the Tandy Level II manual is that there is no index. We now remedy that with contents you can cut out and stick into your manual.

TRS-80 Level II Index

ABS	7/1	FIX	7/2	PRINT # -1	3/10
ASC	5/3	FOR. .TO. .STEP	4/8	PRINT	3/1
ATN	7/1			PRINT@	3/1
AUTO	2/1	FRE	5/5	RANDOM	7/3
CDBL	7/1	GOSUB	4/6	READ	3/9
CHR$	5/4	GOTO	4/5		
CINT	7/2	IF	4/12	RESET	8/2
CLEAR	2/2,4/3	INKEY$	5/5	RESTORE	3/10
CLOAD	2/2	INPUT # -1	3/10	RESUME	4/11
CLOAD?	2/2	INPUT	3/7	RETURN	4/6
CLS	8/2	INT	7/3	RIGHT$	5/7
CONT	2/3	LEFT$	5/6	RND	7/3
COS	7/2	LEN	5/6	SET	8/1
CSAVE	2/3	LET	4/4	SGN	7/4
CSNG	7/2	LIST	2/4	SIN	7/4
DATA	3/8	LLIST	10/2	SQR	7/4
DEFDBL	4/2	LOG	7/3	STOP	4/5
DEFINT	4/1	MEM	8/4	STR$	5/7
DEFSNG	4/2	MID$	5/6	STRING$	5/7
DEFSTR	4/2	NEW	2/4	SYSTEM	2/5
DELETE	2/3	NEXT	4/8	TAB	5/2
DIM	4/3,6/3	ON ERROR GOTO	4/11	TAN	7/4
EDIT	2/4	ON N GOSUB	4/7	THEN	4/13
ELSE	4/13	ON N GOTO	4/6	TROFF	2/5
END	4/4	OUT	8/4	TRON	2/5
ERL	8/2	PEEK	8/5	USING	3/3
ERR/2+1	8/3	POINT	8/2	USR	8/7
ERROR	4/10	POKE	8/5	VAL	5/8
EXP	7/2	POS	8/6	VARPTR	8/8

Reprinted from Orange County TRS-80 Users' Group Newsletter, available by donation, £10 a year from user group, 24232 Tahoe ct., Laguna Niguel CA 92677.

PRACTICAL COMPUTING *April 1979*

Serial-printer drive routine for Tandy TRS-80

ONE OF the attractive features of the TRS-80, after the pleasure of finding that you have 16K of usable RAM and 12K of Basic for less than the price of an 8K Brand X, is the ease with which a printer can be driven.

There is not much doubt that a printer is the first essential accessory, assuming that you start with a reasonable amount of RAM. The low-price printers, such as the Nascom Imp or the Epson are serial printers. The natural companion to the TRS-80 is the Centronics, which uses a parallel port, but the Centronics costs considerably more than a complete TRS-80, and if it happens to carry the Radio Shack label, it may cost even more.

Fortunately, serial printers can be interfaced to the TRS-80 through the cassette port, using a very simple interface consisting of little more than a 741 integrated circuit, a rectifier bridge, and a few resistors and capacitors. The total cost of the interface, if you build it yourself, is about £2.

If you are not in the hardware business, however, you will be asked for about £40

by Ian Sinclair

for an interface which presumably must use a very expensive plug and socket. Some of the cost of the commercial effort is the software which accompanies it, because the use of the cassette port for serial printer output is not one of the considerations which Tandy had in mind when it assembled its package.

There are many serial-printer drive programs, ranging from a bare 100 bytes all the way to fully-fledged printer, screen and keyboard controllers, like the excellent KVP Extender from Microcomputer Applications.

I wanted more than a straightforward routine, because I like to keep listings as I develop programs and nothing is more awkward than keeping long rolls of paper. Since I keep the original program notes on A4 and in A4 files, it seems logical to keep the listings there as well, which means that I needed a left-hand margin, to allow room for the punch-holes.

As the Nascom Imp printer starts well over to the left-hand side on listings, that obviously was a software task. In addition, using A4 files meant that the listing should be in groups of 66 lines, so that I could divide the Teletype roll into pages. That in turn meant a lines-per-page counter in software which could be re-set so that I could set top-of-page when I started a listing, instead of having to accept a random number of lines before a page break.

At that point in the specification, a few bells and whistles started to be added. I wanted to be able to slow down the listing on the screen with a simple command — I hate having to poke numbers into memory — just as that other computer does, and I wanted the usual keyboard delay because my TRS-80 is an old one with keybounce.

To make the specification more awkward, I originally wanted the program re-locatable, so that I could place it anywhere in memory. There was only one way of obtaining all this, and that was to write the program myself.

Since the TRS-80 uses the Z-80 CPU, there is a version of the Zen assembler-editor-debugger available for it. If there are still people who run Z-80 machines and have not encountered Zen, I believe it is the best editor/assembler package available.

It is simple to use — which means that you can stop thumbing through the instructions after a few hours, which means I make for more use of it than of the package I used before.

As an assembler/editor, it is very versatile, allowing quick and easy assembler program writing without having to watch the size of spaces or the length of label-names; editing is very simple. It even permits assembler programs to be combined from tape, and, of course, writes object code tapes in TRS-80 standard code.

In addition, Zen has full monitor facilities, allowing you to read machine-code

Figure 1.

```
 1                      ;LPRINT ROUTINE WITH LHS MARGIN
 2                      ;66 LINES/PAGE AND 72 CH/LINE
 3                              ORG   7F11H
 4                              LOAD  6F00H
 5                      CHRSLN: EQU   4029H
 6                      LNFCE:  EQU   4028H
 7 7F11 E5                      PUSH  HL
 8 7F12 21347F                  LD    HL,LPRT
 9 7F15 22C640                  LD    (40C6H),HL
10 7F18 21FC7F                  LD    HL,KBFIX
11 7F1B 221E40                  LD    (401EH),HL
12 7F1E 2ADE7F                  LD    HL,(CHRLN)
13 7F21 222940                  LD    (4029H),HL
14 7F24 21DF7F                  LD    HL,TOPPAG
15 7F27 227D41                  LD    (417DH),HL
16 7F2A 21E77F                  LD    HL, DELAY
17 7F2D 228341                  LD    (4183H),HL
18 7F30 E1                      POP   HL
19 7F31 C37200                  JP    0072H
20 7F34 F3                      DI
21 7F35 79                      LD    A,C
22 7F36 F5                      PUSH  AF
23 7F37 D5                      PUSH  DE
24 7F38 FDE5                    PUSH  IY
25 7F3A CD3300                  CALL  0033H
26 7F3D FDE1                    POP   IY
27 7F3F D1                      POP   DE
28 7F40 F1                      POP   AF
29 7F41 FE0D                    CP    0DH
30 7F43 2805                    JR    Z,RELOAD
31 7F45 FE20                    CP    20H
32 7F47 D8                      RET   C
33 7F48 181C                    JR    START
34 7F4A E5                      PUSH  HL
35 7F4B F5                      PUSH  AF
36 7F4C 21DE7F                  LD    HL,CHRLN
```

(continued on next page)

tapes, examine the contents, change code, copy tapes, copy code from one memory location to another — which is why I like re-locatable programs — and it can even load code into memory as it is assembled, without having to make a tape first.

I used Zen to create the assembler program of figure 1 which is why the line numbers follow the 1,2,3 order and are placed at the start of each line, rather than the convention used by Radio Shack. Those line numbers, incidentally, are not recorded when you tape an assembler program; they are allocated by Zen when the program is loaded, which is why programs can be merged, no matter what line numbers they used.

It does mean, though, that the Zen assembler tapes are not compatible with tapes created by a disassembler operating to Radio Shack standards.

The program is not annotated. Notes occupy a good deal of memory and tape, and I prefer to keep copious notes on a separate sheet. For the benefit of anyone who wants to modify the program either for the TRS-80 or for another Z-80 based machine, here is an outline of the program operation.

The baud rate is set at 300, since that is the maximum rate recommended for a serial printer operated over a single line, with no provision for handshake — no signals can pass back from the printer to the computer to control the rate of transmission.

Program origin

The origin of the program as shown is 7F11H, 32529D, and two of the printer-control RAM addresses in the TRS-80 are used. They are, of course, not part of the 16K user RAM, but a separate section of about 3K of RAM which can be used for a remarkable range of control operations, and which are an Aladdin's cave for machine-language programmers.

Location 4029H (16525D) is used to store the number of characters per line, and 4028H (16524D) is used for the number of lines per page. Four addresses in the program space itself are also used, and they can be Poked so as to alter the print format as desired. As set up, the program prints 72 characters per line and 66 lines per page, with six blanks between pages to allow for cutting.

The first section of the program from 7F11H to 7F2FH loads the locations in low RAM. 4026H is the address for the start of the printing routine. The address which is normally loaded into 4026H, 4027H is 058DH, the ROM routine for a parallel printer, and we have to substitute 7F34H, so that the LPRINT and LLIST commands of the TRS-80 direct the print commands to our routine starting at 7F34H. 401EH is the start address location for the screen-write routine, normally 0458H, but now changed to 7FF2H.

That allows us to put a delay into the screen-print routine, and performs two useful functions — it allows slow-listing to be controlled, and it also debounces the wretched keyboard of my old TRS-80 — new models seem to have a new keyboard with no bounce problems. The number of characters per line is loaded into 4029H; no figure is loaded into 4028H because it is set at 66 lines per page by the TRS-80

(continued from previous page)

```
 37 7F4F 3A2940          LD    A,(CHRSLN)
 38 7F52 77              LD    (HL),A
 39 7F53 21DC7F          LD    HL,LNCNT
 40 7F56 35              DEC   (HL)
 41 7F57 200B            JR    NZ,RUN
 42 7F59 2A2840          LD    HL,(LNPGE)
 43 7F5C 22DC7F          LD    (LNCNT),HL
 44 7F5F 21DD7F          LD    HL,ENDPAG
 45 7F62 3606            LD    (HL),06H
 46 7F64 F1      RUN:    POP   AF
 47 7F65 E1              POP   HL
 48 7F66 F5      START:  PUSH  AF
 49 7F67 E5              PUSH  HL
 50 7F68 C5              PUSH  BC
 51 7F69 0609    AGAIN:  LD    B,09H
 52 7F6B 37              SCF
 53 7F6C F5              PUSH  AF
 54 7F6D F5              PUSH  AF
 55 7F6E 2101FC          LD    HL,0FC01H
 56 7F71 CD2102          CALL  0221H
 57 7F74 CD797F          CALL  DLY
 58 7F77 1809            JR    ROUND
 59 7F79 21DE00  DLY:    LD    HL,00DEH
 60 7F7C 2B      DLY1:   DEC   HL
 61 7F7D 7C              LD    A,H
 62 7F7E B5              OR    L
 63 7F7F 20FB            JR    NZ,DLY1
 64 7F81 C9              RET
 65 7F82 F1      ROUND:  POP   AF
 66 7F83 1F              RRA
 67 7F84 F5              PUSH  AF
 68 7F85 3005            JR    NC,OUT1
 69 7F87 2102FC          LD    HL,0FC02H
 70 7F8A 1805            JR    OUT2
 71 7F8C C600    OUT1:   ADD   A,00H
 72 7F8E 2101FC          LD    HL,0FC01H
 73 7F91 CD2102  OUT2:   CALL  0221H
 74 7F94 CD797F          CALL  DLY
 75 7F97 10E9            DJNZ  ROUND
 76 7F99 2102FC          LD    HL,0FC02H
 77 7F9C CD2102          CALL  0221H
 78 7F9F CD797F          CALL  DLY
 79 7FA2 F1      OUT3:   POP   AF
 80 7FA3 F1              POP   AF
 81 7FA4 FE0D            CP    0DH
 82 7FA6 2022            JR    NZ,TEST2
 83 7FA8 21DD7F  TEST1:  LD    HL,ENDPAG
 84 7FAB AF              XOR   A
 85 7FAC B6              OR    (HL)
 86 7FAD 2812            JR    Z,BLNK
 87 7FAF 35              DEC   (HL)
 88 7FB0 3E0D            LD    A,0DH
 89 7FB2 1805            JR    AGAIN
 90 7FB4 C1      OUT4:   POP   BC
 91 7FB5 21DE7F          LD    HL,CHRLN
 92 7FB8 35              DEC   (HL)
 93 7FB9 201D            JR    NZ,OUT5
 94 7FBB E1              POP   HL
 95 7FBC F1              POP   AF
 96 7FBD 3E0D            LD    A,0DH
 97 7FBF 1809            JR    RELOAD
 98 7FC1 21DB7F  BLNK:   LD    HL,LHSPCE
 99 7FC4 3603            LD    (HL),03H
100 7FC6 3E20            LD    A,20H
101 7FC8 189F            JR    AGAIN
102 7FCA 3ADB7F  TEST2:  LD    A,(LHSPCE)
103 7FCD B7              OR    A
104 7FCE 28E4            JR    Z,OUT4
105 7FD0 3D              DEC   A
106 7FD1 32DB7F          LD    (LHSPCE),A
107 7FD4 3E20            LD    A,20H
108 7FD6 1891            JR    AGAIN
109 7FD8 E1      OUT5:   POP   HL
110 7FD9 F1              POP   AF
111 7FDA C9              RET
112 7FDB 03      LHSPCE: DB    03H
113 7FDC 42      LNCNT:  DB    42H
114 7FDD 00      ENDPAG: DB    00H
115 7FDE 48      CHRLN:  DB    48H
116 7FDF E5      TOPPAG: PUSH  HL
117 7FE0 21DC7F          LD    HL,LNCNT
118 7FE3 3642            LD    (HL),42H
119 7FE5 E1              POP   HL
120 7FE6 C9              RET
121 7FE7 F5      DELAY:  PUSH  AF
122 7FE8 3AF67F          LD    A,(KBFIX+4)
123 7FEB EE10            XOR   10H
124 7FED 32F67F          LD    (KBFIX+4),A
125 7FF0 F1              POP   AF
126 7FF1 C9              RET
127                      ORG   7FF2H
128 7FF2 F5      KBFIX:  PUSH  AF
129 7FF3 C5              PUSH  BC
130 7FF4 012000          LD    BC,0020H
131 7FF7 CD6000          CALL  0060H
132 7FFA C1              POP   BC
133 7FFB F1              POP   AF
134 7FFC C35804          JP    0458H
135                      END

SYMBOL TABLE

AGAIN   7F69  BLNK    7FC1  CHRSLN  4029  CHRLN   7FDE
DLY     7F79  DLY1    7F7C  DELAY   7FE7  ENDLN   7F41
ENDPAG  7FDD  KBFIX   7FF2  LNPGE   4028  LPRT    7F34
LHSPCE  7FDB  LNCNT   7FDC  OUT1    7F8C  OUT2    7F91
OUT3    7FA2  OUT4    7FB4  OUT5    7FD8  RELOAD  7F4A
RUN     7F64  ROUND   7F82  START   7F66  TEST1   7FA8
TEST2   7FCA  TOPPAG  7FDF
```

ROM when the machine is switched-on.

The remaining two loadings are luxury items. When a disc system is not in use — several commands, such as FIELD, PUT, GET etc., will return "L3 ERROR" if you use them in Basic. According to the manual, that is because they are Disc Basic commands, which can be used only with a disc system.

In fact, each one of those commands causes the ROM to look in RAM for the address of a routine, and if you poke in your own address and your own routine, you can make use of any of the commands — there are 28 of them from which to choose. In the program, I have used the FIELD command to re-set the printer program to top-of-page, so that it will print a full 66 lines before putting in its six blank lines.

Delay times

The address of a page-re-set routine at 7FDFH is, therefore, loaded into 417DH, the address for the FIELD command. Similarly, I have used PUT to change the delay time, by loading the address of a delay-byte insertion routine to 4183H. By typing PUT and entering from Basic, the TRS-80 screen-access speed is considerably slowed so that all operations which involve the screen are run at low speed and that obviously includes listing.

If the TRON command, part of standard TRS-80 Basic, has been entered, so that the line numbers are printed as the program executes, PUT allows me to watch a program executing in slow motion — a most useful debugging aid for elusive faults. Typing PUT for a second time and entering restores normal running speed.

That delay, incidentally, can also be implemented by using the keyboard address, but the screen routine is preferable, because the delay is then operative only when the screen is accessed — the keyboard is scanned continually, and any delay in the keyboard routine slows the computing rate considerably unless the keyboard is disabled during computing time. By this time, re-locatability was abandoned.

The loading section ends with a jump to 0072H, which returns to Basic. That is not the address given in the older Radio Shack manuals, incidentally. The address in the old manuals is not so suitable, and can cause error messages to appear; later editions give this correct address of 0072H.

The print routine starts at 7F34H, with a DI command so that the routine cannot be interrupted in mid-character by the break or any other key. The printer routines of the TRS-80 place the character byte in the C register, so that is transferred to the accumulator, A, for subsequent operations. The section of program from 7F36H to 7F40H saves registers, and calls the screen-print routine, so that whatever is printed is also echoed on the screen.

That is a useful feature, since most dot-matrix printers conceal what is being printed. Curiously enough, that, along with the delay, causes odd effects — a "z" is printed in place of the least significant digit of the line number — when that routine is used along with Zen, so that the version of the program which I have tacked into Zen has 7F3AH and 7F3BH both set at 00, forming a set of NOP instructions.

At ENDLN, the byte in the accumulator is checked to see if it is a carriage return (0DH), in which case the program branches to RELOAD so as to re-set the memory location which governs the number of characters per line and to decrement the number-of-lines count in location LNCNT.

If at ENDLN, the byte in the accumulator has a value less than 20H and is not a carriage return, the program returns for the next character, since the other control characters are not used in this program. For any other character, the program jumps to START.

The registers are saved, and register B is loaded with a bit count — one byte plus a space. A mark bit is sent out by loading FC01 to HL and calling 0221H, which sends the bit in the L register to the cassette port. Incidentally, I have used 01 for mark and 10 for space, because this gives a much larger output signal than the conventional 01 and 00.

To establish the correct baud rate, a delay must follow, and that is achieved in the subroutine labelled DLY. The byte 00DEH which is loaded into HL for the delay subroutine determines the baud rate, so that if you are writing for 110 baud, a larger number must be loaded — try 0267H.

Carry position

At the label ROUND, the lowest bit of the byte is rotated into the carry position by the RRA command, and the other bits are similarly shifted in the accumulator. If the bit in the carry position is 0, C is re-set, and the program jumps to OUT1 to send a space bit. If the bit in the carry position is a 1, hl is loaded with FC02H, the space bit which is output by the routine at OUT2. Those routines are followed by the DLY subroutine to keep the baud rate correct.

The DJNZ command at 7F97H keeps the rotate-and-output routine going until the B-register is decremented to zero in the usual action of the DJNZ command. A final delay subroutine follows, and then at OUT3, which is the end of the main section of the print routine, the original unrotated byte is recovered and compared to the carriage return byte, 0DH. If the byte was a C/R, the program skips to TEST2, otherwise at TEST1 the byte from ENDPAG is loaded, and the accumulator is cleared.

Now if the lines-per-page count reached zero earlier in the program, at RELOAD, there will be a six in ENDPAG, so that the OR(HL) step will give a non-zero answer, and the program will continue to decrement ENDPAG, load in a carriage-return character and run it through the print routine again until ENPAG is zeroed.

If the line/page count has not reached zero, the step at 7FADH directs the program to BLNK, where the store LHSPCE is loaded in with the number 3. The accumulator is loaded with the ASCII blank character, 20H, and the printing routine repeated, so that a space is printed at the left-hand side of the page.

That continues until the store LHSPCE is decremented to zero, giving the left-hand margin which prevents the listing from being mangled.

When the character in A is not a carriage-return, the test routine is TEST2. That tests for the byte in LHSPCE to see if more spaces need to be printed, otherwise the retreat is sounded by jumping to OUT4. At OUT4, the characters/line count is decremented and, if the count is zero, the accumulator is loaded with a carriage return character and re-cycled. If the byte is not at the end of a line, the next stage in the countdown is OUT5, where the registers are restored and the routine returns for the next character.

Storage bytes

The bytes from 7FDBH to 7FDEH are storage bytes for the left-space, lines/page, end-of-page space, and characters/line respectively. At TOPPAG, a short routine restores the lines/page count. That is the routine which is called by typing FIELD and entering. Similarly at 7FE7H, a short routine inspects the delay byte at 7FF6H and XORs it with 10H. If the delay byte was 0, it is replaced by 10H, if it was 10H, it is replaced by 0, such being the action of the XOR command.

The routine is called from Basic by the PUT command. It, like FIELD, can be called from a running program, so that a program which prints to the screen can be slowed in mid-run and speeded up again as needed — useful for instructions. Finally, at 7FF2H, the video delay consists of saving registers, loading BC with the delay bytes, calling the TRS-80 delay subroutine at 0060H, restoring and then jumping to the screen routine at 0458H.

It has been a very useful routine for me, and its usefulness has been enhanced by a short program from A J Harding (Molimerx), which transforms the machine code of a routine into DATA lines of a Basic program. In that way, I can poke it in from any Basic program, so that I do not need to load a separate object code tape.

Furthermore, if I ever need it to be fully re-locatable, I can place the DLY subroutine and the store bytes into low memory between 405C and 407F — that is used only when the TRS-80 is first switched-on. At present, however, the ease with which I can generate object code with Zen makes strict re-locatability unnecessary.

SOMEONE, ON ONE OF THE INTERWEB CHANNELS WAS ENQUIRING ABOUT THIS COMPETITION - IT'S PART OF THE MOLIMERX-LTD CATALOGUE (WHICH IS AVAILABLE FROM THE DOWNLOADS PAGE AT TRS-80.ORG.UK)

Software News

INNOVATIVE
TRS 80-GENIE SOFTWARE

from the professionals

£250 REWARD

Below you will find described a new program entitled Enigma. It is a true simulation of the German wartime cypher machine of that name. It will encipher messages which may be communicated to third parties by any means who, assuming they have the key, will be able to use their Enigma program to decipher.

We will pay the sum of £250 to anyone [who has purchased the program] who can demonstrate an infallible method of deciphering the coded message supplied in the program's instructions. We consider Enigma to be the best program of its kind on the U.K. or U.S. market; contestants may therefore use any orthodox means to crack the code, including microcomputer programs other than Enigma.

The original message and keys will be lodged with our Solicitors for safe keeping in a sealed envelope. In the [hopefully] unlikely event that the code is cracked by more than one person, the reward will be paid to the first customer who demonstrates to us that he has succeeded.

MOLIMERX LTD.

During the 1939/45 war the German Army and Intelligence used a deciphering machine called Enigma. It was a fascinating machine and the stories that have surrounded it are equally interesting. There have been some four or five books written about the machine, and with regard to the way in which the British counter intelligence managed to crack the code.

That they did so was the culmination of some fortuitous circumstances, a lot of luck, but mainly it was due to the fact that the people who did it were extremely clever mathematicians. The fact that it took so much brain power, plus a rudimentary type of computer and a specimen of the machine in order to crack the code is an indication of how complex that code is.

The Enigma microcomputer program that we are selling is a simulation of the original machine, together with one or two improvements which were suggested by Gordon Welchman, who wrote the book "The Hut Six Story" last year and was also the leader of the team that cracked the code.

Although the machine and, therefore, the program is so complicated, its use is amazingly simple. One simply inputs a key and a message and the code is supplied. To decipher, the message is input again with the key and if the key is correct then the decoded message is displayed. With the cassette version it is necessary to input from the keyboard but with disks both inputs and outputs may be to disk files if required. A printer is of course supported.

The code may be transmitted in any way which the written word can be transmitted. Companies who wish to fully protect their communications will no doubt have the program generate the code and then tap it into a telex. Tape users will have to send either the output from their printer or write down the code direct from the screen.

Enigma is a fascinating program designed, not only for those people who are interested in encryption professionally or as a hobby, but also for companies or private persons who wish to communicate with others in an entirely secure manner. As is shown by the above Reward Notice, we have great faith in the powers of this piece of software.

ENIGMA (Tape) ... **£17.25**
ENIGMA (Disk) ... **£23.00**
Inclusive of V.A.T. P&P 75p

TEL: [0424] 220391/223636

MOLIMERX™ LTD

TELEX 86736 SOTEX G

A J HARDING (MOLIMERX)

1 BUCKHURST ROAD, TOWN HALL SQUARE, BEXHILL-ON-SEA, EAST SUSSEX.

TRS-80 & VIDEO GENIE SOFTWARE CATALOGUE £1.00 plus £1 postage.

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



GEORGE PHILLIPS

ANOTHER LITTLE PROGRAM TO PUBLISH IF YOU LIKE. I SAW A POSTING OF THIS CAT ANIMATION ON A COCO AND THEN AN MC-10. I THINK THAT IT HAS BEEN PORTED TO MANY DIFFERENT 8 BIT COMPUTERS. I FIGURED IT NEEDED A TRS-80 MODEL 1/3 VERSION.

=-- GEORGE --=



```
10 CLEAR1000:DEFINTA-Z:DIM C$(5,3):CLS
20 FORF=0TO5:FORY=0TO3:C$(F,Y)=" ":FORX=1TO16:READN:C$(F,Y)=C$(F,Y)
+CHR$(N):NEXT:PRINTC$(F,Y):NEXT:NEXT
30 CLS:F=0:P=-18:PRINT@640,STRING$(64,131)
40
IFP<0THENN=18+P:PRINT@384,RIGHT$(C$(F,0),N);:PRINT@448,RIGHT$(C$(F,1),N)
;:PRINT@512,RIGHT$(C$(F,2),N);:PRINT@576,RIGHT$(C$(F,3),N);:GOTO70
50 IFP>47THENN=64-
P:PRINT@P+384,LEFT$(C$(F,0),N);:PRINT@P+448,LEFT$(C$(F,1),N);:PRINT@P+512,
LEFT$(C$(F,2),N);:PRINT@P+576,LEFT$(C$(F,3),N);:GOTO70
60
PRINT@P+384,C$(F,0);:PRINT@P+448,C$(F,1);:PRINT@P+512,C$(F,2);:PRINT@P+5
76,C$(F,3);
70 F=F+1:IF F=6THENF=0
80 P=P+2:IF P>62 THEN P=-18:FORI=0TO200:NEXT
90 GOTO40
1000 DATA 128,128,128,128,128,160,176,176,176,176,176,140,140,140,164,148
1010 DATA 128,128,160,152,131,129,128,128,128,128,128,168,128,156,164,154
1020 DATA 134,131,183,176,152,140,140,131,131,131,131,131,137,143,156,144
1030 DATA 128,134,181,128,128,128,128,128,128,128,128,128,128,128,130,130
1040 DATA 128,128,128,128,128,176,176,176,144,128,160,176,176,128,160,128
1050 DATA 128,128,160,152,131,128,128,128,130,131,129,160,160,179,131,164
1060 DATA 131,163,185,165,176,140,134,131,131,131,131,175,139,172,179,129
1070 DATA 128,140,129,128,128,128,128,128,128,128,128,138,180,128,128,131
1080 DATA 128,128,128,128,160,152,134,131,131,140,176,176,144,176,180,128
1090 DATA 128,128,128,152,131,148,128,160,164,144,128,128,130,152,176,153
1100 DATA 130,131,131,128,138,152,140,129,128,162,155,139,186,144,128,128
1110 DATA 128,128,128,128,128,134,128,128,128,138,144,128,128,131,164,128
1120 DATA 128,128,128,176,176,176,134,131,131,140,176,128,128,128,144,128
1130 DATA 176,152,134,128,128,138,144,164,128,160,128,137,140,134,137,144
1140 DATA 128,128,128,128,128,128,130,190,169,165,156,140,140,131,131,128
1150 DATA 128,128,128,128,128,128,128,130,135,130,181,128,128,128,128,128
1160 DATA 160,140,140,140,176,176,140,140,164,144,128,128,128,144,128,128
1170 DATA 130,128,128,128,128,150,128,128,128,130,137,140,131,137,144,128
1180 DATA 128,128,128,128,128,169,152,140,140,173,164,140,131,131,128,128
1190 DATA 128,128,128,128,128,174,154,176,136,129,129,128,128,128,128,128
1200 DATA 130,131,131,140,140,164,176,140,140,176,128,128,128,160,184,144
1210 DATA 128,128,128,128,168,129,128,128,128,128,131,137,131,161,176,154
1220 DATA 128,128,128,160,134,176,152,131,137,140,140,140,172,185,128,128
1230 DATA 128,128,128,130,166,180,144,128,128,128,128,128,128,128,128,128
```

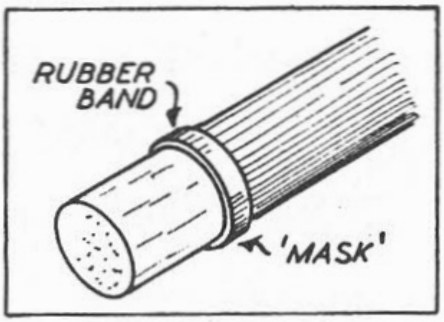
IF YOU WOULD LIKE A COPY OF THE BASIC PROGRAM, JUST TO SAVE HAVING TO TYPE IT IN, PLEASE EMAIL ME AND I'LL SENT ONE TO YOU ASAP. THERE WILL BE AN ANIMATED .GIF RUNNING ON THE WEBSITE (WWW.TRS-80.ORG.UK) SHORTLY. *ED.*



HANDY HINTS

HERE'S SOMETHING JUST A BIT DIFFERENT. IT'S A 'HANDY HINT' FROM THE 1955 'GADGETS ANNUAL'. I MUST ADMIT, I HADN'T THOUGHT OF THIS ONE MYSELF. I HOPE YOU FIND THE TIP USEFUL.

RUBBER BANDS AS PAINTING MASKS



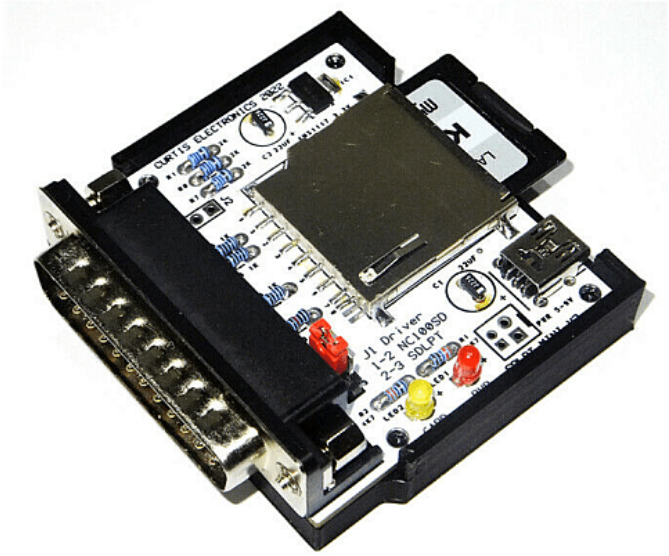
WHEN painting small articles, a flat rubber band will make a good mask to produce a clean, straight edge to the paintwork. Just slip the band in place up against the line marking the end of the paint and then paint right up to the band. Remove the band when the paint is still slightly tacky.

A tip when using two or more colours: paint the lightest colour first, allow to dry, and then apply the next lightest colour, and so on.

AVAILABLE ON THE U.K. EBAY SITE, I THOUGHT THIS ADVERT MIGHT INTEREST THE 'TANDY PC' COMMUNITY. I RECALL A CD DRIVE USING THE PARALLEL PORT IN THE EARLY 1990'S *ED.*



SDLPT SD-Card Adapter
LPT Parallel Port Removable SDCard Drive for Classic PC's



Read/Write Files Through LPT Port
Floppy Drive Transfer Speeds
Driver Supports DOS and Windows 3.1/95

THE SDLPT ADAPTER IS DESIGNED TO ALLOW CLASSIC IBM COMPATIBLE PC SYSTEMS TO USE SD-CARDS AS A DISK FILE STORAGE DEVICE. THE ADAPTER IS AN ALTERNATIVE OPTION TO FLOPPY DISCS AND IS NOT DESIGNED AS A DIRECT REPLACEMENT FOR HARD DRIVES OR OTHER STORAGE DEVICES.

THE DRIVER USES A SLOW SPI (SERIAL PERIPHERAL INTERFACE) CONNECTION THROUGH THE LPT PORT TO COMMUNICATE WITH THE SD-CARD AND THIS LIMITS THE DATA TRANSFER RATE. TYPICAL FILE TRANSFER RATE IS COMPARABLE TO A FLOPPY DISK DRIVE AT AROUND 30-60 KB/S. TRANSFER RATES ALSO DEPEND ON THE PC SYSTEM CONFIGURATION AND LPT PORT SETTINGS.

THE ADAPTER IS COMPACT ENOUGH TO NOT PROTRUDE TOO MUCH WHEN PLUGGED INTO THE LPT PORT.
LOCKING NUTS CAN BE USED TO SECURE THE ADAPTER IN PLACE OR A CABLE CAN ALSO BE USED INSTEAD
OF DIRECTLY PLUGGED INTO THE PORT.

POWER IS SOURCED FROM A 5V MINI USB PLUG OR OTHER 9V TO 5V 30MA POWER SOURCE.
THE ADAPTER CAN ALSO BE POWERED FROM A 9V PP3 BATTERY. MINIMUM 5V REQUIRED.

EBAY.CO.UK 'BARGAINS'

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



£
£20



£99



£300



£37



£110



£311



£210



£105



£230

EBAY.CO.UK 'BARGAINS'

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



£

£78



£205



£62



£135



£576



£51



£112



£190



£4

Retrocomputing Pages: TRS-80 Model 100

Sarah Libman

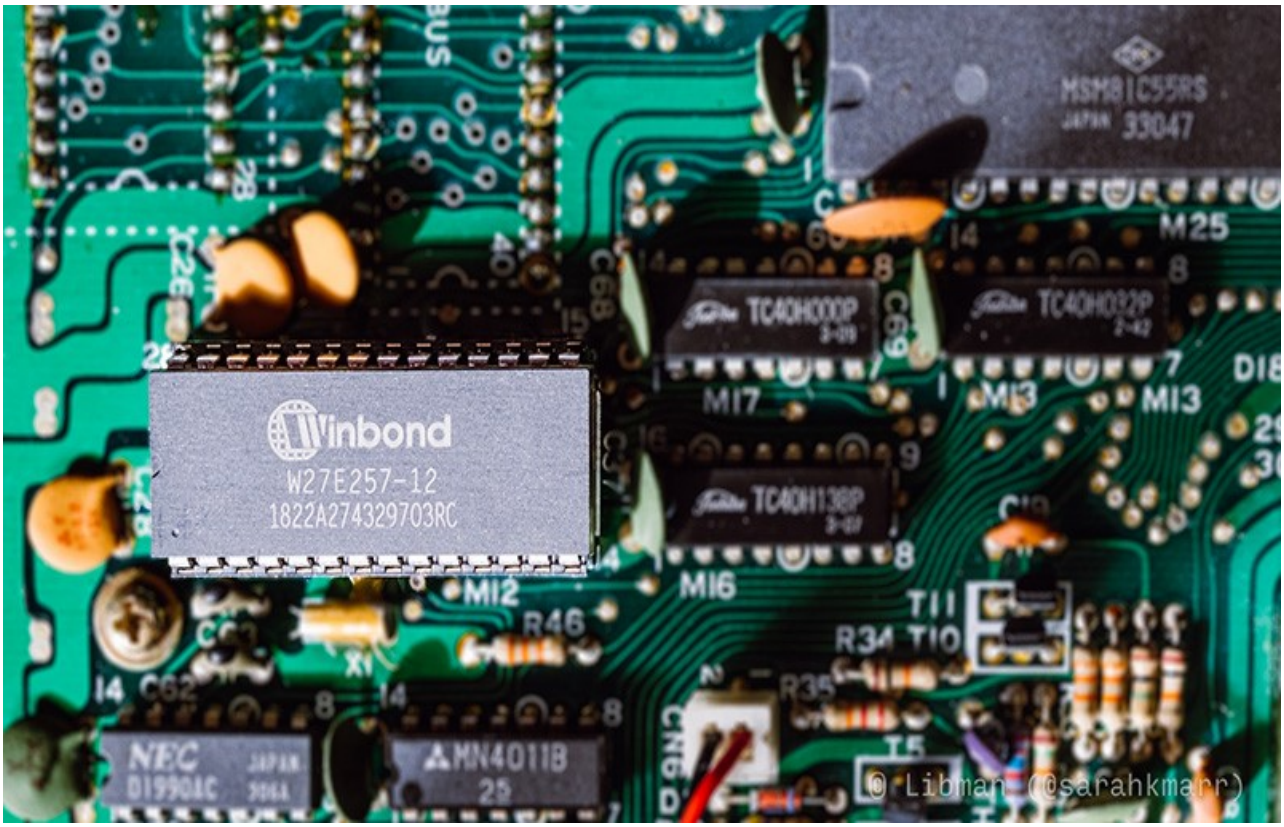


Quick download link: LibROM

When I got my TRS-80 Model 100, a UK model, I discovered that UK Model 100s don't have a modem. Eh, no big deal. But! what was a big deal is that UK ROMs have all their routines in different locations to American ROMS, so nothing which uses ROM calls will run on them; REXCPM isn't compatible, etc. Anyway, fast forward and I've learned about the Model 100, picked up 8085 assembly, got to work on Ken Pettit's annotated ROM listing, disassembled Ron Wiesen's 'Teeny' — with helpful pointers from Steve Adolph — and hex-edited my way to LibROM.

Click here to download the zip file, which contains a .bin file to be program an EPROM and a .asm file which contains a heavily annotated assembly listing of the ROM. Although LibROM was designed for non-American machines, it should work in American Model 100s, too. (Note, however, that early American models used non-standard pins for the ROM chip so it may not be trivial to replace it. Also, you might want to back out the change of date format, for American use.)

I used a XGecu TL866II Plus EPROM programmer, which did the job without any problems. I programmed the ROM onto a Winbond W27E257 EEPROM, which works fine in the machine, and which has the advantage of being electronically erasable and reprogrammable.



I haven't had time to test the ROM as thoroughly as I'd like. Everything seems fine, but I wouldn't use it for mission-critical Model 100 operations. It works in VirtualT, although there seems to be a VirtualT bug which prevents Teeny+ for loading large files: VirtualT doesn't refill the loading buffer once the first filled buffer has been emptied. (I have loaded large files on the actual Model 100 using Teeny+.)

Description

LibROM started as an adaptation of the Model 100 American ROM for UK (and other non-American) machines which lack a modem, so that software and external ROMs which make calls to ROM routines will work. It grew from that to include other bug fixes and enhancements - including in-ROM disk management - replacing modem, address and schedule code to do so.



BASIC start screen
showing: ROM details & UK date formatting



MENU screen
showing: status line (with time, date, day, sound icon, current STAT settings & reassuring message) & TEENY+ item



TELCOM screen
showing: modem settings interpreted as 300 baud RS-232 settings, & file list & free space commands







TEENY+ screens
showing: load, kill, save & check functions; short filenames, auto-capitalization, OK messages; STAT toggling; DSR toggling; free space and files commands

Communications

- Removal of modem functions: LibROM always sets ports to use RS-232 and not the modem. If a STAT is entered beginning with 'M' it is interpreted and stored as the same stat for RS-232 with a baud rate of 300. For example, "STAT M8N1E" results in a STAT of 38N1E. The CALL (i.e. dialing) function has been removed.
- The ADDRSS and SCHEDL functions are removed, to make room for disk functionality.

Changes to TELCOM: the "Find" (F1) and "Call" (F2) instructions are replaced with "File" (F1), to display a list of files in memory, and "Free" (F2), to display free memory.

Localization

DATE$ processes dates in UK (non-American) DD/MM/YY format (as described in the UK manual), and the MENU screen shows the date in that format. (The storage format of the date in memory is unaffected.)

Display

- The MENU screen top line now shows: time, date, day, a sound on/off icon, the current STAT settings and a reassuring message
- Character 88H (136) is changed from a sans serif 'i' to 'π'. This matches the (incorrect) UK version of the "Model 100 Quick Reference Guide" and the Model 102 character set, and seems more useful.
- The text for July is changed from "JLY" to the more usual (and Model 102's) "JUL".



The capitalization of "Bytes free" is standardized to "bytes free".

Sound

Global SOUND ON / SOUND OFF: these functions now turn all sound on/off, not just cassette and modem sounds. This includes system beeps. Run times are unaffected: a 2 minute tune with sound on will be two minutes of silence with sound off. The MENU-screen sound icon is changed accordingly.

The PRINT Button (Sound and Scrolling)

- The PRINT button is repurposed to toggle sound on and off. This does not need to be done when on the MENU screen. The MENU-screen sound icon is changed accordingly.

Shift+PRINT toggles now toggles scrolling on and off. This does not need to be done when on the MENU screen. Note that many applications, including the MENU screen, also turn scrolling on or off, so settings may change without use of shift+PRINT.

Bug Fixes

- The ATN function is fixed to use correct (Model 102) coefficient in calculations.

The "search for file" ROM routine is fixed to allow searching for 6+2-character filenames, not just 4+2.

Disk Management: Teeny+

Ron Wiesen's Teeny disk management software is implemented in ROM and augmented to create Teeny+. An entry for Teeny+ is added to the MENU screen. Teeny+ makes the following changes to Teeny:

- the command memory is cleared before each entry to prevent unwanted results

- there is no longer a need to add spaces to filenames shorter than 6+2 characters

- (e.g. the file HI.BA can be loaded using "L HI.BA" rather than "L HIssss.BA", where 's' is a space)

- commands are automatically capitalized so a mixture of lower- and upper-case characters can be used on entry (e.g. "S fred.do", "s FRED.DO", "S fReD.dO" are all valid and all execute "S FRED.DO")

- there is no longer a need to supply a dummy filename for commands which do not require a filename

- Teeny+ sets the communications STAT to 98N1D when executed, and displays it to the user, but saves the existing system STAT beforehand, so that other STAT values can be used in Teeny+, if required
- the displayed STAT value has a suffix of 'd' if a DSR check is required, and 'x' if not
- the 'B' command toggles between 98N1D and the saved system STAT, and displays the newly active STAT

- exiting Teeny+ restores the system to the saved system STAT (rather than leaving as 98N1D)

- the requirement for a DSR check can be toggled with the 'D' command; the new setting is displayed

- commands have been added to list files in memory ('F') and display empty (free) space in memory ('E')

- the prompt text has been updated to reflect changes from Teeny

- the error message text has been updated to report "msg" rather than "Err" as some messages are not errors

- the K, L & S functions now report an "OK" message when executed successfully - the command to exit Teeny+ is now 'M' (for "MENU")
the command to check for the existence of a file on disk is now 'C' (for "Check")
(In fact, just like Teeny, Teeny+ interprets any command which is not K, L, S, B, D, E, F or M as "check file on disk".)

Full list of commands:

C file - check if file exists on drive
K file - kill (delete) file on drive
L file - load file from drive to local memory
S file - save file from local memory to drive
B - toggle baud and other settings (STAT) between saved system STAT and 98N1D
D - toggle requirement for DSR on ('d') and off ('x')
E - display empty (free) space in local memory
F - display list of files in local memory
M - return to MENU (exit Teeny+)



2022 Competition Wrap-Up
George Phillips

Half of last year's competition was to demonstrate the best compression of the Wordle dictionary and answer list. The programmers who entered have all been gracious enough to share their source code. I've taken some time looking over their code and will talk here a little bit about general ideas in compression and give my understanding as to the theory of operation of each entry.

Before we continue I must apologize to Don Barber who submitted a very good 6809 entry which slipped between the cracks and didn't get reported in the results. He was able to get the size down to 21664 bytes which put him solidly in 4th place. Sorry about that, Don, and thanks for taking the time to enter.

The Wordle dictionary (as was set for the competition) is 12,960 five letter words. A subset of 2,313 of those are specially designated as answers. The game will pick one of those words for the player to guess. To be true to the original game the answers must be remembered in order. Roughly speaking, the game would look at the number of days that have passed since it was released and use that as the answer number to present. This meant that every player would be guessing the same answer on a given day and would get a new puzzle every day.

As you might imagine, it is easier to compress a list of words if the order doesn't matter. There's simply less information to record. One way to approach things is to compress the entire 12,960 word dictionary in whatever order is most favorable. Then represent the answers as an ordered list of the dictionary entries by number. Or you could compress the 10,647 valid-but-not-answer words as best you can and treat the remaining list of 2,313 answer words as a separate problem. In either case the unordered list of words is much bigger and easier to compress and thus a good place to focus our effort. We have the most to gain and the best chance of realizing those gains.

Let's start with the full 12,960 word dictionary. If you look at a text file containing one word one per line it'll be 77,760 or 90,720 bytes in size. You'd think that it should be 64,800 bytes because there are 12,960 words at 5 letters per word. The extra bytes come in due to the characters used to indicate the end of each line. Due to a long standing tradition of incompatibility we in the computing world have never come to a universal agreement of how to denote the end of a line. Windows and their brethren use 2 characters, unixy computers use 1. Despite the disagreement the extra data is necessary for a text file as the lines can vary in length.

In our case we know every word (and thus every line) is 5 letters. We can drop all the line endings and still easily be able to read out the word list. Well, a computer can. I'm not so sure I can quickly rattle off the words in “polypfinalmushyreplygrain”.

While it was a fairly simple observation it was nonetheless a big saving dropping at least 12,960 bytes and getting us down to the expected 64,800. Or at least in the ballpark. After all, we still need to remember how many words are in the list and there is still the code access it.

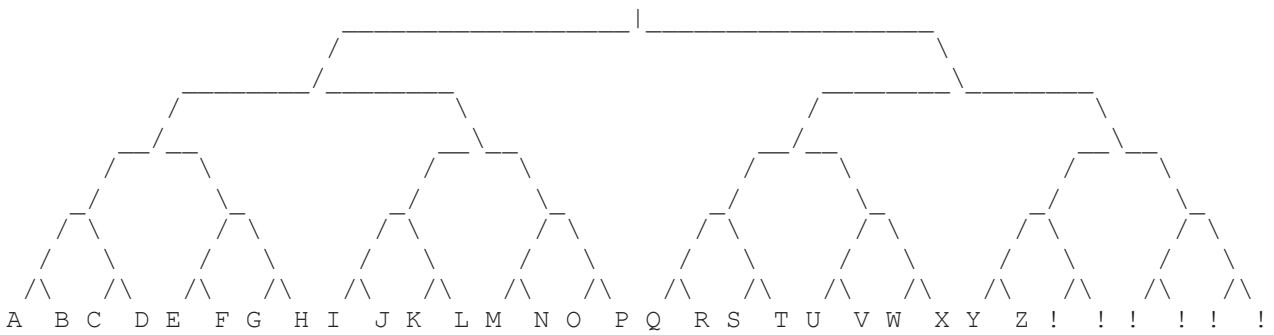
For our next step let's ask if one byte per letter is the best we can do. Considering that a single byte can store 256 (2^8) values and we only have 26 possible letters it sure sounds like a lot of wasted space. With a little trial and error we can figure out that 5 bits is enough. A single bit can store 2 values, 2 bits can store 4 values and so on. In general, n bits can store 2^n values. So it goes that 5 bits can store $2^5 = 32$ values. Mathematically we can get the result directly by calculating $\lceil \log_2(26) \rceil$. Those broken square braces mean “the ceiling” which is a fancy way of saying “round up”. If your calculator doesn't have base 2 logarithms you can replace $\log_2(n)$ with $\log(n)/\log(2)$. $\log_2(26)$ is just a smidge over 4.7 which rounds up to 5.

At 5 bits per letter we're using a mere 25 bits per 5 letter word. That's much better than the 40 bits at 1 letter per byte we were using. We could do the math to figure out how big the compressed data will but it's just a well to keep thinking in terms of bits per word. As long as we think the extra code won't be bigger than the savings we're OK. For 12,960 words even saving 1 bit per word is 1,620 bytes which is an awful lot of assembly code.

Stepping through memory 5 bits at a time is more complicated than going a byte at a time. And certainly it's slower but speed isn't a problem in this competition so never mind that. The Z-80 code required is only about 20 or 30 bytes. Take a look at Mathew Boytim's entry for an example. `shiftb` is the subroutine that pulls `B` bits into `E` register. And just after `fwini:` you'll find the code to initialize the bit reading.

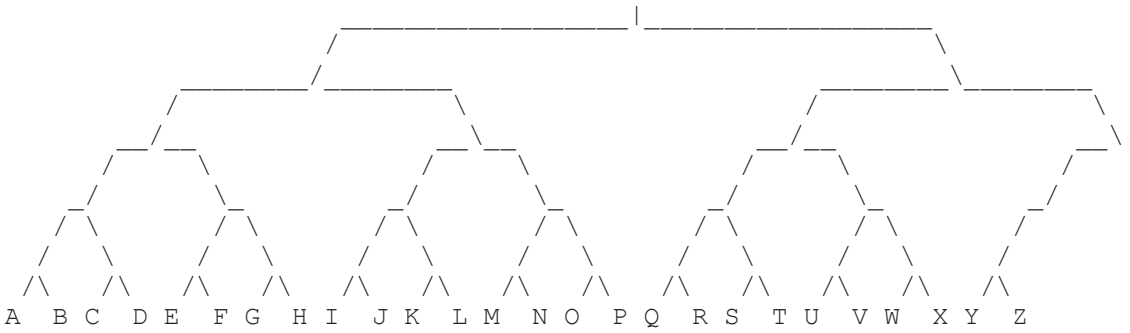
Can we do improve on this? Well, it is clear that 5 bits is more than we need. Only 26 of 32 possible values are used. One idea would be to use the extra 6 values to mean something special. For example, maybe make 27 correspond to some common two letter sequence like “th” or whatever the most common letter pair that occurs in the list. That'll be more assembly code, of course, but feels like we'd come out ahead. For that matter, maybe we should go back and look at those byte values. They wasted a whole lot more but with a bit of thought we could find good uses for the 224 unused codes. It is a fruitful approach and the code will be faster and by extension smaller working a byte at a time. See Kyle Wadsten's entry for a good example of adding special codes.

For now let's confine ourselves to better packing. Instead of thinking each letter as a 5 bit chunk let's lay the codes out in a tree. The idea here is that we start at the root of the tree and read a bit. If the bit is 0 we go left otherwise we go right. That would look like this:

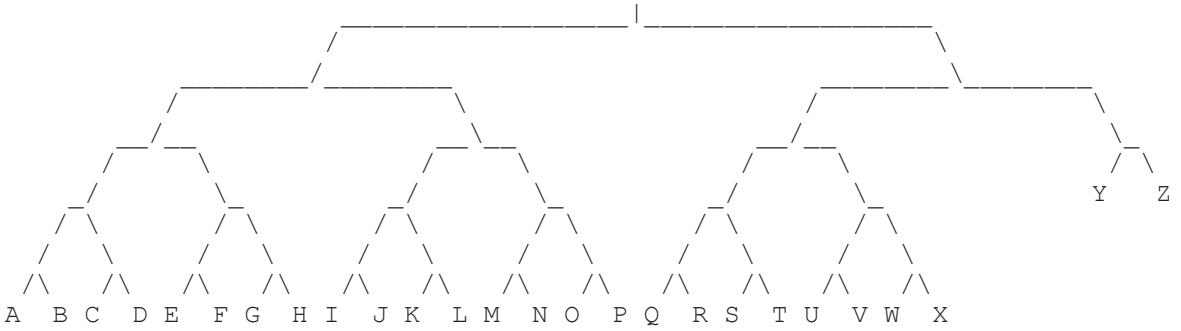


If we see 5 zero bits in a row (00000) we got all the way down the left side of the tree and end up at 'A'. If the next 5 bits are 10101 we wind our way down to 'V'. Going too far to the right of the tree gets us into the unused area. Pure wasted space.

Let's chop out those unused paths to make our tree neat and tidy:



The actual waste now becomes clear. As we head to either 'Y' or 'Z' we hit two nodes where there is no choice as we always turn left. We may as well just move 'Y' and 'Z' up two levels:



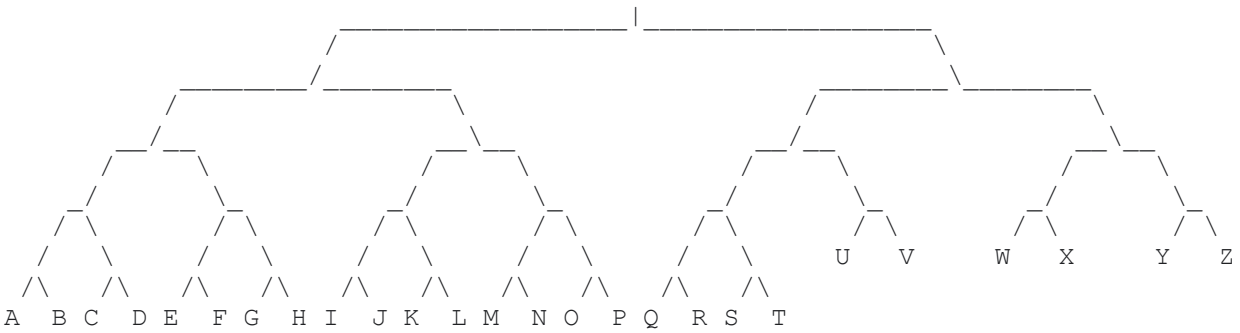
Nice picture, but so what? Well, if we follow our rule we find that 'A' through 'X' still have 5 bit codes but 'Y' and 'Z' have three bit codes: 110 and 111. It feels like this new tree should work but does it really? We can no longer just read 5 bits but need to decide, somehow, if we should read 5 bits or 3. That could be a lot of code or control data.

Looking carefully at the paths it will become apparent a simple rule will suffice. Notice that the letters 'A' .. 'H' start with two zero bits. 'I' .. 'P' start with 01 and 'Q' .. 'X' start with 10. Meanwhile, 'X' and 'Y' start with 11. All we need to do is:

> read 2 bits
> if they're 11 then read one more bit to pick 'Y' or 'Z'
> otherwise, read 3 more bits and use the last 4 bits to choose the 16 letters 'A' .. 'X'.

Will this be worthwhile? There will definitely be less than 5 bits/character. 24 of the letters still have 5 bit codes but 2 of them are 3 bits long. On average there is (24 * 5 + 2 * 3) / 26 bits per letter which is about 4.846 bits/character. 0.154 bits/character saved doesn't sound like a whole lot but it works out to 1,247 bytes.

In fact, we can do a little bit better by balancing the tree. Push 'Y' and 'Z' down so that 'U', 'V', 'W', and 'X' can have shorter codes.



Now we average (20 * 6 + 6 * 4) / 26 = 4.769 bits/letter. The improvement is slimmer but I still think even with a bit of extra code we'll come out ahead.

In either case there's a more important conceptual breakthrough is we've moved from using an integer number of bits per character to a fractional number. It's easy to get stuck on the idea that the smallest unit in the computer is the bit. Not true! Here we are with fractions of a bit.

It really gets you thinking about how we got to 5 bits in the first place by rounding up $\log_2(26)$ to 5 from its actual value of about 4.70044. Maybe that wasn't just a step in figuring out how many bits we needed. Maybe that is *exactly* the number of bits we need. I hate to gloss over the whole discipline of Information Theory but in short, yes, it really is! This is good news; we can do better than 4.769 bits/letter. The bad news (and you knew there was bad news) is that the theory isn't telling us how to get that value.

As it turns out, base 26 encoding of the all the words will get to the 4.7 “minimal” bits per letter predicted. In my example program I encoded each 5 letter word as a base 26 number. HAZEL has letter numbers 7, 0, 25, 4 and 11. As a base 26 number it is $7*26^4 + 0*26^3 + 25*26^2 + 4*26^1 + 11*26^0$ or 3,215,847. The largest possible word Is ZZZZZ which is 26^5-1 or 11,881,375. $\log_2(11{,}881{,}375)$ is about 23.5 so treated as numbers each letter can fit in 24 bits. That works out to 4.8 bits/letter; we get close to the best but lose out because we rounded up to 24 bits.

To get to 4.7 bits/letter we can't stop at a single word. Instead, we must encode all the words as one big number. How big? Well, 12,960 words gives us 64,800 letters. The single number holding all the words will be less than $26^{64,800}$ and the number of bytes will be $\log_{256}(26^{64,800})$ which rounds up to 38,074 bytes. The bits per letter is 38,074*8/64,800 = 4.7005 which is pretty darn close to the theoretical minimum of 4.70044 bits/letter.

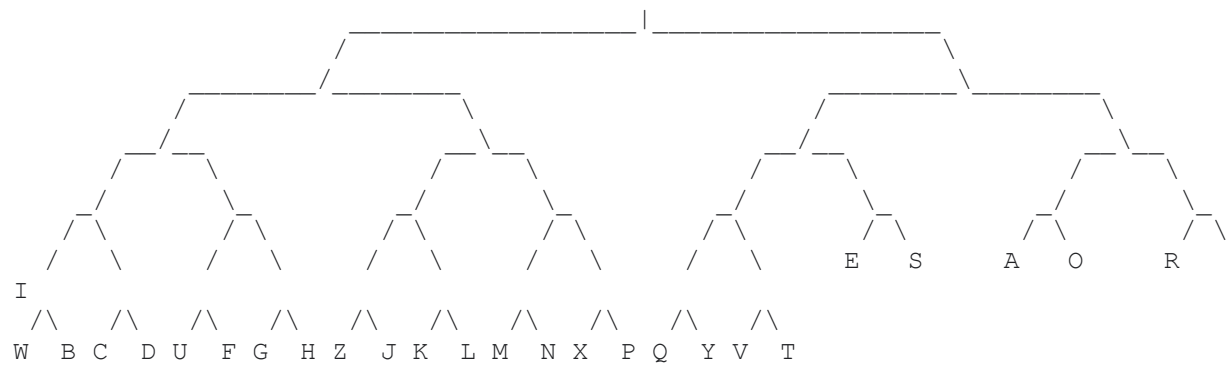
The theory looks good but the practice has a few problems. Every letter extracted requires dividing a 38K digit number by 26. The number will get smaller as we go but roughly speaking that's on the order of 38K * 38K operations. Even if each division step were a single instruction that's an awfully long time to search through the dictionary. Even worse, we need to keep track of the quotient after each division. We'll need 38K of memory for the dictionary and 38K working space as we extract the letters.

That won't fit on a Model 1 and might barely work on a Model 4. At best it might be plausible on the Model 16's 68000.

There may be some sort of middle ground. Base 26 encoding 5 letters at a time wasn't as efficient as the packed tree method but maybe 100 letters at a time would get there without being as expensive. Still, I think it best at this point to consider this a dead end and look to see how the tree method can be improved.

Right away there is a small problem. I calculated the bits/letter as an average. That's fair enough for an approximation but the dictionary is fixed. There's no need to guess at the number of bits since it can be calculated exactly. I'll just whip up a little script to count 5 bits for 'A' .. 'T' and 4 bits for 'U' .. 'Z' (… a bit later ...) and it's 4.89 bits per letter. Uh-oh, that's worse than predicted!

Why is that? If I print out the number of times each letter appears it becomes clear – while 'U' and 'Y' are reasonably common, 'V', 'W', 'X' and 'Z' are some of the least common letters. Yeah, I know, this is hardly a revelation and I'll confess I glossed over the issue for the sake of the story. Most importantly, there wasn't even a need to guess the expected frequency of each letter when the data is known. And having written a little program to count letters it's really easy to show that the six most common letters are 'E', 'S', 'A', 'O', 'R' and 'I'. [Is it mere coincidence that Saoirse Ronan has the most common letters in her first name]. What if we arranged the tree so that those letters used the short 4 bit codes? Sounds like a real pain shuffling all the branches around so instead I'll just swap the leaves around, 'E' ↔ 'U', 'S' ↔ 'V' and so on:



Will this have much impact on the code side? Not really; at worst it will be a few bytes of code to look into a 26 byte table to remap the letters. By choosing the most common letters for the short codes the bits per letter drops down to 4.512. This is excellent but how can it be better than the 4.7 value from theory?

As is usually the case, the math is good but the assumptions have led us astray. That $\log_2(26)$ business assumed that each letter is equally probable which clearly isn't the case. And before we get too mad, we were mostly looking at situations where a letter or a word would be packed into some fixed size unit. That ease of indexing can have a lot of practical value and is nothing to be ashamed of but here we want the best compression.

Having found a loophole of sorts the next thing to ask is if we can exploit it more. The current tree structure was arrived at just to avoid wasted code values.

The occurrences of each letter vary quite a lot. 'E' and 'S' appear about 6600 times each where 'J' and 'X' are only 290 and 'Q' a paltry 112. Surely we could change the tree so that 'E' and 'S' have the shortest possible codes and letters like 'Q' and 'X' longer ones? Feels like we should come out ahead, right?

Yes, yes we can. This idea of building a tree where the most common letters get the shortest codes and uncommon ones get the longest is called Huffman Coding. Read about it here:

https://en.wikipedia.org/wiki/Huffman_coding

Building a Huffman tree based on the occurrences of each letter I got these codes:

```
S 000    G 00110   N 01111     Q 011111101
E 100    H 01110   V 010110    X 111111101
T 0010   P 01001   W 110110
L 1010   C 11001   F 011101
I 0001   M 11110   K 011111
R 0101   Y 01101   B 111111
O 0011   D 01011   Z 0111101
A 0111   U 11011   J 01111101
```

That works out to a pleasing 4.347 bits/letter. It amounts to a 1,336 byte saving over our previous best. We will pay some overhead due to more complex assembly code and some data to describe the tree. As you might imagine there are techniques to minimize the space required to store the tree; start here for some useful ideas:

https://en.wikipedia.org/wiki/Canonical_Huffman_code

Huffman coding is a very good technique and was used in many of the competition entries. It even promises to generate an optimal code for an alphabet given the input probabilities. Mind you, that's a story we've heard before. Are there hidden conditions on that optimality?

Indeed there are. Huffman codes are restricted to outputting a bit at a time. This means they can only match probabilities that are an inverse of a power of 2: $1/2^k$ for integer values of k >= 0. Unless the frequency of every letter is one of these values there will be some loss of compression efficiency. Information theory once again can tell us the minimum number of bits for encoding a letter depending on the probability, P, of that letter appearing. This is called the entropy of the letter and is given by $-\log_2(P)$.

We can take the probability of a letter appearing as the number of times is appears divided by the dictionary size. For example, 'E' and 'S' have a probability of about 6600/64800 or 0.109. If we had a way to encode at the entropy lower bond for each letter we'd reach 4.312 bits/letter. Not a huge difference but amounts to around a 280 byte saving.

The other inefficiency is how we compute the probability of a letter. The total number of occurrences divided by the number of letters in the dictionary is all well and fine but takes no account of context. If the letter 'Q' appears than the next letter is 'U' with probability nearly 1. An 'H' will be more likely than usual if the previous letter was 'T'. In general the idea is to create a model of the input that predicts the probability

of each letter coming next. It could take into account the previous letter or all the previous letters in the word or even the previous word.

That kind of idea can work well but there are two challenges. One is that each context will need its own Huffman tree – say 26 trees; one for each possible previous letter. The compression gains will be offset by the cost of storing all those trees. The other is the inefficiencies in the Huffman code. Suppose our predictions of the next letter get very good and we can confidently state the next letter with 0.75 probability. Ideally that would be encoded as $-\ln_2(0.75) = 0.415$ bits but the Huffman code will always use one bit. Our model is doing a great job but all its work is lost in the encoding.

Happily there is a coding technique that can get very close to the entropy of a letter. It has the slightly strange name of “Arithmetic Coding”. I find the theory of it fairly easy to understand but must admit to not yet being able to grasp how it is implemented. I leave the interested reader the Wikipedia article on it as a start:

https://en.wikipedia.org/wiki/Arithmetic_coding

Alan Petrofsky uses something like it in his competition entry: a particular type of Asymmetric Numeral System or ANS which has the coding efficiency of Arithmetic Coding with the speed of Huffman coding.

At this point we've reached the bleeding edge of data compression theory. The best data compression algorithms involving combing a model that predicts what data follows very well with a good entropy encoding like Arithmetic Coding. However, there are other very good compressors that work in entirely different ways. Often times on the simple notion of data literally repeating itself. Generically these are called LZ compressors and work by emitting codes that say “look back N bytes and repeat those K bytes”. They're very fast and have the advantage of dynamically adapting to the data as they read it. The Deflate compression in .zip files use this technique. They're not as well suited for our needs since a dictionary doesn't have much that is literally repeated and we know exactly what we're compressing up front. There's no need to adaptively compress when we can jump in with the best strategy immediately.

The point being that despite the length of this little journey into data compression I've only scratched the surface. And for every generic data compression that are many more suited to particular types of data that has its own patterns to exploit. There are so many ways to crack this nut.

To end this I'll mention one more approach as it comes up in so many entries. I'll refer to this as a delta list. Convert all the words into numbers using base 26 encoding and put them in ascending order (which will be the same as alphabetical order). Now write out the list as the numeric differences between each word. For the most part the differences will be small – the jump from HAZEL to HAZER is only 6. The biggest jumps will come when the first letter changes like from HYTHE to IAMBI but even those won't be much bigger than $26^4 = 456{,}976$. That fits in 19 bits giving us a decent 3.8 bits/letter. Now use some technique to write the common short differences with fewer bits and the efficiency will go up considerably. See the entries themselves for ideas on how to do so.

The Entries

Here's an overview of how each of the entries work. It's based on my look at the code so any mistakes are mine. Presented in no particular order. You can get all the source code at:

http://48k.ca/comp2022.zip

Don Barber

This was the only submission written in 6809 assembly. The answer list is stored separately from the rest of the words in a straightforward base 26 encoding so 3 bytes per word. The rest of the words are stored as a delta list with a twist. Before being alphabetized the letters of each word are permuted like so:

B C D E F

F E C D B

This has the effect of reducing the average size of the deltas and thus decreases the size of the compressed data. The numeric value of the first word is stored in the code which checks that for a match then proceeds to read a list of deltas. Each delta is a fixed-size 5 bit number giving the number of bits in the delta itself. A delta of 0 is used to terminate the list.

Mathew Boytim

All the words are stored in a single dictionary in alphabetical order as a delta list. The answer list is an array of 14 bit indices into the full dictionary. The deltas use intricate scheme that adds a delta value onto each letter individually. Letter 2 gets a one bit value added, letter 3 a 2 bit value, letter 4 a 3 bit value and letter 5 a 4 bit value. This amounts to a 1+2+3+4 = 10 bit delta value. But since the delta for letters 2, 3 and 4 are often zero there is an initial 2 bit value indicating the first letter with a non-zero delta. This leads to 4 possible deltas:

00 1-bit 2-bit 3-bit 4-bit – 4 values added to letters 2, 3, 4 and 5.
01 2-bit 3-bit 4-bit – 3 values added to letters 3, 4 and 5.
10 3-bit 4-bit – 2 values added to letters 4 and 5.
11 4-bit – 1 value added to letter 5

4 bits isn't enough to always reach the next 5th letter. And that might also be the case for the other letters. When that happens the first delta value in the list will be set to 0. The deltas are added as normal but the decoder knows this is an intermediate value and doesn't consider it a valid word. Instead it gets the next delta to move on to the next word.

The delta values are added onto in wraparound (aka modulo 26) fashion. Adding 3 to letter 24 ('Y') yields letter 2 ('B'). The delta for letter 2 is a bit different. Whenever it wraps around it adds 1 to the first letter. That's how the encoding avoids storing a delta for letter 1 which would be 0 most of the time and 1 only 26 times.

There's another special case where if the delta to letter 2 is 0 then 2 is added onto letter 2 with the carry propagating to letter 1. But the delta is still treated as a 0 and so the word generated is considering an intermediate word and the program moves on to the next delta.

Shawn Sijnstra

Here the entire dictionary is stored as a single delta list. The answers are stored as 14 bit indices into the dictionary. The deltas come in 6 different sizes: 3, 6, 9 12 and 16 bits. The deltas avoid redundancy by starting off at the maximum value for the smaller delta. A 3 bit delta will naturally indicate differences of 1 through 8. A difference of 0 is not needed at it would repeat the previous word. Thus the 6 bit delta starts at 9 and goes to 72 instead of a 0 to 63 range. The deltas are packed into 4 bit chunks (nybbles) and use 1 to 4 bits of the first nybble in denote the size of the delta.

The delta values themselves are moved around a bit as some longer deltas are more common than shorter ones. For instance, a skip of 3 is stored in a 6 bit delta with a skip of 10 taking its place in the 3 bit deltas. Here are the possible delta values and how they are encoded:

0xxx – adds 1, 2, 10, 4, 5, 6, 14, 15.

10xx xxxx – adds 9, 3, 11, 12, 13, 7, 8, 16 ... 22, 168, 24 ... 27, 86, 29 .. 41, 100, 99, 162, 45, ... 48, 104, 89, 51, 52, 130, 98, 90, 56 ... 58, 156, 60, 78, 88, 63, 64, 96, 66, 67, 97, 182, 103, 84.

110x xxxx xxxx – adds 73 ... 77, 61, 79 ... 83, 71, 85, 28, 87, 62, 50, 55, 91 ... 95, 65, 68, 54, 43, 42, 101, 102, 70, 49, 105 ... 129, 53, 131 ... 155, 59, 157 ... 161, 44, 163 ... 167, 23, 169 ... 181, 69 ... 584.

1110 xxxx xxxx xxxx – adds 585 plus the 12 bit value giving 585 .. 4680.

1111 1101 xxxx xxxx xxxx – adds 164,248 to the 12 bit value.

1111 xxxx xxxx xxxx xxxx – adds 70,217 to the 16 bit value.

It may look like there is quite a bit of data required for 3, 6 and 9 bit offsets but they are represented quite compactly by a list of values that swap out from the “natural” offsets.

There is one other special case to mark the end of the list:

1111 1100 – end of list

I did a fair bit of decoding to figure out the offsets so it is quite possible that I missed a case or have the offsets wrong. Look at the original code to be sure.

Bill Kewley

A Huffman code is used for the letters. The answer list is stored separately and encodes a full 5 letters per word. The rest of the words are stored as 26 lists of the 4 letter suffixes for each word. Each list starts out with a 2 byte count of the number of suffixes. And the lists are put in alphabetical order so there is no need to store the first letter. This makes for quite a savings as the first letter in an alphabetical list is highly predictable.

Kyle Wadsten

The answer list is stored separately as base 26 encoded words. The rest of the words are stored with an optimized base 26 packing. The words are broken into 3 letter prefixes and 2 letter suffixes. The prefixes are base 26 packed into two bytes with a maximum value of 17575 = 26^3 – 1 or $44A7 in hexadecimal. The suffixes come in two forms. One is a literal base 26 packing into two bytes which will have a maximum value of $2A3 in hexadecimal. Or it can be a single byte with the high bit set referencing one of the 128 common suffixes stored in a table. The dictionary must start with a prefix code but then the suffixes and prefixes may occur in any order. A suffix means we have a valid word with the current prefix and a prefix replaces the current prefix. Compression is gained because a common 3 letter prefix only has to occur once and common word endings are represented in a compact fashion.

In order to distinguish the prefix codes from the suffix codes the prefix codes have $4000 hex ORed to them. The codes are stored in big endian fashion (high byte first). The code can read a byte and know what it has based on the high two bits of the byte:

01 – a 3 byte prefix.
00 – a 2 byte suffix.
10 or 11 – a single byte common suffix; the bottom 7 bits are the table lookup.

There is one last wrinkle. The 3 letter prefixes can be as large as $44A7 so after the OR with $4000 you can't know if the prefix had that bit set or not. This is handled by putting the prefixes in alphabetical order and noting the address in the table where the first prefix >= $4000 occurs. The address of the prefix is checked and if it is below that transition address the code knows that the bit must be cleared otherwise it can be left alone.

Peter Phillips

The entire dictionary is stored as a delta list. Each delta in the list reports the number of bits required to represent the delta and whether or not the next word is in the answer list. The number of bits required is one less than the actual number since the top bit must necessarily be 1. A zero bit delta represents a 1. A one bit delta can either be 10 or 11 in binary or 2 or 3. A two bit delta covers numbers from binary 100 to 111 or 4 through 7. These bits are added to an answer bit and put into a Huffman tree to minimize the space required to represent them as deltas with smaller numbers of bits tending to be more common and ones with the answer bit set are relatively rare. The delta list is thus a Huffman code that can be looked up to find the answer bit and the number of bits in the delta which come immediately after it.

When validating a word the delta list is traversed without paying any heed to the answer bit. As another space saving measure, the Huffman tree is stored in a compact form requiring only 35 bytes and expanded at run time to Z-80 code to perform the Huffman decoding.

The answer bit can be used to extract all the answers but only in alphabetical order. A permutation is used to get them into required order. The permutation is conceptually a list of 2,313 numbers that say which answer number should appear in each position. These only need to be 12 bits in size since that's enough to refer to any of the 2,313 positions ($\log_2(2313)$ is 11.18). But these numbers can be reduced as we go along since each answer put in place means one less choice. The number of bits goes down to 11 when only 2047 possibilities remain and eventually drops to 0 bits as the last remaining possibility is the answer.

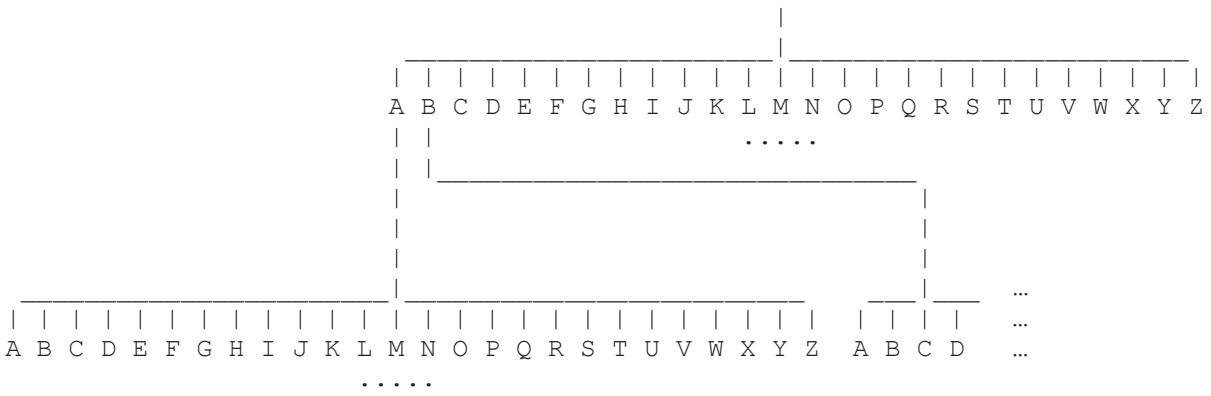
The code starts by unpacking the entire permutation list and then goes through the list swapping the current code with the one indicated in the permutation.

Once the list has been put into the correct order it only remains to look up what position answer N is in the alphabetical answer list and then scan the dictionary while counting down as we see each answer bit go by.

Alan Petrofsky

I'll be honest here in that I don't have a great understanding of this entry and this is more a summarization of the comments than independent thought. Nonetheless, it stores the entire dictionary in a list and the answer list is a compressed array of indices into the dictionary. As we've seen this normally requires 14 bits per index. Here the upper 6 bits are Huffman coded to reduce their size while the lower 8 bits are represented literally. I suspect this saves space as the upper bits can only be from 0 to 50 which is quite a bit smaller than the 64 values you could put in 6 bits.

The dictionary is represented as a walk over a 26-way, 5 deep tree of all words. The full tree is quite large. Here are two levels of it:



The tree is traversed depth first. You start at the top level with an empty word and ask if any words start with 'A'. Since there are you write down a 1 and, note that 'A' is the first letter and move down a level. Now you ask if any words that start with 'A' have 'A' as their second letter. Yes, so write another 1 and note we have two letters 'AA'. Are there any words that start with 'AAA'? No. That means the walk can stop going down and we write a 0. Since there are also no words that start with 'AAB' through 'AAG' we write 6 more zeros. At 'AAH' there is a word so we write a 1 and consider 'AAHA' which isn't valid so we write a 0 and will keep writing zeros until we hit 'AAHE'. We write a 1 for that and then 3 zeros because 'AAHEA', 'AAHEB' and 'AAHEC' are not words. Then we write a 1 because 'AAHED' is a word. And since we're at level 5 we know this is a valid word and there's no going down and simply proceed on to 'AAHEE'. None of the rest of the letters work so we write 22 zeros in a row. Having done all 26 letters we go back up and consider 'AAHF'. Once again no letters work and we do another 21 zeros in a row and go up a level and noticing that 'AAI', 'AAJ', 'AAK' are not the first 3 letters of any word so we have 3 more zeros written down. But 'AAL' does start a word so we write a 1 and consider 'AALA'. This continues on in a similar fashion (the next word will be 'AALII') until we get to the last word 'ZYMIC' and emit a bunch more zeros to “pop up” back to the top of the tree.

This is a fairly decent way to represent all the words as it only walks through parts of the tree necessary to reach all valid words. Many branches of the tree are ignored as they are nothing but dead ends.

Alan's code is a bit different as he starts at the end of a word and in goes in reverse alphabetical order. This is due to his compression scheme naturally producing the output in reverse order. I think the walk might be shorter working from the ends of the words but haven't checked that.

Since using Huffman coding is of no use on a two letter alphabet, a variant of ANS (Asymmetric Numeral System) coding called ABS (Uniform binary variant) is applied. Here's a reference for that:

https://en.wikipedia.org/wiki/Asymmetric_numeral_systems

Suffice to say that like Arithmetic coding it can approach the entropy of a symbol's probability when encoding. Or in other words it can use less than a bit to encode a symbol (the 1's and 0's of the tree walk, in this case).

To give better compression the probability of a 0 or 1 appearing is better predicted than just a simple average of the relative number of 0's and 1's. Instead quite a large number of tables are consulted based on the position in the tree and what letters precede the current one.

At the top level of the tree no prediction is needed since the dictionary has words that end in every letter (recall that we start at the end of the word and consider the 5th letter first). At the next level of the tree (the 4th letter) there are two predictions based on whether the 5th letter was a consonant or a vowel. The 3rd letter has 26 predictions based on the 4th letter. The 2nd letter has 2 * 26 predictions based on the 3rd letter and whether the 4th letter was a consonant or a vowel. Finally, the 1st letter has 2 * 26 * 26 predictions by having different ones for each of the 2nd and 3rd letter combinations and two for whether the 4th letter was a consonant or a vowel.

It's a pretty elaborate scheme and was a refinement of two earlier schemes which used fewer previous letters to make the prediction. Each prediction is the probability that the number is a 1 and is stored as a single byte covering 1/256 to 255/256. There are 2 + 26 + 2 * 26 + 2 * 26 * 26 = 1,432 prediction contexts requiring the same number of bytes of storage.

Unlike most of the entries, Alan chose to use zmac macros to produce the compressed data from the word list. Most often this work is done in an external program where it is considerably more convenient. I'll leave it to the curious to look for themselves to see how that rather unlikely and startling feat is accomplished.

All For Now

Phew, what a dazzling array of inventive and ingenious competition entries. I hope you found them as interesting as I did. While this will do for now there are still some developments to come. I hear Alan is working on a version of his program that will fit in 16K and run fast enough to be practical. I also secretly hope that someone out there will read this and plot some way to compress the dictionary even more. I'll be sure to let you know if there are any new developments. In the mean time, be sure to check out this year's competition!

TRS8BIT 2023 Competition

0K Z-80 Disassembler

A disassembler is a great addition to a debugger. It can show actual instructions instead of hexadecimal opcodes which, if you're like me, are quite tedious to disassemble by hand. Since a debugger must coexist with the program being debugged it should be as small as possible lest you get caught without enough space for the program and the debugger. Waste not, want not, as the old saying goes.

A Z-80 disassembler literally fitting in zero kilobytes is a tad unlikely. Nonetheless, there's every hope it can be squeezed into something less than 1024 bytes and will round down to zero kilobytes.

The smallest entry based on code size and run-time data space used will be the winner. You can use Z-80, 6809, 6803 or 68000 assembly to program your disassembler subroutine. All entries will be in source code form and will be published at the end of the competition for all to learn from and admire.

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

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

For details on exact requirements, other rules and a running leader board please consult:

http://48k.ca/comp2023.html

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

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

INNOVATIVE

TRS-80 SOFTWARE

FROM THE PROFESSIONALS

VISION LOAD

VLOAD your Basic tapes rather than CLOAD them and SEE the program on the screen, byte for byte as it loads in from the recorder! After a VLOAD, the program may be executed in the normal way so having seen a good load you are then able to use it. No more having to load in a whole program before you are sure of the load. If the recorder volume is too high the display will be increasingly made up of large graphic characters. If the volume is too low there will be no data appearing on the screen. The distinction between both extremes is very pronounced, so uncertain loads are a thing of the past! As the program loads you will be able to see the ASCII characters, so identification of the program is immediate if you are not sure which program is on the tape. VLOAD only occupies three hundred bytes and resides at the top of memory. Once loaded under the System command it may be used as often as you wish. Another advantage is that as the capacity of the screen is 1K bytes, it is easy to see how long a program is by counting the number of times the screen is filled. VLOAD is supplied with two copies of the 16K, 32K and 48K versions on cassette.

Vision Load on cassette .....£11.50 + VAT = £13.23 75p P & P

ENHANCED BASIC

EBAS is an important addition to Level II Basic. It contains about 25 new commands including: software control of the cassette motor, automatic entry to the Edit mode when typing a line, instantaneous recall of the last entry, the ability to place a block at the end of a Basic program so that you can add another to it, Hex constants and conversions, insert breakpoints in a Basic program without loosing variables, single step through a program, unlimited USR calls, instring function, line input function, define function and a special command for loading cassettes which is almost independent of the volume control setting. These are powerful commands but the primary importance of EBAS is that it provides a bridge between Basic and machine code programs. If you are familiar with Basic and want to branch out into machine code, this is the program for you. Machine code may be entered as part of the Basic program with register contents returned to Basic variables! Also a Monitor is on board so that you can inspect, change and search memory, all with a return to Basic. System tapes may be created and loaded to tape. Until now there have been Basic programs and there have been machine code programs; at last you can now combine them and manipulate machine code from Basic.

Enhanced Basic on cassette ..... £24.00 + VAT = £27.60 75 p P & P

EMPEROR

Occasionally a game comes along which is of such immensity that it is almost impossible to describe. Such a game is "Emperor". It is entirely a game of strategy, played on a graphic map of the Roman Empire as it was in the first four centuries A.D. The player takes the part of the Emperor and he must pit his wits and forces against invading barbarians, rebellious provincials and treacherous Roman Generals. Even the Plebs of Rome will have to be placated with bread and circuses if the Emperor is to keep his head and his throne. If he can last out for the first eight years of the game, he is judged on the state of the Empire at the end of that time. There are three levels of play. Depending upon his choice, the Emperor has to guide the Empire through the first, third and fourth centuries. To win in the first century he must expand the Empire by two provinces, in the third he must maintain his Empire intact and in the fourth he must lose not more than two Provinces. For each Province the player is given three items of information, the number of loyal Legions, the number of revolting legions and the number of Barbarian Invaders or Local Rebels. During play Legions must be raised, taxes inflicted and troops moved. The choice of Generals can be very critical - some are loyal and good fighters, some are neither. Battles must be fought and invasions repelled. All the while the citizens in Rome must be kept happy and - you must keep an eye on those Barbarians in Britannia!

Emperor on cassette ..... £13.50 + VAT = £15.53 75 p P & P

Send large SAE (27p) for our current catalogue of TRS-80 software. Add £1.85 for a binder

A.J.HARDING (MOLIMERX)

28 COLLINGTON AVENUE, BEXHILL-ON-SEA, E.SUSSEX. TEL: (0424) 220391

TELEX 86736 SOTEX G FOR A. J. HARDING



The *BEST* in TRS-80s

Call The Right Stuff

Ask for Ian
The number is +61 416 184 893

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

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







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

WHERE 8 BITS AND 16K STILL RULE OK

TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

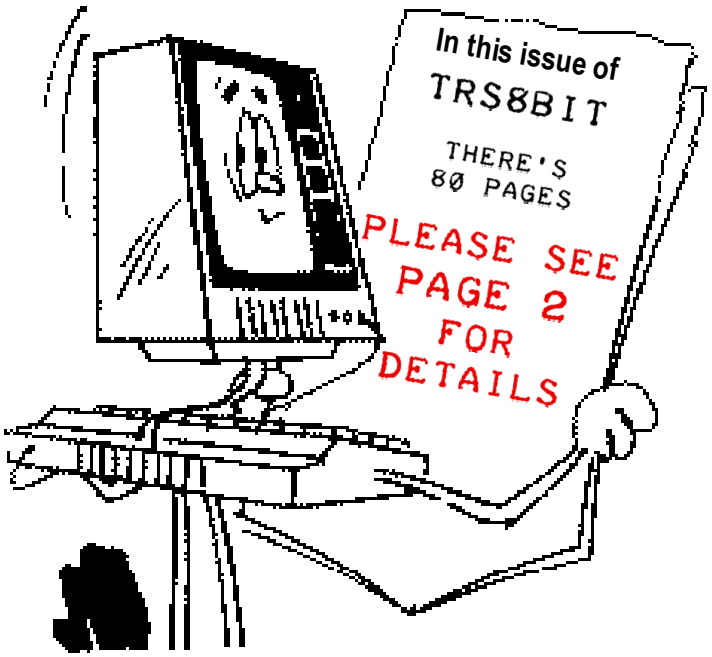


HELLO EVERYONE AND A VERY WARM WELCOME TO THE JUNE 2023 TRS8BIT NEWSLETTER, OUR 66TH EDITION!

ONCE AGAIN, THERE'S A NUMBER OF EXCITING BITS OF NEWS FOR YOU ALL.
CHECK PAGE 3 FOR THE DETAILS OF A **MAJOR UPGRADE** TO THE **TRS80GP** EMULATOR WITH V2.5.

AS YOU WILL SEE ON PAGE 65, GEORGE PHILLIPS HAS PERFORMED ANOTHER FANTASTIC FEAT FOR ALL ACULAB FANS OUT THERE, WITH THE CREATION OF LEVEL 2.5 BASIC FOR THE ACULAB.

WE'VE BEEN LUCKY ENOUGH, PURELY BY COINCIDENCE, TO HAVE A NUMBER OF UNIQUE, BASIC, ONE-LINERS SENT IN. THE ONE WHICH REALLY CAUGHT MY EYE, (NO NOT THE BIRTH DATE MAGIC SQUARE ONE!) BUT THE 'VAN ECK' INTEGER SEQUENCING PROBLEM. I RAN THE PROGRAM ON, OF COURSE, TRS80GP, BUT FOR A LARGER TRIAL, YOU'LL NEED 48K AND, MOST DEFINITELY, THE 'TURBO' OPTION SWITCHED ON!



OK, LET ME REMIND YOU...

Tandy Assembly 2023

HAVE YOU BOOKED YET?
IT'S HAPPENING IN
SPRINGFIELD, OHIO
29SEP23 - 01OCT23

IF THIS IS GOING TO BE YOUR FIRST VISIT, YOU MAY FIND MAV'S ARTICLE, ON PAGE 46, REALLY USEFUL.

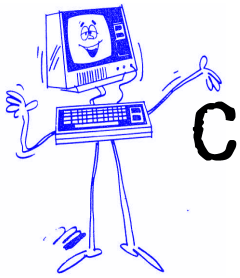
THERE'S A FULL REVIEW OF THE LATEST BASQUE TOURNAMENT FROM EGOITZ. IT'S SO NICE TO SEE THE INTEREST IS STILL THERE IN SPAIN. NO DOUBT THE LATEST RELEASE OF TRS80GP WILL BE OF INTEREST TO THEM.
ALSO, I HOPE YOU NOTICE THAT IN THE REGULAR FEATURE 'TWWW', GIL SPENCER'S ARTICLE FROM 80-MICRO HAS STARTED THE BALL ROLLING (AGAIN), FOR ANY LEVEL 2 CASSETTE BASIC USERS. THE 'NEW' BASIC 2.5 IS AVAILABLE FROM THE DOWNLOADS PAGE AT WWW.TRS-80.ORG.UK.
VINCE OTTEN IS STARTING A NEW SERIES FEATURING SOFTWARE TOOLS IN BASIC.

I'D ALSO LIKE TO WELCOME THE RETURN OF MAV TO TRS8BIT. PLEASED TO HEAR HE'S MUCH RECOVERED AND HE IS LOOKING FORWARD TO SEEING EVERYONE AT TA2023.



THAT JUST ABOUT WRAPS UP THIS EDITORIAL, SO UNTIL SEPTEMBER,

TAKE CARE - DUSTY



CONTENTS

PAGE 04 "BLAST OFF"
PATRICK TTR

PAGE 07 BASQUE TOURNAMENT RESULTS
EGOITZ CAMPO

PAGE 10 SOFTWARE TOOLS IN BASIC
VINCE OTTEN

PAGE 25 WHY DID TANDY CLOSE? - FROM LEON HELLER
NATHANIEL MEYERSOHN, CNN

PAGE 26 WARNING ON "AI"
GEOFFREY HINTON

PAGE 28 AT THE READY> PROMPT

PAGE 30 MAPPING THE MAZE
E.T. FONEHUME

PAGE 34 IN MAV'S WORKSHOP - IAN MAVRIC
FREHD WITH A JOYSTICK ON A MODEL 1

PAGE 37 SYSTEM CASSETTE TAPE NAMES
PETER PHILLIPS

PAGE 40 YOUR OWN BIRTH DATE AS A MAGIC SQUARE!
E.T. FONEHUME

PAGE 42 NEW FROM FRANCE
PASCAL HOLDRY

PAGE 43 BOX SHOT
E.T. FONEHUME

PAGE 46 IN MAV'S TRAVELS - IAN MAVRIC
ARE YOU READY FOR TA 2023?

PAGE 49 THE WAY WE WERE

PAGE 65 BASIC V2.5 FOR THE MODEL 1
GEORGE PHILLIPS

PAGE 68 INTEGER SEQUENCES
NEIL JAMES ALEXANDER SLOANE & E.T. FONEHUME

PAGE 70 HANDY HINTS (FROM THE 1950'S)

PAGE 71 U.K. EBAY BARGAINS FROM THE LAST 3 MONTHS
CAN YOU BELIEVE SOME OF THESE PRICES!

PAGE 73 IN MAV'S WORKSHOP - IAN MAVRIC
A CO-CO 3 RUNNING ON 240 VOLTS?
JUST FOR OZ AND THE UK

PAGE 79 THE 2023 COMPETITION
SUPER PRIZE ON OFFER AGAIN!

NEW, MAJOR UPGRADE
VERSION 2.5

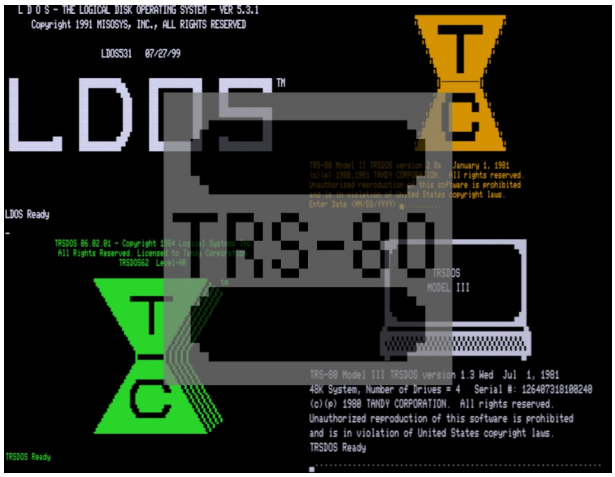
DON'T HAVE A TRS-80?

CRAZY PRICES ON EBAY?

TIRED OF RIFA CAPS BLOWING?

OF FLOPPIES SHEDDING OXIDE?

OR JUST PLAIN DEAD CHIPS?



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

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

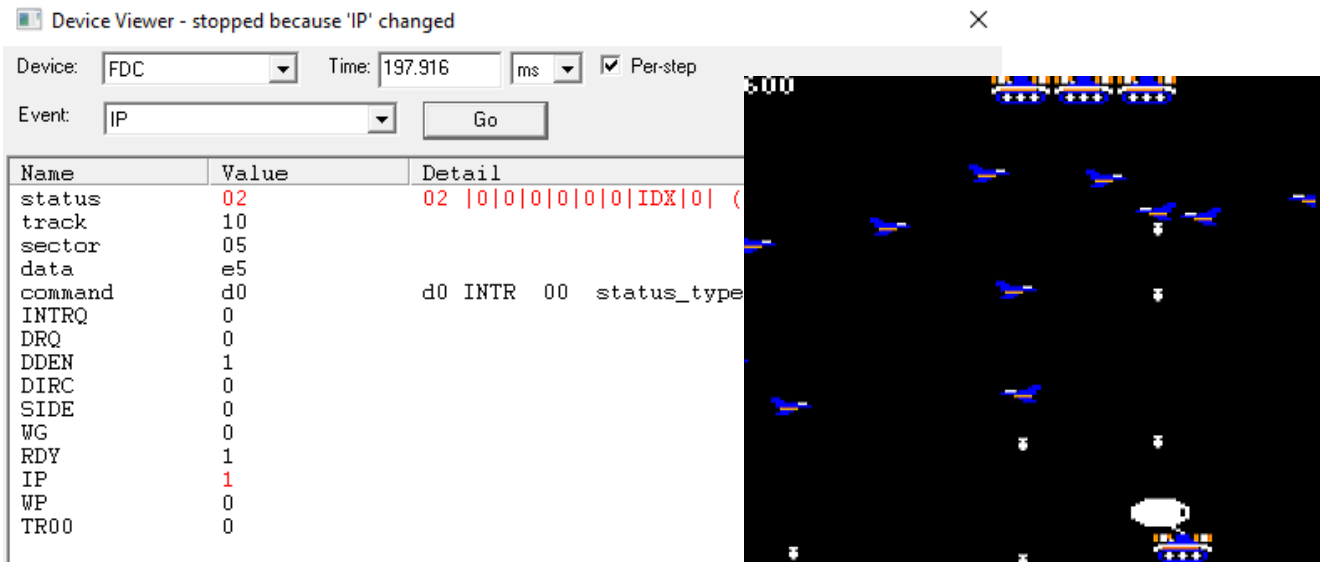
JUST LIKE A REAL TRS-80 WITH NONE OF THE HASSLE. BUILT-IN DEBUGGER AND OTHER FEATURES GREAT FOR PROGRAMMERS.

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

NEW VERSION 2.5, AVAILABLE NOW, FEATURING :-

COLOR COMPUTER EMULATION

DEVICE VIEWER BREAKPOINTS





PATRICK TTR

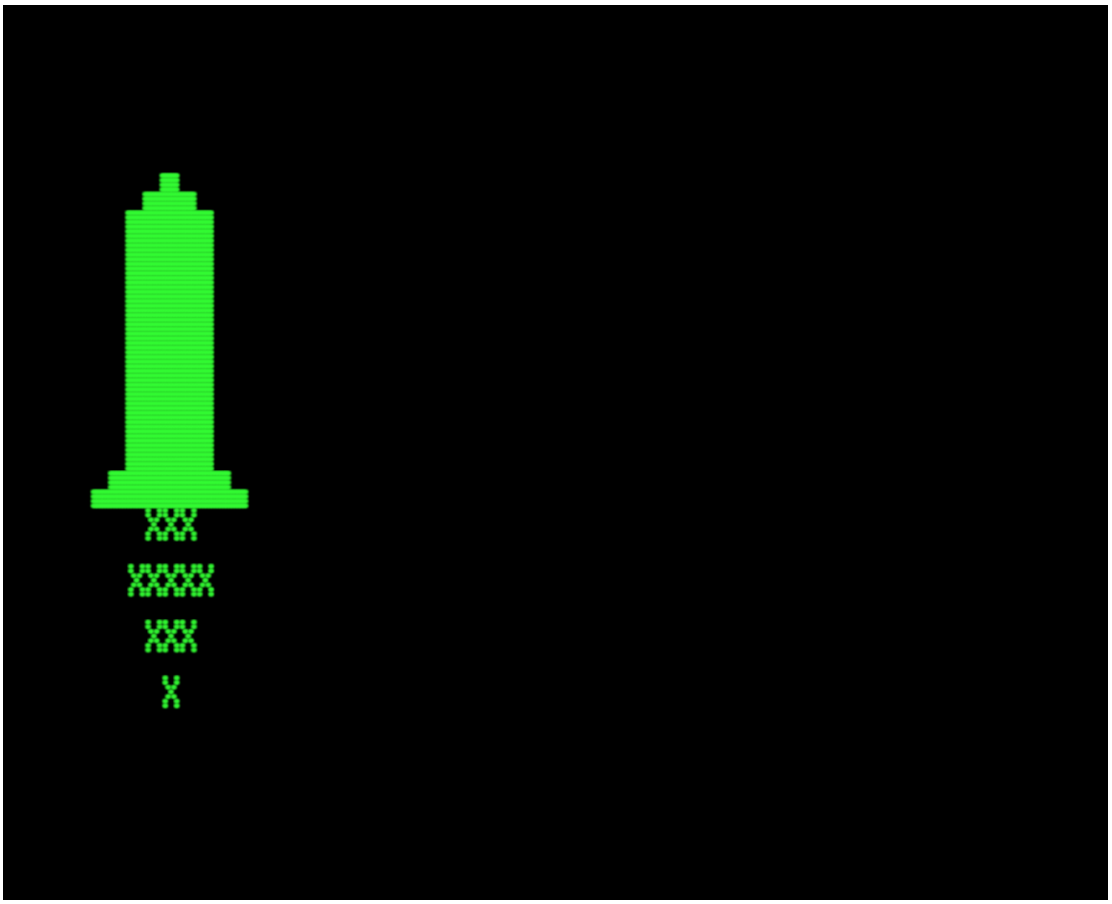
I KNOW YOUR ALWAYS LOOKING TO FUN ONE LINERS FOR YOUR MAGAZINE AND I WOULD LIKE TO SUGGEST ONE.
THIS IS NOT MY DOING BUT I STUMBLED ON IT AND FOUND IT A VERY NICE ROUTINE AND PROVIDES A SURPRISING GRAPHIC SOLUTION, (SEE NEXT PAGE).

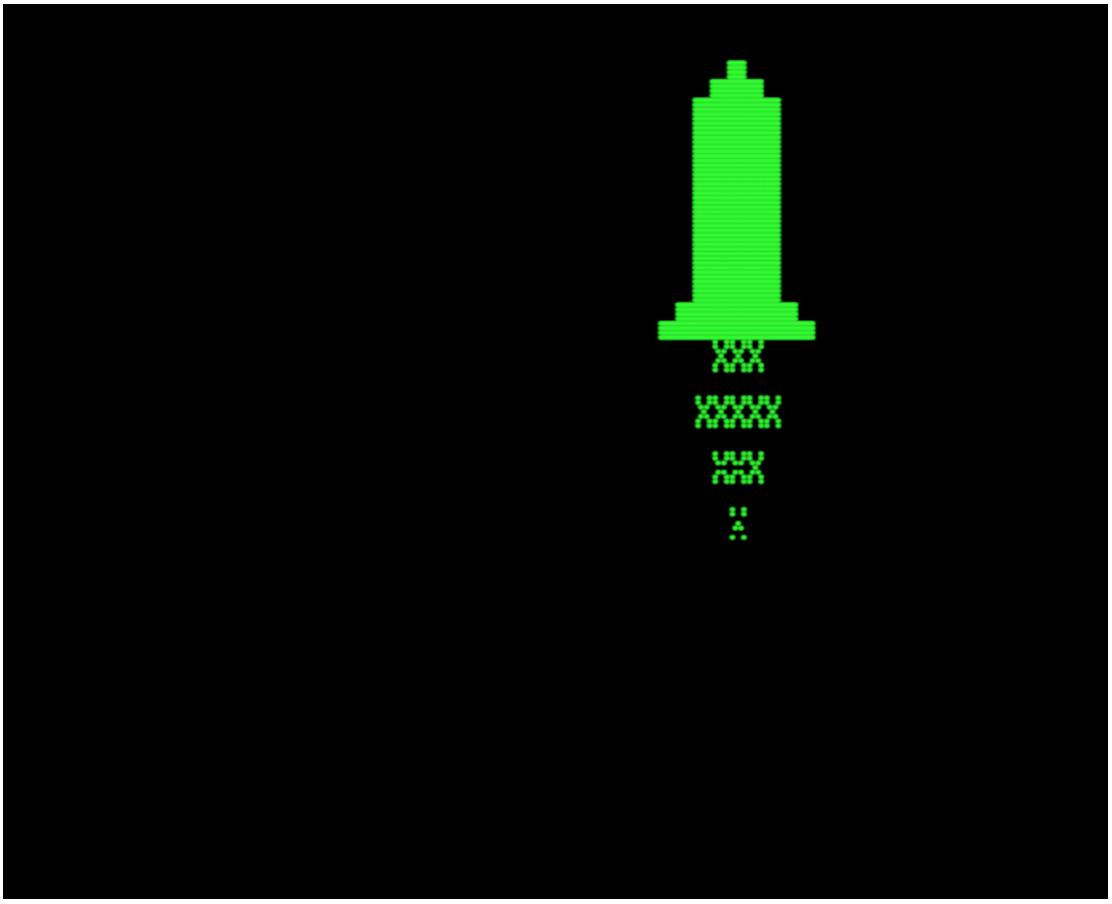
ONE LINER AFTER I HAVE CLEANED UP THE PDF CONTENT FROM THE MAGAZINE IS FOUND BELOW

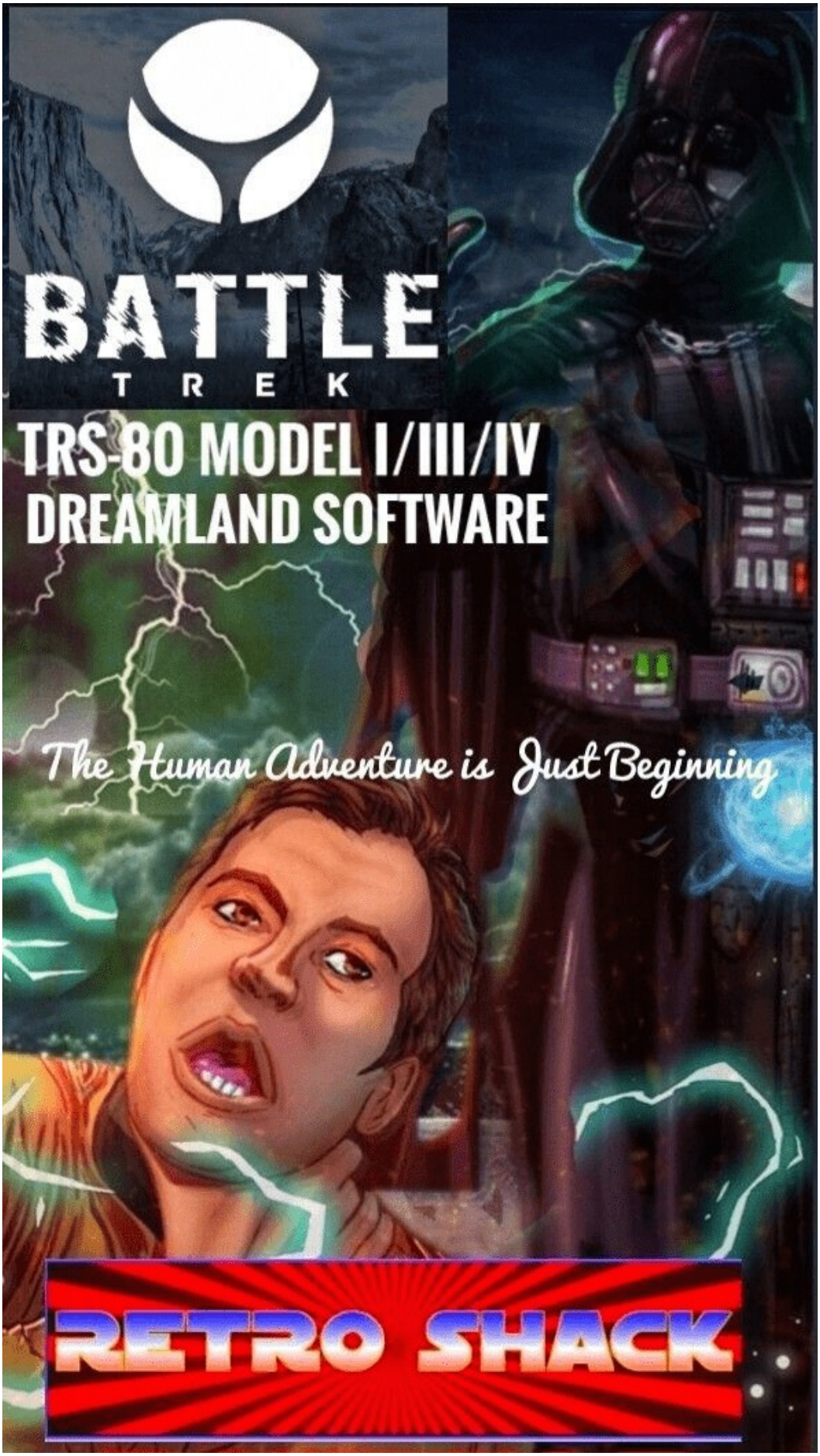
THE CREDIT IS DUE TO "REMARKS" MAGAZINE, POSTED VOLUME 49 P37, 1988 FROM THE NETHERLAND TRS-80 USER GROUP, THERE IS NO PROVENANCE FURTHER THAN THAT EVEN LISTED IN THE REMARKS MAGAZINE.

THE MAGAZINE CAN BE PROOF READ AT HTTPS://TRS80.NL/ARCHIEF/REMARKS88-10/

```
0 PRINT:PRINT:R$=CHR$(191):Z$=CHR$(176):Y$=CHR$(188):X=RND(53)+3
:A$=Z$+Y$+R$+Y$+Z$:FORT=1TORND(6)+3:PRINTTAB(X)A$:A$=STRING$(5,R
$):NEXT:PRINTTAB(X-2)Z$Y$A$Y$Z$:PRINTTAB(X+1)"XXX":PRINTTAB(X)"X
XXXX":PRINTTAB(X+1)"XXX":PRINTTAB(X+2)"X":RUN
```






BATTLE
TREK
TRS-80 MODEL I/III/IV
DREAMLAND SOFTWARE
The Human Adventure is Just Beginning
RETRO SHACK

V. Basque Tournament of TRS-80

Egoitz Campo

The last 5h of April, the Euskal Retro Group organized the V. Basque Tournament of TRS-80, in Bilbao (in the Bar Aldaketa).

It was entertaining and we had a good time. First we played the Amstrad CPC 464 and then we started with the TRS-80. As always the image is not entirely good but we like the American retrocomputer and we have a good time. Possibly from now on we will meet in this new bar because they let us play on the second floor and the place and the owners are very nice.

Egoitz, who trains from home, had the best score, but Daniel did very well. Unai did a good job and it was nice again to see my 6 year old son, Lander, enjoying this pure vintage 80's retro-computer. We played Monster Maze, Clowns and Ballons, Canyon Climber and Skiing.

The idea is to play a new TRS-80 tournament in October and this time we hope to add new games like Dragon Fire.

Here we leave you the results:

Monster Mazer

1-Egoitz Campo 5 points (1.510)
2-Daniel Barrientos 4 points (490)
3-Unai Pérez 3 points (440)
4-Joseba Zubiaurre 2 points (230)
5-Lander Campo 1 point (200)

Clowns & Ballons

1-Egoitz Campo 5 points (1.750)
2-Daniel Barrientos 4 points (480)
3-Unai Pérez 3 points (170)
4-Lander Campo 2 points (60)
5-Joseba Zubiaurre 1 point (60)

Canyon Climber

1-Egoitz Campo 4 points (12.000)
2-Daniel Barrientos 3 points (6.000)
3-Unai Pérez 2 points (4.000)
4-Joseba Zubiaurre 1 point (3.000)

*Skiing **

1-Egoitz Campo 3 points (1’25’’)
2-Joseba Zubiaurre 2 points (1’29’’)
3-Unai Pérez 2 points (1’44’’)

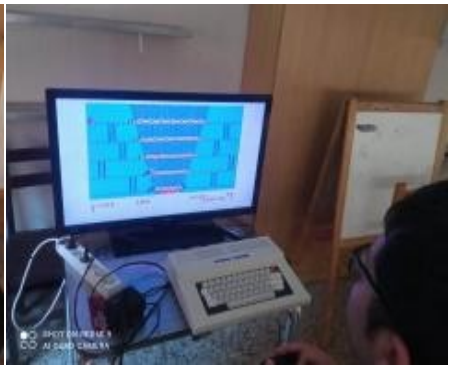
* time but taking into account the doors throught which we did not pass, in that case there was a penalty of one second for each door we did not pass.

Classification

1-Egoitz Campo 17 points (Basque Country)
2-Daniel Barrientos 11 points (Basque Country)
3-Unai Pérez 10 points (Basque Country)
4-Joseba Zubiaurre 6 points (Basque Country)
5-Lander Campo 3 points (Basque Country)

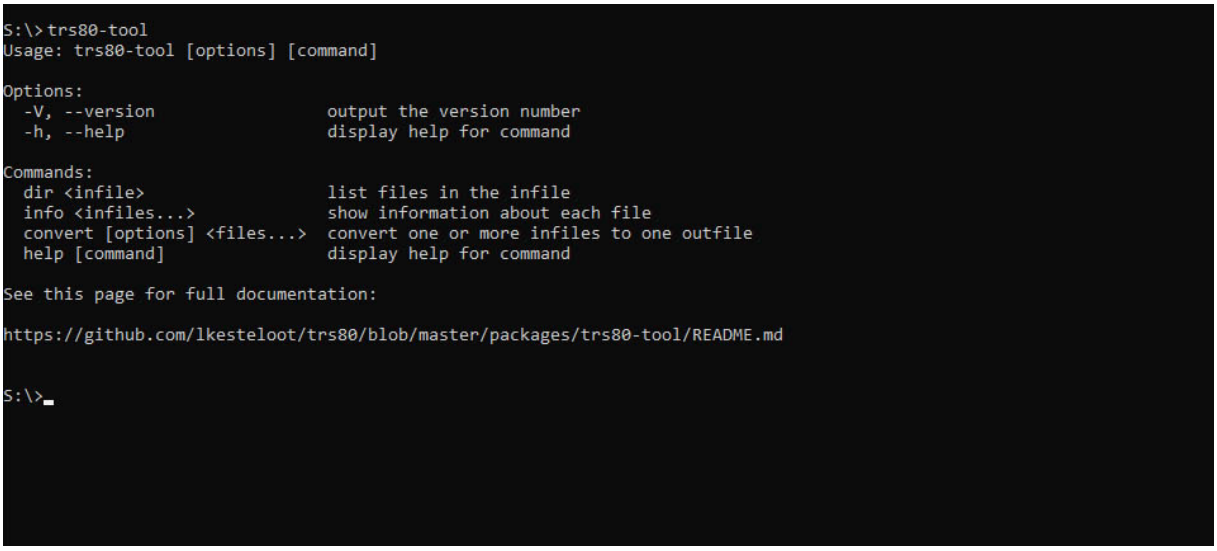







Lawrence Kesteloot has rewritten, his TRS-80 Cassette command line tool to support many new functions. Called "TRS80-TOOL" it is a command-line tool for manipulating TRS-80 files.



Lawrence Kesteloot has updated his "TRS80-TOOL" to v2.3.1

It runs under NODEJ, Linux, macOS, and Windows. It can be downloaded from Lawrence's site or locally from the The Virtual Tape Utilities page on this site.

The new version adds support for multi-file CAS files (such as CLOAD Magazine or Temple of Apshai) and will try to guess (and set, if you want) the starting address of a SYSTEM file.

The main page for TRS80-TOOL can be found at

https://github.com/lkesteloot/trs80/tree/master/packages/trs80-tool

along with instructions.

It runs under NODEJ, Linux, macOS, and Windows. It can be downloaded from Lawrence's site, the The Virtual Tape Utilities page on this site or right here for Windows / Linux / Mac.

Software Tools in BASIC Part 1

Vince Otten

Nearly 45 years ago, I got my first TRS-80: a 4K Level II Model I that I hooked up to a little TV for a 32-column display. Within days, I'd discovered the wonders of PEEK and POKE, and was trying to write a disassembler (with 8080 mnemonics!) to delve into the mysteries of the ROM. I expanded my machine, first to 16K of RAM (for $175 from Ithaca Audio in New York State), then to an Expansion Interface with 2 disk drives, and finally to a Line Printer so I could list my ever-growing programs. I was fascinated by programming languages. I wrote a little demo-display interpreter in BASIC so I could sell banner programs to local Radio Shack dealers. I discovered a LISP interpreter written in Applesoft BASIC that I tried to adapt to my TRS-80. I forked out anywhere from $25 to $600 for APL, C, Forth, more LISP, and Pascal. The way that these different languages warped my thinking still amazes me.

Somewhere in the middle of that voyage of discovery, I found the book "Software Tools" by Brian Kernighan and PJ Plauger. At the time, I didn't realize that they were introducing the basic utilities available in Unix: I was just amazed at how simple and effective their programming style was. I wanted to try this toolkit myself. They started with an example program that copied its input to its output: it sounds useless now, but back then TRS-DOS didn't even have anything like that. However, when I began to emulate these programs in BASIC, I ran into a brick wall: there was no way for my program to "read the command line". See, in Unix (or Linux), you can type

copy myfile

and the contents of "myfile" would be printed on the screen (Unix pros will recognize this as the "cat" utility). But I couldn't type in

COPY/BAS myfile

and expect "COPY/BAS" to understand that it should display "myfile". I gave up. Besides, I was too busy programming for real, for a living, and that took the joy out of coding for a long time.

George and Peter Phillips to the rescue! Using their amazing "trs80gp" emulator, and limiting myself to the system I had in 1979 (Model I, 48K, disk drives, printer, TRSDOS 2.3), I've rediscovered the joy of programming. So, for the fun of it, I decided to try the Software Tools in BASIC project again. What I've come up with is a "shell" program that lets me type in commands like

copy <myfile >copyfile

("copy from 'myfile' to' copyfile'), and it will create a set of instructions for COPY/BAS, which will retrieve those instructions et voila!

I've also created a BASIC "skeleton" In which to drop the code for "COPY". So the original program in "Software Tools" is written in a Fortran dialect called RATFOR ("Rational Fortran):

```
# copy – copy input characters to output
      while (getc(c) != EOF)
            call putc(c)
```

I want to write a BASIC version that looks something like this

```
39999 ' COPY – COPY INPUT CHARACTERS TO OUTPUT
40000 GOSUB 2100: IF EO% GOTO 40200' GETC( C$)
40010          GOSUB 2300 ' PUTC( C$)
40120          GOTO 40100
40200    '
```

Okay: the end result actually looks a bit more like

```
39999 '

      COPY -- COPY INPUT CHARACTERS TO OUTPUT (ST P. 8)

      WHILE (GETC() != EOF)

40000 GOSUB 2100: IF NOT VR%(0) THEN GOTO 40200'

            CALL PUTC( VA$(0))

40010         VA$(0)= VR$(0): GOSUB 2300:
                GOTO 40000'

      STOP

40200 CLOSE

59994 '

      RETURN TO SHELL

59995 RUN"SH/BAS"
```

Very similar: instead of C$ holding each character to copy from getc() to putc(), we use VR$(0) and VA$(0); and instead of "EOF" being a special character, it's an integer value in VA%(0). Why? It seems a bit more complicated.

See,while I think that Level II BASIC is superior to the Fortran used in "Software Tools", there's one place it definitely falls down: GOSUB subroutines lack the "parameters" of Fortran's CALL subroutines.

Notice that the original code had
getc(c)
and
call putc(c)
which told the getc() and putc() subroutines to "use the value of 'c' as your 'parameter' this time". BASIC, at least at the time of Level II, didn't have this capacity. *Every variable* is "global": every part of the program knows what variables like A or B$ or C(3) contain all the time, and can change them at will. I wanted "parameter passing" for each of the subroutines I was going to build into my "skeleton", and I certainly didn't want to use a lot of global variables that could conflict with whatever names a future user might employ. 'C' is a pretty common variable name after all. When all the dust settled around SKELETON/BAS, the framework in which my COPY program is written (lines 39999..40010 above), there were only three (or four) sets of variable names to avoid as a user:
any VA()arrays, ie VA%(), VA!(), or VA$(): these are used to pass "arguments" TO a subroutine;
any VR() arrays, ie VR%(), VR!(), or VR$(): these are used to pass "return" values back FROM the routine;
any SY() arrays, ie SY%(), SY!(), or SY$(): these are used to store "system" values; and optionally,
AV$(): this stores the "argument vector" of options and values passed to the program, but can be re-used by the programmer.
Also, be careful using any "Xn" variables (ie, "X0%", "X9$", or "X5!") since these are temporary storage in the system procedures. [For example, "getc()", GOSUB 2000, uses X9%, X9! and X9$. Using X9$ elsewhere in your program won't mess with SKELETON's operation, but GOSUB 2000 may mess with your X9$.]

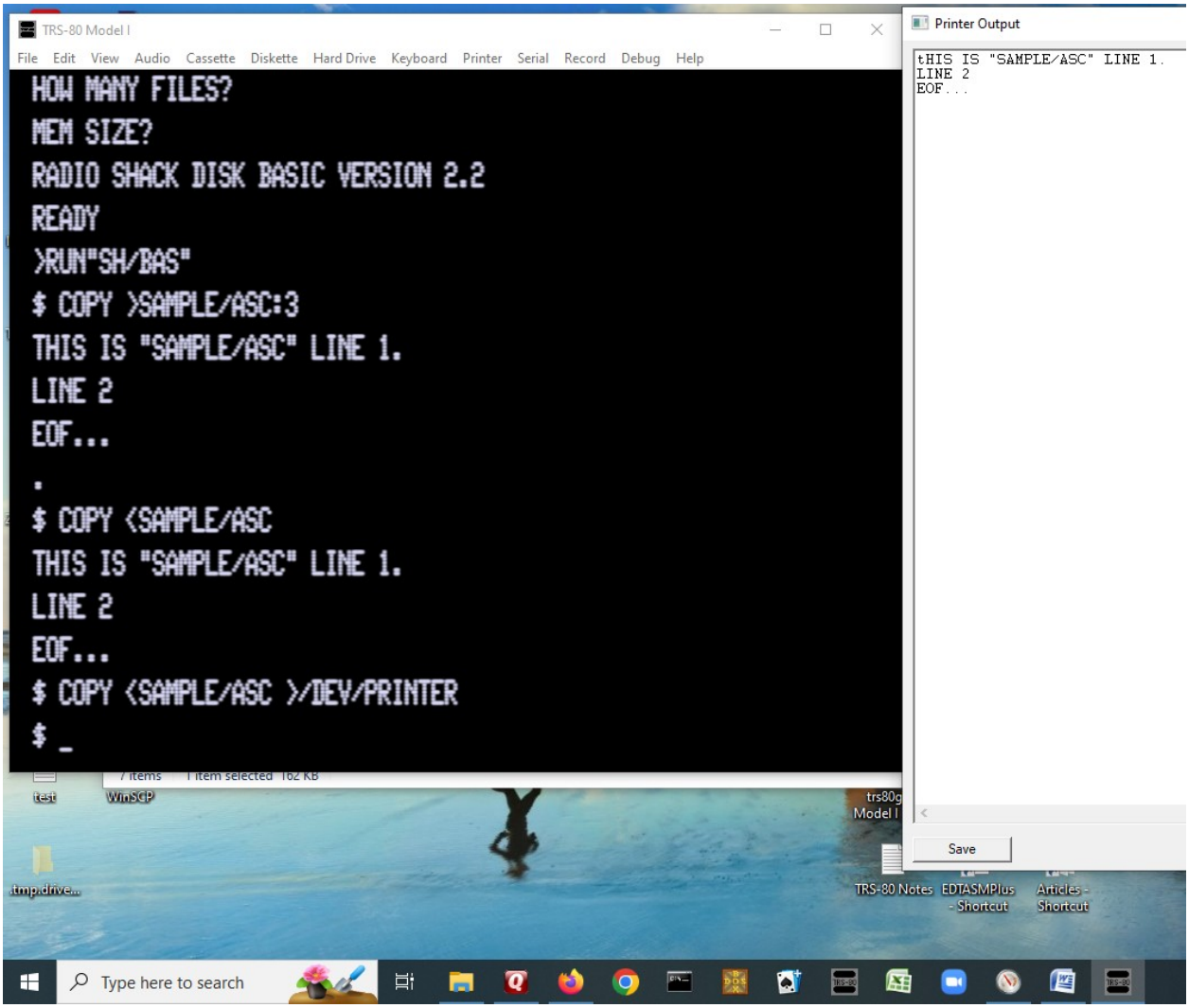
For example, to get a character value back from the "getc()" subroutine in SKELETON/BAS, just read the "value returned" in VR$(0) after you GOSUB 2100. To tell the "putc()" subroutine to write a character value, set it as the "value argument" in VA$(0) and GOSUB 2300. (How do I remember these VA()s and VR()s? Good question. I keep a documentation file for each collection of routines, and the one for getc() (I call it "getchar()" here) looks like this:

```
getchar()          Read 1 char/byte from stdin
-----------
Usage:         GOSUB 2100
Constraints:   SY%(4,0)     -1:  byte-stream I/O
                            0:   text-stream I/O
Result:        VR%(0)  succeed/fail (-1/0).     Fail on EOF
Returns: VR$(0)   one char/byte
Typical usage:     200 GOSUB 2100
                   IF VR%(0) THEN
                            PRINT VR$(0);
```

```
        ELSE
            RETURN

putchar( VA$(0))        Print string or char in VA$(0) to stdout
---------------
Usage:          GOSUB 60200
Quick Link:     GOSUB 2300
Inputs:         VA$(0)    String or char to be output
Typical Usage:      100 VA$(0)= "HELLO, WORLD"
                    200 GOSUB 60200
```

NOTE: All of these programs and support files can be found online at: https://github.com/VinnieMan1/Software-Tools-In-BASIC
That's a little peek "under the hood". Here's what it looks like when I take my "Software Tools" framework out on the road:



After booting into Disk BASIC, I run SH/BAS, the "shell" program, which prompts me with a "$ ". The shell accepts 1 line of input, "parses" it into a command name

any standard input file (identified by a "<" as the first character)

any standard output file (identified by a ">" as the first character) and "everything else" (ie, options, other file names, etc).
First I used COPY/BAS to create a text file. (Notice that, since I hadn't included the "/BAS" extension, the shell automatically appended it.) The default "standard input" for any program created using SKELETON/BAS as its basis will be the keyboard, as it is here. Where does what I typed go? ">SAMPLE/ASC:3" – the standard output has been reassigned from its default, the screen, to a file on drive 3 called SAMPLE/ASC. The program COPY/BAS will keep doing this until I tell it "end of file": from the keyboard, I do this by typing a "." on a line by itself. The "." Is not put into SAMPLE/ASC, but the file is closed, and COPY/BAS automatically runs SH/BAS again, displaying the "$ " prompt.
Now, I confirm what I typed into SAMPLE/ASC. This time, "standard input" won't be from the keyboard, but instead is reassigned with "<SAMPLE/ASC". But "standard output" is to the default, the screen. Once COPY/BAS reaches the end of the file (how does it know that? Stay tuned), it closes it, and runs SH/BAS again.
The third example COPYs the contents of SAMPLE/ASC to a device other than the screen: the printer (in this case, since I use trs80gp, it's an onscreen emulation of the printer, which you can [Save] to a file if you wish). The standard output is redirected to "/DEV/PRINTER" (the shell is smart enough to recognize any device specified with at least "PR" or "LP" in its name, so it could be "/DEV/PR" or "/DEV/LP", for example).
This example gives a good top-level demonstration of what the combination of SH/BAS and a program written using SKELETON/BAS produce. SH/BAS interprets Unix-like commands, including I/O redirection, and produces a file named CMDLINE/SH with these contents:

Line 1 command name + arguments
Line 2 standard input file, device, or "" (empty)
Line 3 standard output file, device, or ""(empty)
Line 4 standard error file, device, or ""(empty)

In the last example of the screen shot above, it looks like this:

Line 1 COPY
Line 2 SAMPLE/ASC
Line 3 -3
Line 4 ""

("-3" is the internal code for "printer").
Any program like COPY/BAS which is built upon SKELETON/BAS will automatically do the following before actually running the user code (like lines 40000-40200 in our COPY example above):
Initialize (default values for standard I/O, etc.)
Look for CMDLINE/SH and read it if it exists.
Read line 1, the "command line", and parse it into a string array containing all the "arguments" (options, extra filenames, etc).
Re-assign any standard I/O based on lines 2-4 or CMDLINE/SH.
Jump to line 40000 for the user program (COPY, in this example).

Provide functions [like "getc()" and "putc()"] that allow "Software Tools"-style programs to access I/O (disk files, keyboard, screen, printer, and eventually cassette data and RS-232 connections).
Close all files and return to SH/BAS when the user program is done.
Why does the user program start at line 40000? Why not line 10, like most BASIC programs? Actually, SKELETON/BAS starts at line 1 with an immediate GOTO 65000. That's where steps 1-5 are performed. One of the quirks of Microsoft's 8-bit interpreted BASIC is that any GOTO or GOSUB scans for the line to jump to from the beginning of the program each time. All the initialization that takes place in steps 1-5 only happens once, so I put it at the end of the program text. Now code that has to run faster can be placed at lower line numbers and the BASIC interpreter will find them quicker. (For example, I used this for the code for "getchar()", at line 2100.) Also, the user can place their subroutines between lines 20000..39990, so that they're "before" their main code in lines 40000..59990.

Granted, BASIC just isn't fast. And when you see COPY/BAS in action, you'll see that: it prints out the contents of TESTFILE/ASC at about 1 character per second! Kernighan and Plauger ("K&P") point this out in Software Tools: most of a program's time is typically spent in I/O. In the case of these tools, it's especially true of "getc() / getchar()". It reads one character or one byte at a time from the input. Actually, we input a line (from the keyboard) or a disk sector into a "buffer" and then step through that buffer. When we reach the end of the buffer—or the "end of file"—we ask for more to put in the buffer, and start all over again from its first entry.

But how do we determine "end of file"? In Unix (Xenix..Linux..) we type a [Ctrl][D] and the system understands that we're done. But [Ctrl][D] isn't available on a Model I. How about [Shift][UpArrow] then? That's the usual "Escape" key sequence used in the editor for BASIC and the Assembler. But BASIC's LINE INPUT just considers that part of the text you're typing in. That's why I chose a single period or "stop" standing by itself on a line as the signal to quit. (It's a convention used in a few other Unix tools, like "ed" and "vi".) See the screenshot above for an example. The period does not get entered into the buffer: it's only an end-of-file (EOF) signal.

How about disk files? Can we just use BASIC's EOF() function to determine whether we've reached the last character? That works if we only use plain text files, like the one in the screen shot. But it would be nice (and compatible with the Software Tools programs) if we could COPY and process files like BASIC code, machine language /CMD files, etc., that might contain unusual byte values. Will LINE INPUT and EOF() correctly process those? I ran a test. First, I PRINTed a file filled with all the ASCII characters from 0..255 like this

```
10 OPEN "O", 1, 'ASCITEST/FIL"
20 FOR C%= 0 TO 127
30 PRINT #1, CHR$( C%);
40 NEXT
50 PRINT #1, ""
120 FOR C%= 128 TO 255
130 PRINT #1, CHR$( C%);
140 NEXT
150 PRINT #1, ""
200 CLOSE
```

I split it up into 2 lines of 128 bytes each, because the maximum length of a string in BASIC is 255 bytes. Then I wrote a program to see if I could just LINE INPUT those lines back in, and peel off the characters one byte at a time.

```
10 OPEN "I", 1, "ASCITEST/FIL"
20 IF EOF( 1) THEN GOTO 500
30 LINE INPUT #1, L$
100 FOR P%= 1 TO LEN( L$)
110 C$= MID$( L$, P%, 1)
120 C%= ASC( C$)
150 PRINT C%;
160 NEXT
170 GOTO 20
500 PRINT
510 CLOSE
```

When I ran this code, I expected to see all the ASCII codes from 0 to 255, plus a couple of extra 13s (ASCII code for the Carriage Returns [CR] appended in lines 50 and 150 of the first program). Instead, I only saw up to 127: then EOF(1) reported "end of file", and that was it.
Hmmm. My guess is that Disk BASIC somehow gets confused by the extra CHR$(13) inserted in line 130 when C% reaches a count of 13. I'd PRINTED out 2 lines, but there were 3 CHR$(13)s there.
How about accessing this as a Random file somehow? Indeed,

```
OPEN "R", 1, "ASCITEST/FIL"
```

works, and I can now access the entire file. But how do I know where the EOF is? BASIC Random I/O doesn't let you use EOF(). It provides its own function, called LOF(). The only problem is, it doesn't tell you where the last byte is, only the last 256-byte "record" in which it occurs.

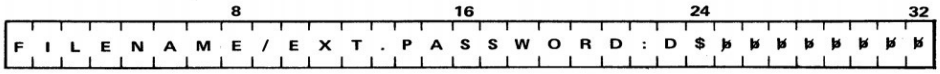
If only there was some way to determine exactly how many bytes are in a given file. (In the case of ASCITEST/FIL, it's 258 bytes: the 256 codes, plus 2 CRs from lines 50 and 150.) It's been decades since I used my pirated copies of NEWDOS/80 and LDOS, but I believe one or both of those included that capability in their BASICs. However, in the name of authenticity…or something…I wanted to see if I could figure this out with TRSDOS 2.3—and I can. I did a little digging, wrote a few lines of code, and now if I

TRSDOS Technical Information

DCB before OPEN and after CLOSE:

The DCB is defined as 32 contiguous bytes of RAM designated by the user. Before OPEN and after CLOSE, it is a left justified, compressed (no spaces) ASCII string, as in a standard TRSDOS filespec:

CONTENTS OF 32 - BYTE DCB



Notes: /EXT, .PASSWORD, :D are optional.
$ stands for a carriage return (X'OD')
ß stands for a blank (X'20')

Explanation of DCB while OPEN:

lsb/msb is least significant byte followed by most significant byte in Z80 RAM format (i.e. addr=7CC8 in RAM is C8 7C).

Addr.	Len.	Explanation
DCB+0	3	Reserved
+3	2	Physical Buffer address (lsb/msb)
+5	1	Offset to delimiter at end of current record
+6	1	File drive number residence
+7	1	Reserved
+8	1	EOF offset of last delimiter in last physical record
+9	1	LRL (logical record length)
+10	2	NRN (next record no. — open sets=X'0000' — lsb/msb)
+12	2	ERN (ending record no. — last in file — lsb/msb)
+14	18	Reserved

NRN Next Record Number defines which record is to be read or written by the next system call for READ or WRITE. It is automatically incremented by one after each system call. In order to process random files, use the POSN call to direct TRSDOS to the record you wish to transfer next.

ERN Ending Record Number is the last record number currently in the file. It is put into the directory at CLOSE time, so if it is expected to be correct, the user *must* close his files after adding records to a file. This value may also be used to position to end of file so that new records may be added to the end of the file. To position to the end of file use a call to POSN with a record number of ERN+1. POSN is described later.

40000 OPEN "R", 1, "ASCITEST/FIL"

and then
40010 PRINT FN DX!(1)

It will print "258".
Where does FN DX!() get its information? According to Tandy's original "TRSDOS & DISK BASIC Reference Manual", each open file has a "DCB", or Disk Control Block, a 32-byte section of RAM that contains details about the file.

To find out where an existing file that you've OPENed currently ends, you need to know:
"ERN (ending record no. – last in file – lsb/msb)"
and
"EOF offset of last delimiter In last physical record"
(see the picture of the TRSDOS Manual on the previous page).

So we need code to "pull out" this information. First of all, we need to know where each DCB is: there's one for each open file, depending on what TRSDOS BASIC calls its "buffer number" (1..15). I got much of the following information from Lewis Rosenfelder's amazing "BASIC FASTER AND BETTER", on page 274, shown below.

If we
OPEN "I", 1, "ASCITEST/FIL"

for example, the DCB for ASCITEST/FIL will start at address 26303 in TRSDOS 2.3, which is the Model I DOS I'm using in all these examples. Now I'd like to set a constant to this address as my starting point for all calculations—but Disk BASIC doesn't have "constants". Or doesn't it?

How about if we use a defined function? So I created
DEF FN D0%(X)= 26303
and whenever I need that base address, I just use
FN D0%(0)
in my calculations.

Now if we
OPEN "O", 2, "ANOTHER/FIL"
its DCB will start at address 26593, 290 bytes farther down the road in RAM, or FN D0(0) + 290. Notice that this "distance between DCBs" varies from DOS to DOS, so I made it a "constant" using
DEF FN D1%(X)= 290

That way, if I end up using a different DOS (say, TRSDOS 1.3 on the Model III), I can just change these two DEF FNs, and my final calculations will still be correct.

274 Appendix 4 - Disk DCB Addresses

Disk Data Control Block Addresses

DISK OPERATING SYSTEM	VERSION	MODEL	1	2	3	4	5	6
TRSDOS Radio Shack	2.3	1	26303 66BF	26593 67E1	26883 6903	27173 6A25	27463 6B47	27753 6C69
TRSDOS Radio Shack	2.0	2	27715 6C43	28549 6F85	29383 72C7	30217 7609	31051 794B	31885 7C8D
TRSDOS Radio Shack	1.2	3	25762 64A2	26122 660A	26482 6772	26842 68DA	27202 6A42	27562 6BAA
TRSDOS Radio Shack	1.3	3	26182 6646	26542 67AE	26902 6916	27262 6A7E	27622 6BE6	27982 6D4E
NEWDOS Apparat	2.1	1	25941 6555	26231 6677	26521 6799	26811 68BB	27101 69DD	27391 6AFF
NEWDOS/80 Apparat	1.0	1	26315 66CB	26616 67F8	26917 6925	27218 6A52	27519 6B7F	27820 6CAC
DOS PLUS Micro Systems Software	3.3D	1	28021 6D75	28567 6F97	29113 71B9	29659 73DB	30205 75FD	30751 781F
DOS PLUS - TBASIC Micro Systems Software	3.3D	1	25418 634A	25964 656C	26510 678E	27056 69B0	27602 6BD2	28148 6DF4
DOS PLUS Micro Systems Software	3.3	3	28007 6D67	28553 6F89	29099 71AB	29645 73CD	30191 75EF	30737 7811
LDOS Lobo Drives, Int'l	5.0.1	1	27205 6A45	27751 6C67	28297 6E89	28843 70AB	29389 72CD	29935 74EF
ULTRADOS Level IV Products	4.2	1	25499 639B	25789 64BD	26079 65DF	26369 6701	26659 6823	26949 6945
DBLDOS Percom	4.23	1	25941 6555	26231 6677	26521 6799	26811 68BB	27101 69DD	27391 6AFF

Now to be able to calculate the DCB start address for any buffer # from 1..15:
*DEF FN DB%(F%)=FN D0%(0) + FN D1%(0) * (F% - 1) ' DB "Disk Buffer"*

Now we need the address of the "End Record". TRSDOS 2.3 files are made up of 256-byte physical records. For our example, we're going to use "ASCITEST/FIL" from above, which was a 258-byte file. ASCITEST/FIL would span 1 256-byte record and go on for 2 more bytes into the 2nd record. Its "End Record" number would be record #2. If we have the address of the DCB, we go down the road 12 more steps to find the 2 bytes that store that "ERN", the End Record Number.

DEF FN DE%(F%)= FN DB%(F%) + 12

Note: this doesn't give us that "record #2" in our example: it simply gives us the address where that "2" is stored, as a 16-bit number. To pull out the End Record Number for file F%, we need to PEEK(FN DE%(F%)) …but it's not that easy. This number is stored in 2 bytes. Wouldn't it be nice if BASIC gave us a WORD-PEEK() along with the PEEK() function? I guess we'll have to "roll our own".
We'll need to PEEK at the address returned by FN DE%(), and then PEEK at the next address, and combine those two bytes. The first PEEK is the "low byte" of a 16-bit word, and the second is the "high byte".

To get a 16-bit value, the high byte is multiplied by 256 and added to the low byte. But there's a sneaky conversion built into Disk BASIC. You can store and retrieve BASIC integer values in a random disk file by using the MKI$() and CVI() functions. These functions can be shanghaied to use in our conversions.

Here's how.
DEF FN SW%(BL%, BH%)= CVI(CHR$(BL%) + CHR$(BH%))

First, we convert each byte value into a single-character string with the CHR$() function. Then we make a 2-character string out of them with + . Finally, we convert that 2-byte string back into a signed integer with CVI ().
But we often don't want a signed integer value. For example, if the low byte contains the value of 255, and the high byte contains 255 also, that would mean either 65535 as an unsigned integer, or -32767 as a signed integer. So how do we convert the latter to the former value? With a bit more complex Defined Function, of course!

*DEFFN IS!(I%)= (I% < 0) * -(65536 + I%) + (NOT I% < 0) * -I%*

Let's start with the last bit: "(NOT I% < 0) * -I%". This tests if the signed integer is 0 or positive [NOT I% < 0]: if so, the result will be -1, and we multiply that by the integer made negative.

For example, say we have a value of 11. It's not < 0, so that gives us a -1, multiplied by 11 made negative, -1 x -11, which gives us "11". So far so good.

What if it was "-22"? This time the test gives us a 0, multiplied by anything including -22 gives us "0". So positive numbers give us their own value, and negative numbers give us 0. That's going to be added to the first part of this function with the "+".

Now for the first bit: "(I% < 0) * -(65536 + I%)". This tests if the signed integer is negative [I% < 0]: if so, the result will be -1, and we multiply that by a little formula (65536 + I%) made negative. For example, say we have a value of -22. 65536 + -22 is the same as 65536 – 22, which is 65514, which we then make negative, "-65514". That gets multiplied by the test result, -1, which gives us the unsigned integer number 65514. What if it was "11"? Tested, 11 is not < 0, so the result is 0, which multiplied by -(65536 + 11) still gives us 0.

Now the two sides of the function get added together. For "11", the left side gave us "0", which we add to the result of the right side, "11", so 0 + 11 gives us 11. Right. For "-22", the left side gave us "65514", which we add to the result of the right side, "0", so 65514 + 0 gives us 65514. Again, correct. We can convert signed integers into unsigned values.
[There's a way to convert back, too:

*DEF FN SI%(S!)= -((S! > 32767) * (S! - 65536)) - ((S! < 32768) * S!)*

This is also done with the "two tests" method. This time the right side and left side are joined by a "-" in the middle.]
Now we're able to define "WordPeek()". I included FN IS!() to make its result unsigned, but if you use it and need a signed integer result, just pass it through FN SI%(). Given an address A%, WordPeek() will combine the values PEEKd at A% and A% + 1 into an unsigned 16-bit value.

DEF FN WP!(A%)= FN IS!(FN SW%(PEEK(A%), PEEK(A%+1)))

With this, we're able to get the End Record Number of file # F% at the address given to us by FN DE%():
ERN%= FN WP!(FN DE%(F%))
For our example here,
OPEN "I", 1, "ASCITEST/FIL"

FN WP!(FN DE%(1)) will return a value of 2. The file ends in record #2. The file contains 1 full 256-byte record + some more from record 2, but how many? That bit of information is found in the single byte at DCB + 8, "EOF offset of last delimiter in last physical record".
DEF FN DL%(F%)= FN DB%(F%) + 8

In other words, where's the last character written to this file in the last record? In the case of ASCITEST/FIL, which is 258 bytes long, the byte stored at DCB + 8, "PEEK(FN DL%(1))", will have a value of 258 – 256 = 2: the last character is the 2nd byte in record #2. Seem simple. To get the total number of bytes in a file, multiply (Last Sector – 1) x 256 and add Last Position, like 256 x 1 + 2 in the case of ASCITEST/FIL. How about

*DEF FN DX!(F%)= (FN WP!(FN DE%(F%)) – 1) * 256 + PEEK(FN DL%(F%))*

But wait…each record contains up to 256 bytes. Character 1 in position 1, character 2 in position 2…all the way up to character 256 in position 256. But how do we represent 256 positions in a single byte? A byte can hold 256 different values, but they run from 0..255, and we've already used 1..255 for positions 1..255 in the disk record. We'll have to use 0 to indicate position 256. That's what the author(s) of TRSDOS did, and it works, but it screws up our simple calculation. If PEEK(FN DL%(F%)) is anything but 0, we can use it straight up; but if it's 0, we need to replace that with 256:

*DEF FN DQ%(F%)= ((PEEK(FN DL%(F%)) = 0) * -256) +*
*(NOT PEEK(FN DL%(F%)) = 0) * -PEEK(FN DL%(F%))*

(Proof positive that BASIC uses almost as many parentheses as Lisp!)

The left-hand side of this function (the part of the definition before the "+") tests if the byte is 0, and if true (-1) multiplies that by -256 to give a value of 256. The right-hand side tests if it is not 0, and if so, multiplies that by the negative value of the byte to give the byte's value as a positive number.

Finally, here's the function which gives us the total number of bytes in file F%:

DEF FN DX!(F%)= FN DQ%(F%) + 256(FN DE!(F%) - 1)*

We add that end byte value that's been corrected by FN DQ%() to 256 x the total number of records – 1.

The hierarchy of function calls for FN DX!() looks like this:

```
FN DX!()
    FN DQ%()
        FN DL%()
            FN DB%()
                FN D0%()
                FN D1%()
    FN DE!()
        FN WP!()
            FN IS!()
            FN SW%()
        FN DE%()
            FN DB%()
                FN D0%()
                FN D1%()
```

Why use all these DEF FNs? I have two good reasons:
It's faster than subroutine calls. As mentioned above, when BASIC has to find a line number (like the "1000" in "GOSUB 1000"), it searches from the beginning of the program. DEF FNs, on the other hand, are stored in the same area as variables, so they can be looked up more quickly.

(Okay: some more unnecessary details—
BASIC actually stores a memory pointer at the beginning of each program line to the next line, so searching for line numbers is a little quicker than just scanning all the text of a program.
Only the name of the FN is stored in the variable space, along with a memory pointer to the text of the function's definition in the actual program line where it's DEFined.)

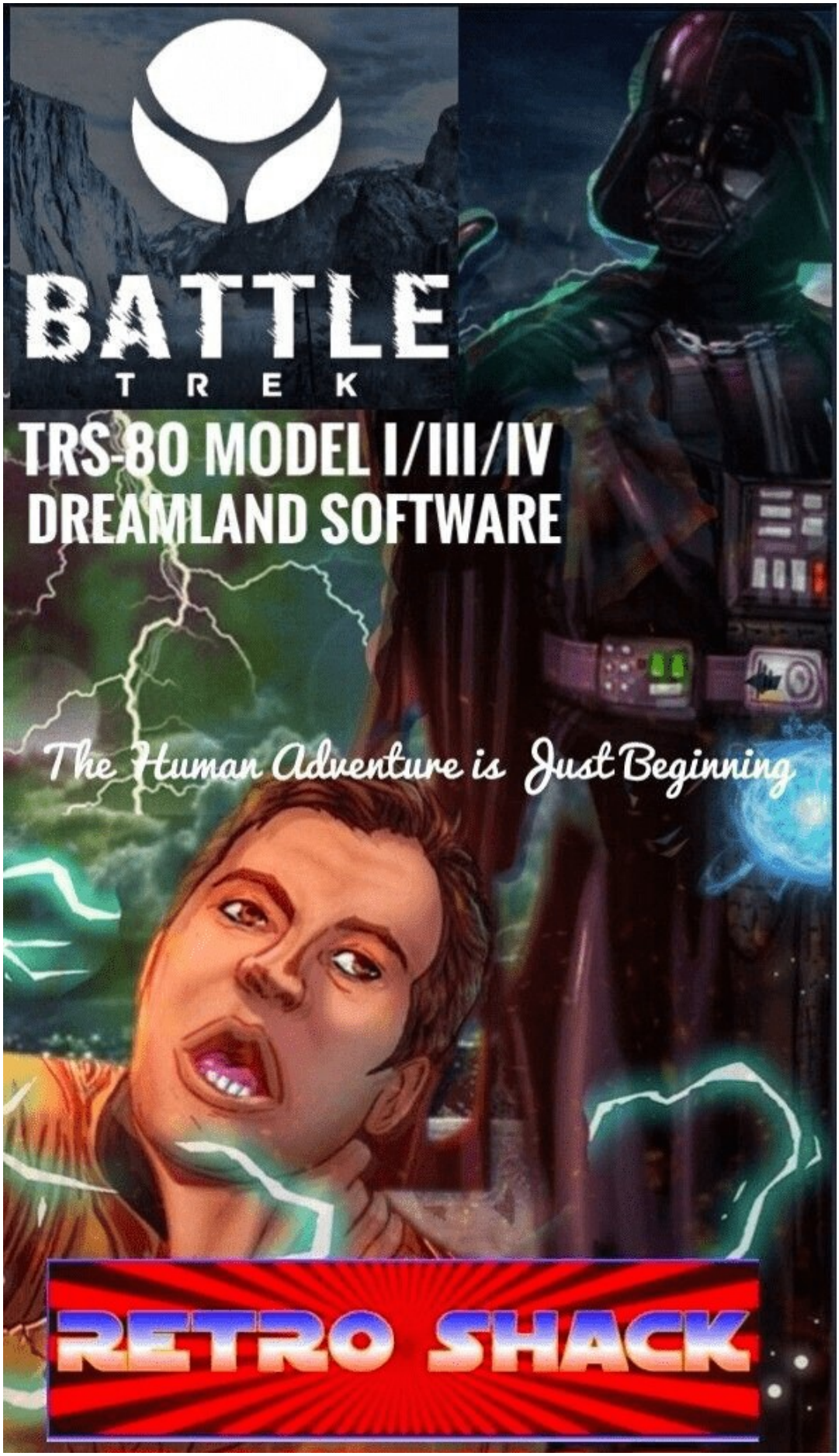
It documents the code a little better than just "GOSUB 1000…GOSUB 2000…" etc. BASIC programs are hard to read at the best of times.
But isn't it slow anyway? FN DX()! uses FN DQ%() uses FN DL%() uses FN DB%() uses FN D0%()... That's true. So what SKELETON does is to store the result of FN DX!() for any OPENed file in the appropriate part of SKELETON's System SY!() array. When getchar() or getc() want to check if they're at the EOF, end-of-file, they can just access that array element rather than running through all the DEF FNs again.

All the BASIC code for SKELETON/BAS, SH/BAS, and COPY/BAS are now up on GitHub at

https://github.com/VinnieMan1/Software-Tools-In-BASIC

along with a "Skeleton for Software Tools User's Guide" and a more detailed "Skeleton Usage Guide". Next time: we look at how SKELETON will OPEN a file.




BATTLE
TREK
TRS-80 MODEL I/III/IV
DREAMLAND SOFTWARE
The Human Adventure is Just Beginning
RETRO SHACK



Toys 'R' Us and RadioShack all shut down for the same reason

Nathaniel Meyersohn CNN

In the 1980s, a new type of specialty retail chain started to emerge: "category killers."

The stores' powerhouse business model was aimed at giving shoppers access to every different size, style and color of a product imaginable – all in one place and at reduced prices.

Category killers, which began to dominate entire merchandise categories, opened stores typically under 50,000 square feet – bigger than independent shops but smaller than Walmart superstores – in strip mall centres all over the suburbs. Shoppers embraced these overstuffed emporiums.

Staples is "a classic 'category killer,' like Toys R Us," and Radio Shack Mitt Romney, then Bain & Co.'s managing general partner reported in 1989.

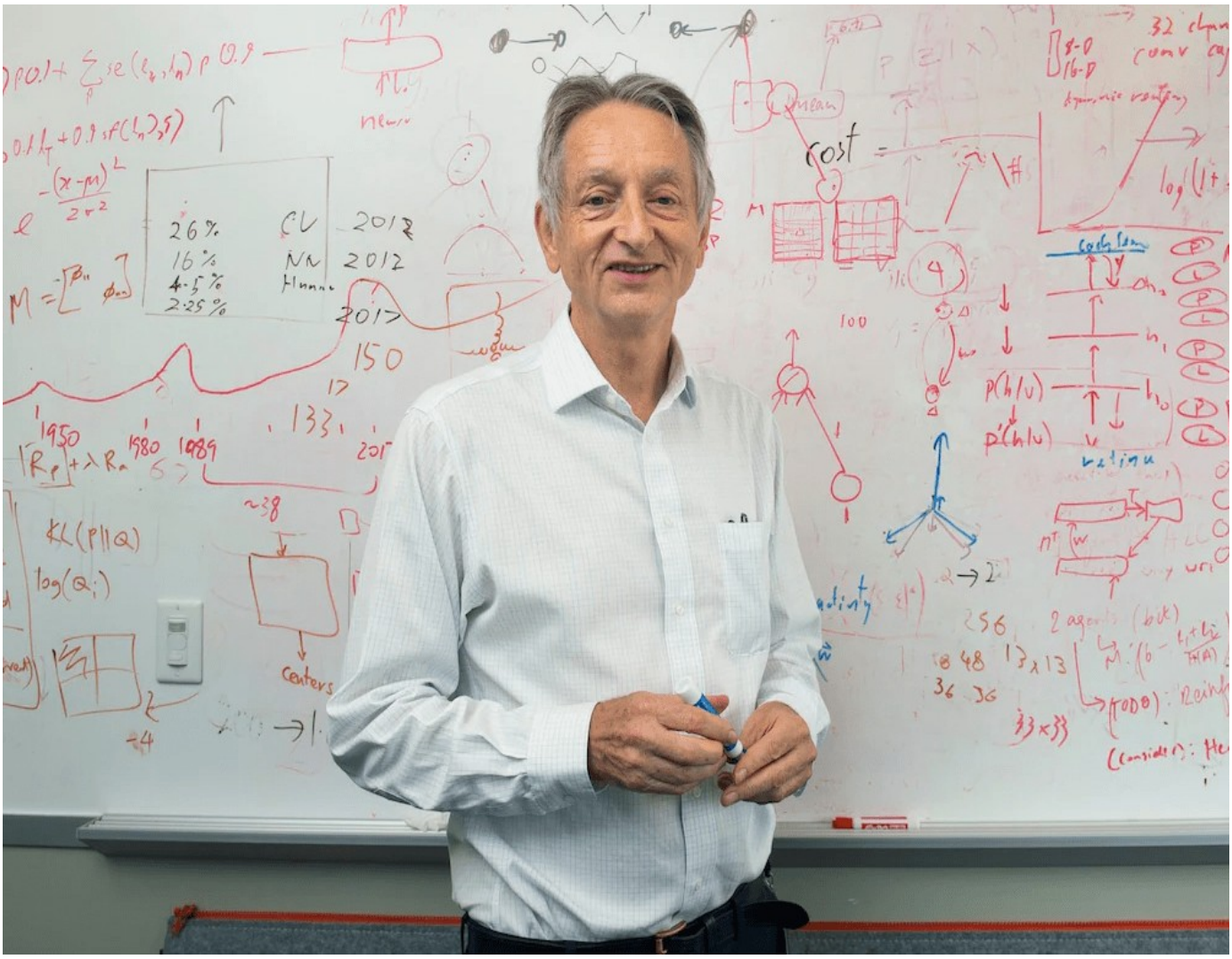
These companies, along with RadioShack, Blockbuster, Barnes & Noble and others, spread into the 2010s, remaking how Americans shopped and steamrolling right over mom-and-pop stores.

But the category killer's time has passed.

Toys "R" Us, Blockbuster and RadioShack are gone. Staples and Barnes & Noble are still around, but they have struggled and closed hundreds of stores.
"We're missing these places now they're gone. There are increasingly few stores where you can go and find any real variety or options for a product," the urbanist writer Addison Del Mastro said on Twitter this week. "We should have more than the single Walmart option or 100 pages of spammy Amazon results."

Godfather of AI' quits Google over fears technology can't be controlled

GEOFFREY HINTON WARNS 'SCARY' ADVANCE OF ARTIFICIAL INTELLIGENCE COULD MEAN INTERNET USERS WILL NO LONGER 'KNOW WHAT IS TRUE'



A BRITISH COMPUTER SCIENTIST KNOWN AS THE "GODFATHER OF AI" HAS QUIT GOOGLE AND EXPRESSED REGRET OVER PARTS OF HIS LIFE'S WORK AS HE FEARS THAT THE GROWTH OF ARTIFICIAL INTELLIGENCE COULD LEAD TO KILLER ROBOTS THAT ARE SMARTER THAN HUMANS....



AS ALWAYS, OF COURSE, I'M ON THE LOOKOUT FOR ANY SOFTWARE FOR THE U.K.'S ACULAB 'FLOPPY TAPE'. THERE WERE VERSIONS SPECIFICALLY 'ZAPPED', FOR USE ON THE AFT, OF ELECTRIC PENCIL, SCRIPSIT, ZEN, LEVEL III BASIC AND VISICALC. THERE WAS ALSO THE ENHANCED BASIC, XBAS. IT'S SO DISAPPOINTING TO THINK THAT THEY HAVE ALL BEEN LOST! PLEASE EMAIL ME AT THE USUAL ADDRESS IF YOU CAN HELP.

SOLVE YOUR BUSINESS PROBLEM WITH A MICROCOMPUTER

We have a variety of microcomputers and available software to suit most types and sizes of Business Applications.

At the lower end of the market is the popular TRS-80 (more than 100,000 sold last year). This is a reliable, effective and versatile business microcomputer. Processing speeds and disk storage are more than sufficient for many business applications.

The word processing (using **THE ELECTRIC PENCIL**, is excellent. We also distribute the **TRIDATA** range of business packages (software written by professionals).

WHY PAY MORE?, if the **TRS-80** will do the job.

But if you require a machine with extra capacity, we have suitable systems available.

We would be pleased to discuss your particular requirements with you.

Katanna Management Services Ltd

kms

(In association with S. J. Trott Ltd.)
22 Roughtons,
Galleywood,
Chelmsford,
Essex, CM2 8PF
TEL: (0245) 76127

(member of the computer retailers' association) (TRS-80 is a TANDY' trademark)

VIDEO GENIE **TRS-80.LI.11.16K**

PLAY THE WORLD FAMOUS BOARD GAME
MONOPOLY
AGAINST YOUR COMPUTER

Your computer will be programmed to play Monopoly as a player, so you can play this game by yourself or up to five other players against the computer. With easy fool-proof entry making it ideal even for children to use.

Among its many features is THE COMPUTER'S SKILL AUTOMATICALLY ADJUST throughout the game to that of the best player, keeping the game close and exciting, but always remaining a real challenge.

The game can also be 'SAVED' on tape to be continued, from where you left off, at another time. This program ACHIEVES AN INTELLIGENCE from your computer and will keep it busy for many hours, giving you, your family and friends a great deal of FUN.

COMPUTICS
MICROSOFT

A great deal for £11.95 inclusive.

1 BELL LANE
WHEATLEY
OXFORD OX9 1XY

PROGRAMS, HINTS & TIPS

My thanks to all those members who took no little trouble to provide you with the following helpful programs, hints & tips:

SCROLLING from S166AT

Scroll Up : PRINT@1023,

Scroll down: a machine code routine:

```
 10   Z$ = STRING$ (15,0)
 20   A = VARPTR (Z$)
 30   B = PEEK (A+1) : C = PEEK (A+2) : A = B + C*256
 40   POKE 16526, B : POKE 16527, C
 50   DATA 33, 191, 63, 17, 255, 63, 1, 192, 3
 60   DATA 237, 184, 201
 70   FOR X = 0 TO 11 : READ D : POKE A + X, D
 80   NEXT X
```

Example run:

```
 90   CLS : PRINT @ 475,"SCROLL"
100   FOR X = 0 TO RND (4) : Z = USR (0): NEXT
110   FOR X = 0 TO RND (4) : PRINT @ 1023, : NEXT
120   GOTO 100
```

MELODY PLAYER from S182AT

The source listing below needs assembling. To use poke 16555 with length and 16556 with pitch then USR.

```
 10 ; MUSIC MELODY MAKER
 20 ; by SATNAM  c
 30 ;
 40 LENGTH  EQU 16555D     ;POKE LENGTH HERE
 50 PITCH   EQU 16556D     ;     PITCH HERE
 60         ORG 32500D     ;TOP OF BASIC
 70 START   LD A,(LENGTH)  ;GET LENGTH
 80         CP 0           ;? 0
 90         JP Z,03E3H     ;RETURN TO BASIC IF 0
100         LD H,A         ;LEN - H
110         LD A,(PITCH)   ;GET PITCH
120         CP 0           ;IS IT ZERO ?
130         JP Z,03E3H     ;RETURN IF SO
140         LD L,A         ;PIT - L
150         LD C,L         ;L - C
160 LOOP1   LD B,H         ;H - B
170         LD A,1         ;A = 1
180         OUT (255),A    ;OUT TRS80 1/2 CYCLE
190         OUT (254),A    ;OUT GENIE 1/2 CYCLE
200 LP1     DJNZ LP1       ;FREQ DELAY
210         LD B,H         ;H - B
220         LD A,2         ;A = 2
230         OUT (255),A    ;OUT TRS80 LAST 1/2 CYCLE
240         OUT (254),A    ;OUT GENIE LAST 1/2 CYCLE
250 LP2     DJNZ LP2       ;FREQ DELAY
260         DEC C          ;DEC LENGTH
270         JR NZ,LOOP1    ;REPEAT TILL DONE
280         LD HL,0        ;ZERO HL
290         LD (LENGTH),HL   ;ZERO LENGTH AND PITCH
300         JP 03E3H       ;RETURN TO BASIC
310 ;SET TOP OF BASIC
320         ORG 16561
330         DEFW START-1
340 ;SET MEM SIZE
350         ORG 16598
360         DEFW START
370 ;SET AUTO START
380         ORG 4015H
390         JP START
400         END START
```

*** NOTE ***
Unfortunately I have not been able to check that these listings work as printed.

AT THE READY PROMPT



HERE'S A COUPLE OF PROGRAMS FROM THE 'RENT A PROG' CATALOGUE, JULY 1982. THE SCROLLING UP IDEA, (PRINT@1023) I THOUGHT WAS A RATHER CLEVER IDEA! ED


Joe Kay
Director
NORTH COMPUTER WEST MUSEUM
Create Experience Learn
joe@nwcomputermuseum.org.uk
01942 582826
Information Technology facilities & training
Workshops for programming, Electronics & VR
Community classes, clubs & retro games
IT Skills for the young & old
City & Guilds Creditation
Supporting education of local people
Creating opportunities & careers
Disable friendly
Volunteer-led training & museum operations
NORTH WEST COMPUTER MUSEUM
Create Experience Learn
www.nwcomputermuseum.org.uk
Leigh Spinners Mills, Park Lane, Leigh, WN7 2LB

MAPPING THE MAZE

E.T. FONEHUME

HI DUSTY,
I WAS NATTERING TO MY DAUGHTER-IN-LAW'S SISTER WHO IS A SCHOOL TEACHER FOR CHILDREN WITH 'SPECIAL NEEDS'. IT SEEMED, ONE OF THE MANY DIFFICULTIES SHE ENCOUNTERS WITH THE KIDS IS WHEN IT COMES TO WRITING. THEIR MAIN PROBLEM WAS JUST HOLDING A PEN/PENCIL IN A CONTROLLED MANNER. SHE THOUGHT PRACTICING BY DRAWING LINES ON A MAZE PUZZLE MIGHT BE USEFUL, BUT THE 'FALSE LEADS' WERE TOO MUCH OF AN ADDITIONAL PROBLEM FOR THEM TO MANAGE WITH.

THIS GOT ME THINKING AND I TRIED TO KNOCK-UP A QUICK BASIC PROGRAM THAT MIGHT CREATE A MAZE-LIKE PRINTOUT FOR THE KIDS TO PRACTICE WITH. MY INITIAL IDEA WORKED OK, BUT I DIDN'T WANT SO MANY 'EXTRA' GAPS, EVEN THOUGH THEY WOULD ALL BE VALID WHEN IT CAME TO EXITING THE MAZE.
ALSO, THEORETICALLY, IT COULD BE POSSIBLE TO HAVE NO GAPS AT ALL (USING THE RND VERB)! THIS WAS AN ABSOLUTE 'NO-NO'

THIS IS WHAT I STARTED WITH :-

```
10 RANDOM:CLEAR2500:DIMA$(30):CLS:S$=" "
20 FORZ=1TO30:FORY=1TO50:R=RND(7)
30 IFR=5THENA$(Z)=A$(Z)+S$ELSEA$(Z)=A$(Z)+"*"
40 NEXTY:NEXTZ
50 PRINT:FORZ=1TO30:LPRINTA$(Z):NEXT
READY
>_
```

THE RESULTING PRINT-OUT (FIG. 1) IS OVERLEAF. JUST TOO MANY GAPS! THEN I THOUGHT, HOW ABOUT CREATING A STRING OF "*", SAY 12 LINE TIMES 50 COLUMNS AND EDIT IN JUST THE ONE GAP.



FIG.1 - AS YOU CAN SEE, TOO MANY GAPS PER LINE AND ALSO, I DIDN'T WANT A GAP AT THE START AND/OR END OF A LINE. I FELT IT LOOKED UNTIDY! IT WAS RATHER SLOW TOO.

I HAD FORGOTTEN, EDITING STRINGS IS NOT EASILY DONE IN L2 BASIC. SO AFTER PICKING YOUR BRAINS VIA A COUPLE OF QUICK EMAILS, YOU SUGGESTED I GO DOWN THE 'VARPTR' ROUTE. THIS WORKED BETTER AND AS IT TURNED OUT, WAS VERY MUCH QUICKER. PROBLEM CRACKED, SEE FIG.2 (NEXT PAGE)

```
10 RANDOM:CLEAR1500:DIMA$(30):X=0
20 FORZ=1TO30:A$(Z)=STRING$(50,"*"):NEXT
25 FORY=1TO30
30 M=PEEK(VARPTR(A$(Y))+1)+PEEK(VARPTR(A$(Y))+2)*256
35 PRINTM;
40 R=RND(45)+3:POKEM+R,32
50 NEXTY
99 FORZ=1TO30:LPRINTA$(Z):NEXT
```

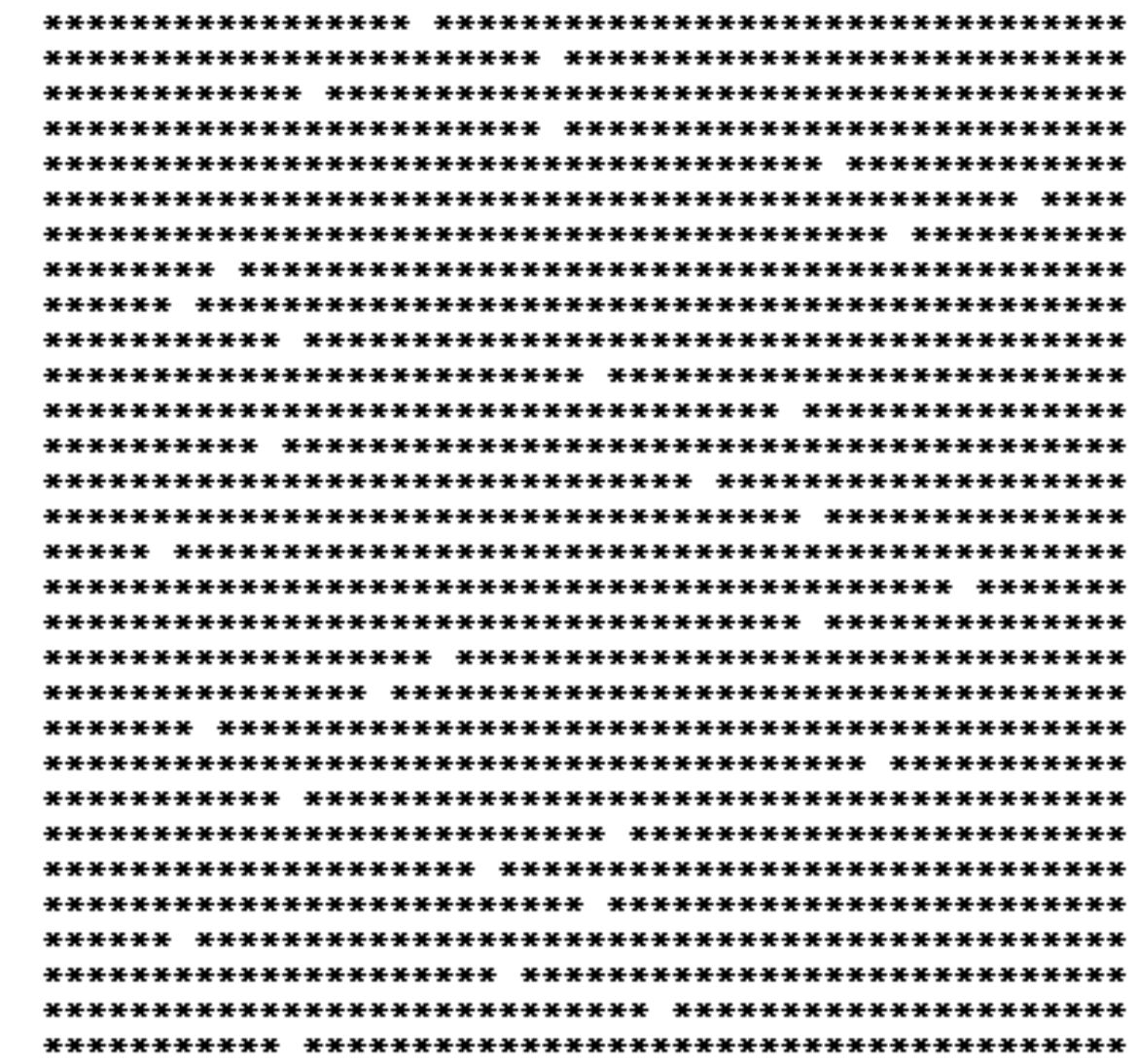


FIG.2 - JUST THE ONE GAP, AND PRODUCED 'LIKE A STREAK OF LIGHTENING FLASHING 'CROSS THE SKY'. (AS THEY SAY IN CTWH!)

IF YOU'RE RUNNING UNDER DISK BASIC OR HAVE A MACHINE WITH GREATER THEN 16K RAM, IT WOULD BE NECESSARY TO CHECK THAT LINE 30 DIDN'T MAKE THE VARIABLE M GRATER THEN 32767. IF YOU DO GET AN 'OV', OVERFLOW ERROR, YOU'LL NEED SOMETHING LIKE M=M-65536. NOT A PROBLEM US ACULAB, M1 L2 16K USERS ARE LIKELY TO ENCOUNTER!

ANOTHER PLUS POINT WAS THAT, THANKS TO GEORGE AND PETER'S TRS80GP'S PRINTING ROUTINE, IT'S VERY SIMPLE TO EDIT THE OUTPUT TO A LARGER FONT SIZE FOR ANY KIDS WITH IMPAIRED VISION PROBLEMS TOO.

AND THEN, OF COURSE, I JUST COULDN'T RESIST GETTING IT DOWN TO A ONE-LINER AND WONDERING, JUST HOW EASY WOULD IT BE TO CARRY ON AND CREATE A 'PROPER' MAZE ONE-LINER? PERHAPS THAT'S FOR ANOTHER DAY EH?

```
0 RANDOM:CLEAR1500:DIMA$(30):X=0:FORZ=1TO30:A$(Z)=STRING$(50,"*"
):NEXT:FORY=1TO30:M=PEEK(VARPTR(A$(Y))+1)+PEEK(VARPTR(A$(Y))+2)*
256:R=RND(45)+3:POKEM+R,32:NEXTY:FORZ=1TO30:LPRINTA$(Z):NEXT
```

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



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

newSOFT

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

IN MAV'S WORKSHOP

FreHD with Joystick - Model I -

Ian Mavric



Did you know it's been 10 years since the FreHD was designed and made available for purchase? The product matured quickly with auto boot Eproms and complete kits ready to plug in and use, growing interest in the Z-80 TRS-80 line, unshackled from being limited to unreliable floppy disk or slow cassette operation.
It's popular because it just plain works. For the few people who had trouble I was there to help and get them back up and running. I'm pleased that no one ever sent a FreHD back for a refund due to it not being as described.

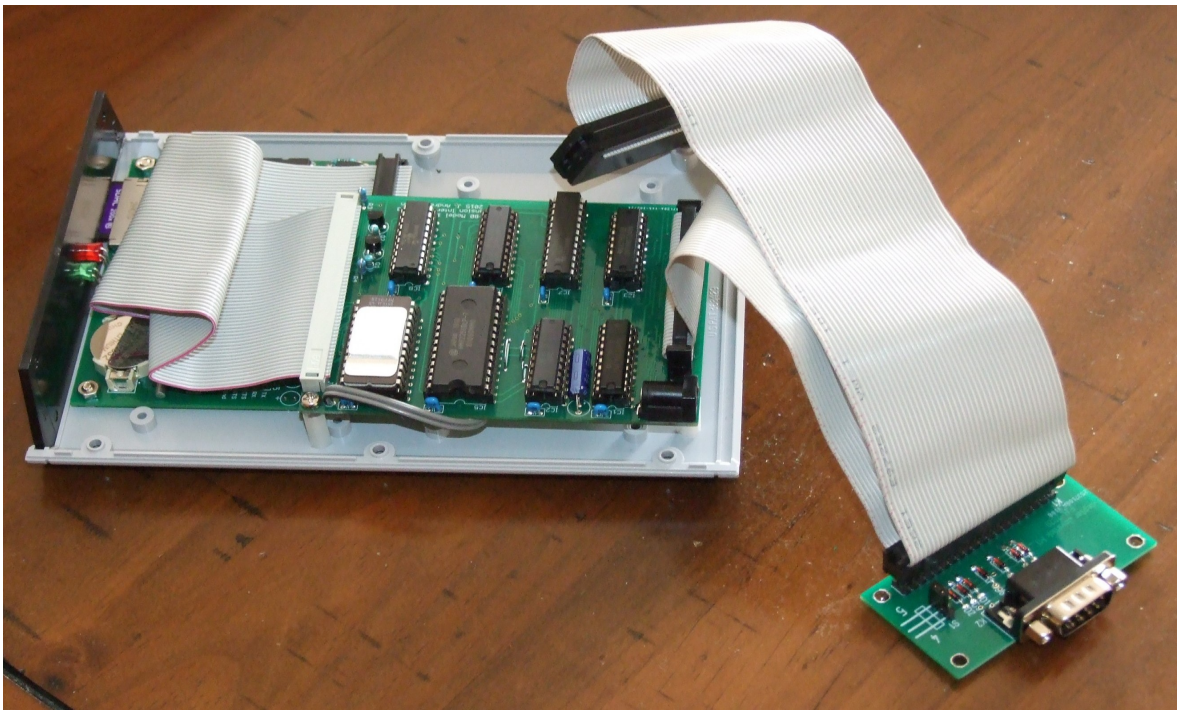
It's basically a hard drive with a battery backed RTC thrown in for good measure. It runs multiple operating systems.... Don't like LDOS because Newdos/80 was your favourite? No problem. You're a CP/M junkie and only use your Model 4 for CP/M apps? Easy.

NOSTALGIC GAME PLAYING IS WHAT MOST TRS-80s DO

Sometimes I am asked by people selling me consignments of vintage computing equipment (not necessarily only TRS-80s) "What do people do with this stuff?". I used to ponder this question with a little anxiety... Am I in a hobby and running a cottage industry catering for a decreasing base that as some finite time not to far away not need what I have to offer? Since I regularly over-think things no only related to vintage computing I decided rather than think about the end of the hobby, focus on what I can do to grow the hobby. So my usual response when people ask me what hobbyists do with the machines once they get them running, I say Nostalgically do what they did with them back when they were new, and play games.

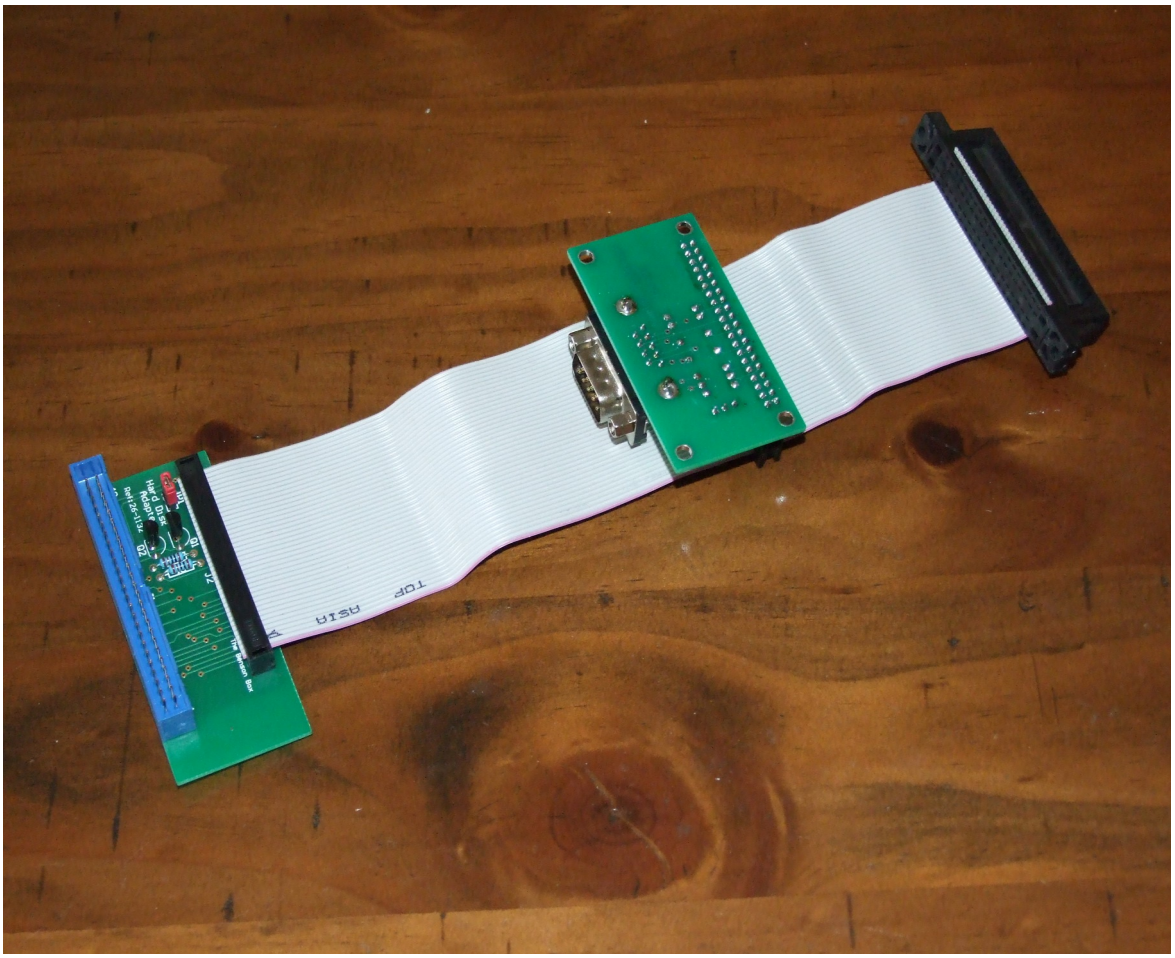
A few years ago Prof-80 reverse engineered the Alpha joystick for the Model I and I have been selling it under the name α-Thing. You can connect it to the screen printer port on the Expansion Interface, or for the handful of people who still use cassettes, directly to the expansion bus on a Model I keyboard unit. Inevitably people would ask me if it worked with the FreHD connected. Since for most enthusiasts the default configuration these days is either an Expansion Interface with a FreHD connected via a hard disk adapter, or directly to the Model I via a Quinnterface, there was some concern about conflicts on the bus.

The good news is that I finally have some definitive answers for those who use a FreHD on a Model I. α-Thing works on systems with a FreHD, all you need to do is crimp another 40-way IDC female connector on the bus cable, and order an α-Thing with a pin connector on it instead of a soldered on 40-way card edge connector.



Picture: α-Thing connected to a FreHD/QI Combo box

Good news for those who have the setup with a Model I Hard Disk Adapter connecting their Expansion Interface to their FreHD, the α-Thing also works that way.



Picture: *α-Thing connected to an (early) M1 HDA direct to Expansion Interface*

WHICH GAMES WORK?

Surprisingly, there isn't a definitive list out there of games which work with a joystick on a Z-80 TRS-80. If someone knows of one please point me in the direction so I can continue testing games to see how well compatible they are with α-Thing.

So far I have tested it with Cosmic Fighter, Volcano Hunter, Frogger* and Zaxxon. I tried Eliminator and it doesn't work... I guess it's fair to say most games advertised as being joystick compatible with the Alpha Products joystick will work as they should. Those that do work are more fun by an order of magnitude than playing them on the keyboard.

* Frogger was interesting in that the version on the dual-boot Frogger sold by Radio Shack (cat. no. 26-1959) works with a joystick just fine, but the original Cornsoft Group version from 1981 doesn't.

As always, it helps to take the time to make sure you are using the most recent version of any program.

CONCLUSION - WHAT'S NEXT, THE MODEL III/4?

The plan for this article was to include the Model III version but like all good plans I ran out of time. The main problem is that I haven't yet found a joystick adapter which works on the Model III as well as the α-Thing does on a Model I. I should have this resolved by the next issue, as I know that many (if not more) people like to play vintage games on their Model III/4 as much more so as Model I owners. So....

NEXT TIME: FreHD with Joystick - Model III

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

---

SYSTEM Tape Name – Peter Phillips

Loading a TRS-80 Level II system tape requires knowing the name of the file. Usually the name is printed on the cassette. But what if you can't read the name or the label is missing?

Using BASIC, you can recover the name of the SYSTEM file with the following program:

```
10 INPUT#-1,A$:PRINT A$
```

It won't print the exact system name, but close. For a SYSTEM tape named `DEPTH` the output looks like:

```
UDEPTH <
```

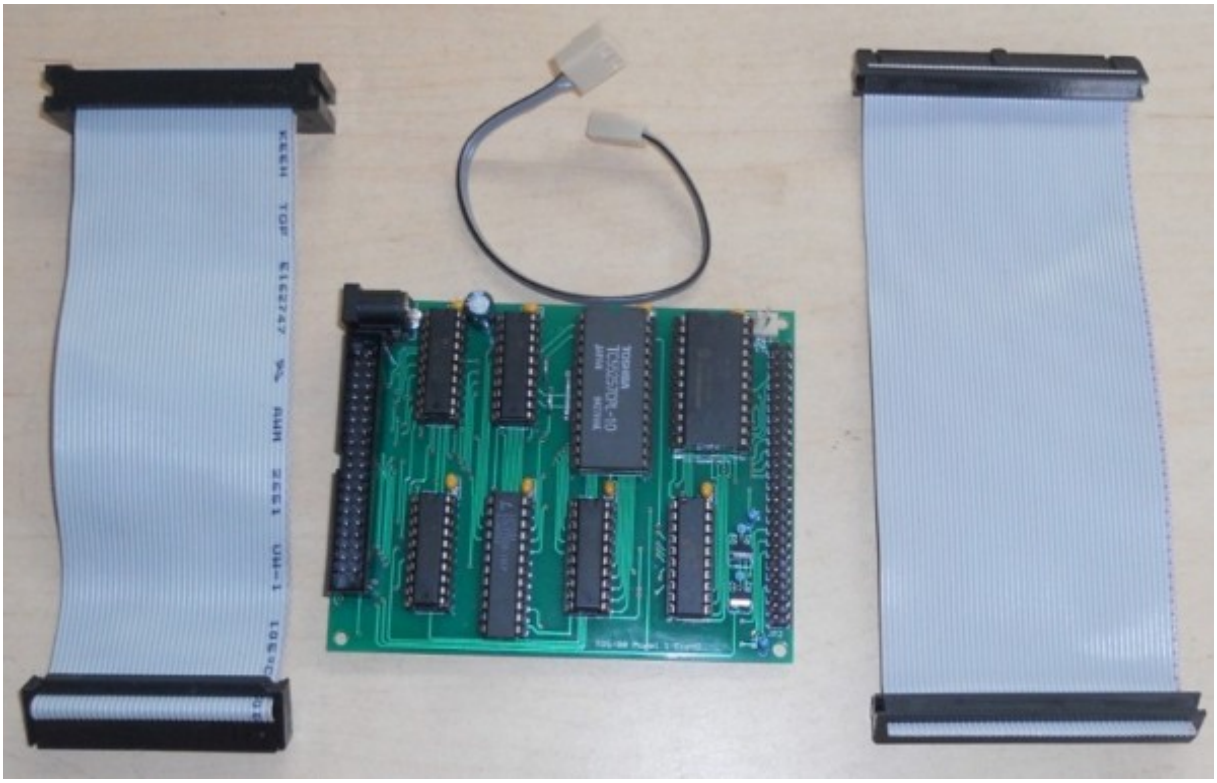
You don't need the whole name, SYSTEM will accept the first letter, e.g. use `D` to load `DEPTH`.

If you're unsure if the tape is a BASIC program, try CLOAD first since it will automatically load the first BASIC program it finds.

This idea in courtesy of "Dave" in a 2004 USENET posting:
https://groups.google.com/g/comp.sys.tandy/c/w4MAQEgSKtQ

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

- COMPLETLY ASSEMBLED AND TESTED -



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

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

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

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

The amazing

"FreHD"



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

Order yours today
http://members.iinet.net.au/~ianmav/trs80/



E.T. FONEHUME

HERE'S A ONE-LINER THAT CREATES A PERSONAL MAGIC SQUARE USING YOUR BIRTH DATE. ENTER THE FOUR 'PARTS' (AS BELOW, FOR THE 07SEP1947) AND YOUR UNIQUE MAGIC SQUARE IS CREATED.
NOTE ALSO, THAT NOT ONLY ALL THE COLUMNS, ROWS AND DIAGONALS ADD UP TO YOUR MAGIC NUMBER, (82 IN THIS CASE), BUT ANY GROUP OF 4 CORNER NUMBER AND THE MIDDLE 4 NUMBERS ALSO ADD UP TO YOUR NUMBER!

THANKS, ONCE AGAIN, TO 'NUMBERPHILE' FOR THE INSPIRATION!

```
TRS-80 Model I
File  Edit  View  Audio  Cassette  Wafer  Hard Drive  Keyboard  Printer  Serial  Record  Debug  Help
? 7,9,19,47
   7   9  19  47

  48  18   6  10

   7   9  49  17

  20  46   8   8

 82
READY
>_

0 CLS:DIMA(16):INPUT D,M,H,Y:A(1)=D:A(8)=D+3:A(10)=D+2:A(15)=D+1
:A(2)=M:A(7)=M-3:A(9)=M-2:A(16)=M-1:A(3)=H:A(6)=H-1:A(12)=H-2:A(
13)=H+1:A(4)=Y:A(5)=Y+1:A(11)=Y+2:A(14)=Y-1:FORX=1TO16STEP4:PRIN
TUSING"####";A(X);A(X+1);A(X+2);A(X+3):PRINT:NEXT:PRINTD+M+H+Y
```



YOU choose which emulation system you want.
Out of:

BBC B
TRS-80 Model I/III/IV
ZX81

State your choice (and choice of keyboard) in "message to seller" upon purchasing.

Whilst we keep all parts in stock we build to order, so allow up to five days for dispatch.
(It's usually done in three.)

ALL systems are preloaded with hundreds if not thousands of games from back in THE day.
And load instantly. (No cassettes!)

Some games support any standard USB controller which you can plug straight into your RX-1000. Keyboard controls are identical to the original now vintage computers of our childhoods.

FREE postage to UK buyers.

Place your order and get ready for the adventure.
Your childhood/youth in a box.
Plug and play.

All software included is in the public domain aside from RX-1000 operating systems which are (C) 2022 Retro Shack.

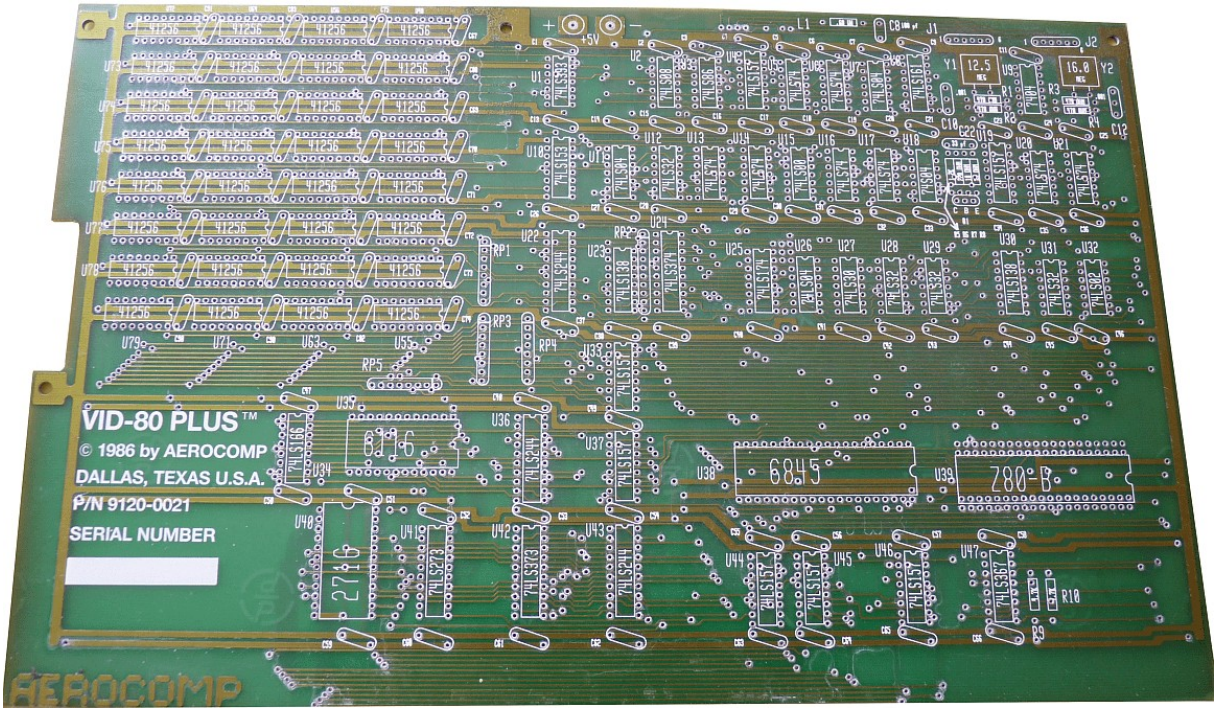
YouTube link below is to a short promo video made when the RX-1000 was developed.

https://youtu.be/eRB8T4bNv6Q

NEWS FROM FRANCE

Liberté • Égalité • Fraternité

PASCAL HOLDRY



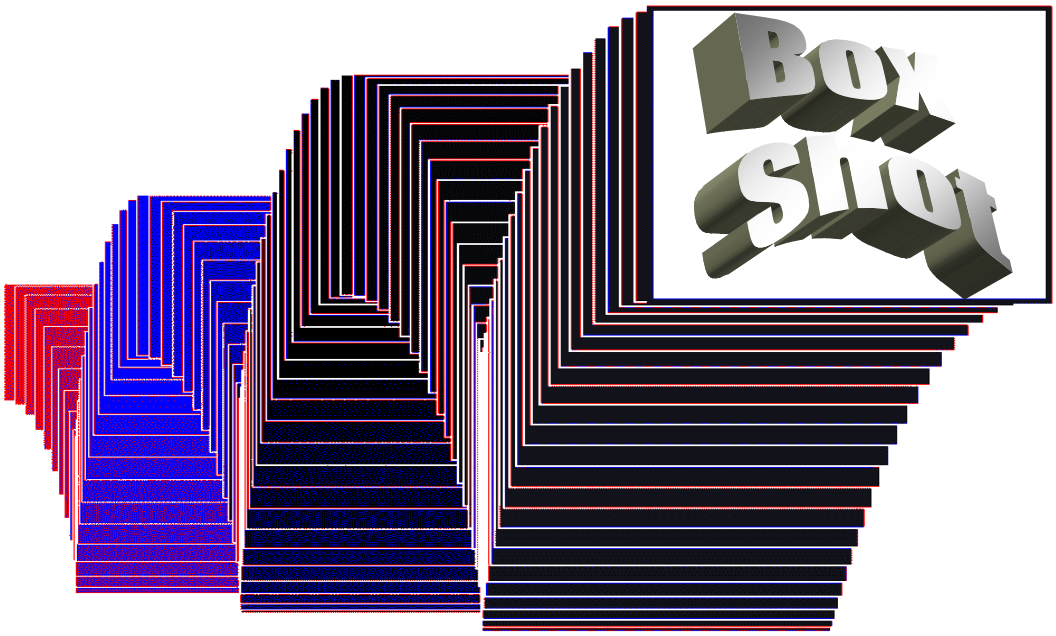
The "PCB" I've had sent to me is an "Aerocomp Vid-80 Plus".
Do you know of anyone who has made this card work, because I make the Shema of this card and I find errors!

It seems that the SRAM IC 6116 is in place of the EPROM 2716, is it a "PCB" printing error or a connection error? The SRAM 6116 seems to be connected as an EPROM and the EPROM 2716 as a SRAM?

Can anyone out there find the contents of the EPROM and the documentation for this card please? It would be great if I have some more information!

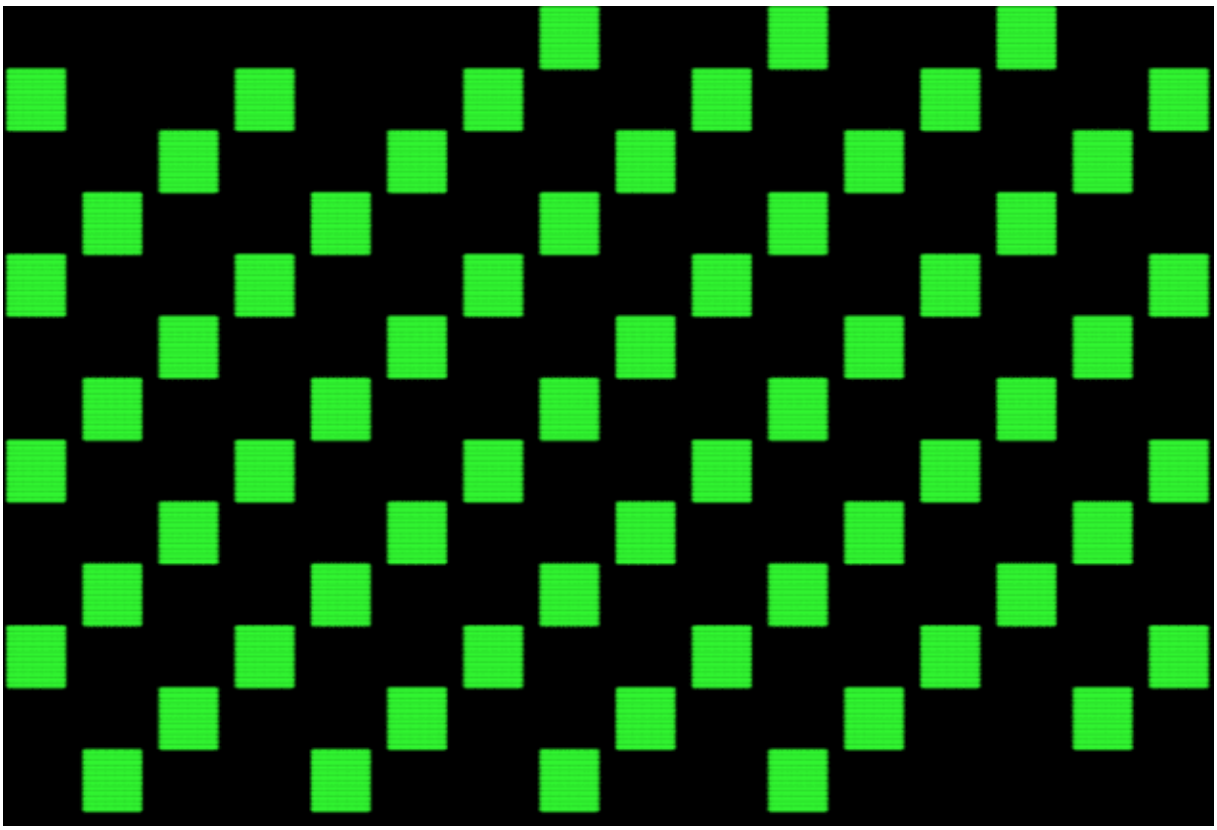
The card is so large that I cannot use my usual tips to clone it and I have to go there per point, it's long, but should get there and it allows me to glimpse possible errors.

The French clone "Guepard" of the TRS-80 M1, works with an IC 6845 for display like this card, and 256 of DRAM memory. I will see if there are similarities.



E.T. FONEHUME

HERE'S A QUICK ONE-LINER JUST FOR FUN. I THINK IT'S ABOUT AS MUCH USE AS A CHOCOLATE TEA-POT BUT WOULD COME IN HANDY IF YOU'RE LOOKING FOR A SCREEN SAVER! (BYW, IF YOU LOOK THE IMAGE BELOW, LONG ENOUGH, DO THEY SEEM TO MOVE ABOUT?)



```
0 RANDOM:DEFINTA-Z:A$=" "+STRING$(3,191)+" ":FORL=0TO1STEP0:CLS:
FORN=RND(63)TORND(411)+612STEPRND(55):PRINT@N,A$:NEXT:FORQ=1TO50
0:NEXT:NEXTL
```



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

Thirsting for adventure?
Brass lantern getting dusty?



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

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



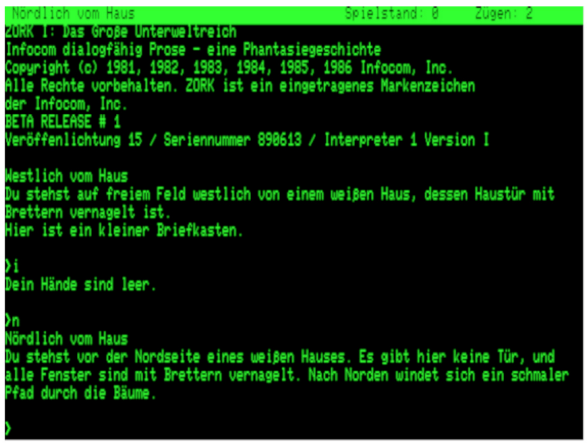
M4ZVM is available for download from https://gitlab.com/sijnstral/m4zvm (includes feature details and source)
Enhanced versions ready for download!

*128K recommended

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



Beyond Zork



Zork 1 (German)

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

IN MAV'S TRAVELS

Ready for TA23?

by Ian Mavric



"Writing about TA23 already? Are you for real, Mav?" No, it's not a miss-print. At the time of publication Tandy Assembly is just under four months away, which means I'll write another article in the September issue of TRS8Bit detailing what is actually happening at TA23. This article is aimed directly at international and long distance travellers to the event, because NOW is the time to start making your plans to attend TA23, and not leave it too late and miss out.

BOOK YOUR FLIGHTS NOW, OR IN THE NEXT MONTH

In case you've been living under a rock of the past decade, it's well known I live in Australia and have the longest distance to the event, and airfare to match. I have been monitoring airfares for the past 3 months and they are stubbornly unmoving so I will be buying my plane ticket in the next 30 days. My experience is that if you leave it to less than two months and especially less than one month you are more likely to see airfares go up rather than down. Last year was particularly brutal. I'll be able to travel for a few hundred dollars less than last year but the airfares for 2017-19 are sadly never returning.

This may not present much a problem for Canadians or people travelling within the USA but getting in early to lock in cheaper fares is still recommended.

People coming from the UK and EU should watch the fares cycle for a week or two to determine which is the cheapest days to book and fly (in Australia Sunday is generally the cheapest day to book a flight and Tuesday or Wednesday the cheapest day to depart the country).

GETTING FROM THE AIRPORT TO SPRINGFIELD

Last year I took a cab from Dayton airport to Springfield at 8 at night and it cost me $US86.00, which was surprisingly good. Previous Google searches for estimations of the taxi ride estimated it would come in between $US110-130. A better plan would be to see who else is flying into DAY and try to co-ordinate car pooling or cab ride sharing to minimise the cost.
TIP: The distance from Columbus (code: CMH) to Springfield is roughly the same as from Dayton, so if it works out cheaper to fly to CMH the cab ride won't be that different.

THEN YOU NEED SOME PLACE TO STAY

I can't stress this enough! Rooms are filling fast, get onto the Marriott soon or risk missing out.

Tandy Assembly will once again be held at the same place it always has been since 2018 which is the Courtyard by Marriott Springfield Downtown (that's the full name of the place) in Springfield Ohio. They have rooms set aside for attendees at a special rate of $US129.00 per night, though they sell out quickly. Best to call them direct on +1 937-322-3600 and say you are attending TANDY COMPUTER CONVENTION (those specific words) because they don't always know what Tandy Assembly means. ¯_(ツ)_/¯

IF YOU DON'T manage to get a room at the Marriott all is not lost. Springfield is a medium sized city with lots of hotels to choose from, many in fact are less expensive than the Marriott so just book one of those and get an Uber or car-pool with other TA23 attendees each day. Don't laugh, people have done that in the past and it works pretty well because we are all looking out for each other.

HOW MANY DAYS FOR A SEASONED (OR UNSEASONED) TRAVELLER

Any time one crosses a time zone you are subject to jetlag and the more time zones you cross the more you become affected by it. When I travel to TA I usually arrive on the Wednesday before the event.
Most people travelling a long distance choose this method also, that way you can settle down for a dinner (if you arrive in the afternoon) or a late dinner if you arrive in the evening), attempt to get some shut-eye and awake refreshed. That's the theory anyway.

Thursday then aims to be a day of combating jetlag by doing something interesting, generally unrelated to computers, to adjust to the new time zone - and in my case new season. Previous years have at visits to the USAF Museum at Dayton, followed by late lunch at Hooters (don't laugh... when in Rome and what-not, the place is an American icon and the meals pretty good as well). Upon returning to the Marriott there is usually afternoon drinks with conversation with the Thursday arrivals, before we are able to set up our Tandy Assembly tables, and the TA organisers put on a nice dinner for the attendees present. This usually results in most of us turning in for the night at between 10:30-11pm. Friday morning comprises a long breakfast and putting the finishing touches on our TA tables before the Tech Track Classes begin at 1pm.

GOING HOME IS USUALLY ON THE MONDAY AFTER THE EVENT

While it may not be something which springs to mind, going home after Tandy Assembly bears consideration. Logically most of the locals or people within a state or two of Ohio will probably go home Sunday night, but people with further to travel will mostly opt to do what I do, which is fly out on Monday. This allows you to get a good night sleep so you aren't racing from the event to either of the airports at Dayton or Columbus to contact a red-eye flight.

It is worth checking with other fly-ins at TA who is returning to either of the airports on Monday and possibly arranging a lift with them to save on a cab fare. Last year at TA22 I was exhausted due to recovering from a cold I had the week before the event, and didn't think to ask about who was headed to Dayton on Monday. While waiting in the airport for my plane out I crossed paths with two attendees who would have gladly driven Jordy and myself to the airport.

NEXT ISSUE - TA23 THE SHOW

I think that covers just about everything there is to know about travelling long distance to Tandy Assembly. 2023 with be the fifth time I have made the trek to TA so I am well versed in the process. If you have any questions feel free to email me. In the next issue of TRS8Bit I'll discuss more of the details which will have been confirmed by then, such as keynote speaker, other presentations, Tech Talk classes, and of course the trivia game.

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



TRS-80
Disc System

Microfloppy disc system giving up to ¼M bytes per disc side and directly compatible with Tandy TRS-80 and Schugart SA400 drives.

Complete self contained unit in smart black case with internal power supply. Trade enquiries only please.

COMPUTER INSTRUMENTATION LTD.
Chandlers Ford, Eastleigh, Hants.,
England, SO5 3YY.
Tel. (04215) 66321. Telex 47326.

**** TRS-80 CHEAP HARD COPY ****
(Also suitable for NASCOM)

A few Olivetti teletypes for private sale at £250 each. Easily connected to TRS-80 keyboard; expansion interface not required! (Connects directly to NASCOM UART). TRS-80 driver package for above: — £50

TRS-80 SOFTWARE

Bargain Games Package!!! 16 games (including Star Trek, Othello, Hangman, Amaze, Slalom eetc) — £25 only!! (+50p post/packing). Disk password cracker — £12.00, TREL (TBUG relocator) £4.00, Labeling Disassembler (outputs to EDTASM) £7.50, KDRVR — allows typing from keyboard to printer. PRINT commands to printer,. keyboard debounce/repeat, lowercase driver — £5.00, DISK EDITOR — alter any sector on disk like Superzap £10.00, Detailed Level 2 memory map — £3.00. All orders add 50p p & p.

Call or write:
Jake Commander,
305, Bronwfield Road,
Shard End,
Birmingham B34 7EA. 747 6964.

Southern Software programs are self-relocating, load-anywhere, machine-code, well documented. They run in 4K Level 2, unless otherwise stated.

SOUTHERN SOFTWARE PO Box 39, Eastleigh, Hants. SO5 5WQ

Increasing capabilities

I HAVE spent a great deal of time, with the aid of the T-Bug monitor program, investigating how the TRS-80 computes, writes J F Hancock of Bristol.

Page E/1 of the Level II manual lists the Basic keywords used in writing Basic programs, but as you have probably discovered many of them can be used only if you have a floppy disc system — they are reserved for use with Level II disc Basic. I hope to show how to use those reserved disc Basic words — and give them new meaning.

When the TRS-80 is switched-on, or re-set, it goes through its housekeeping routine. Part of that routine at 078D Hex, 1933 Dec, programs a section of reserved ram, 4152 to 41A5 Hex, 16722 to 16805 Decimal, with instructions for the words, reserved for disc Basic, giving them their meaning. When you do not have a disc system the instructions in Hex are for example

419A = C3 Jump to NN
419B = 2D LSB (Least significant byte of NN)
419C = 01 MSB (Most significant byte of NN)

That means jump to 012D Hex for your next instruction. The reserved word RSET goes to 419A Hex for its instruction. Type in RSET and enter — press enter — ? L3 ERROR is displayed. Now change those instructions by Poke statements,

419B = 19
419C = 1A

Now type Poke 16795,25: Poke 16796,26 and enter

419A = C3
419B = 19 Jump to 1A19 Hex
419C = 1A

Type RSET and enter and READY is now displayed. Not very useful — but you have given a new meaning to RSET.

For something more useful — a new meaning to the reserved words CLOSE and OPEN, it could be useful to CLOSE off a Basic program already in memory, to cload in another program without losing the first one. There is one stipulation — the program to be cloaded must have line numbers beginning with numbers greater than the last line number of the program already in memory.

To enable us to CLOSE or block-off an existing Basic program, it will help if we know how a Basic program is stored. The housekeeping routine of a Basic program is

42E9 = LSB of start of next Basic line
42EA = MSB of start of next Basic line
42EB = LSB of line number
42EC = MSB of line number

The last byte of your Basic program line is always ZERO. Type in this program line, 10 CLS and enter. Now if you peek at, or use a monitor program, you will see that

ADDRESS: HEX: DEC
42E9 = EF: 239 LSB of start of next Basic line
42EA = 42: 66 MSB of start of next Basic line
42EB = 0A: 10 LSB of Basic line number
42EC = 00: 0 MSB of Basic line number
42ED = 84: 132 CLS
42EE = 00: 0 last byte always zero
42EF LSB of start of next Basic line goes here
42F0 MSB of start of next Basic line goes here

TANDY FORUM is devoted to the Tandy TRS-80. Sometimes we will use it to pass on news about the TRS-80 but, above all, it is for users, and would-be users, of the well-established model I and now the new model II. With your tips, queries, moans and comments, this page can become a market-place for TRS-80 information.



42F1 LSB of line number
42F2 MSB of line number

After switching-on, when your TRS-80 went through its "housekeeping" routine, it programmed

40A4 = E9 LSB of Basic program starting address
40A5 = 42 MSB of Basic program starting address
and
40F9 = EB LSB of address of first line number
40FA = 42 MSB of address of first line number
Now after entering the line: 10 CLS
40A4 = E9 LSB of address start of Basic program
40A5 = 42 MSB of address start of Basic program
but now
40F9 = F1 LSB of address for next line number
40FA = 42 MSB of address for next line number

I hope that you can now see why if you subtract two from the address contained in 40F9 and 40FA which is 42F1, —2 = 42EF and Poke it into 40A4 and 40A5 so that 40A4 = EF: 40A5 = 42, the TRS-80 will assume that your Basic program now starts at 42EE.

So do this, type in Poke 16548,239: Poke 16549,66 and enter. Now type LIST — nothing is listed. The line, 10 CLS, has to all intents and purposes, disappeared. 20 PRINT "THIS IS MY NEXT PROGRAM" and enter and LIST.

To open the Basic in memory so that line 10 CLS becomes available again, all that has to be done is to restore the original starting address in 40A4 and 40A5 Hex. So Poke 16548,233 Poke 16549,66 and enter and LIST.

10 CLS. 20 PRINT "THIS IS MY NEXT PROGRAM" and all is restored again.

A machine code routine to do this is

CLOSE
D9 — EXX — Save what Basic is doing
2A — LD HL,(NN) H = contents of address 40FA
F9 L = contents of address 40F9
40
2B — DEC HL Subtract 2 from HL
2B DEC HL
22 — LD (NN),HL Contents of L into 40A4
A4 Contents of H into
40 40A5
D9 EXX Back to what Basic was doing
C9 RETURN
OPEN
D9 — EXX
21 — LD HL, NN L = E9
E9 H = 42
42
22 — LD (NN),HL L into 40A4
A4 H into 40A5
40
D9 — EXX
C9 — RETURN
Change M/C to Decimal.
CLOSE = 217, 42, 249, 64, 43, 43, 34, 164, 64, 217, 201
OPEN = 217, 33, 233, 66, 34, 164, 64, 217, 201

Decide where to Poke it, so that CLOSE and OPEN can be instructed where to find their new meaning. Suppose you have 16K RAM and that your normally load KBFIX the key de-bounce routine.

CLOSE from 7FB5 to 7FBF Hex
from 32693 to 32703 Dec
OPEN from 7FC0 to 7FC8 Hex
from 32704 to 32712 Dec
WBFTX from 7FC9 to 7FFF Hex
from 32713 to 32767 Dec
CLOSE looks for instructions at 4185-4187 Hex
OPEN looks for instructions at 4179-417B Hex

Here is a Basic program:

```
10 Poke 16774, 181: Poke 16775, 127: Rem Instruct CLOSE
20 Poke 16762, 192: Poke 16763, 127: Rem Instruct OPEN
30 FOR X = 32693 to 32712: READ A:Poke X, A: NEXT
40 DATA 217, 42, 249, 64, 43, 43, 34, 164, 64, 217, 201: REM CLOSE
50 DATA 217, 33, 233, 66, 34, 164, 64, 217, 201: REM OPEN
```

Once this program has been run, it can be deleted, but if you have to re-set the computer the Pokes in lines 10 and 20 will have to be done again. The machine code routines in high memory will stay unchanged.

Remember, whenever you use this program you must protect it in memory by answering MEMORY SIZE in the example with 32692.

TRS8BIT - VOLUME 17 ISSUE 02 - JUNE 2023 - PAGE 50 - TRS8BIT

More power to your TRS-80.

Enhance Your Level II BASIC

Gil Spencer (VK2JK)
Box 300
Spit Junction
NSW 2088, Australia

Every Level II owner worth his salt has by now bought a monitor program like RSM2, Monbug or T-Bug. Any of these fine programs make it tempting to dig around in the Level II ROM to see how it works.

The first thing that you acquire from your searching about is a profound respect for the ROM authors. It's obvious that every byte does plenty of work.

Discoveries

Sooner or later, while cruising through ROM land in your monitor, a lot a familiar road signs will flash past. Pull over to the side of the road near &1660 and have a closer look. Well, for goodness sake! There are all those familiar BASIC words like Run, List and Stop. It must be some sort of look-up table.

And look! There are some other English words hiding in the jumble. There's Merge... and Put...and Open. Wonder what they're there for? Get back into the monitor and let's head for RAM.

As we move through reserved RAM, we pass sign posts all pointing to the same place. There must be 20 or 30 in all, and each one says JP &012D. They're all bunched together near &4170. It must be a popular spot: Let's jump to 8012D and see why it's such an attraction.

Just a hick town, this &012D. A jump there results in this message to the screen—L3 Error. The Level II Manual says this about L3 Error: "Disk Basic only: An attempt was made to use a statement, function or command which is available only when the TRS-80 Mini Disk is connected via the Expansion Interface."

Gosh, I guess you need about $1000 to visit all those exotic L3 destinations. Well, I probably didn't want to go there anyway!

One day a stranger appeared on the shelves at the local Tandy store. No black jacket this! It's a manual with a rich brown jacket and it's called *TRSDOS & Disk Basic Reference Manual* (Catalog No. 26-2104).

A brief reading reveals that all those strange English words in the Level II ROM near &1660 are Disk Basic words available only to those rich enough to afford all that extra gear.

```
10 DEF FNRAD(R)=(3.14159/180)*R
.
.
.
100 X=COS(FNRAD(W))
```

Degree to Radian Conversion Functions

Fig. 1

Digging for Answers

The Level II ROM is a pretty complex can of worms. There's a lot to learn about how it all fits together. If I live long enough it may all become clear.

With the use of a monitor, ROM can be unraveled a small bit at a time. A few things are now obvious:

- The Level II BASIC interpreter can recognize Disk BASIC words.
- While the interpreter recognizes the words it cannot implement them. It's just not quite smart enough. You might say it knows the words...but not the music!
- Most Disk BASIC words result in a jump to &012D via the links near &4170. Thus, an L3 Error results.

A bit more effort and most the mystery is solved. Table 1 shows what goes where. Each of these links is a jump: there are 28 of them. Twenty-four are accessed by a Disk BASIC word. The other four are reached from a particular point in ROM.

We've now established that RAM, from &4152 to &41A5, is reserved for Disk BASIC jump links.

Let's look a little further. There's also a pattern in RAM from &41A6 to &41E4. Dividing this area into three-byte lumps, we notice that each lump starts with a &C9 (hex for the op code RETurn).

This type of three-byte lump looks like it's a link for a Call from somewhere in ROM. The

(Note: In the text, numbers prefixed with & are in hexadecimal notation)

Call has simply been short-circuited with a RETurn. The three-byte Jump could be replaced with JP nn (which is three bytes long). In that way we could link new code into a particular spot in ROM. As long as the new code ends with a RETurn, we'll go right back to the calling routine in ROM.

An investigation of the Call links ultimately gives us Table 2.

Disk BASIC Word Links

RAM ADDRESS	LVL II CONTENTS	LINK FOR	JUMP FROM
&4152	JP &012D	CVI	T*
&4155	"	FN	&2524
&4158	"	CVS	T
&415B	"	DEF	T
&415E	"	CVD	T
&4161	"	EOF	T
&4164	"	LOC	T
&4167	"	LOF	T
&416A	"	MKI$	T
&416D	"	MKS$	T
&4170	"	MKD$	T
&4173	"	CMD	T
&4176	"	TIME$	&2510
&4179	"	OPEN	T
&417C	"	FIELD	T
&417F	"	GET	T
&4182	"	PUT	T
&4185	"	CLOSE	T
&4188	"	LOAD	T
&418B	"	MERGE	T
&418E	"	NAME	T
&4191	"	KILL	T
&4194	"	&	&24C8
&4197	"	LSET	T
&419A	"	RSET	T
&419D	"	INSTR	&2506
&41A0	"	SAVE	T
&41A3	"	LINE	T

JP &021D implements 'L3 ERROR'

*T = JUMP from BASIC word look-up table. Can be reached from Keyboard or from Program.

Table 1

Other Disk BASIC Links

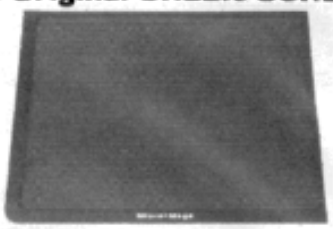
RAM ADDRESS	LVL II CONTENTS	LINK WITH	CALLED FROM
&41A6	RET nn	ERROR	&19EC
&41A9	"	USR	&27FE
&41AC	"	READY	&1A1C
&41AF	"	KB INPUT	&0368
&41B2	"	BASIC INPUT	&1AA1
&41B5	"	" "	&1AEC
&41B8	"	" "	&1AF2
&41BB	"	NEW/END	&1B8C and &1DB0
&41BE	"	VDU SELECT	&2174
&41C1	"	OUTPUT TO DEVICE	&032C
&41C4	"	KB SCAN	&0358
&41C7	"	RUN	&1EA5
&41CA	"	PRINT	&206F
&41CD	"	—	&20C6
&41D0	"	CRLF	&2103
&41D3	"	TAB	&2108 and &2141
&41D6	"	COMMAND "INPUT"	&219E
&41D9	"	—	&2AEC
&41DC	"	READ	&222D
&14DF	"	READ/LIST	&2278 and &2B44
&41E2	RET nn	SYSTEM	&02B2

RET nn = 'Dead End' with room for JUMP or CALL instruction

Table 2

for the TRS-80 from Micro-Mega

The Original GREEN-SCREEN ↵29



The eye-pleasing Green-Screen fits over the front of your TRS-80 Video Display and gives you improved contrast with reduced glare. You get bright luminous green characters and graphics like those featured by more expensive CRT units.

Don't confuse the Original Green-Screen with a piece of thin film stuck to the face of your video tube, such as that advertised by others. The Original Green-Screen is mounted in a full frame perfectly matched to the color and texture of the TRS-80 Video Display. It is attached with adhesive strips which do not mar your unit in any way.

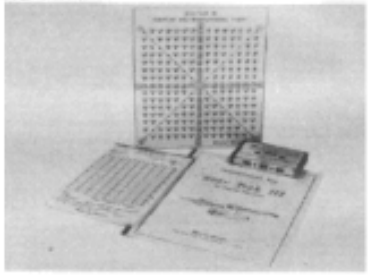
The full frame design of the Original Green-Screen "squares off" the face of your video display and greatly improves the overall appearance of your system.

(Specify whether for Model I or Model III)

THE GREEN-SCREEN.....$13.95

Add $1.00 for postage and handling.

THE ULTIMATE STAR TREK PACKAGE



Tired of trivial computer games? This complete Star Trek package will provide you with endless fascination and challenge. In addition to the program cassette, it includes comprehensive instructions, a pad of "Voyage Log" record sheets, and a free-standing "Torpedo and Maneuvering Chart."

The package is built around the latest version of Lance Micklus' incomparable Star Trek III, a 13,000 byte program with a host of subtle and imaginative features, which include numerous dynamic and spectacular graphic displays. Star Trek III puts you in command of the Enterprise cruising in a galaxy of 192 quadrants filled with uncharted hazards, including hostile Klingons, pulsars, and black holes. You have at your disposal scanners, various weapons and defense systems, on-board computers, and a loyal crew. (You will need them all to survive the Klingons.)

Your mission is to rid the region of Klingons and to locate five inhabitable planets, all within 300 stardays, before returning to Star Fleet Headquarters where your overall effectiveness as a starship commander will be scored. High scores are possible only with careful planning and effective battle tactics. The "Voyage Log" sheets will guide your strategy, and the "Torpedo and Maneuvering Chart" will give you a vital edge in combat. (When you engage three Klingon ships you can't afford to miss.)

STAR TREK PACKAGE (for Level II, 16K only).....$22.95

Add $1.00 for postage and handling

Terms: Check or money order, no CODs or credit cards, please. Add amount shown for postage and handling to price of the item. All items shipped within 48 hours by first class or priority mail. Virginia residents, add 4% sales tax.

Micro-Mega · P.O. Box 6265 · Arlington, Va 22206

↵See List of Advertisers on page 306

Of the 21 links, 18 are each called from only one place in ROM. Three are called from two places. The column in Table 2 headed Link with is a bit suspect. That's because I haven't yet positively figured out what ROM is up to when the calls are given. When you work them out for certain, let me know!

So What?

A fair question. In my case, a study of these link tables started me on the road to a machine language program called Level II and 1/2. The source and object codes (called Twohaf) are reproduced in Program Listings 1 and 2. This short program (not much over 1K bytes) gives access to selected Disk BASIC commands. It also offers the ability to quickly renumber BASIC lines as you go along in a quite sophisticated way.

Finally, Twohaf provides a facility for appending one BASIC program to the end of another—concatenating BASIC programs.

The source of Twohaf was written using Tandy's EDTASM (version 1.2) on a 16K machine. Because of RAM limitations it had to be written in two separate parts. When the object code from both parts is put together, we have Level II and 1/2.

Like most of us, I have looked at many enhancement programs for the Level II TRS-80. Many are very elaborate and represent a lot of work. The trouble is that they take up too much room in RAM while providing a lot of functions that I personally never use. Furthermore, each enhancement is equipped with a multi-page manual that keeps getting misplaced.

It finally came time to make a list of things that would be useful enhancements for Level II for my applications.

The list was short:

1. Enable DEF FN (DEFine FuNction)
2. Enable DEFUSR (DEFine USeR)
3. Enable hexadecimal notation.
4. Enable BASIC line renumbering.
5. Enable appending one BASIC program to another.

All these functions are around as parts of various enhancements on the market. No one utility that I have found offers them all.

DEF FN (DEFine FuNction)

This is perhaps the most useful Disk BASIC command for my kind of programming. It permits the programmer to define a numeric or string function. After definition, the function can be called one or many times in a program. The Disk BASIC manual (referenced earlier) explains DEF FN.

A sample of the use of DEF FN is shown in Fig. 1. Note that the variable used to define a function need not match the variable used when the function is called.

Remember that a function definition is not limited to one variable. It's also useful to realize that one function can be used within the definition of an-

Program Listing 1

```
                00100 ; < TWOHAF >     PART 1                  V 1.2   800401
                00100 ; < TWOHAF >     PART 1                  V 1.2   800401
                00110 ; A TRS-80 LEVEL II BASIC ENHANCEMENT PROGRAM
                00120 ;
                00130 ;                GIL SPENCER (VK2JK)
                00140 ;                BOX 300 / SPIT JUNCTION, NSW  2088
                00150 ;                AUSTRALIA
                00160 ;
                00170 ; THIS PROGRAM ENABLES THESE NEW BASIC FUNCTIONS -
                00180 ;        1.  DEF FN       -DEFINE FUNCTION
                00190 ;        2.  DEFUSR       -DEFINE USER
                00200 ;        3.  &            -PREFIX FOR HEX NOTATION
                00210 ;        4.  LINE         -LINE RENUMBERING COMMAND
                00220 ;        5.  MERGE        -SET CONCATENATE PARTITION
                00230 ;        6.  RSET         -RESET  "        PARTITIONS
                00240 ;
                00250 ; FOR DETAILS, SEE TANDY TRSDOS & DISK BASIC MANUAL
                00260 ;
                00270 ; THIS PROGRAM IS LOADED BEGINNING AT -
42E9            00280          ORG     42E9H   ; (THE USUAL START OF BASIC)
                00290 ;
                00300 ; NOTE - ACTIVE PART IS PROTECTED BY RELOCATING THE
                00310 ;        USUAL START OF BASIC HIGHER IN MEMORY.
                00320 ;        THE INITIALIZATION ROUTINE IS OVERWRITTEN
                00330 ;        AFTER EXIT FROM TWOHAF.
                00340 ;
42E9 3AAF40     00350 AS5      LD      A,(40AFH)
42EC 3D         00360          DEC     A
42ED 3D         00370          DEC     A
42EE 3D         00380          DEC     A
42EF B7         00390          OR      A
42F0 37         00400          SCF
42F1 CA5643     00410          JP      Z,AS6
42F4 4E         00420          LD      C,(HL)
42F5 23         00430          INC     HL
42F6 46         00440          LD      B,(HL)
42F7 C5         00450          PUSH    BC
42F8 FA1343     00460          JP      M,AS7
42FB 23         00470          INC     HL
42FC 4E         00480          LD      C,(HL)
42FD 23         00490          INC     HL
42FE 46         00500          LD      B,(HL)
42FF C5         00510          PUSH    BC
4300 E21343     00520          JP      PO,AS7
4303 23         00530          INC     HL
4304 DA0A43     00540          JP      C,AS8
4307 211D41     00550          LD      HL,411DH
430A 4E         00560 AS8      LD      C,(HL)
430B 23         00570          INC     HL
430C 46         00580          LD      B,(HL)
430D 23         00590          INC     HL
430E C5         00600          PUSH    BC
430F 4E         00610          LD      C,(HL)
4310 23         00620          INC     HL
4311 46         00630          LD      B,(HL)
4312 C5         00640          PUSH    BC
4313 DA6843     00650 AS7      JP      C,AS9
4316 CF         00660 AS1      RST     8
4317 BE         00670          CP      (HL)
4318 3E80       00680          LD      A,80H
431A 32DC40     00690          LD      (40DCH),A
431D B6         00700          OR      (HL)
431E 47         00710          LD      B,A
431F C31226     00720          JP      2612H
4322 CD1643     00730 ASC      CALL    AS1
4325 3AAF40     00740          LD      A,(40AFH)
4328 B7         00750          OR      A
4329 F5         00760          PUSH    AF
432A 22F340     00770          LD      (40F3H),HL
432D EB         00780          EX      DE,HL
432E 7E         00790          LD      A,(HL)
432F 23         00800          INC     HL
4330 66         00810          LD      H,(HL)
4331 6F         00820          LD      L,A
4332 B4         00830          OR      H
4333 CA4E44     00840          JP      Z,AS2
4336 7E         00850          LD      A,(HL)
4337 FE28       00860          CP      28H
4339 C28943     00870          JP      NZ,AS3+1
433C D7         00880          RST     16
433D 22D840     00890          LD      (40D8H),HL
4340 1802       00900          JR      AS4
```

Program continues

UNI-TERM UNIVERSAL TERMINAL PROGRAM

The first and only fully intelligent terminal program for BOTH Mod I and Mod III users! Includes all features found as Terminal programs costing MUCH more, plus many not found anywhere else. Includes extensive documentation and handsome binder.

Price $79.95

RUN YOUR OWN COMPUTER BULLETIN BOARD

Become a CONNECTION-80 SYSOP with an AutoConnection Modem, Message-80 BBS Software (by Richard Taylor) and CONNECTION-80 Enhancements (by Tom Vande-Strouwe) all for only $389.00 ($90 off regular price).

Other Bulletin Board and Electronic Mail Packages available. Call for Info.

LIGHT PEN

Mod I and III Light pen. Needs only a 9 Volt Battery. Attaches through the Cassette recorder for ease of operation. Easy to use and program. Comes complete with hardware and extensive documen-tation and sample routines. All this power for only $12.95

TIC-TOC 8Ø Micro·Clock

Put a wrist watch on your Computer. Gives Time, Date, and Day of Week, without the need of operator input. Runs on batteries, so it never needs to be reset. Gives you the ability to enter data with the knowledge that the Date and Time are correct. Includes complete documentation, and sample routines.

Mod I Version $99.00
Mod III Version $109.00

THE COPYRIGHT KIT

A self-instruction booklet on copyrighting the computer programs you write.

INCLUDES: Step by Step instructions, sample forms, as well as discussions of copyrights, patents and trade secrets, your rights secured by copyright, legal remedies upon infringements, material not copyrightable and MUCH MORE! IF YOU EVER WROTE A COMPUTER PROGRAM YOU NEED THIS BOOK! Written by Attorneys. Published by National Attorneys' Publications, Inc. and distributed exclusively through B.I. Enterprises.

ONLY ..... $11.95 Dealer Inquiries Invited

MODEL III SPEED UP!

Load in the 500 Baud system tapes in 1/3 the time with copy III. This utility will read in you 500 baud system tapes, and then write them out at 1500 baud. This gives you backup protection as well as speeding up your tapes. A must for all Model III Cassette users!

ONLY $14.95

MODEL III DISK DIRECTORY PROGRAM

Catalogue your Diskettes with this easy to use Catalogue program for Model III Disk systems. Can run on as little as a 32K 1 Disk system. Help you keep track of ALL of your programs.

ONLY $19.95

FREE!

Keyboard DeBounce Kit for TRS-80 Model I with Key-bounce Problems. Solves the CAUSE of Key-bounce. No software necessary. Call or Write for your FREE Kit, or [✓ 122]

FREE!

DRAW 80

DRAW-80 is a machine language routine that allows you to draw graphics to the screen, and then save them for future use. Accepts input from Keyboard, Joystick, or Light Pen. Easy to use, with full documentation.

DRAW-80 (Alone) $6.95
When purchased with light pen $4.95

MODEL III MEMORY KIT

Upgrade your Model III to 32 or 48 K with the RIGHT Memory Chips. These are Prime Ceramic 150ns (Fast!!) chips. Beware of slower chips, as future Mod III upgrade may not work with them!

16K set $45.00
32K set $80.00

BE Enterprises

171 Hawkins Road ✓122
Centereach, New York 11720

(516) 981-8568 (Voice)
(516) 588-5836 (Data)
MNET-70331, 105

Dealer Inquiries Welcome

VISA

Add $2.00 S & H
NYS res. add appr. tax

See List of Advertisers on page 306

COMPUTER EQUIPMENT & SOFTWARE BARGAINS



EVERY MONTH

BUY, SELL OR TRADE ALL TYPES OF COMPUTER EQUIPMENT AND SOFTWARE (pre-owned and new) among 20,000 readers nationwide.

FEATURES:

- Low classified ad rates - 10¢ a word
- Hundreds of ads from individuals
- Categorized ads so you can find them instantly
- Large (11 by 14") easy to read pages

Subscribe now for $10 and receive 13 issues/year (one FREE plus 12 regular issues). After receiving your first issue if you're not completely satisfied you may have a 100% refund and you still keep the first issue free. Bank cards accepted.

BONUS: *If you have something to advertise (pre-owned or software) send in a classified ad with your subscription and we'll run it FREE.*

The Nationwide Marketplace for Computer Equipment

COMPUTER SHOPPER ⌐212

P.O. BOX F27 ● TITUSVILLE, FL 32780 ● 305-269-3211

MasterCard & VISA subscriptions only, call **TOLL FREE 1-800-528-6050 Ex. 184**

STOCK MARKET SOFTWARE

By H & H Trading Company for Mod. I, II, & III TRS-80®s

STOCK TRACKER™ times trades on individual stocks and options. Uses a technical volume analysis. Completely objective. Gives buy, sell signals.

MARKET TRACKER™ times tops & bottoms on the Dow Industrials; a composite of six technical indicators. Some judgement required. Gives buy, sell signals.

These two programs are accurate.
Hundreds of satisfied users.

ASK FOR THE EVIDENCE!

FOR MORE INFORMATION OR TO ORDER, CONTACT:

H & H TRADING COMPANY ⌐223

POST OFFICE BOX 23546
PLEASANT HILL, CALIFORNIA 94523
Telephone 415/937-1030

VISA & MASTERCARD

®Regd. T.M. of Radio Shack ©


```
4342 CF       00910 AS10   RST     8
4343 2C       00920        INC     L
4344 0E04     00930 AS4    LD      C,04
4346 CD6319   00940        CALL    1963H
4349 3E80     00950        LD      A,80H
434B 32DC40   00960        LD      (40DCH),A
434E CD0D26   00970        CALL    260DH
4351 EB       00980        EX      DE,HL
4352 37       00990        SCF
4353 C3E942   01000 AS11   JP      AS5
4356 D22324   01010 AS6    JP      NC,2423H
4359 D5       01020        PUSH    DE
435A EB       01030        EX      DE,HL
435B CD8828   01040        CALL    2888H
435E D1       01050        POP     DE
435F AF       01060        XOR     A
4360 E5       01070 AS9    PUSH    HL
4361 F5       01080        PUSH    AF
4362 EB       01090        EX      DE,HL
4363 7E       01100        LD      A,(HL)
4364 FE29     01110        CP      29H
4366 20DA     01120        JR      NZ,AS10
4368 2AF340   01130        LD      HL,(40F3H)
436B CF       01140        RST     8
436C 28E5     01150        JR      Z,AS11
436E 2AD840   01160        LD      HL,(40D8H)
4371 CD0D26   01170 AS13   CALL    260DH
4374 E3       01180        EX      (SP),HL
4375 CD2B1F   01190        CALL    1F2BH
4378 7E       01200        LD      A,(HL)
4379 FE29     01210        CP      29H
437B 2807     01220        JR      Z,AS12
437D CF       01230        RST     8
437E 2C       01240        INC     L
437F E3       01250        EX      (SP),HL
4380 CF       01260        RST     8
4381 2C       01270        INC     L
4382 18ED     01280        JR      AS13
4384 D7       01290 AS12   RST     16
4385 E3       01300        EX      (SP),HL
4386 CF       01310        RST     8
4387 29       01320        ADD     HL,HL
4388 3ED5     01330 AS3    LD      A,0D5H
438A CF       01340        RST     8
438B D5       01350        PUSH    DE
438C CD3723   01360        CALL    2337H
438F 2B       01370        DEC     HL
4390 D7       01380        RST     16
4391 C29719   01390        JP      NZ,1997H
4394 E7       01400        RST     32
4395 282F     01410        JR      Z,AT1
4397 D1       01420 AT6    POP     DE
4398 F1       01430 AT4    POP     AF
4399 283F     01440        JR      Z,AT2
439B 301C     01450        JR      NC,AT3
439D E1       01460        POP     HL
439E C1       01470        POP     BC
439F 70       01480        LD      (HL),B
43A0 2B       01490        DEC     HL
43A1 71       01500        LD      (HL),C
43A2 FA9843   01510        JP      M,AT4
43A5 2B       01520        DEC     HL
43A6 C1       01530        POP     BC
43A7 70       01540        LD      (HL),B
43A8 2B       01550        DEC     HL
43A9 71       01560        LD      (HL),C
43AA E29843   01570        JP      PO,AT4
43AD 2B       01580        DEC     HL
43AE C1       01590        POP     BC
43AF 70       01600        LD      (HL),B
43B0 2B       01610        DEC     HL
43B1 71       01620        LD      (HL),C
43B2 2B       01630        DEC     HL
43B3 C1       01640        POP     BC
43B4 70       01650        LD      (HL),B
43B5 2B       01660        DEC     HL
43B6 71       01670        LD      (HL),C
43B7 18DF     01680        JR      AT4
43B9 D5       01690 AT3    PUSH    DE
43BA F5       01700        PUSH    AF
43BB E7       01710        RST     32
43BC 11D340   01720        LD      DE,40D3H
43BF CC8828   01730        CALL    Z,2888H
43C2 F1       01740        POP     AF
43C3 C31A28   01750        JP      281AH
43C6 2AB340   01760 AT1    LD      HL,(40B3H)
43C9 EB       01770        EX      DE,HL
43CA 2A2141   01780        LD      HL,(4121H)
43CD DF       01790        RST     24
43CE 3805     01800        JR      C,AT5
43D0 CD4328   01810        CALL    2843H
43D3 18C2     01820        JR      AT6
43D5 D1       01830 AT5    POP     DE
43D6 21D340   01840        LD      HL,40D3H
43D9 E5       01850        PUSH    HL
43DA CDF529   01860 AT2    CALL    29F5H
43DD 7E       01870        LD      A,(HL)
43DE 22B340   01880        LD      (40B3H),HL
43E1 E1       01890        POP     HL
43E2 77       01900        LD      (HL),A
```

Program continues

```
43E3 23        01910          INC     HL
43E4 71        01920          LD      (HL),C
43E5 23        01930          INC     HL
43E6 70        01940          LD      (HL),B
43E7 18AF      01950          JR      AT4
43E9 FEC1      01960 DEF      CP      0C1H
43EB 2852      01970          JR      Z,DE1
43ED CD1643    01980          CALL    AS1
43F0 CD2828    01990          CALL    2828H
43F3 EB        02000          EX      DE,HL
43F4 73        02010          LD      (HL),E
43F5 23        02020          INC     HL
43F6 72        02030          LD      (HL),D
43F7 EB        02040          EX      DE,HL
43F8 7E        02050          LD      A,(HL)
43F9 FE28      02060          CP      28H
43FB C2051F    02070          JP      NZ,1F05H
43FE D7        02080          RST     16
43FF CD0D26    02090 DE2      CALL    260DH
4402 7E        02100          LD      A,(HL)
4403 FE29      02110          CP      29H
4405 CA051F    02120          JP      Z,1F05H
4408 CF        02130          RST     8
4409 2C        02140          INC     L
440A C3FF43    02150          JP      DE2
440D F1        02160 USR      POP     AF
440E CD2D44    02170          CALL    US1
4411 D5        02180          PUSH    DE
4412 CD2C25    02190          CALL    252CH
4415 E3        02200          EX      (SP),HL
4416 4E        02210          LD      C,(HL)
4417 23        02220          INC     HL
4418 46        02230          LD      B,(HL)
4419 21E726    02240          LD      HL,26E7H
441C E5        02250          PUSH    HL
441D C5        02260          PUSH    BC
441E 3AAF40    02270          LD      A,(40AFH)
4421 F5        02280          PUSH    AF
4422 FE03      02290          CP      3
4424 CCDA29    02300          CALL    Z,29DAH
4427 F1        02310          POP     AF
4428 EB        02320          EX      DE,HL
4429 212141    02330          LD      HL,4121H
442C C9        02340          RET
442D D7        02350 US1      RST     16
442E 010008    02360          LD      BC,8
4431 3005      02370          JR      NC,US2
4433 D630      02380          SUB     30H
4435 17        02390          RLA
4436 4F        02400          LD      C,A
4437 D7        02410          RST     16
4438 EB        02420 US2      EX      DE,HL
4439 211F4A    02430          LD      HL,4A1FH
443C 09        02440          ADD     HL,BC
443D EB        02450          EX      DE,HL
443E C9        02460          RET
443F CD2D44    02470 DE1      CALL    US1
4442 D5        02480          PUSH    DE
4443 CF        02490          RST     8
4444 D5        02500          PUSH    DE
4445 CD022B    02510          CALL    2B02H
4448 E3        02520          EX      (SP),HL
4449 73        02530          LD      (HL),E
444A 23        02540          INC     HL
444B 72        02550          LD      (HL),D
444C E1        02560          POP     HL
444D C9        02570          RET
444E 1E2E      02580 AS2      LD      E,2EH
4450 C3A219    02590          JP      19A2H
4453 110000    02600 HEX      LD      DE,0
4456 D7        02610          RST     16
4457 2B        02620          DEC     HL
4458 0605      02630          LD      B,5
445A 23        02640 HX6      INC     HL
445B 7E        02650          LD      A,(HL)
445C CD3E1E    02660          CALL    1E3EH
445F EB        02670          EX      DE,HL
4460 380A      02680          JR      NC,HX3
4462 FE3A      02690          CP      3AH
4464 3819      02700          JR      NC,HX4
4466 D630      02710          SUB     30H
4468 3815      02720          JR      C,HX4
446A 1806      02730          JR      HX5
446C FE47      02740 HX3      CP      47H
446E 380F      02750          JR      NC,HX4
4470 D637      02760          SUB     37H
4472 29        02770 HX5      ADD     HL,HL
4473 29        02780          ADD     HL,HL
4474 29        02790          ADD     HL,HL
4475 29        02800          ADD     HL,HL
4476 B5        02810          OR      L
4477 6F        02820          LD      L,A
4478 05        02830          DEC     B
4479 CA0207    02840          JP      Z,0702H
447C EB        02850          EX      DE,HL
447D 18DB      02860          JR      HX6
447F CD9A0A    02870 HX4      CALL    0A9AH
4482 EB        02880          EX      DE,HL
4483 C9        02890          RET
4484 00        02900 LAST     DEFB    0
0000           02910          END
00000 TOTAL ERRORS
```


PROSOFT™

Dept C, Box 839 / No Hollywood, Ca 91603 / (213) 764-3131

INTRODUCES

NewScript



PROFESSIONAL WORD PROCESSING

<*> for the <*>

TRS-80® Models I and III

Powerful editing and formatting features formerly available only on large IBM Time-Sharing computers

Easy-to-use true Full-Screen Editing

Typeahead - never loses keystrokes

Flexible - big documents, Form Letters, Index, Table of Contents, block move/copy, global change

EPSON® MX-80 - all 12 fonts, including EMPHASIZED

Line Printer IV - Right-justified Proportional font (ours was the first complete system to do so)

PLUS: underlining, sub scripts, super scripts

double-width, Centering, and much more

Diablo® and others: underlining and good support

Excellent documentation - hundreds of examples

Introductory Price: $79.95

(until 7/31/81, then $99.95, and still a bargain!)

Requires 48K, runs under TRSDOS®, NEWDOS, NEWDOS/80®, VTOS® 4.0, LDOS®. (Upgrade privilege from SUBSCRIPT applies to current licensees.)

DVORAK

Keyboard translator with press-on labels! $19.95
Typing Tutor (requires a translator)! $19.95
Special! Both DVORAK programs! $34.95
(32K disk systems)

Software Speedups

FASTER $29.95

Analyses executing BASIC programs, then identifies a simple program change to improve their execution speed. NO hardware changes are involved, works with packages as well as your own code and can reduce run-times by 10—50%. Example: "move selection" in "Othello" dropped from 48 to 32 seconds. Runs on 16-48K Level II tape or disk. Models I and III. Written in Z-80. This will be one of your most valuable utilities!

XTEND40 $19.95

Quickly upgrades Model I 35-track disks to 40 tracks with your 40-track drives and DOS.

<*> ORDERING <*>

We accept checks, C.O.D's, charge, and even cash.
Telephone orders accepted for M/C and VISA.

Please add $3.00 for shipping to orders under $25.00. Add 6% tax in Calif., and 10% outside U.S.A.

441

See List of Advertisers on page 306

other.

DEFUSR (DEFine USeR)

Ten different USR routines (USR0 through USR9) can be defined. The purpose of DEFUSR is to tell the BASIC interpreter the start address of a machine language subroutine.

Values can be passed from BASIC to USR and vice versa. Consult both the Level II BASIC manual and the Disk BASIC manual for the complete story.

Hexadecimal Notation

Disk BASIC offers you a way to use both hexadecimal and octal notation in addition to decimal notation. This is a great help to the programmer. It avoids the errors made when code is available in hex (or octal) and needs to become part of a BASIC program (which requires decimal).

Level II and 1/2 skips the octal option. There just doesn't seem to be much demand for it, which simplifies matters. A number without a prefix is assumed to be decimal. A number with an ampersand (&) prefix is assumed to be hexadecimal. Thus, in Level II and 1/2 the interpretation of these two statements is identical: 10 DEFUSR3 = 27648, 10 DEFUSR3 = &6C00. Level II and 1/2 does not support hex notation in data or input statements.

Program Listing 2

```
                02920 ; < TWOHAF >     PART 2                    V 1.2   800401
                02930 ;
                02940 ;                GIL SPENCER (VK2JK)
                02950 ;                BOX 300 / SPIT JUNCTION, NSW  2088
                02960 ;                AUSTRALIA
                02970 ;
                02980 ; THE FOLLOWING LABELS, DEFINED IN PART 1, ARE NEEDED IN
                02990 ; PART 2 -
4322            03000 ASC      DEFL    4322H
43E9            03010 DEF      DEFL    43E9H
448D            03020 USR      DEFL    448DH
4453            03030 HEX      DEFL    4453H
4484            03040 LAST     DEFL    4484H
                03050 ;
4484            03060          ORG     LAST    ;CONNECT TO 1ST PART OF TWOHAF
                03070 ;
4484 CD611B     03080 LINE     CALL    1B61H
4487 011E1D     03090          LD      BC,1D1EH
448A 2B         03100          DEC     HL
448B D7         03110          RST     16
448C C5         03120          PUSH    BC
448D 010A00     03130          LD      BC,000AH
4490 C5         03140          PUSH    BC
4491 50         03150          LD      D,B
4492 58         03160          LD      E,B
4493 2826       03170          JR      Z,NU1
4495 FE2C       03180          CP      2CH
4497 2809       03190          JR      Z,NU2
4499 D5         03200          PUSH    DE
449A CD4F1E     03210          CALL    1E4FH
449D 42         03220          LD      B,D
449E 4B         03230          LD      C,E
449F D1         03240          POP     DE
44A0 2819       03250          JR      Z,NU1
44A2 CF         03260 NU2      RST     8
44A3 2C         03270          INC     L
44A4 CD4F1E     03280          CALL    1E4FH
44A7 2812       03290          JR      Z,NU1
44A9 F1         03300          POP     AF
44AA CF         03310          RST     8
44AB 2C         03320          INC     L
44AC D5         03330          PUSH    DE
44AD CD5A1E     03340          CALL    1E5AH
44B0 C29719     03350          JP      NZ,1997H
44B3 7A         03360          LD      A,D
44B4 B3         03370          OR      E
44B5 CA4A1E     03380          JP      Z,1E4AH
44B8 EB         03390          EX      DE,HL
44B9 E3         03400          EX      (SP),HL
44BA EB         03410          EX      DE,HL
44BB C5         03420 NU1      PUSH    BC
44BC EB         03430          EX      DE,HL
44BD 22E545     03440          LD      (NU0),HL
44C0 EB         03450          EX      DE,HL
44C1 CD2C1B     03460          CALL    1B2CH
44C4 D1         03470          POP     DE
44C5 D5         03480          PUSH    DE
44C6 C5         03490          PUSH    BC
44C7 CD2C1B     03500          CALL    1B2CH
44CA 60         03510          LD      H,B
44CB 69         03520          LD      L,C
44CC D1         03530          POP     DE
44CD DF         03540          RST     24
44CE EB         03550          EX      DE,HL
44CF DA4A1E     03560          JP      C,1E4AH
44D2 D1         03570          POP     DE
44D3 C1         03580          POP     BC
44D4 F1         03590          POP     AF
44D5 D5         03600          PUSH    DE
44D6 180E       03610          JR      NU3
44D8 09         03620 NU5      ADD     HL,BC
44D9 DA4A1E     03630          JP      C,1E4AH
44DC EB         03640          EX      DE,HL
44DD E5         03650          PUSH    HL
44DE 21F9FF     03660          LD      HL,0FFF9H
44E1 DF         03670          RST     24
44E2 E1         03680          POP     HL
44E3 DA4A1E     03690          JP      C,1E4AH
44E6 D5         03700 NU3      PUSH    DE
44E7 5E         03710          LD      E,(HL)
```

Program continues

BASIC Line Renumbering

This is certainly one facility that every programmer wishes he had at his finger tips. A line renumberer should be easy to use; should be fast; should fix all pointers (GOTO, GOSUB, etc.); and should be flexible (variable start points, variable increments, etc.).

Level II and 1/2 offers all these advantages. About the only thing it does not do is change any reference to line numbers that you may have buried in REM statments.

The Disk BASIC word that activates the line renumbering routine in Level II and 1/2 is Line. Used by itself, it assumes these default values:

- New first line number = 10
- Renumber *all* lines
- Increment = 10

The Line command can control all three of these parameters. For example, LINE 1000, 500, 20 means: new first line number = 1000; start renumbering with old line number 500; increment = 20.

Such control is extremely handy when you need to insert a large block of numbers in the middle of a BASIC program to accommodate a new routine.

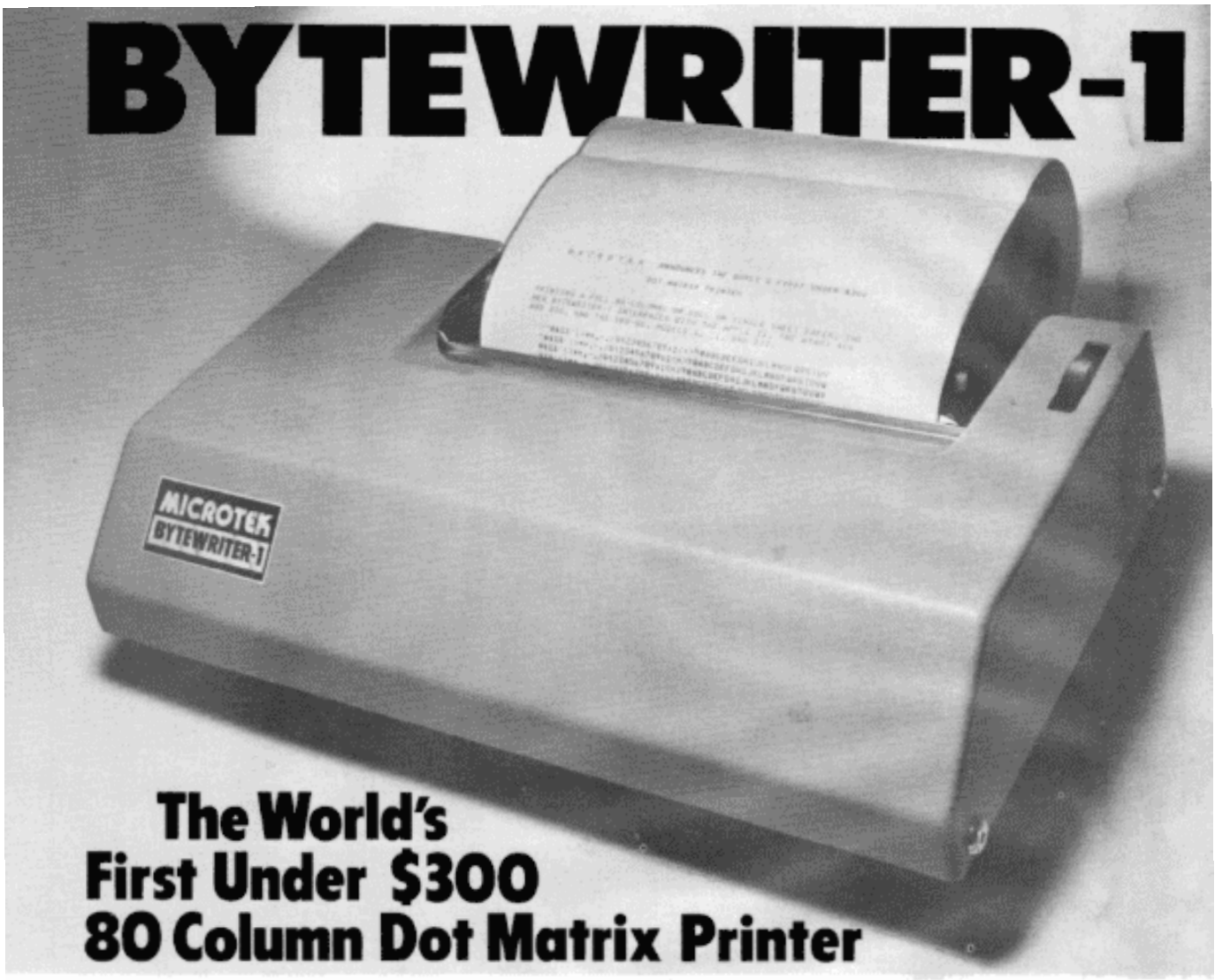
Default values and specific values can be mixed in the Line command. For example – LINE, 100 means: new first line number = 10; renumber *all* lines; increment = 100.

Concatenating BASIC Programs

In EDP the word Merge means to shuffle or blend together. Append or Concatenate is a subset of Merge, meaning place next to each other.

As the word Append is not in the Disk BASIC vocabulary, I had to settle for Merge. Here's how you use it: You have a BASIC program called John in RAM. John's highest line number is

BYTEWRITER-1



The World's First Under $300 80 Column Dot Matrix Printer

PRICE BREAKTHROUGH FROM MICROTEK

Bytewriter-1, the $299 printer that has taken the industry by storm! A product made possible through cost-effective design, efficient manufacturing operations, a proven low-cost print mechanism and the simplest possible control electronics. We have done extensive testing with all three models of the TRS-80, the Apple II, and the Atari 400 and 800. That is why we can guarantee you won't have any interface problems with the Bytewriter-1.

TRS-80 is a trademark of Radio Shack, Div. of Tandy Corp.
Apple II is a trademark of Apple Computer, Inc.
Atari 400 & 800 are trademarks of Atari, Inc.
Bytewriter-1 is a trademark of Microtek, Inc.

BYTEWRITER-1 SPECIFICATIONS

Printing Technology:	7-wire bi-directional impact wire matrix
Printing Speed:	60 lines per minute (80 characters per second) continuous
Character Set:	96 character ASCII (upper and lower case)
Character Size:	10 characters per inch (80 columns per line) plus expanded printing
Paper:	Friction feed (synchronous), accepts single sheet and roll paper up to 9½ inches maximum width. Prints original plus 3 copies.

By the way, our replacement print-head costs less than $30 too!
Call or write today for more information.

Microtek, Inc. ~360
9514 Chesapeake Drive
San Diego, CA 92123
Tel. 714-278-0633
Outside Calif. call toll free: 800-854-1081
TWX. 910-335-1269

MICROTEK inc.

990. You have another BASIC program called Les on cassette, and you want to add Les to the bottom end of John. In Level II and 1/2, simply enter the word Merge. The VDU will say OK. RAM is now apparently empty (like after New). CLOAD Les in the usual way. At this stage the line numbers of Les do not matter. After CLOAD it will appear as though only Les is in RAM. You can run it, change it, anything you like. You must renumber Les so that the lowest line number is higher than 990, which you can do quickly using the Level II and 1/2 Line command. The Line command will not affect John.

If you had a third program on cassette named Bruce that you want to add beneath Les, use the word Merge again. Then CLOAD Bruce and continue as with Les. Remember that Bruce must start at a higher line number than Les finished at.

When you've got all the pieces put together (and the only limit is how much available RAM you have), you're ready for the last command in the sequence, which is the Disk BASIC word RSET. When you enter RSET, the VDU will say OK. Now you've got one big BASIC program! Easy as that!

Details

You can add Level II and 1/2 to your arsenal in several ways.

- Use Listings 1 and 2. These listings have both the source code and the object code. If you have EDTASM I'd suggest you key the source into the EDTASM buffer. After all, you are sure to decide to improve it.
- If you don't have EDTASM, you can key in the object code using T-Bug or a similar utility. Use the two left-most columns of Listings 1 and 2. Be sure to get the right code in the right addresses.
- If you can't type (or don't want to) write to me and we can arrange either a trade or sale of a cassette version of the program.

After you have a copy of Level II and 1/2, it's easy to use. Load it into your Level II machine via System. The name is Twohaf. Hit the slant line command and everything's done. You're back in BASIC with extra functions at your fingertips.

To remind you that the enhancement is in there, a message will be printed before each ready prompt. This is nice when you've got as many versions around as I have!

The Level II and 1/2 code starts at the beginning of Level II free RAM (&42E9). Positioning it here means that you can almost forget about it. All Level II commands work as always. You can even set memory size for machine language routines in high memory.

You do lose about 1100 bytes of free RAM. Of course, if you find yourself with the memory size query again, you'll have to reload Twohaf, as the links are all set back to normal Level II by the bootstrap.

Knowing how the links in Tables 1 and 2 work will spark other ideas for you. It's nice to have some extra commands available that you can give from the keyboard.

For example, it is not at all unusual for me to have RSM2 loaded in high memory (&6C00 and higher) while jumping back and forth between BASIC and machine language. It's easy enough to get from RSM2 to BASIC—G 1A19 does that. Getting back to RSM2 used to be a hassle. I have to first type System, then slant line, followed by 27648 (or is it 26748?). That's now all changed. Only six bytes did it. The beginning of RSM2 is &6C00 and fortunately it's also the entry point. So six bytes were added just before &6C00.

```
6BFA  :  21 00 6C  :  LD HL,6C00
6BFD  :  22 80 41  :  LD (4180),HL
```

After the addition, RSM2 was re-recorded using: P 6BFA 7E00 6BFA. The result of this addition is to link RSM2 with the Disk BASIC word Get.

Now all that's needed from BASIC is to enter Get—and, presto, we're back in RSM2. The same approach can be used for your favorite monitor or other frequently needed machine language utility.

Now, if we could only speed up the baud rate of the cassette I/O, maybe we won't need disks after all... ■

```
44E8 7B       03720          LD      A,E
44E9 23       03730          INC     HL
44EA 56       03740          LD      D,(HL)
44EB B2       03750          OR      D
44EC EB       03760          EX      DE,HL
44ED D1       03770          POP     DE
44EE 2807     03780          JR      Z,NU4
44F0 7E       03790          LD      A,(HL)
44F1 23       03800          INC     HL
44F2 B6       03810          OR      (HL)
44F3 2B       03820          DEC     HL
44F4 EB       03830          EX      DE,HL
44F5 20E1     03840          JR      NZ,NU5
44F7 C5       03850 NU4      PUSH    BC
44F8 CD1846   03860          CALL    NU0
44FB CD2345   03870          CALL    NU6+1
44FE 2AE545   03880          LD      HL,(NU0)
4501 EB       03890          EX      DE,HL
4502 CD2C1B   03900          CALL    1B2CH
4505 60       03910          LD      H,B
4506 69       03920          LD      L,C
4507 C1       03930          POP     BC
4508 D1       03940          POP     DE
4509 D5       03950 NU8      PUSH    DE
450A 5E       03960          LD      E,(HL)
450B 7B       03970          LD      A,E
450C 23       03980          INC     HL
450D 56       03990          LD      D,(HL)
450E B2       04000          OR      D
450F 280D     04010          JR      Z,NU7
4511 EB       04020          EX      DE,HL
4512 E3       04030          EX      (SP),HL
4513 EB       04040          EX      DE,HL
4514 23       04050          INC     HL
4515 73       04060          LD      (HL),E
4516 23       04070          INC     HL
4517 72       04080          LD      (HL),D
4518 EB       04090          EX      DE,HL
4519 09       04100          ADD     HL,BC
451A EB       04110          EX      DE,HL
451B E1       04120          POP     HL
451C 18EB     04130          JR      NU8
451E 011A1A   04140 NU7      LD      BC,1A1AH
4521 C5       04150          PUSH    BC
4522 FEF6     04160 NU6      CP      0F6H
4524 AF       04170          XOR     A
4525 32E745   04180          LD      (FB),A
4528 2AA440   04190          LD      HL,(40A4H)
452B 2B       04200          DEC     HL
452C 23       04210 NU9      INC     HL
452D 7E       04220          LD      A,(HL)
452E 23       04230          INC     HL
452F B6       04240          OR      (HL)
4530 C8       04250          RET     Z
4531 23       04260          INC     HL
4532 5E       04270          LD      E,(HL)
4533 23       04280          INC     HL
4534 56       04290          LD      D,(HL)
4535 D7       04300 NW7      RST     16
4536 B7       04310 NW3      OR      A
4537 28F3     04320          JR      Z,NU9
4539 4F       04330          LD      C,A
453A 3AE745   04340          LD      A,(FB)
453D B7       04350          OR      A
453E 79       04360          LD      A,C
453F 2857     04370          JR      Z,NW1
4541 FE9E     04380          CP      9EH
4543 2025     04390          JR      NZ,NW2
4545 D7       04400          RST     16
4546 FE8D     04410          CP      8DH
4548 20EC     04420          JR      NZ,NW3
454A D7       04430          RST     16
454B FE8E     04440          CP      8EH
454D 20E7     04450          JR      NZ,NW3
454F D5       04460          PUSH    DE
4550 CDFE45   04470          CALL    NW4
4553 7A       04480          LD      A,D
4554 B3       04490          OR      E
4555 281B     04500          JR      NZ,NW5
4557 E5       04510          PUSH    HL
4558 2AE845   04520          LD      HL,(FC)
455B 2B       04530          DEC     HL
455C 212888   04540          LD      HL,28H
455F 2B       04550          DEC     HL
4560 212888   04560          LD      HL,28H
4563 2B       04570          DEC     HL
4564 213888   04580          LD      HL,38H
4567 E1       04590          POP     HL
4568 182A     04600          JR      NW6
456A FE8E     04610 NW2      CP      8EH
456C 28C7     04620          JR      NZ,NW7
456E D5       04630          PUSH    DE
456F CDFE45   04640          CALL    NW4
4572 E5       04650 NW5      PUSH    HL
4573 CD2C1B   04660          CALL    1B2CH
4576 0B       04670          DEC     BC
4577 3E8D     04680          LD      A,8DH
4579 384B     04690          JR      C,NW8
457B CDF920   04700          CALL    20F9H
457E 21D645   04710          LD      HL,NW9
4581 D5       04720          PUSH    DE
```

Program continues

```
4582 CDA728   04730          CALL    28A7H
4585 E1       04740          POP     HL
4586 CDAF0F   04750          CALL    0FAFH
4589 C1       04760          POP     BC
458A CD8A46   04770          CALL    NX1
458D E1       04780          POP     HL
458E E5       04790          PUSH    HL
458F C5       04800          PUSH    BC
4590 CDA70F   04810          CALL    0FA7H
4593 E1       04820 NX6      POP     HL
4594 D1       04830 NW6      POP     DE
4595 2B       04840          DEC     HL
4596 189D     04850          JR      NW7
4598 FE0D     04860 NW1      CP      0DH
459A 2099     04870          JR      NZ,NW7
459C D5       04880          PUSH    DE
459D CDFE45   04890          CALL    NW4
45A0 EB       04900          EX      DE,HL
45A1 23       04910          INC     HL
45A2 23       04920          INC     HL
45A3 23       04930          INC     HL
45A4 4E       04940          LD      C,(HL)
45A5 23       04950          INC     HL
45A6 46       04960          LD      B,(HL)
45A7 EB       04970          EX      DE,HL
45A8 60       04980          LD      H,B
45A9 69       04990          LD      L,C
45AA CDEB46   05000          CALL    NX3
45AD EB       05010          EX      DE,HL
45AE 13       05020          INC     DE
45AF 2AE845   05030          LD      HL,(PC)
45B2 2B       05040          DEC     HL
45B3 2B       05050          DEC     HL
45B4 2B       05060          DEC     HL
45B5 0E05     05070          LD      C,5
45B7 1A       05080 NX5      LD      A,(DE)
45B8 B7       05090          OR      A
45B9 283B     05100          JR      Z,NX4
45BB 77       05110          LD      (HL),A
45BC 23       05120          INC     HL
45BD 13       05130          INC     DE
45BE 0D       05140          DEC     C
45BF 20F6     05150          JR      NZ,NX5
45C1 D1       05160 NX7      POP     DE
45C2 2B       05170          DEC     HL
45C3 C33545   05180          JP      NW7
45C6 219345   05190 NW8      LD      HL,NX6
45C9 E5       05200          PUSH    HL
45CA 2AE845   05210          LD      HL,(PC)
45CD E5       05220          PUSH    HL
45CE 2B       05230          DEC     HL
45CF 70       05240          LD      (HL),B
45D0 2B       05250          DEC     HL
45D1 71       05260          LD      (HL),C
45D2 2B       05270          DEC     HL
45D3 77       05280          LD      (HL),A
45D4 E1       05290          POP     HL
45D5 C9       05300          RET
45D6 42       05310 NW9      DEFM    'BAD BRANCH TO '
45E4 00       05320          DEFB    0
0002          05330 NU0      DEFS    2
0001          05340 FB       DEFS    1
0002          05350 PC       DEFS    2
45EA B7B6     05360 PJ       DEFW    0B6B7H
45EC 9D9F     05370          DEFW    9F9DH
45EE C28E     05380 NX2      DEFW    8EC2H
45F0 B4B5     05390          DEFW    0B5B4H
45F2 8DCA     05400          DEFW    0CA8DH
45F4 9195     05410          DEFW    9591H
45F6 3620     05420 NX4      LD      (HL),20H
45F8 23       05430          INC     HL
45F9 0D       05440          DEC     C
45FA 20FA     05450          JR      NZ,NX4
45FC 18C3     05460          JR      NX7
45FE 23       05470 NW4      INC     HL
45FF 5E       05480          LD      E,(HL)
4600 23       05490          INC     HL
4601 56       05500          LD      D,(HL)
4602 23       05510          INC     HL
4603 22E845   05520          LD      (PC),HL
4606 2B       05530          DEC     HL
4607 C3781D   05540          JP      1D78H
460A D5       05550 NX1      PUSH    DE
460B C5       05560          PUSH    BC
460C 3E20     05570          LD      A,20H
460E 1E05     05580          LD      E,5
4610 0B       05590 NX8      DEC     BC
4611 02       05600          LD      (BC),A
4612 1D       05610          DEC     E
4613 20FB     05620          JR      NZ,NX8
4615 C1       05630          POP     BC
4616 D1       05640          POP     DE
4617 C9       05650          RET
4618 2AA440   05660 NX0      LD      HL,(40A4H)
461B 7E       05670 NX2      LD      A,(HL)
461C 23       05680          INC     HL
461D B6       05690          OR      (HL)
461E 2830     05700          JR      Z,NX1
4620 23       05710          INC     HL
4621 23       05720          INC     HL
```

Program continues

SPECIAL DELIVERY

WORDPROCESSING
- POWER -
for the TRS-80®

"...If you're presently looking for a mailing list processor, this represents the current state of the art."
80 MICROCOMPUTING - 80 REVIEWS - JULY 1980

MAILFORM is data entry at its best, just fill in the form! FAST, EASY to use functions include: search, sort, extract, page forward and back, 'Transparent cursor', insert/delete characters, and MORE!

MAILRITE prints 'personalized' form letters by inserting information from MAILFORM into Electric Pencil®, Scripsit®, or BASIC text files. Print letters, labels, even envelopes! Boldface, underscore, change margins, pause, print 'unprintable' characters, and MORE!

XTRA! includes: MAILFORM; MAILRITE - with capability of printing variable text from a 'key' file; MAILABEL - 1, 2, 3, or 4 across label printer; and MAILSORT - sort a full 40 track double density data diskette in only 48K!

ALL MACHINE LANGUAGE
means unsurpassed
SPEED, RELIABILITY & EASE OF USE

For VISA, Master Card & COD orders only
Call NOW - TOLLFREE
(800) 824-7888
ASK FOR OPERATOR 203
California (800) 852-7777
Hawaii & Alaska (800) 824-7919
For more information call (214) 233-3998
(Requires min 32K single disk drive)

FOR THE MODEL I & III
SPECIAL DELIVERY $125
XTRA SPECIAL DELIVERY ..$199

FOR THE MODEL II
SPECIAL DELIVERY $199
(Requires Mod II DOS version 2.0)

TRS-80 is a registered trademark of Tandy Corp.

VISA

software concepts
13534 Preston Rd. Suite 142
Dallas, Texas 75240

Dealer Inquiries invited ↙299

↙See List of Advertisers on page 306

THE PROGRAMMER'S GUILD MEANS ADVENTURE! !

SPIDER MOUNTAIN

SEARCH FOR GOLD AND GLORY IN THE TUNNELS OF SHELOB'S LAIR.

TAPE $14.95 **DISK $19.95**

LOST DUTCHMAN'S GOLD

YOU AND THE GHOST OF BACKPACK SAM AGAINST THE TREACHEROUS TERRAIN AND HOSTILE INDIANS OF THE SUPERSTITION MOUNTAINS.

TAPE $14.95 **DISK $19.95**

LOST SHIP

DISCOVER THE SECRET OF THE FLOATING PIRATE VESSEL IN THE MYSTERIOUS DEVIL'S TRIANGLE.

TAPE $14.95 **DISK $19.95**

DEATH DREADNAUGHT

CAN YOU ESCAPE THE CREATURE'S DEADLY PURSUIT AND MAKE IT TO THE SHUTTLE ALIVE? RATED R DUE TO VIOLENT DESCRIPTIONS.

TAPE $14.95 **DISK $19.95**

DRAGONQUEST

RESCUE THE PRINCESS BEFORE SUNDOWN FROM SMAEGOR THE TERRIBLE. MACHINE CODE, KEY—BOARD MACRO.

TAPE $15.95 **DISK $21.95**

THUNDER ROAD

HILARIOUS TAKEOFF ON THE "DUKES OF HAZARD" CAN YOU GET YOUR MOONSHINE TO KNAWBONE BEFORE SHERIFF BUBBA SHUTS YOU DOWN?

TAPE $14.95 **DISK $19.95**

DEADLY DUNGEON

DISCOVER THE SECRET OF THE DEADLY DUNGEON AND EMERGE VICTORIOUS AND WEALTHY.

TAPE $14.95 **DISK $19.95**

★ ★ ★ **NEW RELEASE** ★ ★ ★

TEMPLE OF THE SUN

EXPLORE AN ANCIENT AZTEC TEMPLE SEARCHING FOR THE SHAMAN'S SECRET.

TAPE $14.95 **DISK $19.95**

COMPUTER MAIL ORDER IS HERE! ! !

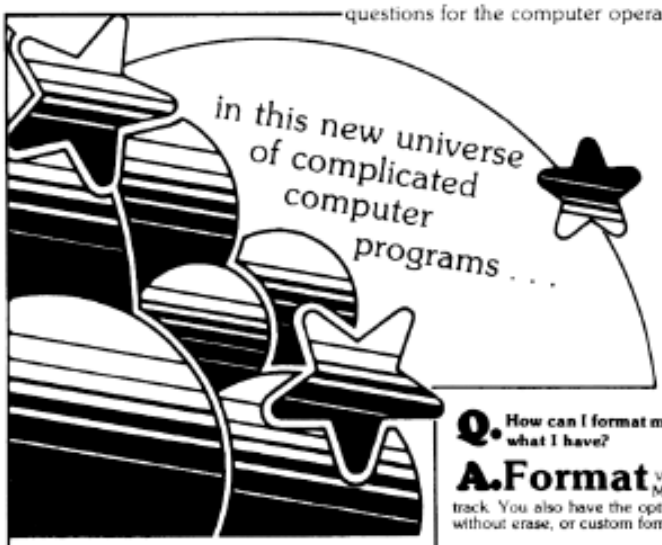
THE Programmer's Guild™ 364

P.O. Box 66 • Peterborough, NH 03458 • 603-924-8065 Voice • 603-924-7920 Data


```
4622 23       05730          INC     HL
4623 7E       05740 NA3      LD      A,(HL)
4624 B7       05750          OR      A
4625 23       05760          INC     HL
4626 28F3     05770          JR      Z,NA2
4628 F22346   05780          JP      P,NA3
462B 2B       05790          DEC     HL
462C 11EA45   05800          LD      DE,FJ
462F 0E0C     05810          LD      C,0CH
4631 1A       05820 NA5      LD      A,(DE)
4632 BE       05830          CP      (HL)
4633 2845     05840          JR      Z,NA4
4635 13       05850          INC     DE
4636 0D       05860          DEC     C
4637 20F8     05870          JR      NZ,NA5
4639 7E       05880          LD      A,(HL)
463A FE89     05890          CP      89H
463C 23       05900          INC     HL
463D 28E4     05910          JR      NZ,NA3
463F E5       05920          PUSH    HL
4640 2B       05930          DEC     HL
4641 D7       05940          RST     16
4642 FE23     05950          CP      23H
4644 280F     05960          JR      NZ,NA6
4646 D7       05970          RST     16
4647 FEF3     05980          CP      0F3H
4649 200A     05990          JR      NZ,NA6
464B D7       06000 NA7      RST     16
464C 2807     06010          JR      Z,NA6
464E FE2C     06020          CP      2CH
4650 20F9     06030          JR      NZ,NA7
4652 F1       06040          POP     AF
4653 1825     06050          JR      NA4
4655 E1       06060 NA6      POP     HL
4656 18CB     06070          JR      NA3
4658 2AA440   06080 NA1      LD      HL,(40A4H)
465B EB       06090          EX      DE,HL
465C 62       06100 NA2      LD      H,D
465D 6B       06110          LD      L,E
465E 7E       06120          LD      A,(HL)
465F 23       06130          INC     HL
4660 B6       06140          OR      (HL)
4661 2867     06150          JR      Z,NA8
4663 23       06160          INC     HL
4664 23       06170          INC     HL
4665 23       06180          INC     HL
4666 7E       06190 NA1      LD      A,(HL)
4667 23       06200          INC     HL
4668 FE8E     06210          CP      8EH
466A 2809     06220          JR      Z,NA9
466C B7       06230          OR      A
466D 20F7     06240          JR      NZ,NA1
466F EB       06250          EX      DE,HL
4670 73       06260          LD      (HL),E
4671 23       06270          INC     HL
4672 72       06280          LD      (HL),D
4673 18E7     06290          JR      NA2
4675 23       06300 NA9      INC     HL
4676 23       06310          INC     HL
4677 18ED     06320          JR      NA1
4679 2B       06330 NA7      DEC     HL
467A D7       06340 NA4      RST     16
467B E5       06350          PUSH    HL
467C CD5A1E   06360          CALL    1E5AH
467F C1       06370          POP     BC
4680 7D       06380          LD      A,L
4681 91       06390          SUB     C
4682 2824     06400          JR      Z,NA3
4684 D5       06410          PUSH    DE
4685 5F       06420          LD      E,A
4686 3E02     06430          LD      A,2
4688 F5       06440          PUSH    AF
4689 3E05     06450          LD      A,5
468B 93       06460          SUB     E
468C 3007     06470          JR      NC,NA4
468E 2F       06480          CPL
468F 3C       06490          INC     A
4690 D1       06500          POP     DE
4691 C602     06510          ADD     A,2
4693 F5       06520          PUSH    AF
4694 AF       06530          XOR     A
4695 C5       06540 NA4      PUSH    BC
4696 C4D446   06550          CALL    NZ,NA5
4699 E1       06560          POP     HL
469A 360E     06570          LD      (HL),0EH
469C 23       06580          INC     HL
469D C1       06590          POP     BC
469E D1       06600          POP     DE
469F 73       06610          LD      (HL),E
46A0 23       06620          INC     HL
46A1 72       06630          LD      (HL),D
46A2 23       06640 NA6      INC     HL
46A3 3628     06650          LD      (HL),28H
46A5 10FB     06660          DJNZ    NA6
46A7 23       06670 NA8      INC     HL
46A8 2B       06680 NA3      DEC     HL
46A9 D7       06690          RST     16
46AA 3C       06700          INC     A
46AB 3D       06710          DEC     A
46AC 38CB     06720          JR      C,NA7
46AE 23       06730          INC     HL
```

Program continues

questions for the computer operator, concerned with the versatility of their TRS-80 programs



How in heavens name, can SUPER UTILITY provide answers?

Super Utility is a powerful and sophisticated zapping program that allows you to go to the heart of the disk and read or modify data with ease, engaging simple one-key commands that threads through all of your logical decision choices. Super Utility, written by Kim Watt of Breeze Computing, Inc., is a stand alone program containing seven main menus, which are the answers to frustrated questions you have while struggling through your TRS-80 programs.

Q. Isn't there an easier way to examine and modify data?

A. Zap has an easy to read printout that reveals information in both HEX and ASCII and simultaneously moveable dual cursors. You can modify data using Hex, Decimal, ASCII or Binary input, and any changes are automatically updated on both sides of the readout. You can search through disk or file sectors, stopping anywhere to copy, compare and verify data on your disk.

Q. Isn't there an easier way to get rid of this data I don't want?

A. Purge enables you to clear a disk of unwanted data. Kill files by file-spec or have the computer list them one at a time for deletion.

Q. How can I format my disk wtthout erasing what I have?

A. Format your disk and add tracks. Make a 35 track disk a 40 track. You also have the option of formatting with or without erase, or custom formatting your disk.

Q. This disk is protected. Isn't there some way I can copy it?

A. Disk Copy enables you to back up most TRS-80 readable disks, regardless of efforts to protect it. So, back up your original and back up your modified version, too, or . . .

A. Tape Copy if you wish to back up your tape.

Q. My disk won't boot. Now what do I do?

A. Disk Repair will recover killed files, if the file was accidently killed by this utility. Repair GAT table, HIT table and Boot. Read protect directory track and check directory.

Q. How can I get more access to my memory?

A. Memory allows you to move, test, compare, zero, exchange edit, or jump to, memory. Load memory to/from disk and input or output a byte to any port.

You will love the simplicity and freedom of modifying programs to suit your needs. Now, the only question left is . . .

Q. How do I order?

A. Send check or money order for only $49.95 plus $2.50 shipping and handling to:

↙76

Breeze Computing, Inc. P.O. Box 1013, Berkley, MI 48072

Foreign orders, please add $5.00 additional postage. Michigan residents add 4% sales tax.

Breeze Computing, Inc. will send every owner, upon registration of Super Utility, one back up copy. **$2.50.**

DISCOVER THE 6809 IN YOUR COLOR COMPUTER

Now you can explore the Radio Shack Color Computer's impressive potentials—as an inexpensive development system, a color peripheral, a process controller—ad infinitum. The Micro Works introduces these powerful software tools for utilizing the color computer at the assembly language level.

MONITOR TAPE: A cassette tape which allows you to:

- Examine or change memory using a formatted hex display
- Save areas of memory to cassette in binary (a "CSAVEM")
- Download/upload data or programs to a host system
- Move the video display page throughout RAM
- Send or receive RS-232 at up to 9600 baud
- Investigate and activate features of your computer, such as hi-res graphics or machine-language music
- Use your color computer as an intelligent peripheral for another computer, a color display or a 6809 program development tool

The monitor has 19 commands in all, and is relocatable and re-entrant.

80C Monitor Tape Price: $29.95

MONITOR ROM: The same program as the monitor tape, supplied on ROM. This allows BASIC to use the entire RAM space. And you don't need to re-load the monitor each time you use it. The ROM plugs into the Extended Basic ROM Socket or a modified ROMPACK.

80C Monitor ROM Price: $39.95

INSIDE THE COLOR COMPUTER: This package is a disassembler which runs on the color computer and enables you to generate your own source listing of the BASIC interpreter ROM. Also included is a documentation package which gives useful ROM entry points, complete memory map, I/O hardware details and more. Disassembler features include cross-referencing of variables and labels; output code which can be re-assembled; output to an 80-column printer, small printer or screen; and a data table area specification which defaults to the table boundaries in the interpreter ROM. A 16K system is required for the use of this cassette.

80C Disassembler Price: $49.95

C BUG IS HERE!!

THE MICRO WORKS ↙109

Mastercharge and BankAmericard

P.O. BOX 1110 DEL MAR, CA 92014 714-942-2400

↙See List of Advertisers on page 306



2 games per cassette

for the TRS-80 Model I or III, 16 K Level II or Model III BASIC microcomputers. all our programs have

ACTION SOUNDS & GRAPHICS



SPACE ACE 21

Scifi Game of Tactical Space Combat

You design your own space fighter and then blast off into battle. Human or 10 Computer Opponents. Fight in 2 or 3 dimensions. Three scenarios: "Smuggler", "Refuel Option", and "Phoenix Decathlon".

THE NEW STARSHIP VOYAGES

A brilliant "trek" type space war. 3D galaxy with wrap around. Moving enemy craft, some are "Cloaked". 16 commands. Rescue starbase Delta from the Rogues.

catalog No. 2001 $ 19.95

PARSECTOR V



Still The Ultimate Space War

Unique split screen gives each player a private display. Launch fleet battle craft and watch them fight. Fire high powered energy beams or short range weapon spreads. Human or computer opponent.

PARSECTOR 8 Tournament Version. Giant galaxy to conquer. Intense Strategy!

catalog No. 2002 $ 19.95

!!! ANTS !!!



Amazing War Between Ant Colonies

Fast, machine language speed. Three game variations: "Open Field", "Nest Barrier", and "Digging Ants". 2 players or computer opponent. Easy to play, Challenging to master. Fun packed game for all ages!!!

THE NEW STARSHIP VOYAGES

catalog No. 2003 $ 14.95

HIGH SPEED LIFE

The FASTEST, most ADVANCED version of J.H.Conway's famous mathematical game. Spectacular kaleidoscopic animation. 500 gen/min typical. 32 preprogrammed patterns. Multiple control functions.

(Rated No.1 in 80 Software Critique, Issue 3)

NAME THAT STATE QUIZ

A top notch educational game. Draws 50 states and asks five questions. Three quiz types. Easy to use.

catalog No. 2004 $ 14.95

Disk Owners

On special request the above games are available on cassette (you transfer), same price, compatible with TRSDOS Disk Basic 48K.

Add $1.00 postage & handling. COD add $2.50 Fla. Res. add 4% tax. Make Check or M.O. payable to:

Synergistic Solar, Inc.

PO Box 560595, Miami FL 33156

231

Please write for more info. Dealer inquiry invited


```
46AF CA1B46   06740          JP      Z,NX2
46B2 2B       06750          DEC     HL
46B3 FE28     06760          CP      28H
46B5 28F8     06770          JR      Z,NA8
46B7 FE2C     06780          CP      2CH
46B9 28EC     06790          JR      Z,NA8
46BB FECE     06800          CP      0CEH
46BD 28E8     06810          JR      Z,NA8
46BF FED4     06820          CP      0D4H
46C1 3804     06830          JR      C,NA9
46C3 FED7     06840          CP      0D7H
46C5 38E8     06850          JR      C,NA8
46C7 C32346   06860 NA9      JP      N23
46CA 2AF940   06870 NX8      LD      HL,(40F9H)
46CD 22FB40   06880          LD      (40FBH),HL
46D0 22FD40   06890          LD      (40FDH),HL
46D3 C9       06900          RET
46D4 E5       06910 NA5      PUSH    HL
46D5 2AF940   06920          LD      HL,(40F9H)
46D8 EB       06930          EX      DE,HL
46D9 2600     06940          LD      H,0
46DB 6F       06950          LD      L,A
46DC 19       06960          ADD     HL,DE
46DD 22F940   06970          LD      (40F9H),HL
46E0 44       06980          LD      B,H
46E1 4D       06990          LD      C,L
46E2 E1       07000          POP     HL
46E3 1A       07010 NB1      LD      A,(DE)
46E4 02       07020          LD      (BC),A
46E5 DF       07030          RST     24
46E6 C8       07040          RET     Z
46E7 1B       07050          DEC     DE
46E8 0B       07060          DEC     BC
46E9 18F8     07070          JR      NB1
46EB CD9A8A   07080 NX3      CALL    0A9AH
46EE AF       07090          XOR     A
46EF CD3410   07100          CALL    1034H
46F2 B6       07110          OR      (HL)
46F3 C3D90F   07120          JP      0FD9H
46F6 21FD46   07130 READY    LD      HL,GI1
46F9 CDA728   07140          CALL    28A7H   ;DISPLAY A STRING POINTED @ BY HL
46FC C9       07150          RET
46FD 0B       07160 GI1      DEFB    0BH
46FE 8785     07170          DEFW    8587H   ;GRAPHICS CHARACTER
4700 4C       07180          DEFM    'LEVEL II-1/2 (V 1.2) IS -'
4719 00       07190          DEFB    0       ;NUL= END OF STRING FOR CALL 28A7
471A ED5BF940 07200 MERGE    LD      DE,(40F9H)      ; ( )= END BASIC
471E 1B       07210          DEC     DE
471F D5       07220          PUSH    DE
4720 C1       07230          POP     BC
4721 1B       07240          DEC     DE
4722 1B       07250          DEC     DE
4723 1B       07260          DEC     DE
4724 1A       07270          LD      A,(DE)
4725 B7       07280          OR      A
4726 2005     07290          JR      NZ,ME2
4728 ED43F940 07300          LD      (40F9H),BC
472C 1B       07310          DEC     DE
472D 13       07320 ME2      INC     DE
472E 13       07330          INC     DE
472F ED53A440 07340          LD      (40A4H),DE
4733 214B47   07350          LD      HL,ME1          ; OK MESSAGE
4736 CDA728   07360          CALL    28A7H           ; DISPLAY A STRING
4739 C3191A   07370          JP      1A19H           ; JP TO BASIC
473C 215247   07380 RSET     LD      HL,TWOHAF
473F 22A440   07390          LD      (40A4H),HL      ; RESET 'BASIC BEGIN'
4742 214B47   07400          LD      HL,ME1
4745 CDA728   07410          CALL    28A7H
4748 C3191A   07420          JP      1A19H
474B 3C       07430 ME1      DEFM    '< OK >'
4751 00       07440          DEFB    0
              07450 ;
              07460 ; NOTE- PROGRAM FROM HERE DOWN IS LOST WHEN JP TO BASIC
              07470 ;       IS DONE.  IT IS ONLY NEEDED FOR INITIALIZATION.
              07480 ;
4752 212243   07490 TWOHAF   LD      HL,ASC
4755 225641   07500          LD      (4156H),HL
4758 21E963   07510          LD      HL,DEF
475B 225C41   07520          LD      (415CH),HL      ;LINK TO DISK BASIC 'DEF'
475E 215344   07530          LD      HL,HEX
4761 229541   07540          LD      (4195H),HL
4764 3EC3     07550          LD      A,0C3H          ;'JP' FOR USR & READY
4766 32A941   07560          LD      (41A9H),A
4769 210D44   07570          LD      HL,USR          ;LINK TO DISK BASIC 'USR'
476C 22AA41   07580          LD      (41AAH),HL
476F 218444   07590          LD      HL,LINE         ;LINK TO DISK B. 'LINE'
4772 22A441   07600          LD      (41A4H),HL
4775 32AC41   07610          LD      (41ACH),A
4778 21F646   07620          LD      HL,READY
477B 22AD41   07630          LD      (41ADH),HL      ;LINK WITH 'READY'
477E 211A47   07640          LD      HL,MERGE
4781 228C41   07650          LD      (418CH),HL      ;LINK WITH D.B. 'MERGE'
4784 213C47   07660          LD      HL,RSET
4787 229B41   07670          LD      (419BH),HL      ;LINK WITH D.B. 'RSET'
478A 215247   07680          LD      HL,TWOHAF
478D 22A440   07690          LD      (40A4H),HL      ;LOAD NEW 'BASIC BEGIN'
4790 CD491B   07700          CALL    1B49H           ;CALL LVL II 'NEW'
4793 C3191A   07710          JP      1A19H           ;JP TO BASIC
4752          07720          END TWOHAF
00000 TOTAL ERRORS
```

Color computer owners,

32K PLUS DISKS*

$298.00

Yes, that's right - for as little as $298.00 you can add 32K of dynamic RAM, and a disk interface, to your TRS-80 Color Computer! If you just want the extra memory it's only $199.00, and you can add the disk interface later for $99.00.

Just plug the *Color Computer Interface (CCI)*, from Exatron, into your expansion socket and "Hey Presto!" - an extra 32K of memory. No modifications are needed to your computer, so you don't void your Radio Shack warranty, and Exatron give both a 30 day money-back guarantee and full 1 year repair warranty on their interface.

The *CCI* also contains a 2K machine-language monitor, with which you can examine (and change) memory, set break-points, set memory to a constant and block-move memory.

So what about the *CCI Disk Card*? Well as we said it's only an extra $99.00, but you'll probably want Exatron's *CCDOS* which is only $29.95 – unless you want to write your own operating system. The *CCI Disk Card* uses normal TRS-80 Model I type disk drives, and *CCDOS* will even load Model I TRSDOS disks into your color computer – so you can adapt existing TRS-80 BASIC programs.

As a further plus, with the optional *ROM Backup* adaptor, you can dump game cartridges to cassette or disk. Once the ROM cartridge is on cassette, or disk, you can reload, examine and modify the software. The *ROM Backup* adaptor is only $19.95.

For more information, or to place an order, phone Exatron on their Hot Line *800-538 8559* (inside California 408-737 7111), or clip the coupon.

excellence in electronics

exatron

DEALER ENQUIRIES INVITED

Exatron, 500
181 Commercial Street,
Sunnyvale, CA 94086

☐ Please send a 32K Color Computer Interface for $199.00

☐ Please send a CCI Disk Card for $99.00

☐ Please include CCDOS and manual for $29.95

☐ Also include a ROM Backup adaptor for $19.95

Please add $5.00 for shipping to all orders, and 6 percent sales tax in California.

Name .....

Address .....

City .....

State ..... Zip .....

Charge my:

☐ MasterCard Interbank Code .....

☐ Visa Expiration Date .....

Card .....

☐ Check enclosed for .....

☐ Ship COD ($2.00 extra)

Signature .....

*32K RAM plus Disk Interface TRS-80 is a trademark of Tandy

Extending Aculab BASIC

George Phillips

Just recently Dusty asked for my help in adding some BASIC extensions to the Aculab floppy tape system. The Aculab floppy tape (or AFT) is a third party add-on to the Model 1 based on the Exatron stringy-floppy drive. It has built-in firmware that manages tape wafers like a floppy disk allowing the user to save and load files without having to worry about the current position of the tape or any other low-level details. You can try it out for yourself by running:

```
trs80gp -m1 -aft -dx -nld
```

You'll see a new "Wafer" menu which includes a brief manual on how to operate the AFT.

The AFT was a good option for anyone who couldn't afford an expansion interface and floppy drive system. However, except for the file management commands it did not add any of the extra BASIC features available in disk BASIC. Dusty found an article in the July 1981 issue of 80 Microcomputing on page 202 that added a few of the disk BASIC functions to Level II ROM BASIC.

Dusty OCR'ed each page of the program listings from the article and sent them on to me. I put them all into a single source file and fixed the usual OCR errors where it confuses number one/lower-case L and number zero/upper-case O. There was a tricky `1NC HL` which took me longer than I'd like to admit to notice it should be `INC HL`.

The result worked fine with Level II BASIC but crashed with the AFT firmware activated – even after relocating it at the new start of free memory required by AFT. After a bit of study I found the source of the conflict. The extender was using the DOS READY hook to change the BASIC `READY` prompt to indicate the extender was active. In ROM BASIC the three byte READY hook is a `RET` instruction followed by two unused bytes. The AFT firmware uses those two unused bytes to point to its working storage. I took out the code that hooked into READY and all was well.

There was one more thing to do. The assembled extender (called `TWOHAF` as it is like Level 2.5 BASIC) had to be loaded from cassette. Not a huge difficulty, especially under emulation, but rather misses the whole point of having a BASIC extender for the AFT when you can't load it from the AFT itself.

I added a little bit of extra code to `twohaf.z` to print the AFT command needed to save the BASIC extender to the AFT drive. The extra code is self-patched out when twohaf runs so it doesn't needlessly make the executable bigger when saved to the AFT wafer.

With the extensions saved to a wafer there is only one extra command beyond the normal AFT startup to use them:

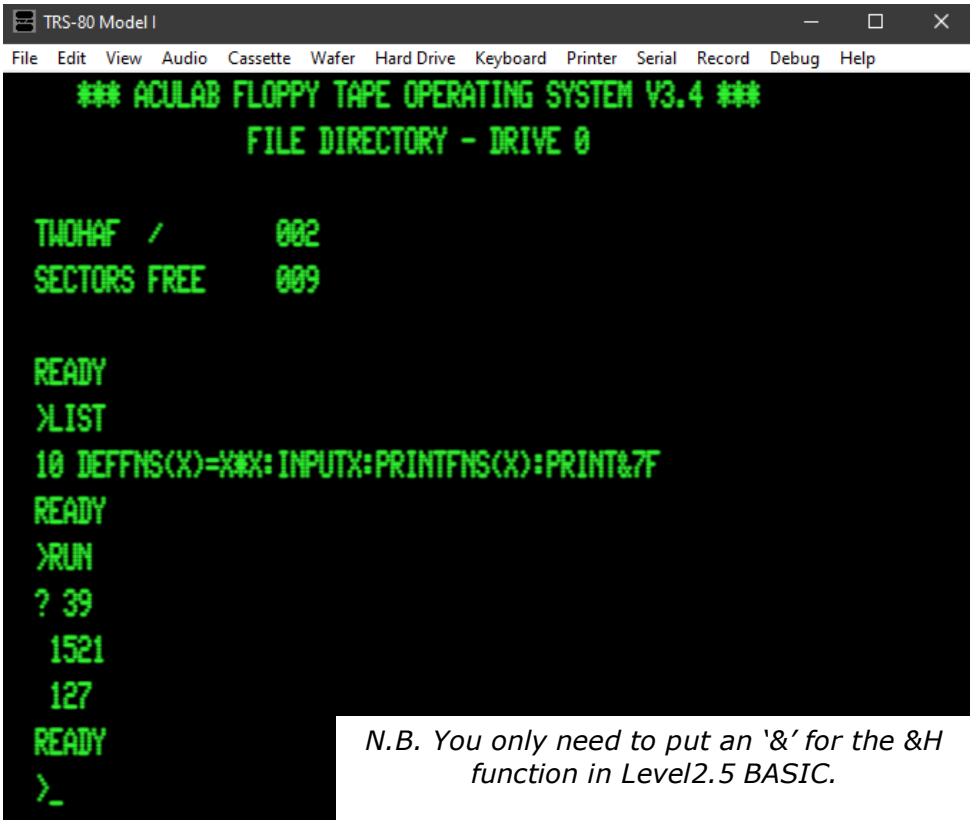
```
trs80gp -m1 -dx -nld -aft -w twohaf.tape
SYSTEM
?* /12345
@RUN"TWOHAF"
```

If twohaf is changed the procedure to create the new wafer image is:

```
zmac -Daculab twohaf.z
trs80gp -m1 -dx -nld -aft -c zout/twohaf.500.cas
SYSTEM
?* /12345
insert a blank wafer
SYSTEM
?* TWOHAF
?* /
```

Follow the directions printed to save `TWOHAF` to the blank wafer.

The source code and pre-built cassette and tape images can be found at **Model 1, Level 2.5 BASIC**



N.B. You only need to put an '&' for the &H function in Level2.5 BASIC.

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

NEIL JAMES ALEXANDER SLOANE



DUSTY :- I CAME ACROSS THIS GUY AND THE 'OEIS' ON 'NUMBERPHILE'. ANOTHER REALLY FASCINATING DISCOVERY! NEEDLESS TO SAY, I JUST HAD TO PRODUCE A ONE-LINER TO REPRODUCE THE VANECK SEQUENCE! IT'S JUST SUCH A SIMPLE IDEA BUT JUST WHERE DOES IT END EH?

(IT'S BEST RUN IN TURBO-MODE ON THE EMULATOR AND, IN THE FIRST 1000 NUMBERS LOOK OUT FOR 56!) E.T.

NEIL JAMES ALEXANDER SLOANE WROTE THAT HE STARTED COLLECTING SEQUENCES AS A GRADUATE STUDENT AT CORNELL UNIVERSITY IN 1964.

"THE HANDBOOK OF INTEGER SEQUENCES" (ACADEMIC PRESS) APPEARED IN 1973; IT CONTAINED 2372 SEQUENCES.

"THE ENCYCLOPAEDIA OF INTEGER SEQUENCES" (WITH SIMON PLOUFFE; ACADEMIC PRESS) APPEARED IN 1995; IT CONTAINED 5487 SEQUENCES.

SEQUENCES CONTINUED TO POUR IN. WHEN THE DATABASE REACHED 10,000 SEQUENCES IN 1996, HE PUT IT ON THE WEB AS THE "ON-LINE ENCYCLOPAEDIA OF INTEGER SEQUENCES" (THE OEIS).

OVER THE NEXT 15 YEARS, NEW SEQUENCES ARRIVED (AND STILL ARRIVE TODAY) AT A RATE OF 10,000 TO 18,000 NEW SEQUENCES EACH YEAR.

BY 2009 THE OEIS HAD GROWN TO A POINT WHERE IT WAS DIFFICULT FOR ONE PERSON TO MANAGE IT. THAT YEAR I SET UP A NON-PROFIT FOUNDATION TO OWN, MAINTAIN AND COLLECT FUNDS TO SUPPORT IT. THIS IS THE OEIS FOUNDATION INC..

THE NEXT STEP WAS TO SET UP A VERSION OF THE OEIS WHICH WOULD RUN AUTONOMOUSLY ON A COMMERCIAL WEB SITE, OPERATING AS A MODERATED WIKI, AND CONTROLLED BY A BOARD OF EDITORS. UNFORTUNATELY, 18 MONTHS AGO WE DISCOVERED THAT THE STANDARD MEDIAWIKI SOFTWARE WAS INCAPABLE OF HANDLING THE KIND OF QUERIES THAT THE OEIS GENERATES. THIS PROBLEM SEEMED INSURMOUNTABLE, AND I HAD ALMOST GIVEN UP HOPE THAT IT WOULD EVER BE SOLVED.

BUT THANKS TO THE ENORMOUS EFFORTS OF RUSS COX, WHO HAS COMPLETELY REWRITTEN ALL THE OEIS SOFTWARE, THE NEW OEIS WAS LAUNCHED ON NOV 11 2010. DAVID APPLEGATE HAS ALSO BEEN OF GREAT HELP IN GETTING THE NEW SYSTEM WORKING. AT THE TIME OF THE LAUNCH, THE OEIS CONTAINED 180284 SEQUENCES. THIS MEANS THAT AFTER 46 YEARS, I NO LONGER HAVE TO PROCESS EVERY SINGLE CONTRIBUTION TO THE DATABASE. THANK YOU, RUSS AND DAVE: YOU HAVE SAVED THE OEIS!



Van Eck's sequence:

For n >= 1, if there exists an m < n such that a(m) = a(n), take the largest such m and set a(n+1) = n-m; otherwise a(n+1) = 0. Start with a(1)=0.

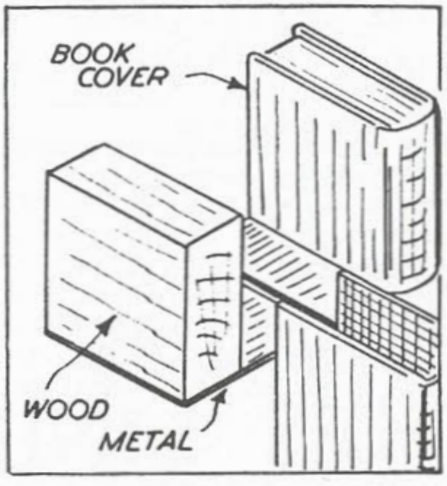
```
0 DEFINT A-Z:DIME(1000):FORI=1TO999:FORJ=I-1TO0STEP-1:IF E(J)=E(
I) THEN E(I+1)=I-J:PRINTE(I);:NEXTIELSENEXTJ:E(I+1)=0:PRINTE(I);
:NEXTI
```

```
READY
>RUN
 0  1  0  2  0  2  2  1  6  0  5  0  2  6  5  4  0  5  3  0  3
 2  9  0  4  9  3  6  14  0  6  3  5  15  0  5  3  5  2  17  0
 6  11  0  3  8  0  3  3  1  42  0  5  15  20  0  4  32  0  3
 11  18  0  4  7  0  3  7  3  2  31  0  6  31  3  6  3  2  8
 33  0  9  56  0  3  8  7  19  0  5  37  0  3  8  8  1  46  0
 6  23  0  3  9  21  0  4  42  56  25  0  5  21  8  18  52  0
 6  18  4  13  0  5  11  62  0  4  7  40  0  4  4  1  36  0  5
 13  16  0  4  8  27  0  4  4  1  13  10  0  6  32  92  0  4  9
 51  0  4  4  1  14  131  0  6  14  4  7  39  0  6  6  1  12  0
 5  39  8  36  44  0  6  10  34  0  4  19  97  0  4  4  1  19
 6  12  21  82  0  9  43  0  3  98  0  3  3  1  15  152  0  6
 17  170  0  4  24  0  3  12  24  4  6  11  98  21  29  0  10
 45  0  3  13  84  0  4  14  70  0  4  4  1  34  58  0
```

HANDY HINTS



HERE'S SOMETHING JUST A BIT DIFFERENT. IT'S A 'HANDY HINT' FROM THE 1955 'GADGETS ANNUAL'. ANOTHER GREAT IDEA!



THIS IS THE WAY TO USE

BOOKS AS BOOK-ENDS

A WHOLE row of books stand up by themselves and never slide over, simply because the two end books are dummies—book-ends, in fact.

You need two old books with good covers, preferably identical in binding and shape. The covers only are needed and are carefully removed from the rest of the book. The body of each of these books is then replaced by hardwood blocks of appropriate size. These blocks are screwed to thin metal plates (with wood-screws, from underneath). Trim the inner edge of each cover to fit snugly over the metal plate and glue the covers to the blocks. Your book-ends are then complete.

EBAY.CO.UK 'BARGAINS'

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



£
£286



£25



£112



£75



£46



£122



£187



£135



£180



£225

EBAY.CO.UK 'BARGAINS'

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



£



£40



£150



£77



£155



£54



£40



£59



£106



£150

IN MAV'S WORKSHOP



Ex-Pat Color Computer 3 Moves to Australia



by Ian Mavric



"Have you got your Coco 3 yet?" Not much has been written in TRS8Bit about the Coco 3, so I thought I would detail how I converted a US 120V machine for use in Australia.

THIRD TIME LUCKY - AND WHAT AN UPGRADE!

The Color Computer 3 was released in late 1986 (first appearing in RSCC-17 for 1987) and was the first and only proper upgrade to the Coco line released in 1980. Between the original Coco and Coco 3 was the Coco 2, which was very popular but basically was a cost-reduced version of the original Coco with a better keyboard.

The Coco 3 was a ground-up re-design of the Coco platform and improved in virtually all areas... it was faster, had better graphics, more memory and a better keyboard.

It was the one to have and when people are asked if they could only have one Color Computer they invariably go with the CC3. In Australia it was sold only for approximately 2 years, and was available in the USA for a bit longer though by 1991 it was pretty much over (RSCC-22 being the last catalog to feature the CC3).

Software compatibility with the Coco 1/2 was quite good and new software to take advantage of the improvements of the computer soon appeared and was plentiful. The computer was upgradeable to 512K Ram, an amount considered unbelievable during the Coco 1/2 era. The CC3 sold well considering it's competition in the marketplace not just from the likes of C= with the 64 and Amiga, Atari with the ST, but even within Radio Shack with it's PC compatibles like the Tandy 1000 HX.

TRY FINDING ONE NOW IN 2023

Being at the tail end of a platform's life fewer units were sold regardless of how good the machine was. There are stories of Coco enthusiasts in the US wanting to buy one going into their local Radio Shack only to find nil stock, and having to convince the salesperson to order one into the store, after they had been given a monologue about the features Tandy's other machines... This is understandable as the profit margin for the RS store would be higher for a PC compatible than a CC3.

The upshot of this is that in present times they are simply not as plentiful to buy on the used market and often fetch premium prices, especially if they are hardly used or still in their original boxes. The short time frame they were on sale in Australia makes them particularly hard to find here. A fellow Coco enthusiast in Australia acquired a CC3 from the USA and used it for a while, but he also wanted to expand his Coco collection with a 64K long-white case Coco 1 (which were only available in the USA) and I just happened to bring one home with me from TA22. A trade was organised and soon I had this American 120V CC3 in my workshop.

US Cocos are interesting because they connect to NTSC television sets, and can do colour artefacting that PAL televisions can't. It's why games that use the highest resolution on the Coco 1 and 2 appear in colour on NTSC and B&W on PAL... Australians regularly mumbling under their breath "You call this a *colour* computer?". However, if you have one of the late model CRT TVs from around the mid-to-late 90s, many of them can recognise an NTSC signal even though their manufacturers (namely Teac, NEC and Sony) didn't advertise it as such. If you happen to have one of those you can use a US Coco in Australia and get full artefact colour on those games.

THAT 120V IS A LITTLE ANNOYING

The usual method to use a US Coco here is via a small step-down transformer from 240V to 120V. The CC3 doesn't use much power so even the cheapest eBay step-down at $30 would do the trick. A better solution would be to replace the transformer inside the computer with a proper 240V one and change the power cord to a normal Australian Type -I (that's Type-'Eye' not Type-'1') plug.

Normally to replace a transformer one would just look up what the required AC output is, and buy a transformer with enough wattage from an electronics store like Jaycar.

The problem with this method is that some carpentry is usually required to fir the new transformer. I've done it before with other computers and external disk drives and the result is messy, I'm never really 100% happy with the result. What I would really prefer is a 240V transformer from an Australian PAL CC3.

The closest thing I had was a manky-looking yellowed 64K CC2. I figured I could use the 64K board to upgrade a pristine 16K CC2 I had here, use the keyboard in a CC1, and put the power transformer into the US CC3.
Stopping this being a drop-in solution was that the CC2 uses a different power connector at the mainboard to a CC3:



Photo: 240V Coco 2 has an E-shaped power connection

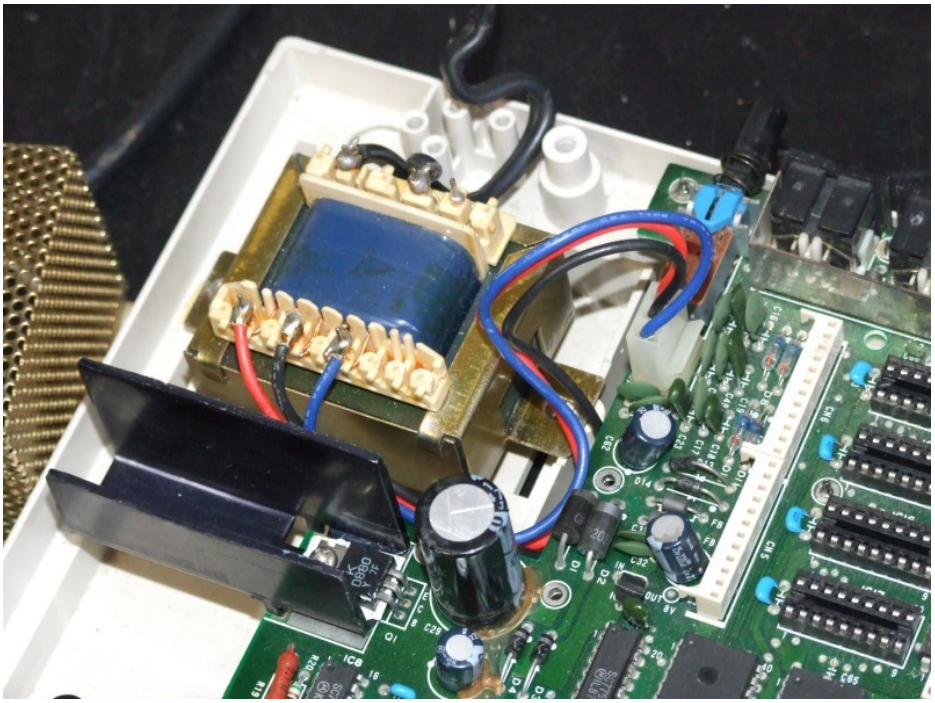
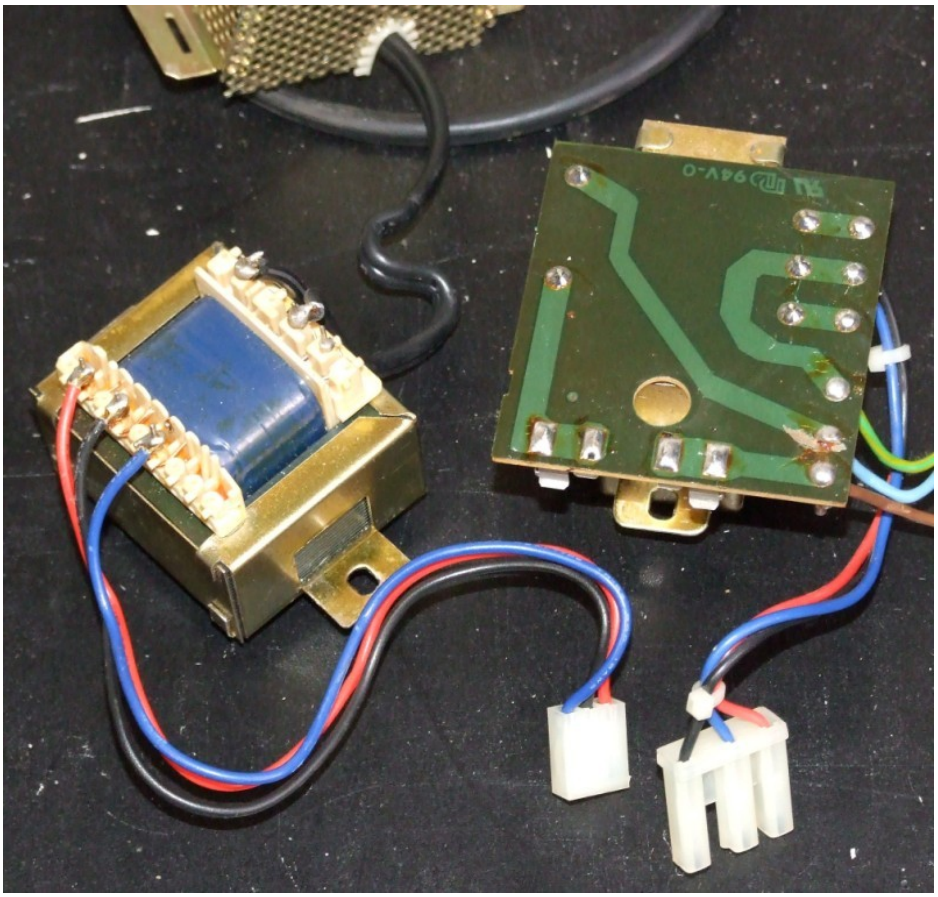


Photo: CC3 120V PSU and different, but still 3 pin, connection

Using the CC2 power supply has the benefit that is mounts exactly in the right place using the existing screws and hardware, making for a very slick fit. The solution is to transfer the 3 wires and connector from the



CC3 PSU to the CC2 PSU:

Photo: CC3 transformer (on the left), CC2 240V transformer (right)

One benefit of the CC2 power supply set up is that the 240V transformer has a proper earthing connection and a fuse protecting the system, the 120V CC3 transformer has neither.

SIX SOLDERED WIRES AND IT'S ALL DONE

Here it is completed. I transferred the 3-wire (low voltage) connector over. Black wire is the earth wire, and the red and blue are either sides of the A/C. While at it I added a new Type-I power cable and the conversion is complete. All up six soldered wires.

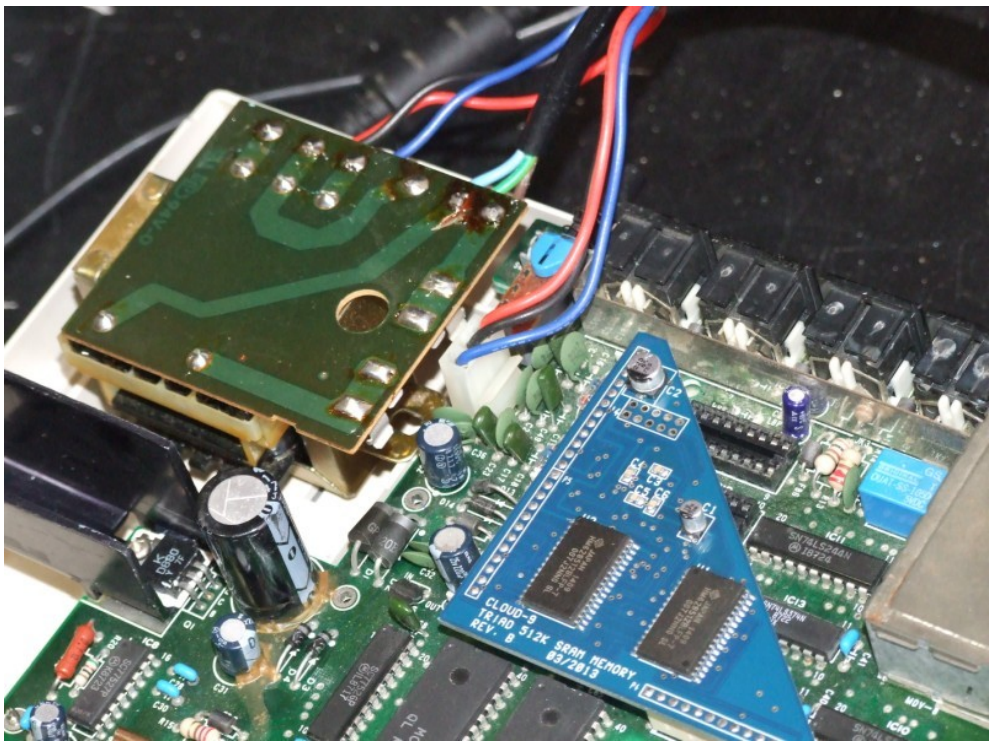


Photo: CC2 240V transformer converted to CC3

After I took this picture I properly screwed down the transformer and replaced the protective cardboard on top. I did notice the fuse leans against the power connector, so I bent that over a few degrees to clear the fuse.

NEXT TIME: I RE-CAP THE COCO 3 AND UPGRADE IT'S CPU

If you wish to see more Coco projects please do reach out to me with your ideas and I'll see what I can do. Remember TRS8Bit is not just about the monochrome Z80 TRS-80s, but aims to support all Tandy/TRS-80 platforms including, but not limited to Cocos, Tandy MS-DOS machines, and pockets/portables. See you all at TA23.

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.

Software for TRS80®

ACCEL Compiler for TRS80 BASIC

Speeds execution of correct Level 2, BASIC programs by compiling the common statements to Z80 machine-code.

- Develop and debug normally.
- Compile for production work.

ACCEL (16K) **Great value at.....£19.95**

DLOAD	Load program segments	£4.95
ZBUG	Debug Z80 programs	£11.95
TSAVE	Prepare system tapes	£4.95
RENUM	Renumber BASIC programs	£5.95
FGRAF	Fast Graphics	£4.95
XREF	Program cross-reference	£5.95
SDUMP	Symbolic dump	£2.95
LIFE	(16K) 48 x 64	£3.00
USRN	Multiple USR calls	£2.95

Southern Software programs are self-relocating, load-anywhere, machine-code, well documented. They run in 4K Level 2, unless otherwise stated.

SOUTHERN SOFTWARE PO Box 39, Eastleigh, Hants. SO5 5WQ

TRS8BIT 2023 Competition

0K Z-80 Disassembler

A disassembler is a great addition to a debugger. It can show actual instructions instead of hexadecimal opcodes which, if you're like me, are quite tedious to disassemble by hand. Since a debugger must coexist with the program being debugged it should be as small as possible lest you get caught without enough space for the program and the debugger. Waste not, want not, as the old saying goes.

A Z-80 disassembler literally fitting in zero kilobytes is a tad unlikely. Nonetheless, there's every hope it can be squeezed into something less than 1024 bytes and will round down to zero kilobytes.

The smallest entry based on code size and run-time data space used will be the winner. You can use Z-80, 6809, 6803 or 68000 assembly to program your disassembler subroutine. All entries will be in source code form and will be published at the end of the competition for all to learn from and admire.

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

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

For details on exact requirements, other rules and a running leader board please consult:

http://48k.ca/comp2023.html

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

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



The *BEST* in TRS-80s

Call The Right Stuff

Ask for Ian
The number is +61 416 184 893

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

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







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

WHERE 8 BITS AND 16K STILL RULE OK

TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

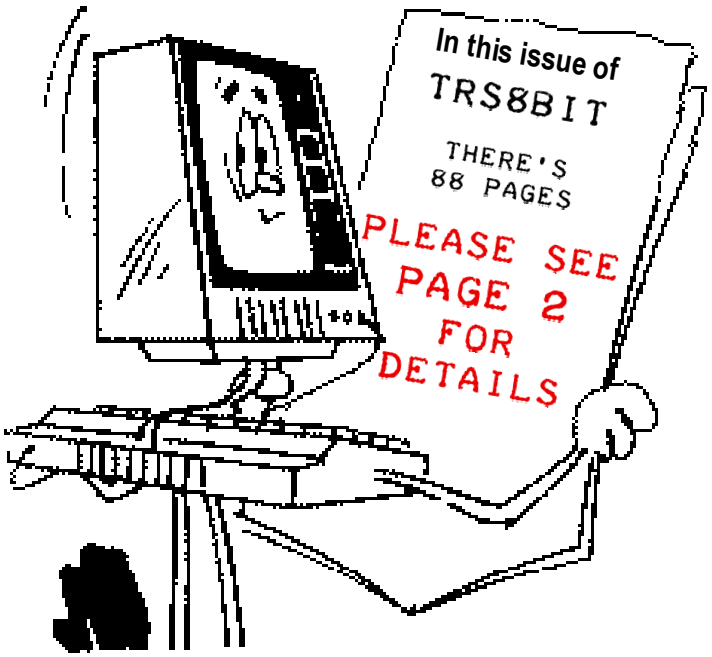


HI EVERYONE AND WELCOME TO ONE OF THE LARGEST SEPTEMBER EDITIONS TO DATE. 88 PAGES, WOW. THANKS EVERYONE WHO HAS SENT IN CONTRIBUTIONS.

THE FIRST ITEM I WISH IS TO REMIND YOU OF IS :-
THIS YEARS COMPETITION.

GEORGE PHILLIPS IS LOOKING FOR THE SMALLEST POSSIBLE Z80 DISASSEMBLER, WITH THE MOST FEATURES.
HE HAS UPDATED HIS COMPETITION PAGE WITH INFORMATION ON THE STRUCTURE OF Z-80 OPCODES AND HAS AN EXAMPLE DISASSEMBLER FOR YOU TO TAKE INSPIRATION FROM.

ONCE AGAIN, MAV IS OFFERING ONE OF HIS TOP PRODUCTS, AS THE PRIZE TO THE WINNER.
THE CLOSING DATE IS THE 23RD NOVEMBER 2023, SO IF IT'S SOMETHING YOU'VE BEEN PUTTING OFF UNTIL LATER, WAITING FOR THE DARKER NIGHTS OF AUTUMN TO GET THOSE LITTLE GREY CELLS WORKING, PLEASE REMEMBER, TIME IS MARCHING ON!



SECOND MENTION IS OF COURSE,

Tandy Assembly 2023

HAVE YOU BOOKED YET?

IT'S HAPPENING IN SPRINGFIELD, OHIO

29SEP23 - 01OCT23

CHECK-OUT MAV'S SPECIAL ARTICLE ON PAGE 4, FOR FULL DETAILS. ALTHOUGH I'M NOT ABLE TO ATTEND, I'LL BE WITH YOU ALL IN SPIRIT, NOT TO MENTION LOOKING FORWARD TO THE LIVE-LINK AND VIDEOS TO WATCH ON YOUTUBE ONCE AGAIN.

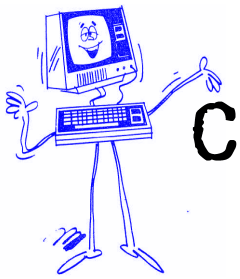
THERE HAS BEEN A LARGE NUMBER OF TANDY RELATED ITEMS ON THE UK'S EBAY. SOME BUY-IT-NOW PRICES ARE RATHER FANCIFUL, HOWEVER IT'S JUST NICE TO THINK THERE'S STILL A DEMAND FOR 42 YEAR OLD TANDY COMPUTERS.

AS YOU WILL SEE ON THE NEXT PAGE, THERE'S A FANTASTIC SELECTION OF ARTICLES ON BOTH HARDWARE AND SOFTWARE JUST TO INTEREST AND AMUSE YOU.

IT'S STRANGE TO THINK THAT THE NEXT EDITION OF TRS8BIT WILL BE THE XMAS ONE! SO UNTIL THEN,



TAKE CARE EVERYONE
DUSTY.



CONTENTS

PAGE 04 — IN MAV'S TRAVELS
T.A. 2023 AND WHAT TO LOOK FOR

PAGE 10 — YOUTUBE 'HACKS'

PAGE 14 — JULY 2023 BASQUE TRS-80 TOURNAMENT REPORT
EGOITS CAMPO

PAGE 17 — SOLGOR. A MIND-BENDING, SORTING GAME
TU-BARDH

PAGE 21 — IN MAV'S WORKSHOP -
M3 & 4 POWER CONNECTORS
IAN MAVRIC

PAGE 30 — 'ZAP THE STARS' - ONE-LINER BASIC GAME
E.T. FONEHUME

PAGE 31 — SOFTWARE TOOLS IN BASIC - PART 2
VINCE OTTEN

PAGE 47 — AT THE READY> PROMPT.
JUST A LITTLE 'QUIRK' I CAME ACROSS

PAGE 50 — HOW IT HAPPENED.
ANDY

PAGE 53 — THE WAY WE WERE
COMPILERS, TRS-80 SOFTWARE, MODEL 2000 REVIEW

PAGE 64 — HANDY HINTS
JUST FOR THE MODEL RAILWAY BUFFS

PAGE 65 — AN 'AMAZING BUG'
PETER PHILLIPS

PAGE 71 — M1 FAILURE AND MISS-PRINTS TOO!
WELLSWANG

PAGE 74 — EBAY BARGAINS

PAGE 76 — RESTORING THE CHEAPEST AT THE 'SWAP-MEET'
TOM NARDI

PAGE 79 — IN MAV'S WORKSHOP -
CO-CO 3, RESTORING AND UPGRADING
IAN MAVRIC

PAGE 85 — NEWS FROM FRANCE
PASCAL HOLDRY REPORTS ON HIS LATEST, EXCITING,
HARDWARE PROJECT

PAGE 86 — STRANGER THEN FICTION!

PAGE 87 — **THE 2023 COMPETITION**
SUPER PRIZE ON OFFER AGAIN!

LATEST UPGRADE AVAILABLE
VERSION 2.5.1

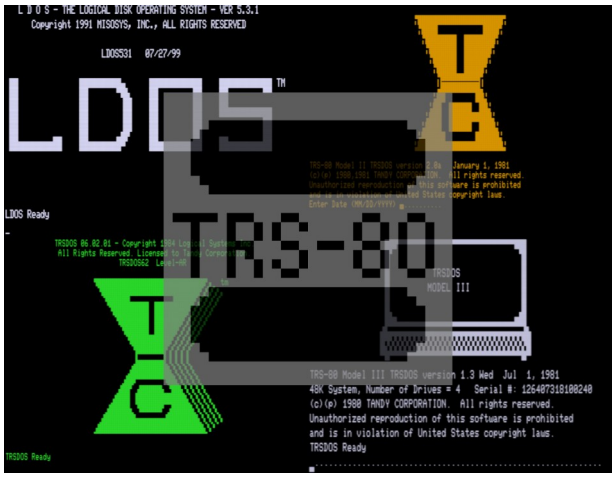
DON'T HAVE A TRS-80?

CRAZY PRICES ON EBAY?

TIRED OF RIFA CAPS BLOWING?

OF FLOPPIES SHEDDING OXIDE?

OR JUST PLAIN DEAD CHIPS?



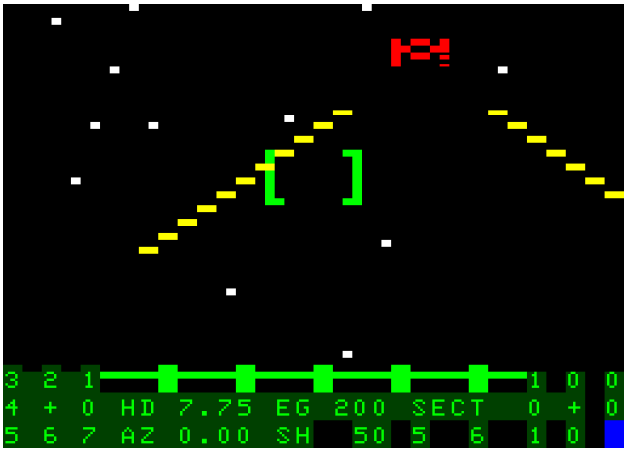
THEN WHAT YOU NEED IS TRS80GP.
THE ULTIMATE TRS-80 EMULATOR

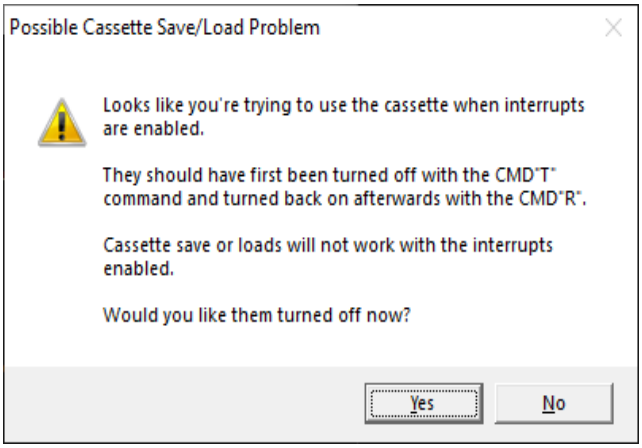
RUNS ON WINDOWS, MACOS, LINUX & RASPBERRY PI.
EMULATES MODELS 1, 2, 3, 4, 12, 16, 6000,
MC-10, COLOR COMPUTER, DT-1 AND VIDEOTEX.

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

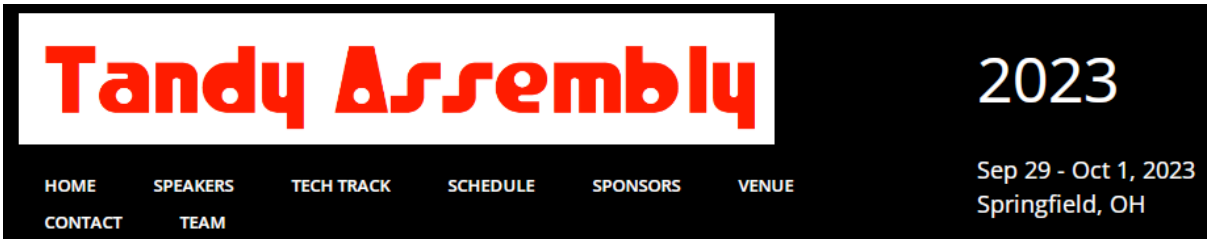
GET IT NOW AT HTTP://48K.CA/TRS80GP.HTML

LATEST VERSION 2.5.1, AVAILABLE NOW, FEATURING :-
LINUX JOYSTICK SUPPORT
COLOUR COMPUTER SG-8 SEMIGRAPHICS MODE
WARNING ON MODEL 1 IF CMD"T" NEEDED
68000 SPEED SELECTABLE, BETTER TIMED AND DECOUPLED FROM Z80





IN MAV'S TRAVELS
Who's Going To TA23?
by Ian Mavric



By the time this goes to publication Tandy Assembly for 2023 (TA23) will be less than 30 days away. My last article was written 3 months ago so this is an update on who will be attending and what we know will be at the show.

THE DAY BEFORE THE SHOW - THURSDAY 28th SEPTEMBER

At the time of printing the group activity for the early arrivals will be a visit to the USAF museum at Dayton. We usually go in a group of two or three cars so if you want to be included ask any of us on Wednesday night or Thursday morning in the breakfast room in the Marriott. We usually leave right after breakfast, between 9 and 9:30. Afterwards we usually go for a late lunch at an Owl-themed diner infamous throughout the United States. Entry to the museum is free.



It's a pretty good place to visit even if you're not into military aircraft, there is also space age exhibits from the 60s-90s, and presidential planes, as well as rare experimental planes.

FRIDAY ARE THE TANDY TECH TRACK (TTT) CLASSES

A good idea is to keep an eye on the Tandy Assembly website for regularly updates. Four of the five TTT instructors have been announced, and the 5th should not be far away.

So far we have Ruud Broers from the Netherlands flying in to give a talk about how he designed a 512K memory expansion for the TRS-80 Model II.



Following Ruud we have TA regular Jeff Birt who will be taking about the cosmetic restoration of vintage computers. Many times the electronics of a computer can be easily repaired, but the overall resulting working computer can be a bit underwhelming if it looks knocked-about and unloved. Jeff will help you make it look great!



I am up next with my usual sidekick Eric Dittman. Our talk will be about 10 TRS-80 items which were, to put it bluntly, a bit crap. You eagerly buy it, get it home, and quickly realise it doesn't live up to it's expectations. Some of these items are quite collectable these days, mainly for their craptasticness... everyone should find something here to laugh at. Where possible we should have examples to show.





The fourth Tech Track will be Matt Boytim and Arno Puder who will be talking about their latest and great invention TRS-IO++, which is an all-in-one card that incorporates TRS-IO and PocketTRS functionality.



The last Tech Track presenter has yet to be announced but it would only be a matter of days away, so keep your eyes on the website:
www.tandyassembly.com
Likewise the main speakers will also be announced soon for the Saturday and Sunday of the show.

OTHER ATTRACTIONS:
Last year improvements were made to the Tandy Assembly auction and the trivia game. I'm now the auctioneer. The auction has always been one of the highlights of TA and when I run it I try to aim for around 1

minute or less per item, which includes any description or notes on the item, or anything else I care to say about it. At around 2 hours in duration it means we can get through around 100-120 items, so there should be something there for everyone.



Just in case you have been living under a rock for the last 11 months, I now run the trivia game. I have included elements of pub trivia games, so now the audience is involved, via an answer sheet, to answer 9 incredibly difficult TRS-80 related questions. This is in addition to the usual three contestants up the front.



Prizes are awarded to the winners, who last year was Eric Dittman in the 60-question 3-experts game, and Peter Cetinski was the winner of the audience game. Can you beat Pete? Again, the trivia game is a not-to-be-missed part of the show. Remember to bring a pen though.

TASTY SWEET BAKED TREATS FROM PASTRIES&PEARLS

A new addition for 2023 will be the baked treats from Canadian ladies Andrea and Amy Grimwood, with the assistance of Elizabeth Hayes, have a range of sweet tasting treats to keep you and your family from feeling famished. Famous for their appearances at Comic Con and other special events in southern Ontario, the girls will have something to appeal to everyone.



Pastries & Pearls
andrea@pastriesandpearls.com
www.pastriesandpearls.com

THEME AND OTHER INFO HANDY TO KNOW



The theme of Tandy Assembly most years represents the computers released during the timeline they were sold. 40 years ago was 1983 and the major computer releases were the Model 4, the Color Computer 2, and the Model 12/16B business computers. Each year apparel to celebrate the TA event is released, and this is the proposed design for TA23:

As usual they'll be available in T-shirts and Hoodies. Some of the merch from previous TAs are also still available, so if you missed out last year it's worth asking at the merch table what is left from previous years.

IT'S A LONG FLIGHT BUT I GO BECAUSE LIFE IS SHORT

Though it's possible some would spend more than 30 hours to get to TA, I hold the record for the longest transit to the event, taking three planes to get to Ohio and three more to get home. That's around 50 hours flying and 10 hours just hanging around airports waiting for connecting flights.

I'm used to it now and the chance to meet up with friends yearly, greatly outweighs the minor inconvenience of the travel required. I'd rather look back and think I did the best I could as part of the TRS-80 community to be as involved as I can be in events when they are organised, such as TA. See you all there at the end of the month.



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.

INTERESTING HACKS FROM YOUTUBE

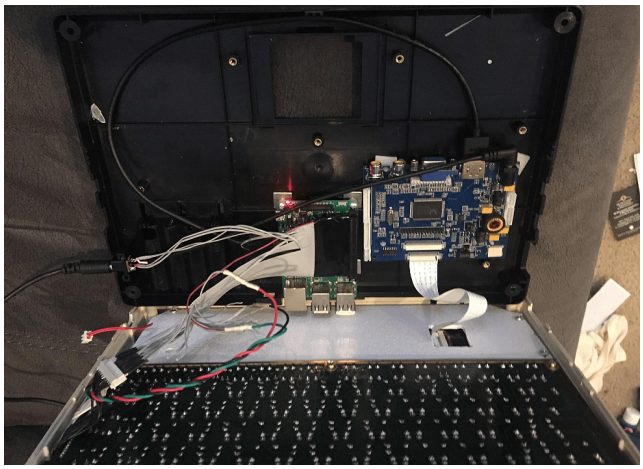
Thanks to Leon Heller for sourcing the information

RASPBERRY PI3 IN A MODEL 100









I GUESS THIS WAS A CASE FOR A PI.
A BROKEN TANDY M-100
SCREEN IS A 7.8" 400X1280
KEYBOARD WORKS WITH A SMALL PYTHON SCRIPT.

THE LCD IS FROM EARTHLCD.COM PART# EARTHLCD-7-4001280 CODEMAN

FURTHER DETAILS, CHECK OUT

ISN'T THIS JUST BEYOND FANTASTIC. YOU COULD RUN TRD80GP ON IT TOO ED.

Raspberry Pi in a Model 4P

https://youtube.com/watch?v=hT-XvVvhYkY&feature=share9





32K M1 MEMORY

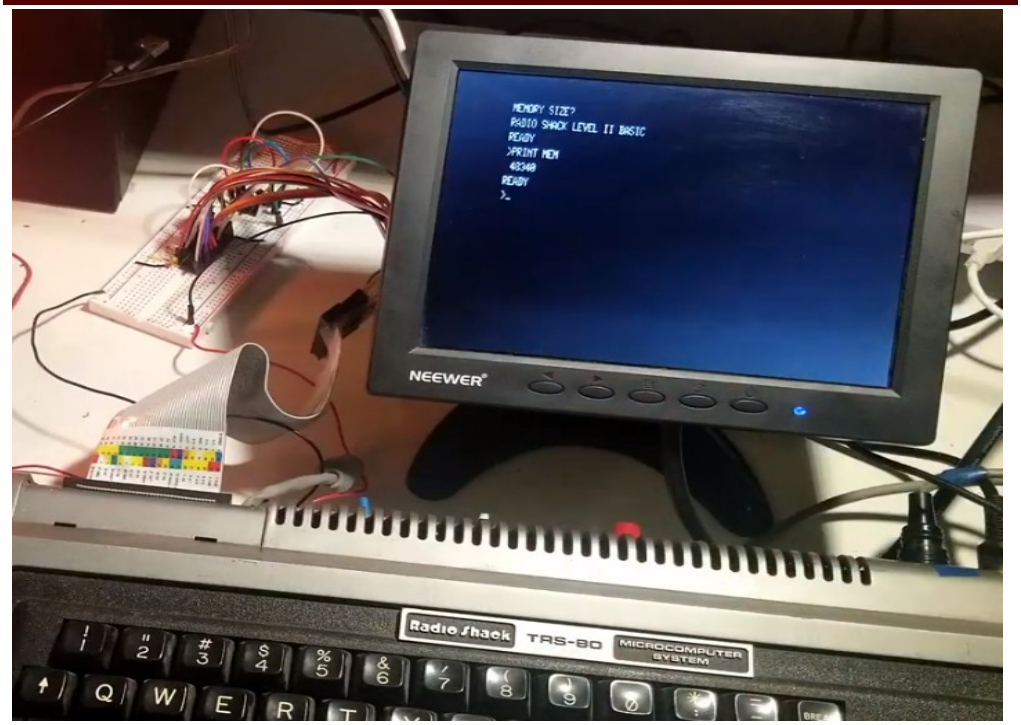
UPGRADE USING 2 CHIPS

NON-MIRROR

SYSRES	2	40	39	1	RAS*
A10	4	38	37	3	CAS*
A13	6	36	35	5	A12
GND	8	34	33	7	A15
A14	10	32	31	9	A11
OUT*	12	30	29	11	A8
INTAK*	14	28	27	13	(MEM) WR*
MUX	16	26	25	15	(MEM) RD*
D4	18	24	23	17	A9
D7	20	22	21	19	IN*
D1	22	20	19	21	INT*
D6	24	18	17	23	TEST*
D3	26	16	15	25	A0
D5	28	14	13	27	A1
D0	30	12	11	29	GND
D2	32	10	9	31	A4
A3	34	8	7	33	WAIT*
A7	36	6	5	35	A5
A6	38	4	3	37	GND
A2	40	2	1	39	NC/GND

MIRRORED

RAS*	1	39	40	2	SYSRES
CAS*	3	37	38	4	A10
A12	5	35	36	6	A13
A15	7	33	34	8	GND
A11	9	31	32	10	A14
A8	11	29	30	12	OUT*
(MEM) WR*	13	27	28	14	INTAK*
(MEM) RD*	15	25	26	16	MUX
A9	17	23	24	18	D4
IN*	19	21	22	20	D7
INT*	21	19	20	22	D1
TEST*	23	17	18	24	D6
A0	25	15	16	26	D3
A1	27	13	14	28	D5
GND	29	11	12	30	D0
A4	31	9	10	32	D2
WAIT*	33	7	8	34	A3
A5	35	5	6	36	A7
GND	37	3	4	38	A6
NC/GND	39	1	2	40	A2



https://www.youtube.com/watch?v=skFTp3Tn0-8

CP/M IN A MODEL 100

https://www.youtube.com/watch?v=eZQLcTmW1sE





VI. Basque Tournament of TRS-80
Egoitz Campo

The last 12th of July, the Euskal Retro organized a new tournament of TRS-80, the VI. Basque Tournament. We met in Derio.

It was a fun afternoon, and we had a great time playing the TRS-80 again. Unfortunately among the games we played, we could barely play one of our favorite games; Monster Mazer because the image was rapidly getting worse.

After playing all four games; Monster Mazer, Clowns & Balloons, Demolition Derby and Canyon Climber, Egoitz goy the highest score. For the next tournament, possibly in april 2024, we hope to have new games like Dragon Fire.

Here you have the results:

Demolition Derby

1-Unai Pérez 4 points (2.400)
2-Egoitz Campo 3 points (1.700)
3-Daniel Barrientos 2 points (1.200)
4-Andi Erromo 1 point (200)

Monster Mazer

1-Egoitz Campo 2 points (750)
2-Daniel Barrientos 1 point (110)

Canyon Climber

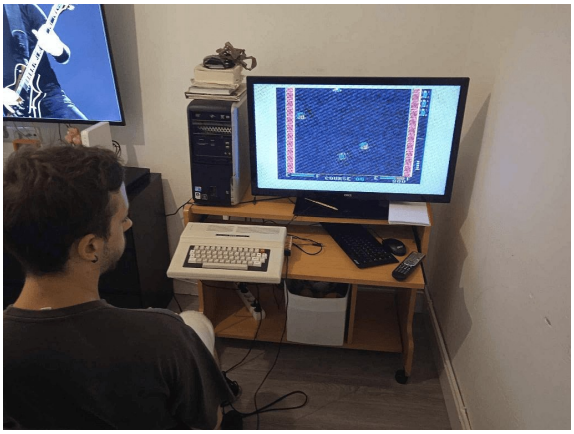
1-Egoitz Campo 3 points (14.000)
2-Unai Pérez 2 points (8.000)
3-Daniel Barrientos 1 point (2.400)

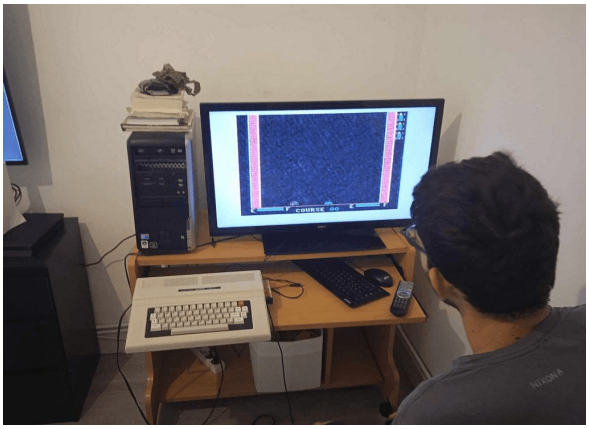
Clowns & Ballons

1-Egoitz Campo 4 points (5.060)
2-Daniel Barrientos 3 points (2.530)
3-Unai Pérez 2 points (2.320)
4-Andi Erromo 1 point (60

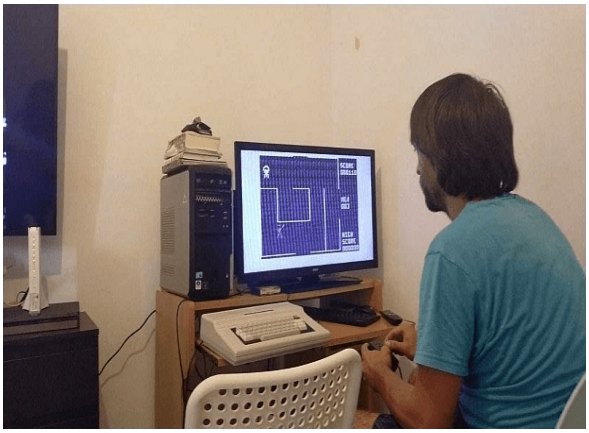
Classification

1-Egoitz Campo 12 points (Basque Country)
2-Unai Pérez 8 points (Basque Country)
3-Daniel Barrientos 7 points (Basque Country)
4-Andi Erromo 2 points (Basque Country)

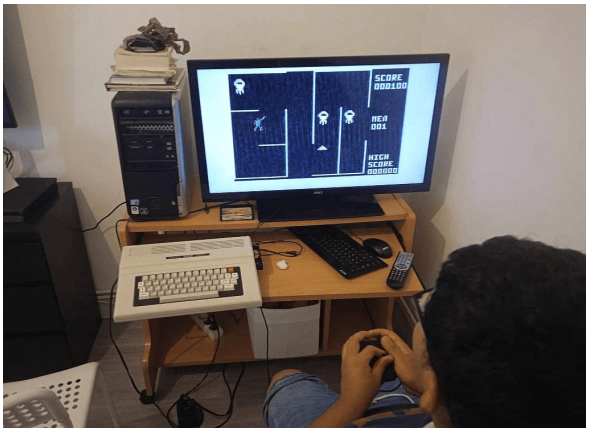




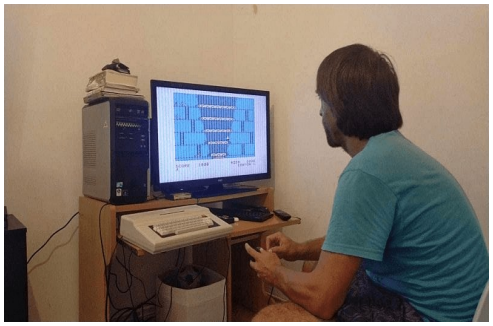


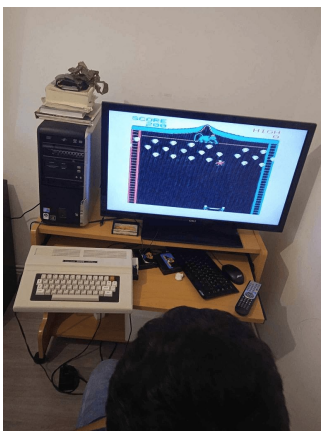


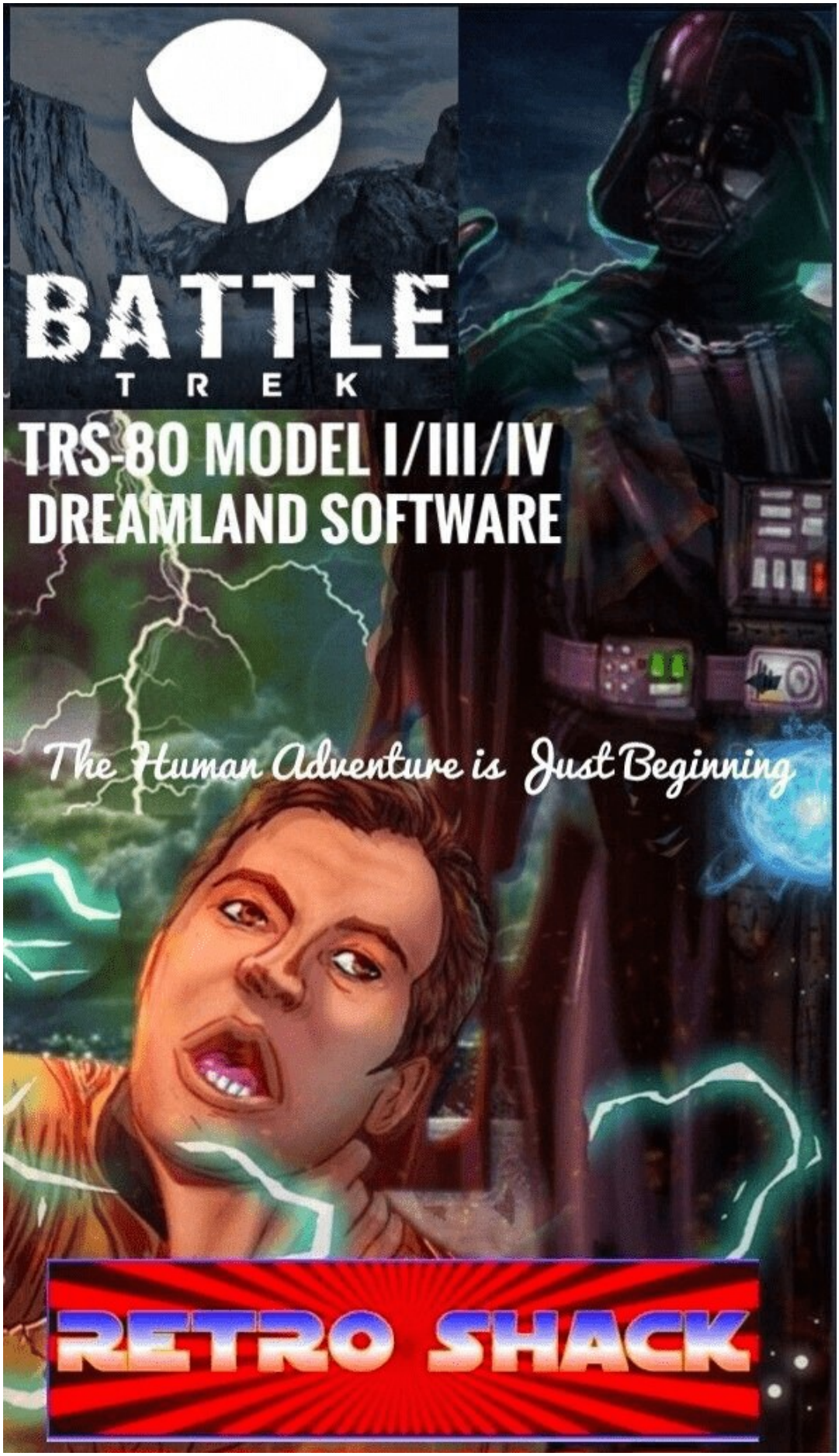



SCORE
000100
MEN
001
HIGH
SCORE
000000








BATTLE
TREK
TRS-80 MODEL I/III/IV
DREAMLAND SOFTWARE
The Human Adventure is Just Beginning
RETRO SHACK

HI DUSTY. HERE'S AN INTERESTING GAME TAKEN FROM A MAGAZINE, ORIGINALLY WRITTEN FOR THE COMMODORE 'PET', BY **ANDY HENNIGAN.**
I THINK THERE WAS SOMETHING SIMILAR FOR OTHER MICROS BACK IN THE DAY, BUT ANDY ENTITLED IT ...



I THOUGHT YOU MIGHT LIKE TO INCLUDE IT IN THE NEXT EDITION OF TRS8BIT. I'VE SET OUT THE SIMPLE RULES BELOW AND SHOWN AN EXAMPLE RUNNING ON TRS80GP. *(ON THE NEXT PAGE. ED.)*

KEEP UP THE GOOD WORK WITH TRS8BIT.

TU-BARDH

SOLGOR IS A SIMPLE NUMBERS GAME THAT'S QUITE EASY TO PROGRAM BUT REQUIRES A BIT OF THOUGHT TO PLAY.

AT THE BEGINNING OF THE GAME, YOU'RE PRESENTED WITH A SET OF NUMBERS FROM 0 TO 9, IN RANDOM ORDER, VIS. -
3564279081

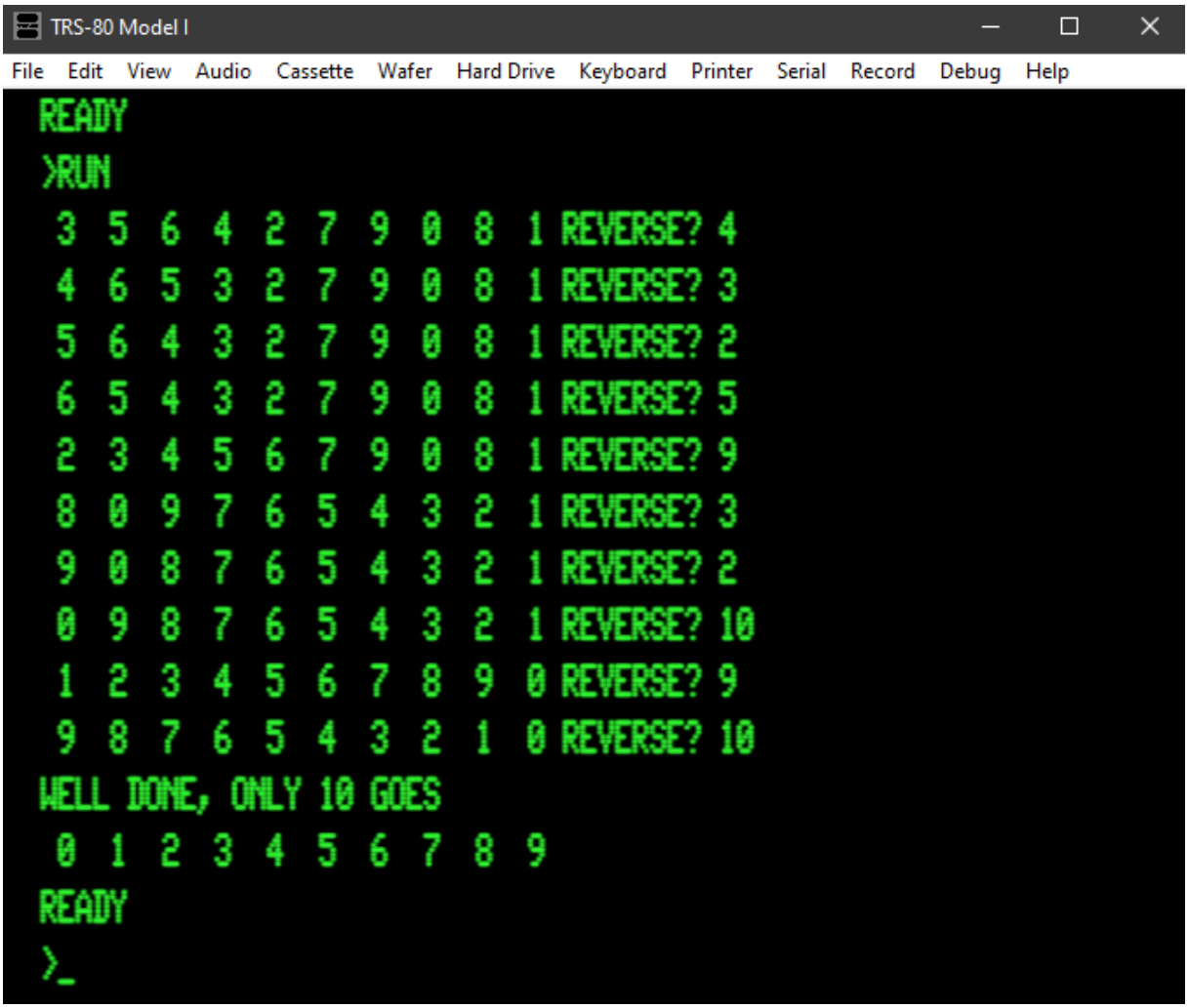
ALL YOU HAVE TO DO IS SORT THEM INTO ASCENDING ORDER 0 TO 9, **S**MALLEST **O**N THE **L**EFT, **G**REATEST **O**N THE **R**IGHT, (HENCE THE NAME). THE CATCH IS THAT YOU'RE ONLY ALLOWED TO REVERSE THE ORDER OF 2 TO 10 DIGITS COUNTING FROM THE LEFT WITH EACH MOVE.

FOR EXAMPLE, IF YOU DECIDE TO REVERSE THE FIRST THREE DIGITS, THE NUMBERS WILL BE BECOME -
6534279081

THE GAME FINISHES WHEN THE NUMBERS ARE IN SOLGOR ORDER AND YOU CAN THEN JUDGE YOUR PERFORMANCE BY THE NUMBER OF MOVES IT HAS TAKEN.

THE SAMPLE RUN SHOWS ONE WAY OF SOLVING THE ARRANGEMENT, GIVEN ABOVE. THE EASIEST AY TO ATTACK THE PROBLEM IS TO JOIN UP NEIGHBOURING NUMBERS WITHOUT BREAKING UP EXISTING RUNS.

IT'S IMPORTANT TO THINK AHEAD SO THAT YOU CAN RE-UNITE TWO NUMBERS WITH AS MANY MOVES AS POSSIBLE.




```
10 RANDOM:CLS:FORI=1TO10:N(I)=-1:NEXT
20 FORI=1TO10
30 X=RND(10):IF N(X)>-1THEN30ELSE N(X)=I-1:NEXTI
40 FORZ=1TO10:PRINTN(Z);:NEXT
50 INPUT"REVERSE";R:IFR<2 OR R>10 THEN50ELSEG=G+1
60 FORI=1TOINT(R/2):P=N(I):N(I)=N(R-I+1):N(R-I+1)=P:NEXTI
70 FORI=1TO10:IFN(I)<>I-1THEN40ELSENEXT
80 PRINT"WELL DONE, ONLY";G;"GOES"
90 FORZ=1TO10:PRINTN(Z);:NEXT
```

THE ORIGINAL PROGRAM WAS OVER 50 LINES LONG, SO TU-BARDH NOT ONLY CONVERTED IT FROM COMMODORE BASIC, ENABLING IT TO RUN ON A TRS-80 BUT SHORTENED IT SOMEWHAT TOO!

DON'T FORGET, IF YOU'RE LUCKY ENOUGH TO BE RUNNING TRS80GP, THE EASIEST WAY TO GET A COPY OF THE PROGRAM IS TO COPY THE 9 LINES FROM THE ABOVE BASIC PROGRAM AND JUST SIMPLY PASTE THEM IN TO BASIC.

ED.



As always, of course, I'm on the lookout for any software for the U.K.'s **Aculab 'Floppy Tape'**. There were versions specifically 'zapped', for use on the AFT, of Electric Pencil, Scripsit, Zen, Level III BASIC and VisiCalc. There was also the enhanced BASIC, XBAS. It's so disappointing to think that they have all been lost!

Please email me at the usual address if you can help.

ALGOL—60
Z80-CP/M, PDP11, PDP8

A family of high level languages
from RHA (Minisystems) Ltd.

ALGOL—60, the language from which PASCAL is derived. A mature implementation with comprehensive operating system and machine code interfaces.

SYSTEJ-ALGOL, the subset of Algol-60 in which all the compilers are written. Compiled code is shorter, execution faster.

Z80 based CP/M systems including TRS80
RML Algol-60, includes the option of 32 bit integers instead of floating point. About 7 times faster than TRS80 level II BASIC, speed comparable with Microsoft Fortran. Document £10, system £99 + VAT. System-Algol £50 + VAT, free leaflet.

PDP11 WITH RT-11, RSTS, RSX or IAS and PDP8 with OS/8 or stand alone
Complete package including both compilers in machine readable source form ... £250 + VAT
consultant.

83, Gidley Way, Horspath, Oxford OX9 1TQ (08677) 3625

TRS 80 16K L11

THE FUNCTIONAL BUSINESS SOFTWARE for CASSETTE
BREAKTHROUGH. BRITISH COMPOSED WORD PROCESSOR TYPER IV AT £15.00 INCLUDING MANUAL
YEAR AUDIT/ BANK ACCOUNT
Shows Bank Balance instantly. Shows accumulated expenditure in 36 columns, Balances to date, month & year, from which can draw up a balance sheet. Stores all names & numeric records. £21.95.

ORDERS PROGRAMME	14.95
SALES LEDGER	21.95
MAILING LIST	9.95

All programs are menu driven needing no operator expertise. Most responses require only single key-strokes. Operator errors correctable.

We believe ou software to be unequalled in Function-ality & ease of use. Causes great excitment from users. Fascinating Games with high quality graphics, by one of Britain's leading Authors from £5.00

2000 Sheets 11"x9½" Fanfold paper £14.00 + 2.55 p&p. Multi copy forms in stock

Printers from: Epsom MX70 £259.00. OKI 80. £289.00

S.A.E. for catalogue. *ALL SOFTWARE PLUS . 75 pence P&P*
ALL PRICES PLUS VAT
ACCESS COMPUTERS
2 ROSE YARD, MAIDSTONE, KENT: 0622 58356



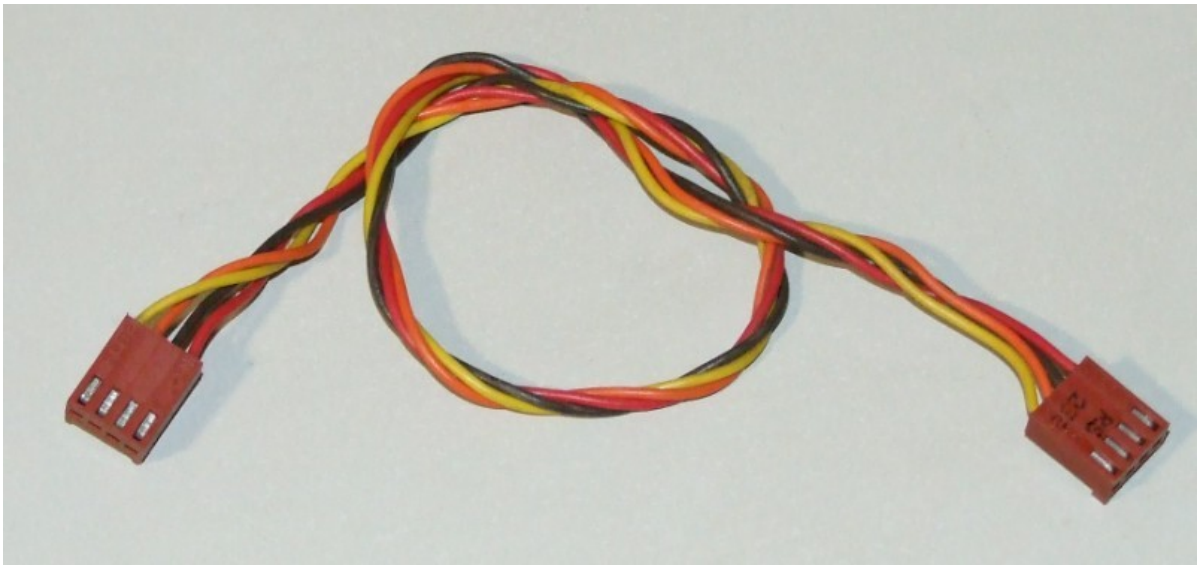
I CAME ACROSS THIS PHOTO ON BILL DEGNAN WEBSITE.

DREW HELPING WHEEL OUR NEW DEC PDP-8 COMPUTER, DONATED TO KENNETT CLASSIC MUSEUM.


Joe Kay
Director
NORTH COMPUTER WEST MUSEUM
Create Experience Learn
joe@nwcomputermuseum.org.uk
01942 582826
Information Technology facilities & training
Workshops for programming, Electronics & VR
Community classes, clubs & retro games
IT Skills for the young & old
City & Guilds Creditation
Supporting education of local people
Creating opportunities & careers
Disable friendly
Volunteer-led training & museum operations
NORTH WEST COMPUTER MUSEUM
Create Experience Learn
www.nwcomputermuseum.org.uk
Leigh Spinners Mills, Park Lane, Leigh, WN7 2LB

IN MAV'S WORKSHOP
Model III/4 Internal Power Connections Explained

Ian Mavric



Every few years I get an email from someone doing a restoration on a Model III or 4 (the early Non-Gate Array version is the focus in this article), which goes something like "It doesn't work at all now and has a screen full of garbage" or "It works but only says Cass? prompt", or in extreme cases "I lost some or all of the power cables, what can I do now?".
My old mate Jay Newirth asked me recently if I can properly explain the power connections on the RS232 and Floppy Disk controller (FDC) on the Model III and I thought it would a good time to explain it all once and for all, in a place easy to find for all future reference. Firstly, Radio Shack made it harder than it needed to be, but once you get your head around the anomaly, it's easy to remember when working on a Model III or 4.

MODEL III AND UPGRADED 16K MODEL 4 (26-1067)

Both these machines, as far as power cabling are concerned, are the same computer. The 26-1067 (16K Diskless Model 4) has the same power supply (PSU) as the Model III. Like the Model III, the PSU powers the motherboard, monitor, and RS232 board.

When one of these diskless Model 4s are upgraded with disk drives (or indeed a diskless Model III is upgraded with disk drives) a second PSU is added to the side of the drive tower to power the FDC and disk drives.

I call these the dual PSU setup, and they have individual wiring looms which connect to their devices. 4-pin connectors are used, with 3 or 4 wires coloured red, yellow, brown, and orange.

Here is the first thing to remember, the FDC only has three wires (5V, 12V, Gnd), so that one is easy to locate if your parts are in disarray after disassembly of the computer. The cable with a red, orange and brown wires only is for the FDC. It can be plugged into the RS232 board or motherboard but it won't work.

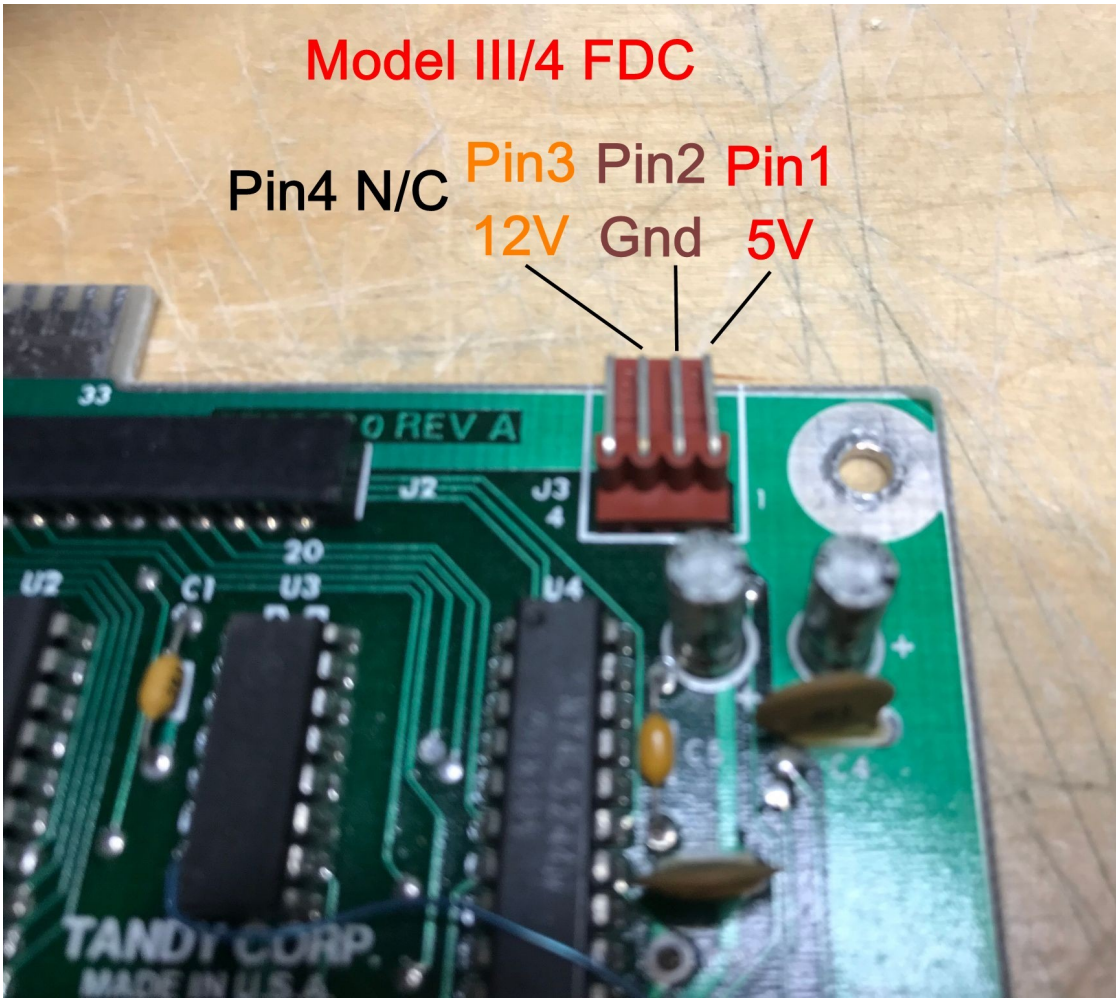


Figure 1: FDC power pin designation (colours represent wire colours)

So that's the easy one out of the way. The confusion, nay, anomaly, comes from the 4-wire cables connecting the motherboard and RS232 to the PSU.
When you first notice it you wonder, as I did, what were they thinking? What happened is somewhere between laying down the design for the Model III motherboard and serial board, the power pin designation got mirrored. The upshot of which is that the 4-wire power cables for the serial and motherboard are slightly different. I'll show you.

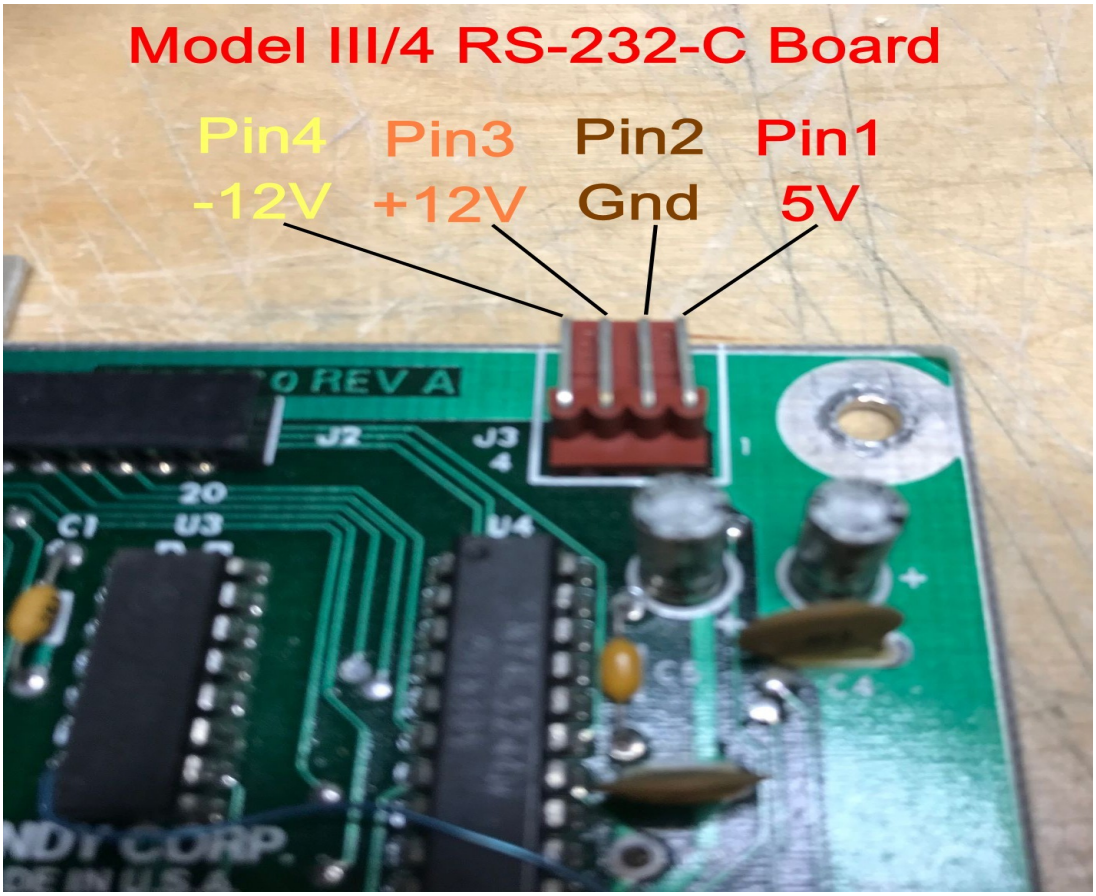


Figure 2: RS232 board power pin designation (colours represent wire colours)

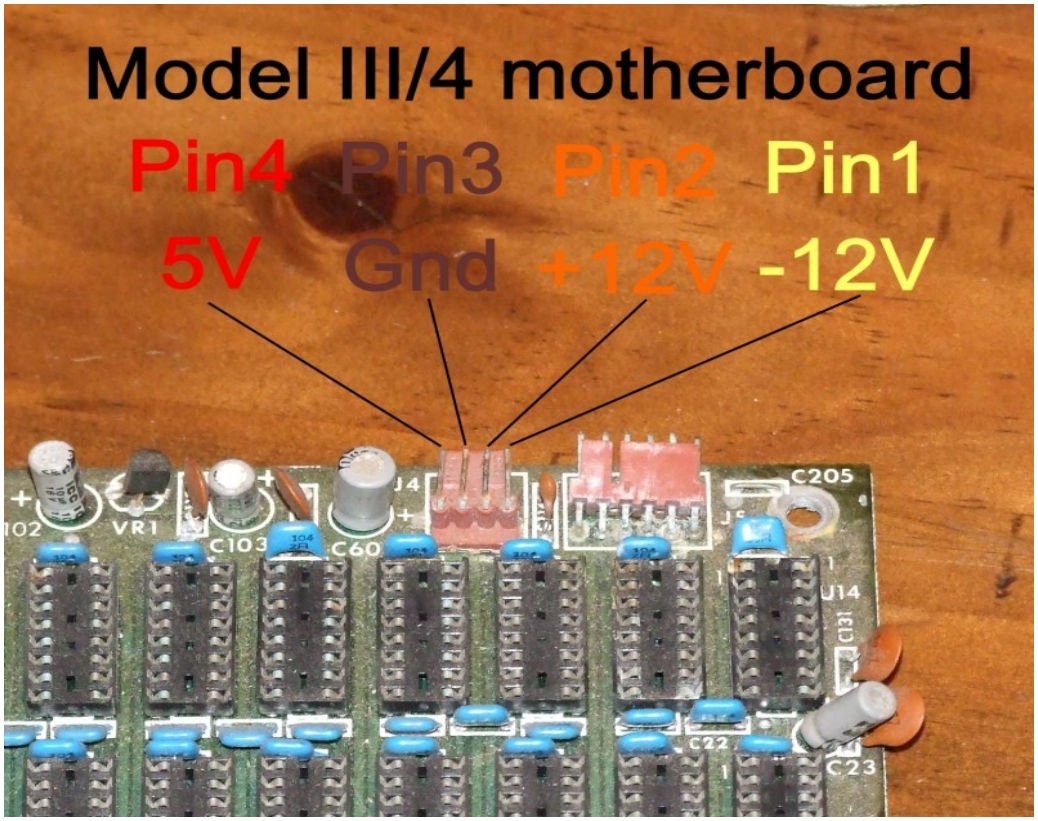
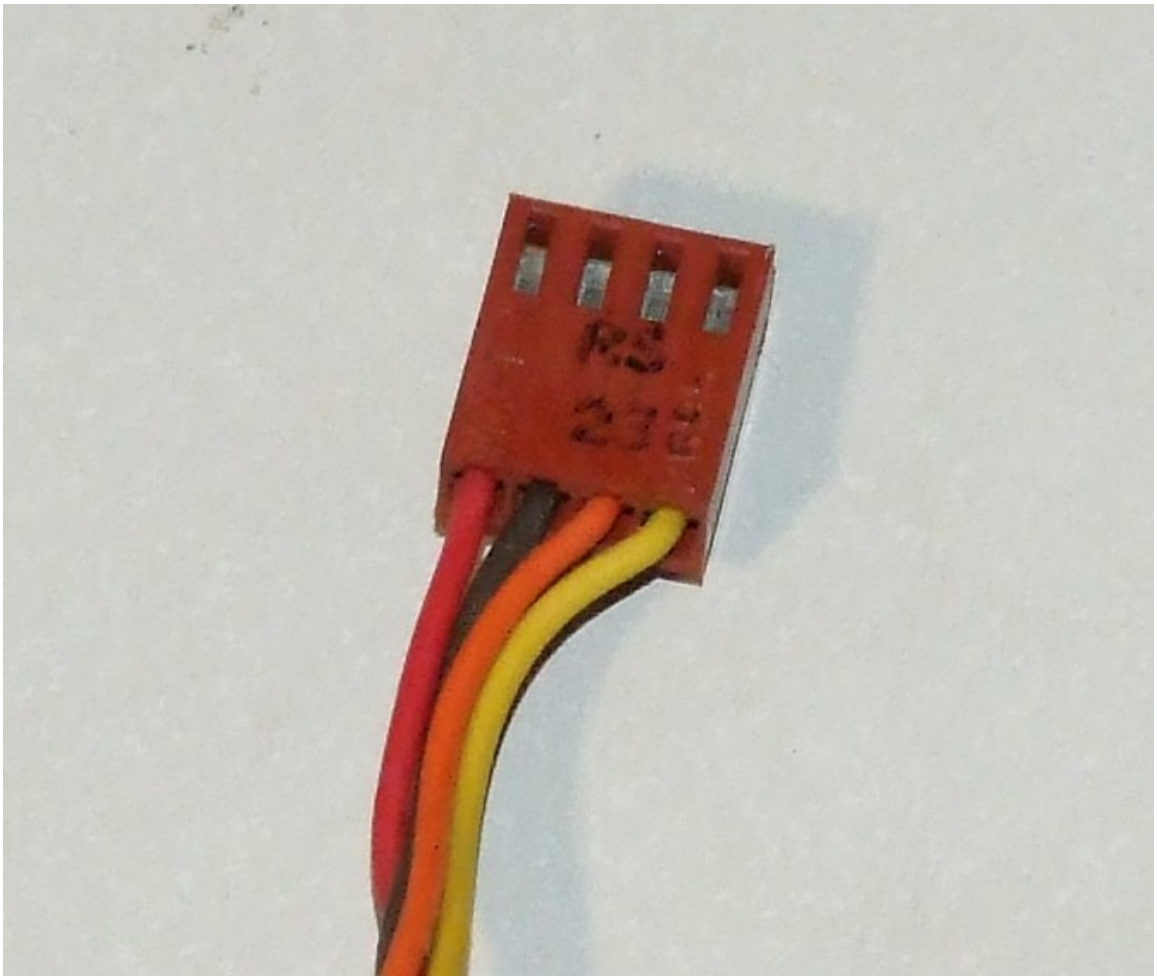


Figure 3: motherboard power pin designation (colours represent wire colours)

See how the power inputs to each of the 4 pins are mirror imaged on the serial compared to the motherboard? What does it mean? To me someone made an error in the design but the problem wasn't picked up until thousands of serial boards were produced and it was easier to just re-arrange the wires on the power cable loom.

It only has to be done at one end because the power output pins on the PSU to not change. Recognising this might be a problem down the line, RS stamped in black ink which is the CPU power cable and which is for the RS232. It sucks that after 40+ years the black in fades on some of them.



Picture: you can just make out the RS232 stamped on it

WHAT HAPPENS IF YOU PLUG THEM IN THE WRONG PLACES?

Luckily, nothing will be damaged. Plugging the serial power into the motherboard will result in a short, so the system will not start.
Plugging the serial power into the FDC will actually still run and boot because the yellow -12V wire goes nowhere.

People usually then plug the 3-wire cable which is supposed to be for the FDC into the serial board, but the serial will not work without -12V. No data will flow out of it be received into the RS232 port. Computer and disk drives will still run. If you never use the RS232 port you may never know the serial board is not powered properly.

MODEL 4 DISK SYSTEM (26-1068, 26-1069)

Disk-based Model 4 systems differ from the Model III and diskless Model 4 mainly in the PSU department. To save money Radio Shack quite rightly changed from two smaller power supplies to one large 65W PSU. Along with this change included a new all-in-one wiring loom with 5 connectors: motherboard, RS232 serial, FDC, and two disk drives. (The monitor was powered by a separate 2-wire cable.)

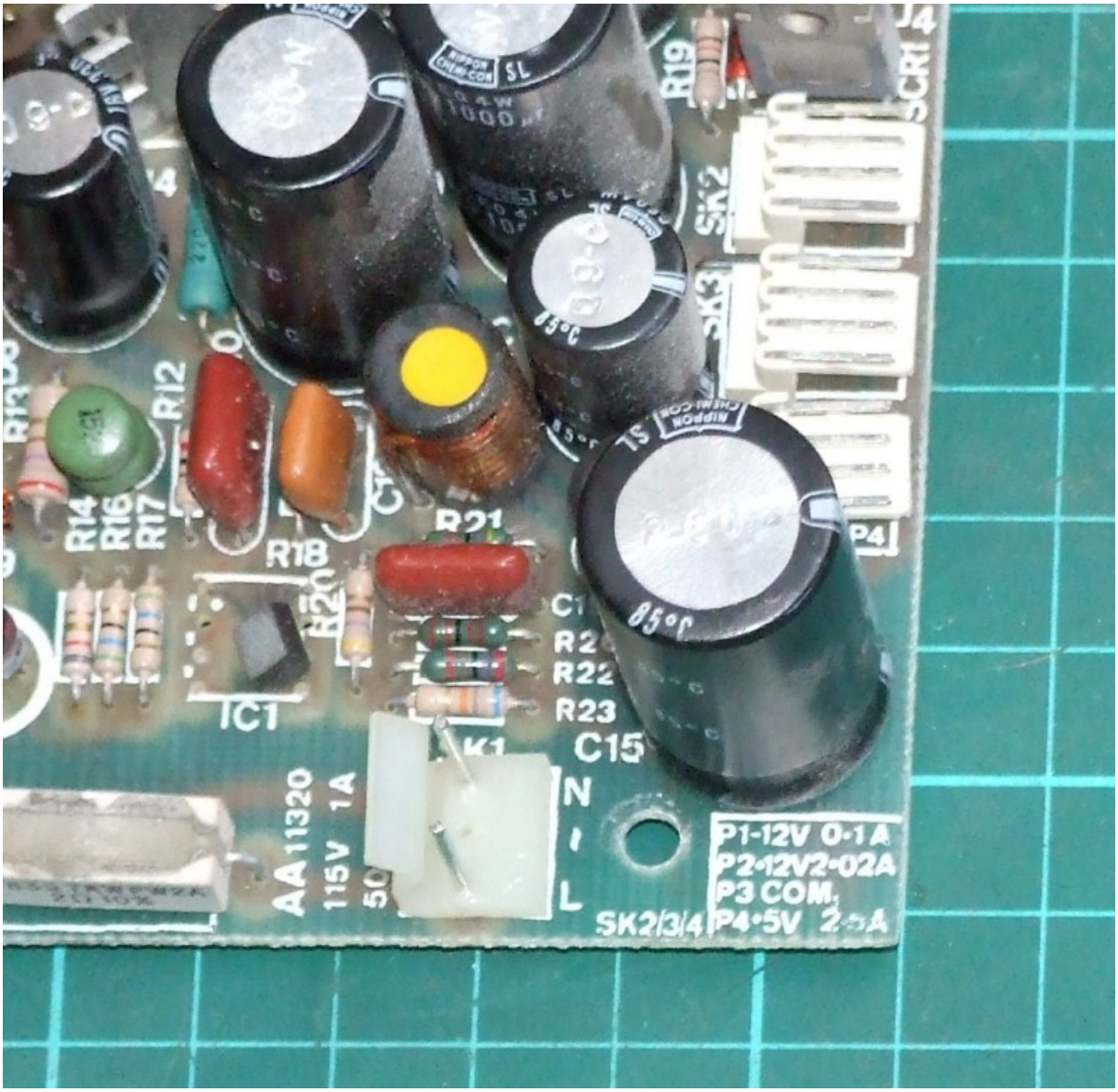
The same rules apply as above and so the same pictures above are again used to decipher which plugs go to which place. The 3-wire goes to the FDC, and the two remaining connectors with 4 wires you need to look at the wires closely, and compare them with the pictures above. If you are lucky the might still have the original black stamped 'CPU' and 'RS232' on them... it only needs to be readable on one of them.

Like on the previous Model III and diskless Model 4 systems, plugging them in the wrong place will result in a non-running computer with a garbage screen, or a short and no-start (you might hear a faint ticking in the power supply which is the overload cut-off rapidly switching).

DISASTER - I'VE LOST MY POWER WIRING LOOM(S)!

It was there before, or the other day, or months ago when you disassembled the computer, but now one or more of the power cables from the PSU to a board are missing, never to be seen again. Fortunately, they are pretty easy to make. This mostly applies to the Model III and diskless Model 4, as the all-in-one wiring loom of the disk-based Model 4 is rather complex to build. We are helped because the 35W PSUs clearly mark which voltages are on which pins at the three PSU output connectors:

This picture is actually upside-down compared to how it would be mounted in a Model III or 4, but in the lower corner you can see the voltages for each of the 4 pins on the 3 power output connections (the white connectors). So by making a cable is as simple as crimping the four wires and plugging them into a header, and then crimping the other end as per Figures 1, 2 and 3, above. Using the same colour for the wires and you should be back up and running quickly.



CONCLUSION

So now that you know, next time you disassemble your computer for repairs, the quirks of the power distribution cables inside the Model III and 4, you should be able to get it back up and running with much less frustration. For those who have bought a computer off eBay or FB marketplace which works great but the RS232 isn't working, check the power cables to the FDC and serial board aren't switched before delving in deeper, removing the motherboard, and scoping around the serial board.

Ian Mavric

ianm@trs-80.com



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

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



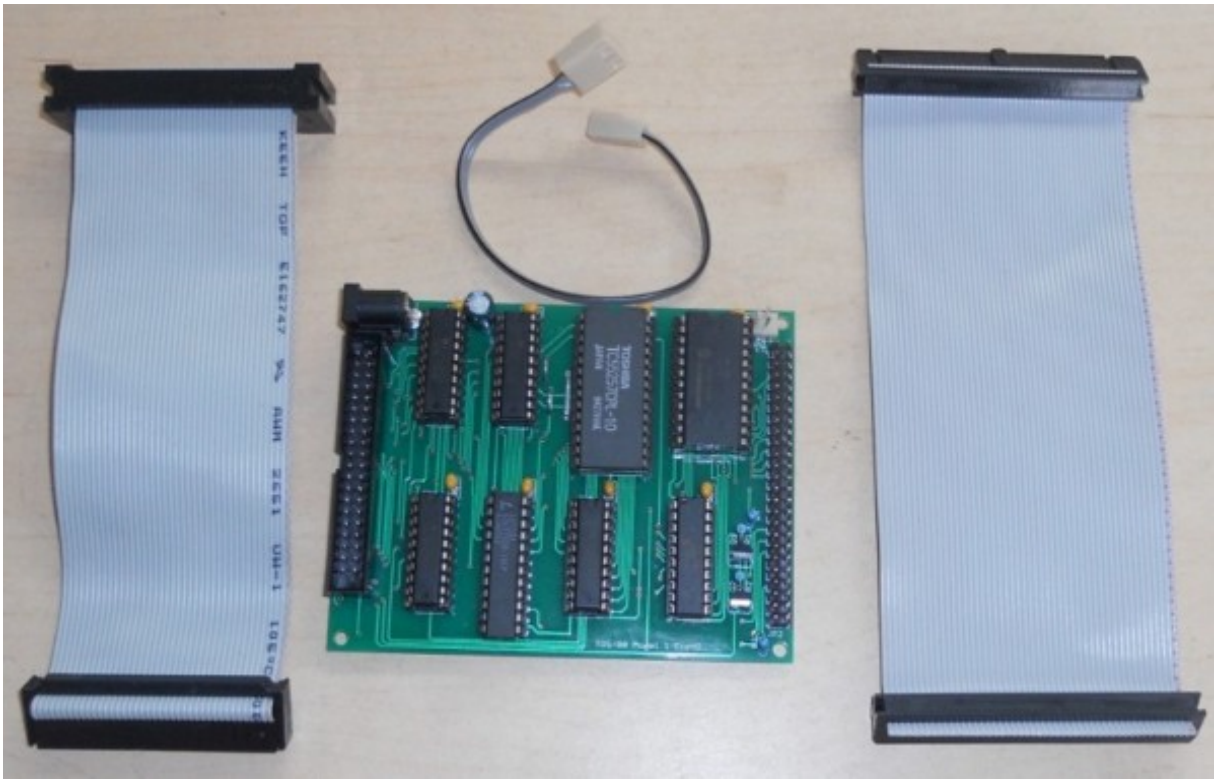
* Uses modern day MEAN WELL power supply units
* 50w for Model 3 version and 65w for the Model 4 version
* The Model 4 version uses a SEPARATE +12V POWER SUPPLY (prevents video "spasm")
* No more RIFA CAP explosions
* No more COLD solder joints
* Units are DIRECT REPLACEMENTS for the original power supplies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

newSOFT

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

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

- COMPLETLY ASSEMBLED AND TESTED -



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

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

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

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

The amazing

"FreHD"



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

Order yours today
http://members.iinet.net.au/~ianmav/trs80/



E.T. FONEHUME

HI DUSTY, HERE'S A ONE-LINER, A GUESSING GAME WITH A TWIST. THE COMPUTER 'THINKS' OF A NUMBER AND YOU'RE ASKED TO GUESS WHICH ONE IT IS, BETWEEN 1 AND 100. THE MORE INCORRECT YOU GUESS, THE MORE STARS ARE PRINTED. ZAP ALL THE STARS TO WIN, BUT, OF COURSE, YOU ONLY GET 7 TRIES!

```
0 RANDOM:CLS:X=RND(80)+20:M=7:FORK=1TOM:PRINT:INPUT"YOUR GUESS";
G:IFG=XTHENPRINT"WELL DONE IN";K;"TRIES":ENDELSED=ABS(G-X):B=-(D
>=1)*-((D)=1)+(D)=3)+(D)=7)+(D)=15)+(D)=31)+(D)=63)):B=B*B:FORZ=
1TOB:PRINT"*";:NEXT:NEXTK:PRINT:PRINT"SORRY ALL 7 TRIES USED :("
```

```
TRS-80 Model I
File  Edit  View  Audio  Cassette  Wafer  Hard Drive  Keyboard  Printer  Serial  Record  Debug  Help

YOUR GUESS? 50
**************
YOUR GUESS? 35
****
YOUR GUESS? 30
*
YOUR GUESS? 29
WELL DONE IN 4 TRIES
READY
>_
```

Software Tools in BASIC: Episode 2

Vince Otten

OPENing Pandora's Box

If you saw the Introduction to "Software Tools in BASIC" in the previous issue of TRS8Bit, you know that I wanted to create a version of Unix's "sh" shell program which would allow me to specify input and output files, and pass along a "command line" to BASIC programs built using a SKELETON/BAS framework. So I could run SH/BAS, and enter

COPY <INFILE >OUTFILE –OPTION1

and COPY/BAS, coded within the SKELETON/BAS outline, would automatically know that INFILE was its standard input, OUTFILE was its standard output, and that its list of commands would include

"COPY"
"-OPTION1"

I shared the DEFined FuNctions (DEF FNs) that I use to gather important information about each OPEN file, such as "how many characters or bytes are stored in INFILE". My "Software Tools" programs need to know this information in order to read 1 character or byte at a time. I can't use Disk BASIC's INPUT or LINE INPUT statements, because they "interpret" characters like ASCII 13 (Carriage Return).

So how do I get one byte at a time, regardless of its numeric value, from a disk file, for example? I have to use Random I/O, using OPEN "R", FIELD, LSET, and GET instead of the Sequential method that uses OPEN "I" and INPUT. That means GETting a 256-byte record from the file, and accessing 1 character each time I GOSUB to the "getc" subroutine. Once I've read the 256th byte, I'll need to GET the next record and read its first character. My program will keep doing this until it's read as many characters as my DEFined FuNctions report are in that file, like this:

1. OPEN the file in "R"andom mode.
2. Have we read less than the total number of bytes in this file? If not, we report "EOF" and we're done.
3. Have we already read the last character in this record (ie, the 256th)?
4. If we have, time to get another record and reset our "position" where we get a character. Go back to step 2.
5. If not, we locate the character in this record. (say, at position 100). That's going to be our return value. Then we increment the "position".
6. We're done for now!

That doesn't seem too difficult. But notice that this procedure has some unusual features. I'll digress in order to illustrate here.

If you had to write a "max()" function that returned the larger of two values, you might code it like this (in a sort-of-C-language-I-use-for-pseudocode):

```
max( a, b) {
        if a > b
                return a
        else
                return b;
        }
```

This function takes two "arguments", 'a' and 'b'. It returns one value. It isn't affected by anything "outside" itself, and it doesn't affect anything outside itself.

Now consider the C function "getchar()", which returns 1 character or byte from the standard input. It takes no arguments. It returns two kinds of values: either a byte value (an ASCII character, typically), or the special EOF end-of-file value (typically "-1"). Many C library functions do this: they return their typical values, but on special occasions like EOF or an error, they return some special value outside the range of the normal values.

getchar() is also affected by conditions outside itself. It gets its characters from the standard input: that may be a disk file, or the keyboard, for example.

My first example here, "max()", can be written in TRS-80 Disk BASIC using DEFined FuNctions:

```
64010 DEF FN MX%( A%, B%)= (A% < B%) * -B% + (NOT A% < B%) * -A%
```

A bit more complicated than my pseudocode example above, but like that it takes 2 arguments, returns 1 value, and isn't affected by anything outside itself.

getchar(), on the other hand, is a subroutine, and the big drawback of TRS-80 Disk BASIC is that subroutines neither take arguments nor return values. For that reason, as I described in the last article, I've created a set of BASIC arrays both for arguments and for return values:

- VA()arrays, ie VA%(), VA!(), or VA$(): these are used to pass "arguments" TO a subroutine;
- VR() arrays, ie VR%(), VR!(), or VR$(): these are used to pass "return" values back FROM the routine;

You may be saying, "I can understand using an array for arguments. But surely you only need one return value. Why are the VRs arrays, rather than a single return value?"

I learned this from studying the ancient programming language SNOBOL4: there's a difference between what a procedure like getc() *returns* (a character or byte) and what it *reports* (EOF). Languages like SNOBOL4 and Icon state that the function getc() normally returns a character and *succeeds*, but when it reaches the end of the file, it *fails* and returns nothing. In my BASIC skeleton, I try to emulate SNOBOL4's approach, and always report success (-1) or failure in VR%(0). The actual return value (if there is one) is saved in another VR() location.

getc() actually has to keep track of several values. In pseudocode I call them

Name	in BASIC	Description
file#	VA%(0)	the current file we've OPENed ('1' in OPEN "R", 1..]
CharPos[file#]	SY%(f,2)	space before current character in current record
RecordCount[file#]	SY!(f,1)	current 256-byte record we're reading from
Return$	VR$(0)	character returned by getchar()
Buffer$[file#]	SY$(f,1)	LINE INPUT storage for text-stream input
Result	VR%(0)	Succeed / Fail report: FAIL (0) on EOF

getc() has no arguments or "inputs", but it has 2 "outputs": the return value "Return$" in VR$(0), and the success/fail "Result" in VR%(0), which reports "0" at EOF. Everything else falls under the category of "state" variables. If we were programming in an Object-oriented Language, we might call them "properties" of the file. Alas! We're programming in old-fashioned 8-bit Microsoft BASIC, and *everything* is a global variable. The actual code for the disk byte-stream portion of getc() looks like this:

```
2006 VR%(0)= SY!(X9%,0) < SY!(X9%,2): IF NOT VR%(0) RETURN
2008 IF SY%(X9%,2) = 256 THEN X9!= SY!(X9%,1): GET X9%, X9!: SY%(X9%,2)= 0: SY!(X9%,1)= X9! + 1:
GOTO 2006
2010 X8%= SY%(X9%,2): FIELD X9%, X8% AS X9$, 1 AS VR$(0): SY%(X9%,2)= X8% + 1: SY!(X9%,0)=
SY!(X9%,0) + 1: RETURN'
```

Ugh! (In my final program versions, I remove all spaces to make it just a little faster and smaller.) Not only is Disk BASIC ugly, when it looks like this I think it's less readable than Assembly code. Might as well be using machine-language bits and bytes! So I tend to write almost everything first in my pseudocode language. Here's the same fragment in pseudocode:

```
while (Result=
 (CurrentCharCount[file#] < TotalCharCount[file#]) {
        if (CharPos[file#] == RECORD_END) {
                NextRec#= RecordCount[file#];
                GET file#, NextRec#;
                CharPos[file#]= 0;
                RecordCount[file#]++;
                }
        else {
                FIELD file#, CharPos[file#] AS Spacer$,
                 1 AS ReturnChar$;
                CharPos[file#]++;
                CurrentCharCount[file#]++;
                }
        }
```

and here it is again with the BASIC code interspersed:

```
while (Result=
```

```
(CurrentCharCount[file#] < TotalCharCount[file#]) {
/* "VR%(0)= SY!(X9%,0) < SY!(X9%,2)"                    2006    */
        if (CharPos[file#] == RECORD_END) {
        /* "IF SY%(file#,2) = 256"                      2008    */
                NextRec#= RecordCount[file#];
                /* "X9!= SY!(X9%,1)"                    2008    */
                GET file#, NextRec#;
                /* "GET X9%, X9!"                       2008    */
                CharPos[file#]= 0;
                /* "SY%(X9%,2)= 0"                      2008    */
                RecordCount[file#]++;
                /* "SY!(X9%,1)= X9! + 1"                2008    */
                }
        else {
                /* temp8= CharPos; "X8%= SY%(X9%,2)"            */
                FIELD file#, CharPos[file#] AS Spacer$,
                 1 AS ReturnChar$;
                /* "FIELD X9%, X8% AS X9$,
                 1 AS VR$(0)"                         2010    */
                CharPos[file#]++;
                /* "SY%(X9%,2)= X8% + 1"                        */
                CurrentCharCount[file#]++;
                /* "SY!(X9%,0)= X8% + 1"                        */
                }
        }
```

(Note: these are fragments of the getc() code, so BASIC temporary variables like X8% and X9% have been initialized in earlier lines.)

Why? I find that, within days of writing it, I find it impossible to remember what my BASIC code is doing, so I've gotten into the habit of

1. writing it in pseudocode first to see if I understand what I want to do and what I'm doing
2. keeping a record in a "Guide" for each subroutine and DEFined FuNction of the pseudocode name for the procedure, what BASIC line it starts on, what are its arguments, what other global variables or "constraints" affect it, an example of its usage, etc. Here's the entry for getc() in the "Skeleton Usage Guide":

   getc(VA%(0)) Read 1 char/byte from device/file# specified by VA%(0)

   Prerequisite: If reading from a disk file, it must have first been fOpen()ed.
   Usage: GOSUB 2000

```
Argument:       VA%(0) Device (-1..-3) or FD# to read from
Constraints:    SY%(4,0) aka "GetclsByteStream"      -1:    byte-stream I/O
                                                     0:     text-stream I/O
Result:         VR%(0) succeed/fail (-1/0).     Fail on EOF
Returns:        VR$(0)  one char/byte
Typical usage:  100     VA%(0)= -1 ' Read from keyboard
                200     GOSUB 2000
                        IF VR%(0) THEN
                                PRINT VR$(0);
                        ELSE
                                RETURN
Note:           Pseudocode in "Skeleton in PseudoCode"
```

3. Keeping a file of pseudocode names (like "CharPos[file#]") and the corresponding BASIC element (like "SY%(f,2)") . In the case of SKELETON/BAS, it's in the file named "PseudoCode Aliases" which you can find, along with "Skeleton Usage Guide" , "Skeleton in PseudoCode", a text listing of SKELETON/BAS, and a .dmk disk image file, at https://github.com/VinnieMan1/Software-Tools-In-BASIC

(Why all this extra work? I mean, if somebody else had written this article… wait, *nobody else would write this stuff*. But if they had, and I was reading it, I'd think, "What a lot of extra effort. Just write it in a C cross-compiler and port the resulting Z-80 code to your TRS-80!" Yup. But what I hope eventually to do is to rewrite the shell and the file-system support in Assembler, and I'll need at least this much documentation just to keep everything straight.)

In this "episode" of Software Tools in BASIC, I want to discuss the code for the fOpen() procedure, which is used by the BASIC Skeleton to open every disk file, including those redirected by stdin, stdout, and stderr. At a minimum, you need to give fOpen() a filespec (like "MYDATA/FIL:1") and a mode (either an integer from 1..6 or one of "IORGPA", corresponding to "Input", "Output," "Random", "Getc", "Putc", or "Append"). You can also optionally specify a file number from 1..15 which you want to correspond to the file you're OPENing. fOpen() will report Success or Failure in VR%(0), and if successful, will return the file number in VA%(1). The entry for fOpen() in the Skeleton Usage Guide looks like this:

```
fOpen( VA$(0),          Open disk file with Name in VA$(0),
       opt VA$(1),          Mode described as a char in set 'IORGPA'in VA$(1)
       opt VA%(1))                  or 1..6 in VA%(1)
       opt VA%(0))          file# in VA%(0)
-----------------           or system-assigned if VA%(0) is 0
Usage:          GOSUB 60500
Inputs          VA$(0)      Filename
                VA$(1) **or**  Mode,  1 of 'IORGPA' **or**
                VA%(1)              1 or1..6
                VA%(0)      Buffer #        1..15, **or**
                                    0 for system-assigned buffer #
Constraints:    Maximum # files that can be OPENed reported by FN DN%( 0)
Results:        VR%(0)      Success/Failure (-1/0)
Returns:        VR%(1)      assigned file#
Notes:          1. File may be opened in one of the following modes:
                        in VA$(1)     or in VA%(1)    Mode
                        I             1               Input
                        O             2               Output
                        R             3               Random
                        G             4               getc()
                        P             5               putc()
                        A             6               Append[not currently implemented]
                2. A disk buffer number can be indicated in VA%(0).
                        If that is left with a value of 0, fOpen() will assign
                        a buffer number.
                3. The buffer number will not be allowed to be greater than the number
                        given to the FILES?_ prompt when BASIC was booted.
                4. The final assigned file# is returned in VR%(1)
Typical Usage:  100     VA$(0)= "MYFILE/DAT:1"
                        VA$(1)= "O"
                        VA%(0)= 0 ' Have fOpen() assign FD#
                200     GOSUB 60500
                300     IF VR%(0) THEN
                                FL%= VR%(1) '
                                Save Buffer # in FL%
                        ELSE
                                RETURN '
                                File Error
Note:           Pseudocode in "Skeleton in PseudoCode"
```

fOpen() takes several arguments, reports success or failure, and returns one item. But most of its work is in what are called “side effects”: variables or states that exist outside the fOpen() procedure itself. In

this way, BASIC code is much more like Assembly than it is like Java or Python, and I find it vital to keep detailed documentation like the entry in Skeleton User's Guide, the psuedocode, and the pseudocode aliases.

fOpen() does seven things:

1. Checks if there's actually room to OPEN another file;
2. Keeps tab on the number of files that have been fOpen()ed;
3. Assigns a file number if none was specified;
4. Checks to make sure that file number hasn't already been assigned;
5. Confirms that we're OPENing it with a valid Mode;
6. Sets up and Error Exception to trap "file doesn't exist" errors; and
7. OPENs the file with the appropriate mode.

fOpen() also initializes some state variables for the file if it's OPENed in "getc" or "Append" mode. As well, it will report errors to stderr and exit if there's any problem in any of these steps.

Here's how fOpen() starts, first in pseudocode:

```
fOpen( filename$, opt mode$, opt mode#, opt file#) {
/*      VA$(0),          VA$(1),  VA%(1),   VA%(0)       */
        /* OPEN disk file with filename$ for OpenType specified either as
                a) one of "IORGPA" in mode$; or
                b) a corresponding value of 1..6 in mode#
                in FD/buffer file#.
                        If file# is set to 0, fOpen() will assign a file#.
        */
        Result= FAIL;
```

"filename$" is a required argument, in VA$(0), also known as a "filespec". It's necessary to specify at least one of either "mode$" [VA$(1)], (one of "I", "O', "R', "G", "P", or "A"), or the corresponding number 1..6 in "mode%", VA%(1). Finally, the user can either specify a file # in VA%(0) or set it to 0 to have fOpen() choose a file #. The BASIC code corresponding to this starts at line 60500:

```
 VR%(0)= 0: X9%= VA%(0)
```

"VR%(0)" is the place we report that a procedure has SUCCEEDed or FAILed. It defaults to FAIL, and only when we've successfully gone through steps 1..7 is it changed to SUCCEED. "X9%" is a temporary variable used to hold the file#.

```
if (file# < 0)
                return;
                /* only fOpen() disk files
```

```
        "IF X9% < 0 THEN RETURN"        */
```

No point in trying to OPEN a device, for example (even though some other BASICs do that).

```
        else
                ErrorFname$= "OPEN";
        /* 'ELSE VA$(5)= "OPEN"'          */
```

If we have to report an error, the function name we're reporting about is "OPEN". You'll see a number of occasions further on where we can call

```
fOpenErrorReport( ErrorFname$, ErrorType$, file#);
```

which will combine this "OPEN" with a description of the error and also the file # and report it to stderr.

That was the "setup" portion of fOpen(). Now we get to step 1: making sure that there's enough room for another OPEN disk file.

```
        if (! FilesOpen < MaxOpenFiles()) {
                "IF NOT SY%(5,0) < FN DN%( 0) THEN                      60505   */
                ErrorType$= "NO ROOM";
                /* 'VA$(6)= "NO ROOM"'          */
                fOpenErrorReport( ErrorFname$, ErrorType$, file#);
                /* "GOSUB 60951"                                  60951 <-60505  */
                return;
                /* "RETURN"                                                      */
                }
```

By the way, this is my standard: once I've written the actual BASIC code, I place it as a comment underneath the pseudocode, along with the line number where the BASIC statements are found. Line-by-line: "SY%(5,0) is a "system" array element where I store "FilesOpen", that is, the total number of files that have been OPENed by fOpen(). If that number is not less than "MaxOpenFiles()"—the number the user entered to the

```
FILES?_
```

question when starting Disk BASIC, then we can't OPEN another, so we'll have to report the error and exit. "MaxOpenFiles()" in BASIC is "FIN DN%(0)".

```
MaxOpenFiles() {        /* Return max # of OPEN files (answer to FILES?_ prompt)        */
        return PEEK( 0x408E);
        /* "DEF FN DN%( X)= PEEK( &H408E)"                                    64034  */
        }
```

In Model I Disk BASIC, the maximum number of OPEN files is stored at address &H408E. If we've already OPENed this many files, we'll call fOpenErrorReport, and it'll print on stderr that there's "NO ROOM". Then we exit fOpen().

Now on to step 2: we increment that OPEN file count.

```
        /*
                "2. INCREMENT OPEN FILE COUNT"
        */
        FilesOpen++;
        /* "SY%(5,0)= SY%(5,0) + 1                                            60510  */
```

Easy peasy. Next, let's see if we need to assign a file or "FD" ("file descriptor") number. If the caller hasn't assigned a value to the "file#" argument—ie, they've left it at 0—then fOpen() will try to assign one.

```
        if (file# == 0) {
        /* "IF NOT X9% = 0 THEN 60520"                                        60510  */
                /*
                        "3. DO WE NEED TO HAVE A FD# ASSIGNED?
                                CYCLE THRU FD#S, LOOKING FOR EMPTY SLOT"
                */
                for (s= 1 to FilesOpen; ; s++) {
                /* "FOR X8%= 1 TO SY%(5,0)"                                   60512  */
                        if (OpenType[s] == 0) {
                        /* "IF SY%(X8%,1)=0 THEN"       */
                                file#= s;
                                /* "VA%(0)= X8%"        */
                                break;
                                /* "GOTO 60520"         */
                                }
                        }
                if (file# == 0) {
                        /* was not able to assign       */
                        ErrorType$= "NO ROOM TO OPEN";
                        /* 'VA$(6)= "NO ROOM TO OPEN"'                        60514  */
```

```
                fOpenErrorReport( ErrorFname$, ErrorType$, file#);
                /* "GOSUB 60950"        */
                return;
                }
        }
```

We simply search through all the "file slots"—that is, SY%(x, 1)—for one that doesn't have an assigned "open type" or Mode. We should be able to use that one, so we now assign that slot number to the file # that had been set to 0 when fOpen() was called, VA%(0). It may not be excellent programming practice to assign a value to a procedure's argument, but it works well here. If we've searched through all the slots without finding an unused one, however, we'll set the error type to "NO ROOM TO OPEN", report it, and exit.

This is probably a good time to talk about file "slots", or the various bits of information we keep in SY() arrays for each file #.

SY%(file#,1)	OpenType[file#]	1..6	
SY%(file#,2)	CharPos[file#]	0..256	for disk
SY!(file#,0)	CurrentCharCount[file#]	0..EOF	
SY!(file#,1)	RecordCount[file#]	0..EOF	next record to-- -GET in disk getc() when 256 chars read in or after file OPENed. -PUT in disk append putc() when sector full or file to be CLOSEd.
SY!(file#,2)	TotalCharCount[file#]	0..EOF	
SY$(file#,1)	Buffer$[file#]	""..256 chars	

All SY() arrays have 15 rows, so file# can have a value from 1..15. For each file#,
"OpenType" corresponds to one of "IORGPA", the Mode with which this file was OPENed.
"CharPos" is the position of the next character within a record to getc() or append in "G" or "A" mode.
"CurrentCharCount" is the total number of characters read so far by getc(), or written out to a file in "A" mode.
"RecordCount" is the next record to GET in getc() mode after a full 256 characters have been read in, or to PUT in "A" mode when the record is full.
"TotalCharCount" is the length in bytes of the file. This is particularly significant in getc() and "A"ppend mode.
Buffer$ is a string buffer used in getc() and "A"ppend modes. It can contain an empty string or up to 256 characters.

SY%(5,0)	FilesOpen	0..FN DN%(0)	# of files fOpen()ed

fOpen() has to deal with many of these parameters for each file that it OPENs.

Back to the code. If the caller already specified a file # in VA%(0), we need to check that that "slot" isn't already in use. An easy way to do that is to see if that slot already contains a valid "mode" or "OpenType" value, obviously set by a previous invocation of fOpen().

```
        /*
          "4. DOES FD# MATCH AN ALREADY-OPEN FILE?
        */
        if IsValidMode( OpenType[file#]) {
        /* "IF FN DF%( SY%(VA%(0),1)) THEN"                                  60520  */
                ErrorType$= "ALREADY IN USE";
                fOpenErrorReport( ErrorFname$, ErrorType$, file#);
                /* 'VA$(6)= "ALREADY IN USE": GOSUB 60950: RETURN'                 */
                return;
                }
```

And IsValidMode(), which simply confirms that its argument is in the range 1..6, looks like this:

```
IsValidOpenMode( mode#) {      /* T if mode# is a subscript of "IORGPA" [1..6]      */
        return (mode# > 0) && (mode# < 7);      /* ie, 1..6                  64038  */
        /* "DEF FN DF%( M%)= M% > 0 AND M% < 7"     */
        }
```

Next, we check if the caller has specified a Mode "code" (1..6) in "mode#". If not, we try to convert the Mode "character" (one of "IORGPA") into a "code". Then we'll confirm that it's a truly valid Mode (what if the caller specified "J" instead of "I", for example):

```
/*
          'CONFIRM "OPEN" MODE
                A. DOES VA%(1) CONTAIN VALID MODE?'
        */
        if !IsValidOpenMode( mode#) {
        /* "IF NOT FN DF%( VA%(1))"                                          60525  */
                mode#= INSTR( "IORGPA", toUpper( mode$));
                /* where mode$ is in set "IORGPA" becomes mode#                    */
                /* 'VA%(1)= INSTR( "IORGPA", CHR$( FN TU%( ASC( VA$(1)))))'       */
                }
        /*
                "B. IS IT VALID NOW?"
        */
        if (!IsValidOpenMode( mode#) {
```

```
        /* "IF NOT FN DF%( VA%(1))"                                    60530  */
                ErrorType$= "NOT VALID 'OPEN' MODE";
                fOpenErrorReport( ErrorFname$, ErrorType$, file#);
                /* 'VA$(6)= "NOT VALID 'OPEN' MODE": GOSUB 60950'   */
                return;
                }
```

We use another new function, "toUpper()", to ensure that if the caller specified a lowercase Mode char (one of "iorgpa"), it will still be recognized as one of "IORGPA".

```
toUpper( char) {          /* comverts all alpha ASCII char to uppercase. Returns all other
                                  ASCII unchanged.                                  64000  */
        return char + 32 * CharLowerCaseQ( char);
        /* "DEF FN TU%( C%)= C% +32 * FNCL%( C%)"                                    */
        }

CharLowerCaseQ( char) {          /* returns T (-1) if char is lowercase            64000  */
        return( char > 0x060) && (char < 0x078);
        /* "DEF FN CL%( C%)= C% > &H60 AND C% < &H78"                                */
        }
```

By this point, we've checked that everything is valid. We'll go ahead and setup our only return value as the file#.

```
Return= file#;
        /* "VR%(1)= VA%(0)"     Return value in VR%(1) is file# from VA%(0)    60532  */
```

We haven't changed the "report" value in VR%(0) from FAIL yet. We won't do that until the file has actually been OPENed and any default file parameters in SY() arrays have been set. Now, before we actually OPEN the file, we better set up some error-trapping in case a non-existent file was specified for an Input or Getc operation.

```
        /*
          "6. SET UP EXCEPTION IF FILE DOESN'T EXIST"
        */
        ErrorTrap( 65500);
        /* "ON ERROR GOTO 65500"                                       60534  */
```

Here's the ERROR code at line 65500:

```
65499 '
```

```
     * FILE NOT FOUND ERROR HANDLER *

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

"FN E4(ERR)" returns the actual Error Code as listed in the TRS-DOS 2.3 manual.

```
64020 DEF FN E4%( E%)= E% / 2 + 1' ERROR CODE
```

If the Error Code is 54, then we can't Find The File, and we flag this in VR%(0). Back in the BASIC code which tried to OPEN the file, the next line ("RESUME NEXT") will immediately send us to the appropriate error handler. For example:

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

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

If the file named in VA$(0) is OPENed normally, we GOTO 60700 and carry on. If the Error was 54, "File Not Found", RESUME NEXT will bring us to line 60615, which sends us to 60690:

```
'
     REPORT OPEN MODE ERROR

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


60949 '
     FILE OPEN ERROR RPT

60950 SY%(5,0)=SY%(5,0)-1
60951 VA$(10)="ERROR: '"+VA$(0)+"'-"+VA$(5)+" "+VA$(6)+" IN"+STR$(X9%)
60952 VR%(0)=0
60954 GOSUB65520
60958 RETURN
```

The routine at line 65520 is "putErr()", which writes to stderr.

Finally, we get to OPEN the file. This is a switch statement [ON...GOTO], based on "mode#", 1..6, held in VA%(1):

```
        /*
          "7. OPEN THE FILE"
        */
        switch( mode#) {
        /* "ON VA%(1) GOTO"                                          60540  */
                case 1:
                        /* '7A. OPEN "I"'          */
                        OPEN "I", file#, filename$;
                        /* 'OPEN "I", VA%(0), VA$(0)'                60610  */
                        break;
                        /* "GOTO 60700"                 */
                case 2:
                        /* '7B. OPEN "O"           */
                        OPEN "O", file#, filename$;
                        /* 'OPEN "O", VA%(0), VA$(0)'                60620  */
                        break;
                        /* "GOTO 60700"                 */
                case 3:
                        /* '7C. OPEN "R"           */
                        OPEN "R", file#, filename$;
                        /* 'OPEN "R", VA%(0), VA$(0)'                60630  */
                        break;
                        /* "GOTO 60700"                 */
                case 4:
                        /* '7D. OPEN "G"           */
                        if GetcIsByteStream
                        /* "IF SY%(4,0) THEN"                        60640  */
                                OPEN "R", file#, filename$;
                                /* "60630" (see case 3: above)   */
                        else
                                OPEN "I", file#, filename$;
                                /* "60610" (see case 1: above)   */
                        break;
                case 5:
                        /* '7E. OPEN "P"           */
                        OPEN "O", file#, filename$;
                        /* "GOTO 60620"                                      60650  */
                        break;
                case 6:
```

```
                /* '7F. OPEN "A"'          */
                AppendOpen( file#);
                if (!Result) {
                        fOpenErrorReport( ErrorFname$,
                                "'A'PPEND MODE: CAN'T OPEN", file#);
                return;
        default:
                ErrorType$= STR$( mode#) +
                        "IS NOT A VALID MODE #";
                fOpenErrorReport( ErrorFname$, ErrorType$, file#);
                return;
        }
```

First, cases 1..3 are simply the OPEN modes available in Disk BASIC: "I"nput, "O"utput, and "R"andom. Why duplicate them here?

1. So that the caller can just ask for a file# without worrying if it's already in use by stdin, stdout, or stderr.
2. So the BASIC Skeleton can keep track of the total number of files OPENed. There's a matching fClose(file#) procedure that will release the file's "slot" and decrement the number of OPEN files stored in SY%(5,0).

Here's what the BASIC code for those first three Modes looks like:

```
'

    7. OPEN THE FILE

60540 ON VA%(1) GOTO 60610, 60620, 60630, 60640, 60650, 60660: VA$(6)= STR$(VA%(1)) + " IS NOT
VALID OPEN MODE #": GOSUB60950: RETURN
60599 '
        7A. OPEN "I"

60610 OPEN "I", VA%(0), VA$(0): GOTO 60700
60615 GOTO 60690'
        7B. OPEN "O"

60620 OPEN "O", VA%(0), VA$(0): GOTO 60700
60625 GOTO 60690'
        7C. OPEN "R"

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

First, about the ON..GOTO in line 60540, where we switch destinations according to the Mode code in VA%(1)—you notice that there are 6 possible destinations, and then an "ErrorType" message assigned to VA$(6) and a call to fOpenErrorReport() with GOSUB 60950, before exiting. If by chance VA%(1) doesn't contain a value from 1..6, this will be executed instead of trying to OPEN the file.

For each Mode type (1..3 / "I", "O", "R"), if there are no errors OPENing the file, control is passed immediately to line 60700. The very next line for each Mode is "GOTO 60690". If there's a "File Not Found" error raised when trying to OPEN, the ON ERROR handler will "RESUME NEXT to this line (ie, line 60615, 60625, or 60635), and pass on to a "Can't Open" message at line 60690.

Now, to examine the three additional modes I've added to implement "Software Tools in BASIC". First, "getc()" Mode:

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

Note that a file OPENed in "G" Mode can be read in one of two ways: "byte-strream" or "text-stream". Byte-stream is just that: it reads one 256-byte sector from disk at a time into a buffer, and then reads one byte at a time from that buffer. Text-stream simply LINE INPUTs a string at a time from the file, appends a newline [Carriage Return, CHR$(13)], and then feeds one character at a time from that string. By default, byte-stream getc() is used. Here's the BASIC code corresponding to the above:

```
        7D. OPEN "G"

60640 IF SY%(4,0) THEN 60630 ELSE 60610 ' BYTE STREAM:"R"; TEXT STREAM: "I"
```

In other words, 'if "byte stream", OPEN "R"; otherwise, for "text stream", OPEN "I".'

For "putc()" Mode, we simply

```
        7E. OPEN "P"

60650 GOTO 60620 ' USE "O"
```

We OPEN it for "Output" in line 60620.

Finally, we have the option to OPEN a file in "Append Mode". For example, if the user typed this line into the shell:

COPY >>EXISTING/FIL

it will be possible for them to *add* more to the end of "EXISTING/FIL". This is not natively supported in TRSDOS 2.3 or Disk BASIC (but if I recall correctly, at least one of the later OSes like "LDOS" did). In the next installment of "Software Tools in BASIC", I'll detail how we can do this with the rudimentary tools of TRSDOS 2.3.

TRS-80

Repairs & Upgrades

HNC qualified electronics engineer
with 40 years TRS-80 experience.
Enquiries to: trs80@shanefoster.co.uk

AT THE
READY
PROMPT



JUST A QUICKIE, AS THERE'S LITTLE ROOM TO SPARE.
I'VE JUST DISCOVERED THAT YOU CAN STEP RANDOMLY WITHIN A FOR-NEXT LOOP!
WHY WOULD YOU WOULD EVER WANT TO I HEAR YOU ASK.
WELL, I JUST DID!!
INTERESTINGLY, THE STEP IS NOT RE-RANDOMISED AT EACH RUN OF THE LOOP!

```
>RUN
 1  9  17  25  33  41  49
READY
>RUN
 1  6  11  16  21  26  31  36  41  46
READY
>LIST
10 RANDOM:FORZ=1TO50STEPRND(9):PRINTZ;:NEXT
READY
>_
```



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

Thirsting for adventure?
Brass lantern getting dusty?



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

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



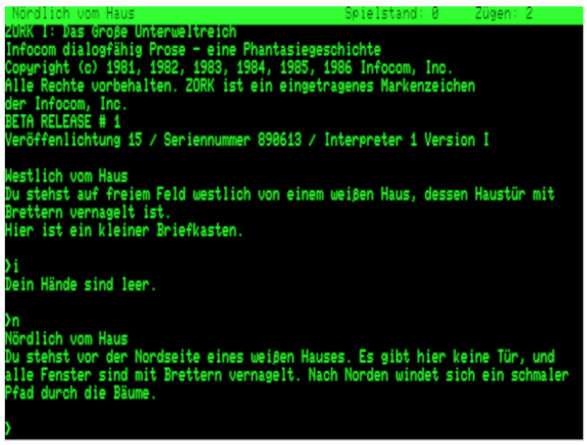
M4ZVM is available for download from https://gitlab.com/sijnstral/m4zvm (includes feature details and source)
Enhanced versions ready for download!

*128K recommended

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



Beyond Zork



Zork 1 (German)

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

How it happened...

It all started, as most quests except those of Hobbits and some dragons do, on Tuesday - Mondays are after all too busy with other things, and Wednesdays just too middle-of-the-week. But the objective was to find a Cambridge Z88. I know, it's true, you couldn't actually find a working one of these in 1987, so there was really no hope in 2020.

Even so, having had one back then which I used extensively until the inevitable early demise occurred, I woke up that Tuesday morning, having decided for some reason in the middle of the night, it was time I got another.

This, via a not even circuitous route, is how I ended up with a TRS-80 Model 100. Absent a working Z88 anywhere on eBay, the abundance of Model 100s was quite noticeable and, I thought, how different could these two machines really be?

Well, obviously, one of them doesn't work and never did, and the other sold about 6 million, of which all but three are still in use. That's how different. But I reasoned why not a Model 100, which I could play with for a bit, until I found the Z88-grail at the end of a rainbow.

Up until that moment, I could count the number of computers I owned on the fingers of one hand, and of those, the oldest would not yet have graduated kindergarten.

The 100 arrived, and Hobbits or not, the quest for vintage computing had begun, simply because it wasn't an artifact of some kind, but actually quite usable. And what instantly made it all the more certain it was the right thing to buy, was that as I unpacked the 100, , in which the eBay seller had inadvertently left batteries, I accidentally powered it up and saw it was ready for use before I had even set it on the table.

As omens go, this was a good one.

The idea was that I would use the 100 to re-familiarise myself with programming in BASIC, a language I had first used and roundly cursed in 1978, when I was exposed to it on a Model 1 (16k, Level 2, thanks for asking).

Somewhat surprisingly, I found myself transfixed by the 100, though not because of BASIC. I wrote some programs and was

quite pleased with myself for that, but the 100 had a surprise waiting for me - it was actually quite useful for work too.

At the time, I was IT manager for a law firm, and wrote a lot of instructional materials and proposals, and the 100's simple text editor proved a great way to get the first drafts done, not least because it was distraction free, and I could use it anywhere. For hours on AA batteries. Seriously, in modern times we have lost the plot quite badly!

But suddenly, the idea of a vintage computer being an actual productivity tool in my modern world took shape, and the next obvious choice was my always-wanted/never-actually-achieved 4P. Back in the early 1980s, these were not common in the UK, and I had drooled over one in the local Tandy store in Manchester enough times that the salesman would point me over to it whenever I walked in the door.

And I did walk in the door quite often. Often enough in fact that the manager asked one day if I wanted a job there - which would have been tempting if the staff discount had made that 4P more sensibly affordable.

It is a modern miracle that eBay makes all these old things new again, though when my 4P eventually arrived, looking rather yellowed and quite fragile, it didn't appear or even work like new. It did power on, which was all I expected, and once every five or six times, it would boot, so was quite a tease, but it was, after almost 40 years of waiting, *my* 4P after all, and that's what mattered.

A FreHD, two replacement floppy drives, and a new power supply made it not just the drool-worthy shop-dweller of the 80s, but my very own holy grail. It is simplicity and excellence in engineering, and a reminder of what the micro-computing revolution was really all about.

And at about 29 pounds and a set up time of 10 minutes once the dining room table is free to use, a joyful experience to use - assuming I have the strength to carry it from its back room repose.

But that is, in fact, the entire point. Certainly the whole business of computing passed Radio Shack/Tandy quite thoroughly by, and there is much to be said for the DOS boxes and Windows systems which came after, but for the real experience of how computing came out of the punch card and

tape drive era of lab-coated high priests in air-scrubbed rooms, and arrived on the desk and tabletops of the rest of us, there is nothing to narrate that journey quite like a TRS-80 can, and when you get one, actually does.

In my case, I rather lost my way thanks to an original PowerBook I had mostly forgotten I owned, and which then sent me off on a tangent gathering 68K Macs and a few classics like a Compaq 'lunchbox' 286, which can certainly run rings around the 4P in terms of what it can do, but lacks every ounce of soul and purpose the Radio Shack machine can bring - quite literally - to the table.

Now, my collection of computers numbers in the dozens. The quite-a-few dozens, in fact, and I can't say I could recount them all from where I'm sitting right now on the couch, though admittedly I *can* see what I think is 27 of them, without having to move a muscle.

That doesn't even include the two Raspberry Pis, which host MacOS 7.1 and 7.5, or the actual, *working* Cambridge Z88, which came from the UK of course and I keep in the damp and chill of the family room downstairs, because, well, it's British so that climate is familiar to it, and it does seem to rather quite like it, though it crashes quite often too of course.

But, the system I'm liking a great deal, and which makes me happy I started this journey into old computers a few years ago, because it's the coolest thing since a January day on the Matterhorn, is my Tandy 200.

Not that Tandy (by then) meant it to be so, but it is the most unapologetically perfect example of how micro-computing built the modern world that there ever was. It is simple, functional, does the exact job it was meant to do, and makes you wonder why it is that all computers were not actually built this way.

It would have been so much better if they had been. The testament to this is all those who have engineered modern solutions to using them. It is impossible not to be rather humbled.







Have you put a CLIP in your Winchester?

CLIP — Compressed Library Interchange Program

CP/M CP/M-86 PC DOS MS DOS £105.00

- **Backs up a Winchester on to floppies.**
- **Compresses text or data to less than half size.**
- **Large files can span multiple discs.**
- **Selective backup or retrieval, on an inclusive or exclusive basis.**

CLIP has no equal in reputation, convenience, power or economy.

EFFORTLESS BACKUP
You can save commands for later use, or type commands directly.

CLIP has its own menu, with seven prepared commands. Customise the commands if you wish, or edit the menu text with a word processor. Or keep the standard forms. All your housekeeping — save/restore/review — by pressing two keys.

CLIP comes standard with Winchester systems supplied by Olympia, Cifer Systems, Research Machines, Philips and British Telecom. And is highly recommended by other major manufacturers

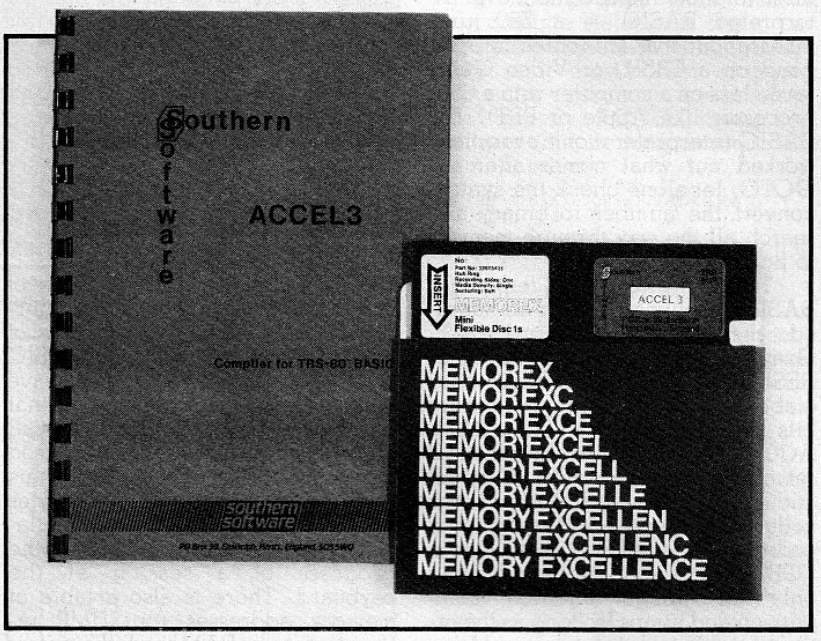
All prices excl. VAT, post free in U.K.
Most popular disc formats from stock.

KEELE CODES LTD
University of Keele, Keele, Staffordshire, U.K. Tel: (0782) 629221 Telex: 36113

COMPILERS

Simon Goodwin

After last month's piece on interpreters we take a look at the other method of getting BASIC programs into a form that can be used by the micro.



In the last issue we explained some of the features of compilers for BASIC personal computers. This month we will look at one ingenious commercial product — the ACCEL2 compiler for TRS 80 and Video Genie. This program demonstrates many of the strengths and weaknesses of compilers in general. We're grateful to Southern Software, the UK authors, for permission to describe some of its inner workings as they illustrate many of the points that a user should consider when attempting to enhance the features of almost any BASIC micro. There is no reason why a compiler along the lines of ACCEL2 could not be written for any of the popular computers.

Briefly, a compiler is a program that translates BASIC statements from the 'tokens' in memory (eg FOR, GOTO, etc) into machine code. Once a program has been translated it is much more difficult to modify than when it was 'interpreted' but it can be run ten or more times faster!

Loading And Using ACCEL2

The compiler was supplied on digital cassette, accompanied by a strikingly-bound 18-page manual (it is compatible with disc systems). Two identical copies of the program are recorded on the tape, which is loaded using the Level 2 BASIC SYSTEM command since it is a machine-code routine. When first run, the message "TARGET ADDR?" appears. This feature allows you to store the compiler anywhere in your computer's RAM. The program will automatically move itself to the target address that you specify in answer to the prompt. That means it is compatible with all 'memory sizes' (so long as there is enough room for it to load in the first place). Normally you should reserve an area of memory out of reach of BASIC when the machine is switched on. The program doesn't check that the memory address specified contains free memory but only the most confused programmer would consider that a problem. The manual contains a useful table of suggested storage areas for different systems.

The ACCEL2 compiler is remarkably small — it only occupies 5½K of user RAM and loads in less than two minutes. About three-quarters of it can be deleted once compilation has taken place (most of the program does the translating rather than support the operation of the final compiled BASIC program). Once the compiler is in memory, you may load or type in any BASIC program that will fit into your remaining RAM. You may even RUN the program in the normal way and it will execute as if the compiler was not there. When you think there are no more errors in the program, you may compile it by typing the new command /FIX. A 14K program should compile in about two minutes. There is no way to interrupt the compiler while it is working; if you press the Reset button before the prompt re-appears, you will have to reload both programs before you can get any sense out of your system.

How It Works

The compiler works by looking through the BASIC program in memory and trying to find instructions that could easily be performed in machine code. It would not be possible to fit a full-feature compiler into only 5632 bytes of Z80 machine code, so ACCEL2 sets out to translate only those functions that can be enhanced most dramatically by compilation. It relies upon the fact that once the program is compiled, it cannot be edited. That means that most of the searches carried out by the interpreter can be replaced by simple instructions that fetch data directly from some pre-calculated memory address. These addresses are worked out by ACCEL2 while compilation takes place and are stored immediately inside the compiled program. The Z80 processor used in Video Genie can only directly process numbers in the range — 32768/+32767 (the range that can be stored in one of its 16 bit registers). Integer (whole number) values on most mini and micro-computers fall into this range and are useful as general-purpose stores in programs. Loop counters, graphics co-ordinates and 'flags' are generally integer values and Apple, Tandy and PET computers all support special 'integer variables'.

The Z80 instruction set includes integer addition and subtraction but not multiplication, division or complex functions such as square-root. These must be performed using a large number of machine code instructions and consequently ACCEL2 does not directly compile

them. All GOTOs and other line number branches can be compiled since the target of a branch is static. ACCEL2 replaces those instructions by machine code JUMPs. This is a most valuable improvement over interpreted BASIC — a Z80 jump takes about five millionths of a second on a TRS80 or Video Genie (even less on a computer with a 6502 processor like Apple or PET). The BASIC interpreter won't even have worked out what comes after the GOTO, let alone check the syntax, convert the number to binary and search all the way through memory to find the line required!

There are quite a few other BASIC functions that can be handled directly by the processor. Graphics commands such as SET, RESET and POINT are really only elaborate ways of setting or clearing bits in the computer's video RAM. ACCEL2 replaces them by calls to a set of simple subroutines. GOSUB and RETURN have direct machine code equivalents, and very little code is needed to simulate a FOR...NEXT loop since it usually only uses integer addition, comparison and jumps (so long as the index variable is an integer — and this is usually the case). PEEK and POKE are simply compiled and constants (such as "FRED", -1000, 3.14, etc) are stored as binary values.

As well as translating those statements into machine code, ACCEL2 is faced with a large number of other more complex operations: floating-point calculations; string-handling; and so forth. These are compiled in a rather different way. The data to be processed is fetched as detailed above but rather than operate upon it directly, ACCEL2 calls up routines within the ROM interpreter to actually work out the results. Even though it is the same code 'doing the work', the BASIC interpreter normally spends so much time searching for data that ACCEL2 can usually speed up those functions by three or four times. Most functions that are very complex (such as SIN and all the varieties of PRINT) are left in their interpreted form, keeping the size and complexity of the compiler down. You may choose to compile only a part of your program at a time and ACCEL2 will automatically take over from the interpreter when it comes to a compiled statement, relinquishing control later. It is this feature that makes ACCEL2 interesting to the programmer since it permits a compiler to be written piecemeal, one function at a time. In fact ACCEL2 was written in this way — the original 'ACCEL' being a 2¾K program that only compiled branches and most integer operations. It is still available (for half the price of ACCEL2) and is ideal for games and similar programs.

ACCEL2 converts compiled statements into REMs followed by machine code. To prevent confusion and to reduce the size of the compiled program, genuine comments are removed from the BASIC before it is compiled. Despite this, compiled programs are usually larger than their interpreted equivalents.

To accomplish the handover of control between compiler and interpreter, a very valuable feature of Microsoft (and most other) BASIC interpreters is used. You may have been disappointed to discover that on your PET, TRS 80 or Apple not all of the internal memory can be used to store BASIC programs. The interpreter borrows a few hundred bytes to store partial results of calculations and various other notes on the progress of a session at the keyboard. There is also a table of memory addresses (or JUMP instructions) in RAM and the interpreter calls at those locations at various key points in the execution of a program. There are about fifty 'vectors' in reserved memory on a Video Genie or TRS 80, and two of them are of special interest to compiler authors since they are used just before each direct command is processed and before statements are executed. On a Video Genie or TRS 80 (Model 1 or 3) these vectors are stored at 41B2 Hex and 41C4 Hex — other interpreters will have equivalent vectors at different addresses.

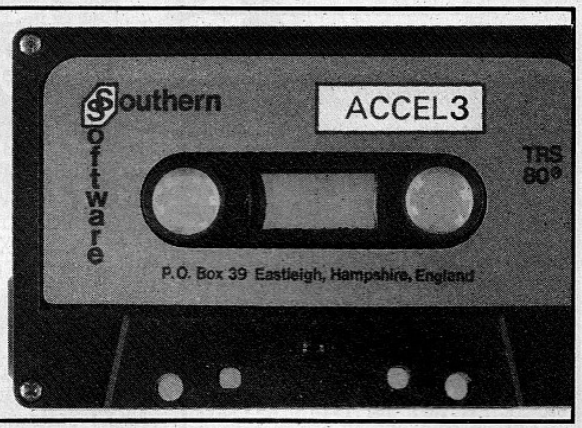
Normally the vector locations contain nothing more than a jump back into the depths of the BASIC ROM but if a machine code program wants to take over at times, it need merely replace the vector in RAM with one pointing to its own routines. It can check what's going on when it is called up and either jump back to ROM if it doesn't want to interfere or process the statement in its own way and then re-enter the interpreter at the point where the next instruction is fetched. Extended BASICs and 'Toolkits' use these vectors to add to the commands on a computer — once the relevant locations have been found, a user can expand the system software of his computer little by little; there are books available describing the workings of most of the popular microcomputer interpreters. My recommendation for TRS 80/Video Genie users is 'Pathways through the ROM' distributed by The Softwarehouse.



The latest version of Southern Software's ACCEL program is ACCEL 3. Available on tape or disc it is accompanied by a startling orange manual, definitely not the thing for sore eyes!

To ensure that ACCEL2 will use the same variable storage area as BASIC, it checks through memory during compilation working out where each variable will be stored and where each line of the program will eventually end up. To make the compiled program exactly compatible with normal BASIC, ACCEL2 has to do a certain amount of 'housekeeping' as it runs. For this reason the compiled program is usually somewhat larger than the original version. To try to minimise that effect, ACCEL2 removes comments and unneccessary spaces from the BASIC before compiling it. ACCEL2 will issue an OUT OF MEMORY error if the compiled program ends up too large to fit in your computer. Often you can get around this by only compiling the part of the program that is executed most often. This keeps code expansion to a minimum and with care, the mixed code will run almost as fast as if it were all compiled. After such an error you have to reload a copy of the original 'source' BASIC entered using the interpreter; you cannot edit the partially-compiled program to cut it down.

ACCEL2 has facilities to allow a compiled program to load another from disc. It automatically looks for a compiled version first and then for an interpreted one if the first search fails. That feature allows a disc user to compile a set of linked programs one at a time (without having to recompile at each stage). The new disc commands are /LOAD, /SAVE and /RUN.



The cassette version of ACCEL is totally relocateable, a useful facility allowing you to have a variety of other programming or editing aids resident in memory at the same time.

Example Program

Figure 1 shows a simple program before and after compilation. The '%' signs indicate that X and Y are integer variables. These should be used whenever possible in compiled programs since they can be processed much faster than normal floating-point variables. It is not very valuable to try to compare the speed of individual instructions when interpreted and compiled, since the larger the program the longer the interpreter spends finding a given variable or line. A compiler will take a long time to translate such a program as it is doing at one step all the searching and converting that the interpreter does bit by bit (and generally over and over again!). This tiny example was compiled by ACCEL2 in less than a second and ended up using rather more memory than the original — 169 bytes instead of 126. The routine simply turns the entire computer display white by individually turning on each of 6144 graphics pixels. Interpreted BASIC took 36.5 seconds to execute the program; after compilation it ran in just five seconds. This speed improvement of around seven times is obviously not typical since most programs will use complex functions such as PRINT and decimal arithmetic. There again, ACCEL2 is at its least impressive when compiling small programs, and most programs will be accelerated by at least a factor of two or three times if they are compiled.

The compiled program may look rather odd; the original REM has been removed from line 100 but the line number alone is left there in case it is referenced by other parts of the program. The REMs conceal the compiled machine codes; ACCEL2 deliberately prevents the machine code from being listed (it wouldn't make sense, either to you or to the LIST routine . . .). END is never compiled so it appears unchanged.

```
100  REM ** ACCEL2 DEMO PROGRAM
110  FOR X%=0 TO 127
120  FOR Y%=0 TO 47
130  SET (X%,Y%)
140  NEXT Y%
150  NEXT X%
160  END

(126 BYTES, RUNS IN 36.5 SECONDS)

100
110  REM
120  REM
130  REM
140  REM
150  REM
160  END

(169 BYTES, RUNS IN 5.0 SECONDS)
```

Fig. 1. The example program before and after compilation by ACCEL 2.

Compiler Restrictions

ACCEL2 cannot compile array references with more than one dimension but this is not normally a major restriction. The size of an array must be known at compilation time (10 INPUT A 20 DIM CD(A) is not legal) but this can be avoided by always dimensioning arrays to the largest size required. ACCEL2 will be of little use in programs limited by the speed of peripherals such as disc, tape or printer. A version of Conway's game of LIFE has been written and found to be 39 times faster under ACCEL (the original 2¾K subset of ACCEL2) than when the BASIC interpreter alone was used. However, the program was written especially to be compiled and consequently a slightly faster interpreted routine could probably be found.

READ and DATA statements are not compiled since ACCEL2 cannot tell whether DATA is going to be stored in string or numeric format when the program is run. READ and DATA can be invariably replaced by assignment statements so this is not a major problem. ACCEL2 will only compile statements that it can translate completely. Although SIN, TAN and most other complex arithmetic operations are not compiled directly, ACCEL2 uses a neat dodge to compile comparisons using them. The line:

```
300  IF SIN(X)=SQR(X) THEN 100
     ELSE 200
```

is compiled into:

```
300  TO%=SIN(X)=SQR(X):REM
```

(That first statement is perfectly legal interpreted BASIC!) The compiler uses TO% as a temporary variable to store the result of the comparison (try PRINT 2=2 and PRINT 2=1 on your computer). The compiled code behind the REM tests the value of TO%; it will be zero if the values are not equal, −1 if they are. TO% is not a legal user variable anyway because the BASIC editor treats it as a reference to the reserved word TO.

Most compilers generate a good many problems as well as advantages by comparison with an interpreter. ACCEL2 is a special case since it remains dependent upon the computer's ROM interpreter. It is possible to test a program 'in slow motion' with the interpreter and the usual BASIC debugging aids, and then to compile the program when it is (hopefully) more or less free of errors. This is important since there is not much room for error-trapping code in the 1280 bytes of run-time routines. After ACCEL2 has been used you cannot edit the program, delete lines or change the text in any way. If you try to do so, the interpreter will 'fall over' the machine code generated by ACCEL2 and the system will probably re-boot. You may still set or print variable values in immediate mode and then GOTO the start or middle of the program to test specific routines. It can be risky to GOTO lines in the middle of the program since ACCEL2 will not recognise an accidental 'Return without Gosub' error and will probably jump away to some indeterminate location quite possibly crashing in the process. GOSUB and REM are not allowed as immediate commands since ACCEL2 uses a different type of GOSUB from BASIC and it uses REM to signify compiled code. When the machine

holds a compiled program, you should not use the commands EDIT, AUTO, CLOAD?, CSAVE, DELETE, MERGE and SAVE, since they assume that the program is in interpreted format.

The compiler gains some of its extra speed by dispensing with part of the 'housekeeping' done by the interpreter; it doesn't check the current line number while executing compiled code so ON ERROR GOTO may not go where you want it to! Likewise TRACE will only display the line numbers of statements that have not been compiled. ACCEL2 doesn't check the 'Break' key while it executes compiled lines; if you 'get stuck' in a compiled loop you will have to use Reset to get out.

We were only able to find one minor bug in the purchased compiler; if INKEY$ was followed by certain statements a cumulative Out of Memory error could develop. However, this problem has been fixed in the current issue of the compiler.

ACCEL2 imposes a number of subtle restrictions upon the programmer. Lazy BASIC programmers have been known to write code that jumps out of a loop without terminating it, as in Fig. 2. This may fail under ACCEL2 since the compiler never realises that the loop has been terminated. The code in Fig. 3 works correctly whether interpreted or compiled and as BASIC on the Apple 2 also requires this construct, the limitation is quite a reasonable one. The program will also fail if the default type of variable is changed (eg from integer to string). This can sometimes happen by accident when Microsoft 12K BASIC is being used since variables are assumed to be floating-point until declared otherwise. Arrays should be dimensioned at the start of our program.

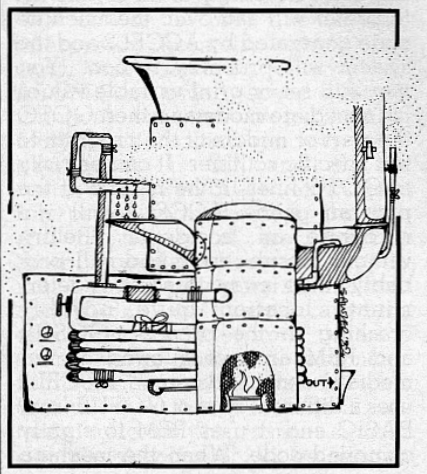
```
10  REM ** ACCEL2 COMPILER
    RESTRICTION
20  GOSUB 50
30  PRINT "RETURNED"
40  STOP
50  FOR I=1 TO 10
60  IF I>X THEN RETURN
70  NEXT I
80  RETURN
```

Fig. 2. A lazy programmers loop which night fail under compilation.

```
10  REM ** ACCEL2 REVISED LOOP
    CODING
20  GOSUB 50
30  PRINT "RETURNED"
40  STOP
50  FOR I=1 TO 10
60  IF I>X THEN J=I:I=10:REM ** SAVE
    VALUE OF I BEFORE RETURN
70  NEXT I
80  RETURN
```

Fig. 3. The correct way to do it for both interpreted and compiled programs.

With the exception of the INKEY$ bus, these restrictions are clearly listed in the ACCEL2 manual. Generally the compiler will work faultlessly on programs that have been written with it in mind and the effort needed to conform with its idiosyncracies is not great. A machine code monitor is required to save compiled programs on tape since they are a mixture of BASIC and machine code, and must be saved along with the 1280 byte ACCEL2 run-time routines. If this is done a compiled program becomes a self-contained file that can be loaded from SYSTEM and then RUN as if it were high-speed BASIC. If no monitor is available then all of ACCEL2 must be loaded whenever a compiled program is required and each program has to be re-compiled from the source before use.



Conclusions

ACCEL2 is a British development that illustrates a fascinating system of 'selective compilation'. The same techniques could be applied to almost any BASIC microcomputer and in fact, an even more powerful compiler could be developed step by step. Program compilation seems certain to become a popular technique in the future — ACCEL2 demonstrates an ingenious approach that combines many of the best points of compilers and interpreters.

Postscript

Since ACCEL2 was reviewed we have received a pre-release copy of the latest Southern Software compiler, imaginatively named ACCEL3. The new program is apparently a complete re-write of ACCEL2 and incorporates some new features.

ACCEL3 will compile non-structured FOR...NEXT loops making it possible to compile programs containing jumps out of loops, conditional NEXT statements and so forth. ACCEL2 didn't do this often leading to changes being made to a program before it could be compiled. The snag is that the extra code to handle unstructured loops slows up compiled programs — FOR... NEXT statements used with the new compiler are about half as fast as they were under ACCEL2. Similarly ACCEL3 now compiles references to arrays with more than one dimension but the speed of access to one-dimensional arrays has suffered.

ACCEL3 compiles some functions that ACCEL2 used to leave for the BASIC interpreter to handle. In particular, floating-point FOR... NEXT loops and functions such as INT and SQR are now compiled into ROM calls. The compilation of the functions (eg SIN, etc) doesn't really speed them up since they take much longer to process than to interpret but it does mean that expressions using them can be compiled. This would speed up the multiplication in X=SIN(X)*3.1416, for example. The USR(n) function, used to call a machine-code routine, is no longer compiled. ACCEL3 will also compile programs which use variable-bound arrays, such as – 20 INPUT N : DIM A$(N,2).

The ACCEL3 sales literature claims it is faster and generates more compact code than ACCEL2, but the difference in performance does not seem to be that great; the new compiler no longer has the disc commands /SAVE, /RUN and /LOAD. ACCEL3 allows compiled programs to be SAVEd, RUN and LOADed just as if they were normal BASIC although they will not work unless the run-time routines of ACCEL3 are in memory. Even the cassette commands, CSAVE and CLOAD, can now be used to store and retrieve compiled programs.

The pre-release version of the ACCEL3 compiler has been tested using it to speed up a few well-known programs — it even found one or two unnoticed syntax errors! More than half of the programs compiled first time and most of the rest could be compiled once a few lines were shortened or expressions simplified.

TRS-80

Title	Program Type	Program Produced By	How Sold		Supplier Codes	Prices
Ascertain	Ut	Kansas	16K	C	IH	£6.25
Assembly Package	Ut	Molimerx	32K	D	AJ	£74.75
Asteroids	Gm	Molimerx	16K	C	AJ	£10.06
Asteroids	Gm	Molimerx	32K	D	AJ	£13.50
Astro Navigator	Gm	Molimerx	16K	C	AJ	£14.95
Astroball	Gm	Molimerx	16K	C	AJ	£13.77
Astrolog	Ed	Molimerx	16K	C	AJ	£10.93
Astronomical Calendar	Ed	Molimerx	16K	C	AJ	£10.93
Asylum	Gm	Med Systems	16K	C	IS	£9.50
					BT	£12.00
Atlantian Odyssey	Gm	Guild	16K	C	IS	£11.50
					GB	£19.75
Atlantian Odyssey	Gm	Guild		D	GB	£21.50
					GB,BT	£10.95
Attack Force	Gm	Big 5 S/W	16K	C	IS	£11.50
					OH	£11.25
BI Nuclear Bomber	Gm	Instant S/W	16K	C	GB	£12.50
					BT	£12.95
					OH	£11.50
BQ81	Bs	Baust Computing	48K	D	HA	£79.95
Back 40	Gm	Molimerx	16K	C	AJ	£10.06
Back 40	Gm	Molimerx	32K	D	AJ	£13.50
Back 40 III	Gm	Adventure Int.	16K	C	AB	£14.35
Backgammon	Gm	Adventure Int.	16K	C	IS	£7.50
					AB	£7.99
Backgammon	Gm	Kansas	16K	C	IH	£8.50
Balrog Sampler	Gm	Adventure Int.	16K	D	AB	£33.70
Bandito	Gm	Acorn	16K	C	AJ	£7.49
Barricade	Gm	MA	16K	C	IS	7.50
Bandit/Maze Chase	Gm	Kansas	16K	C	IH	£7.50
Banner	Ut	Kansas	16K	C	IH	£7.50
Basic Bidding	Gm	Bryants S/W	16K	C	HW	£9.00
Basic Compiler Accel.	Ut	Microsoft	16K	C	AJ	£22.94
Basic Compiler Accel. 3	Ut	Microsoft	16K	C	AJ	£45.95
Basic Compiler Accel. 3	Ut	Microsoft	16K	D	AJ	£49.95
Basic Insert	Ut	Molimerx	16K	C	AJ	£10.95
Basic Insert	Ut	Molimerx	32K	D	AJ	£14.39
Basic Programming System	Ut	Instant S/W	16K	C	AB	£12.50
Basketball	Gm	Molimerx	16K	C	AJ	£10.95
Basketball	Gm	Molimerx	32K	D	AJ	£14.39
Battle Bastogne	Gm	MA	16K	C	IS	10.50
Battle St. Vith	Gm	MA	16K	C	IS	10.50
Battle of Britain	Gm	Molimerx	16K	C	AJ	£15.55
Battle Ground	Gm	Instant S/W	16K	C	GB,AB	£8.50
Bearings	Ed	Bryants S/W	16K	C	HW	£3.75
Bedchest III	Gm	Adventure Int.	16K	C	AB	£20.95
Beewary	Gm	Molimerx	16K	C	IS	£7.50
					AJ	£10.06
Beginner's Russian	Ed	Instant S/W	16K	C	GB,AB	£8.50
Biorhythm	Ed	Molimerx	16K	C	AJ	£8.63
Biorhythms	Do	Kansas	16K	C	IH	£7.50
B-I Nuclear Bomber	Gm	Avalon Hill	16K	C	IA	£11.95
Blackboard	Ut	Molimerx	16K	C	AJ	£10.06
Blackjack Master	Gm	Molimerx	16K	C	AJ	£21.15
Blackjack Master	Gm	Molimerx	32K	D	AJ	£25.42
Blink 1	Ut	Molimerx	32K	D	AJ	£17.00
Blockade	Gm	Molimerx	16K	C	AJ	£7.48
Bradley Estate Mystery	Gm	Ogre	16K	C	IC	£10.00
Bridge Partner	Gm	MA	16K	C	IS	£13.50
Bridge 2.0	Gm	MA	16K	C	IS	£13.50
Bright & Early	Ed	Molimerx	16K	C	AJ	£8.90
Build High						
Knights Tour	Gm	Kansas	16K	C	IH	£7.50
Business Analysis	Bs	Instant S/W	16K	C	GB	£59.25
Business Analysis	Bs	Instant S/W	16K	D	GB,AB	£82.00
Buzzword	Ed	Molimerx	16K	C	AJ	£5.75
Byte Saver	Ut	Kansas	16K	C	IH	£6.25
CTN Stock	Bs	Baust Computing	48K	D	HA	£79.00
Calendar	Ut	Molimerx	16K	C	AJ	£10.29
Car Doctor	Do	Kansas	16K	C	IH	£7.50
Cash Register	Bs	Molimerx	16K	C	AJ	£10.29
Catalogue	Do	Bryants S/W	16K	C	HW	£3.75
Character and Monster Generator	Gm	Ogre	16K	C	IC	£12.00
Checker King	Gm	Personal S/W	16K	C	AJ	£16.10
Chemical Formulae	Ed	Molimerx	16K	C	AJ	£10.95
Cheque Book	Dm	Molimerx	16K	C	AJ	£15.53
Chess Partner	Gm	MA	16K	C	IS	£9.50
Claws	Ed	Bryants S/W	16K	C	HW	£3.75
City Encounters	Ut	Molimerx	16K	C	AJ	£17.20
Climate Comp	Bs	Instant S/W	16K		AB	£20.50
Cload Cassette	Gm	Cload Inc	16K	C	IA	£5.75
Cloze	Ed	Bryants S/W	16K	C	HW	£3.75
Cluster War	Gm	J.K. Gosden S/W	16K	C	LB	£11.00
Code Breaker	Gm	Molimerx	16K	C	AJ	£7.49
Combat	Gm	Adventure Int.	16K	C	AB	£12.50
Company Director	Gm	Molimerx	16K	C	AJ	£17.19

TRS-80

A-Maz-Ing	Gm	Molimerx	4K	C	AJ	£6.33
Acquire	Gm	Avalon Hill	16K	C	IA	£13.95
Adventure I	Gm	Adventure Int.	16K		AJ	£13.77
Adventure III	Gm	Adventure Int.	16K	C	AJ	£13.77
Adventure III	Gm	Adventure Int.	16K	D	AJ	£13.77
Adventures No I	Gm	Cload Inc	16K	C	IA	£11.95
Advanced Programming Aid	Ut	Kansas	16K	C	IH	£12.50
Adventureland	Gm	Adventure Int.	16K	C	IS	£12.50
					AJ	£18.99
					OH	£15.25
Adventure Mysterious	Gm	Adventure Int.	16K	C	AJ	£10.06
Adventure Mysterious	Gm	Adventure Int.	32K	D	AJ	£13.50
Aid, Basic	Ut	J.K. Gosden S/W	16K	C	LB	£9.00
Air Raid	Gm	MA	16K	C	IS	£7.50
Air Traffic Control	Gm	MA	16K	C	IS	£9.50
Airflight Simulation	Gm	Instant S/W	16K	C	GB,AB	£9.95
Airmail Pilot	Gm	Instant S/W	16K	C	GB,AB	£8.50
Ajedit	Bs	Molimerx	32K	C	AJ	£49.95
Aladdin	Ed	Bryants S/W	16K	C	HW	£3.75
Alien Armada	Gm	Molimerx	16K	C	IS	£11.50
					AJ	£10.06
Alien Armada	Gm	Molimerx	32K	D	AJ	£13.23
Alien Attack Force	Gm	Instant S/W	16K	C	GB,AB	£8.50
Alien Attack	Gm	Kansas	16K	C	IH	£9.50
Alien Defence	Gm	Soft Sector Mkting	16K	C	IS	£11.50
					GB	£15.50
Alien Defence	Gm	Soft Sector Mkting	16K	D	IA	£11.95
All Stars	Gm	Instant S/W	16K	D	GB	£26.50
					AB	£28.95
All Turret Gunner	Gm	Instant S/W	16K	C	GB,AB	£8.50
Amateur Log	Ut	Molimerx	32K	D	AJ	£18.40
Amazing Maze Touchdown	Gm	Kansas	16K	C	IH	£7.50
Angel Worms	Gm	Adventure Int.	8K	C	AB	£14.35
Animation	Ut	Molimerx	16K	C	AJ	£17.20
Animation	Ut	Molimerx	32K	D	AJ	£20.65
Android Nim	Gm	Molimerx	16K	C	IS	£7.50
					AJ	£10.06
Arcade Invaders	Gm	Kansas	16K	C	IH	£9.50
Arcade Scramble	Gm	Kansas	16K	C	IH	£9.50
Armoured Patrol	Gm	Adventure Int.	16K	C	IS	£15.00
					GB	£15.00
Armoured Patrol	Gm	Adventure Int.	16K	D	AB	£18.99
Compression Utility	Ut	Instant S/W	16K	C	GB	£15.75
					AB	£16.50
Comproc	Ut	Molimerx	16K	C	AJ	£15.80
Comput-A-Organ	Ed	Molimerx	16K	C	AJ	£6.85
Computer Acquire	Gm	Adventure Int.	16K	C	IA	£10.15
Computer Acquire	Gm	Avalon Hill	16K	C	IA	£17.25
					OH	£13.50
Computer Pools	Do	Molimerx	16K	C	AJ	£16.68
Conflict	Gm	MA	16K	C	IS	£10.50
Conflict 2500	Gm	Avalon Hill	16K	C	IA	£12.95
Consequest of Chester Wood	Gm	Adventure Int.	16K	C	AB	£18.99
Constellation	Ed	Molimerx	16K	C	AJ	£14.30
Constellation	Ed	Molimerx	32K	D	AJ	£17.83
Conv/CPM	Ut	Molimerx	32K	D	AJ	£20.70
Convoy Raider	Gm	Avalon Hill		C	IA	£10.95
					OH	£11.50
Cop 16K	Ut	Molimerx	16K	C	AJ	£12.95
Copsys	Ut	Molimerx	16K	C	AJ	£12.95
Corplan	Gm	Molimerx	16K	C	AJ	£32.79
Cosmic Fighter	Gm	Big 5 S/W		C	IA	£10.95
Cosmic Fighter	Gm	MA	16K	C	IS	£11.50
					OH	£11.25
Cosmic Patrol	Gm	Instant S/W	16K	C	GB,AB	£12.50
Cosmic Patrol	Gm	Instant S/W	16K	D	GB,AB	£16.25
Cowboys	Gm	P.C.S.	8K	C	IE	£4.25
Creole Lobster Catcher	Gm	Molimerx	16K	C	AJ	£10.06
Cribbage Player	Gm	MA	16K	C	IS	£9.50
Cribbage	Gm	Molimerx	16K	C	AJ	£14.38
Crocodiles	Ed	Bryants S/W	16K	C	HW	£3.75
Cross Reference	Bs	Molimerx	16K	C	AJ	£8.45
Crowley Manor	Gm	MA	16K	C	IS	£15.00
Crown of CWL IMDRAS part I	Gm	Ogre	16K	C	IC	£10.00
Crown of CWL IMDRAS part 2	Gm	Ogre	16K	C	IC	£10.00
Crusaders	Gm	Molimerx	16K	C	AJ	£13.51
Cube Hunt	Gm	Molimerx	16K	C	AJ	£10.35
DLDIS	Ut	Instant S/W	16K	D	GB	£15.50
					AB	£16.50
DSM	Ut	Racet Computers	32K	D	AJ	£46.00
D-Day	Gm	MA	16K	C	IS	£10.50
Danger in Orbit	Gm	Instant S/W	16K	C	AB	£12.50
Danger in Orbit	Gm	Instant S/W	16K	D	GB	£15.50
Daredevil	Gm	Instant S/W	16K	C	IA	£8.50
Dark Void	Gm	Molimerx	16K	C	AJ	£10.06
Darts	Gm	Molimerx	16K	C	AJ	£10.06

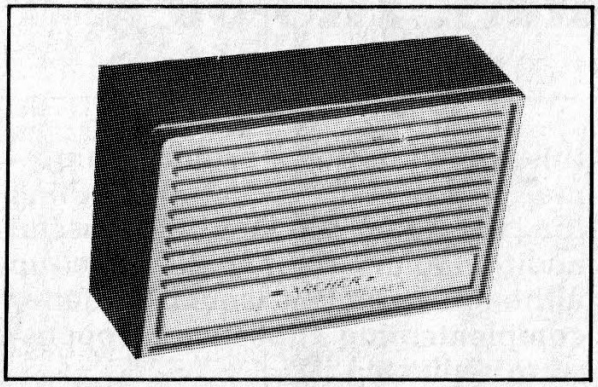
Data Base Management	Bs	Hubert Howe	32K	C	AJ	£28.75
Data Base Management	Bs	Hubert Howe	32K	D	AJ	£32.20
Datadisk	Bs	Kansas	32K	D	IH	£24.50
Data File	Bs	Kansas	16K	C	IH	£9.50
Datestone of Ryn	Gm	Automated Simulation	16K	C	GB	£13.50
Datestone of Ryn	Gm	Automated Simulation	16K	C	IS	£14.50
Dataquiz	Ut	Bryants S/W	16K	C	HW	£3.75
Day of the Week	Bs	Molimerx	4K	C	AJ	£5.75
Death Dreadnaught	Gm	MA	16K	C	IS	£10.50
Deathmaze	Gm	MA	16K	C	IS	£10.50
Deathmaze 5000	Gm	Med Systems	16K	C	IA	£10.95
Decision Master	Ut	Sytonic S/W	32K	D	GB	£35.50
Decimal Dan	Ed	Bryants S/W	16K	C	HW	£3.75
Defence Command	Gm	Big 5 S/W	16K	C	IS	£11.50
					GB,BT	£10.95
					OH	£11.25
Defend	Gm	Molimerx	16K	C	AJ,CH	£14.95
Defend	Gm	Molimerx	32K	D	AJ	£18.40
Deflections	Gm	Adventure Int.	8K	C	AB	£14.35
Delay	Ut	Kansas	16K	C	IH	£6.25
Demo II	Gm	Instant S/W	16K	C	IE	£6.50
Descriptive Statistics	Bs	Molimerx	16K	C	AJ	£15.40
Differential Equations	Bs	Molimerx	16K	C	AJ	£16.10
Direction Finder	Ed	Molimerx	16K	C	AJ	£15.40
Disassembler	Ut	Molimerx	16K	C	AJ	£10.35
Disassembler	Ut	Buy Byte	16K	C	GB	£8.25
					AB	£8.50
Diassembler	Ut	Kansas	16K	C	IH	£8.50
Disc Directory	Ut	Molimerx	32K	D	AJ	£20.13
Disc Editor	Ut	Instant S/W	16K	D	GB	£30.50
Disc Scope	Ut	Instant S/W	16K	D	GB	£15.50
Discaid	Ut	Molimerx	32K	D	AJ	£19.49
Discator	Ut	Molimerx	32K	D	AJ	£29.90
Displus	Ut	J.K. Gosden S/W	16K	C	LB	£10.00
Dnieper River Line	Gm	Avalon Hill	32K	C	IA	£12.95
Dnieper River Line	Gm	Avalon Hill	32K	D	IA	£17.55
Doctor Chips	Gm	Instant S/W	16K	C	GB	£12.50
Doctor Chips	Gm	Instant S/W	16K	D	GB	£14.75
Domes of Kilgari	Gm	MA	16K	C	IS	£11.50
Dominoes	Gm	Molimerx	16K	C	AJ	£9.15
Dominoes	Gm	Molimerx	32K	D	AJ	£12.65
Doomwatch	Do	Kansas	16K	C	IH	£7.50
Dosort	Ut	Molimerx	32K	C	AJ	£27.60
Double Take 3741	Ut	Molimerx	64K	D	AJ	£44.85
Double Up	Ut	Kansas	16K	C	IH	£6.25
Dragon Quest	Gm	Instant S/W	16K	C	IS	£10.50
Dragon Quest	Gm	Instant S/W	16K	D	GB	£17.50
Dragons of Hong Kong	Gm	Adventure Int.	32K	C	AB	£18.99
Draughts	Gm	Molimerx	16K	C	AJ	£14.95
Draughts	Gm	Molimerx	32K	D	AJ	£18.40
Draw and Graphics Resolver	Bs	Molimerx	16K	C	AJ	£10.95
Dreamworld	Gm	Molimerx	16K	C	AJ	£10.06
Driver	Gm	Molimerx	4K	C	AJ	£6.33
Duel-n-Droids	Gm	Molimerx	16K	C	IS	£9.50
Dungeon Escape	Gm	Computer Shack	16K	C	GB	£11.95
Dungeon Escape	Gm	Computer Shack	16K	D	GB	£15.50
Duplex	Ut	J.K. Gosden S/W	16K	C	LB	£14.00
Dynamic Device Driving	Ut	Instant S/W	16K	D	GB	£20.50
Dynamic Device Driving	Ut	Instant S/W	16K	C	GB	£15.50
					AB	£16.50
Early Numbers	Ed	Bryants S/W	16K	C	HW	£3.75
Edas	Ut	Molimerx	32K	D	AJ	£53.49
Edit	Ut	Molimerx	16K	C	AJ	£20.13
Editor Assembler Plus	Ut	Molimerx	16K	C	AJ	£24.95
Editor Assembler and Debugger	Ut	Kansas	16K	C	IH	£19.50
Edjust	Ut	Molimerx	32K	D	AJ	£18.40
Eights	Gm	Emjay	16K	C	IG	£7.00
Electric Accountant	Bs	Molimerx	48K	D	Aj	£28.75
Electronic Breadboard	Ut	Instant S/W	16K	C	GB	£39.75
					AB	£40.95
Elemental Maze	Ed	Molimerx	16K	C	Aj	£10.93
Eliminator	Gm	Adventure Int.	16K	C	IS	£15.00
					OH	£15.50
Eliminator	Gm	Adventure Int.	16K	d	AB	£18.99
Emperor	Gm	Molimerx	16K	C	AJ	£15.53
Empire	Gm	MA	16K	C	IS	£10.50
Empire of the Overmind	Gm	Avalon Hill	48K	C	IA	£17.95
Enhanced Basic	Ut	Molimerx	16K	C	AJ	£27.60
Enhanced Basic	Ut	Molimerx	32K	D	AJ	£31.05
Epson Patch	Ut	Molimerx	16K	C	Aj	£13.80
Epson Patch	Ut	Molimerx	32K	D	Aj	£17.25
Escape from Tramm	Gm	Adventure Int.	16K	C	IS	£15.00
					AB	£18.99
Everest Explorer	Gm	Molimerx	16K	C	AJ	£10.06
Escape	Gm	Kansas	16K	C	IH	£7.50
Everest Explorer	Gm	Molimerx	32K	D	AJ	£13.50
Everyday Russian	Ed	Instant S/W	16K	C	GB,AB	£8.50
F for Freddie	Gm	Kansas	16K	C	IH	£9.50
Facto Bingo	Ed	Bryants S/W	16K	C	HW	£3.75
Facto Finder	Ed	Bryants S/W	16K	C	HW	£3.75
Fairytale Adventure	Gm	Molimerx	16K	C	AJ	£10.06
Fastgammon	Gm	MA	16K	C	IS	£14.00
Family Tree	Do	Molimerx	32K	D	AJ	£20.70
Farmer Brown	Gm	Molimerx	16K	C	AJ	£7.49
Fed	Ut	Molimerx	32K	D	AJ	£27.60
Fetch	Ut	Molimerx	32K	D	AJ	£15.53
File Handling for Cassette	Ut	Molimerx	16K	C	AJ	£15.93
File Handling for Disc	Ut	Molimerx	32K	D	AJ	£28.75
Filter Package	Ut	Molimerx	32K	D	AJ	£40.25
Financial Analysis	Bs	Molimerx	32K	D	Aj	£46.00
Find it Quick	Ut	Instant S/W	16K	D	GB	£39.75
					AB	£40.95
Flight Path	Gm	Instant S/W	16K	C	GB,AB	£8.50
Flight Plan	Ut	Molimerx	32K	D	AJ	£20.70
Flippy	Gm	MA	16K	C	IS	£10.50
Flight Simulator	Gm	MA	16K	C	IS	£20.00
Flying Circus	Gm	Instant S/W	16K	D	AB	£24.50
					GB	£25.00
Forbidden City	Gm	Fantastic S/W	16K	D	IA	£24.95
Forbidden Planet	Gm	Fantastic S/W	48K	D	IA	£24.95
Forest of Mordor	Gm	Molimerx	16K	C	AJ	£8.63
Fortran	Ut	Molimerx	32K	D	AJ	£74.75
Fortress	Gm	Soft Sector Mkting	16K	D	IA	£15.50
Fortress	Gm	Soft Sector Mkting	16K	C	IA	£11.95
Fortress	Gm	MA	16K	C	AJ	£10.50
Fox and Hounds	Gm	Molimerx	16K	C	AJ	£7.48
French Vocabulary	Ed	Molimerx	16K	C	AJ	£17.19
Frog	Gm	Adventure Int.	16K	C	AB	£12.50
Frog	Gm	Kansas	16K	C	AJ	£7.48
Frogs/Logic	Gm	Kansas	16K	C	IH	£7.50
Galatic Empire	Gm	Adventure Int.	16K	C	IS	£10.50
					AJ,AB	£9.78
Galatic Firebird	Gm	Kansas	16K	C	IH	£9.50
Galactic Revolution	Gm	Adventure Int.	16K	C	AB	£18.99
Galactic Revolution	Gm	Adventure Int.	16K	C	IS	£10.50
Galactic Trader	Gm	Adventure Int.	16K	C	IS	£10.50
Galactic Trader	Gm	Adventure Int.	32K	D	AB	£18.99
Galaxy Invasion	Gm	Big 5 S/W	16K	C	IS	£11.50
Gamesaver	Ut	J.K. Gosden S/W	16K	C	LB	£6.50
Game Of Life	Gm	MA	16K	C	IS	£6.50
Gammon Challenger	Gm	MA	16K	C	IS	£10.50
Gammon Challenger	Gm	Molimerx	16K	C	AJ	£10.06
Gammon Challenger	Gm	Molimerx	32K	D	AJ	£13.51
Gauntlet of Death	Gm	Programmer's Guild	16K	C	GB	£15.50
Gauntlet of Death	Gm	Programmer's Guild	16K	C	GB	£15.50
G.C.E. O'Levels						
Tape 1	Ed	Bryants S/W	16K	C	HW	£9.00
Tape 2	Ed	Bryants S/W	16K	C	HW	£9.00
Gencop	Ut	Molimerx	16K	C	AJ	£10.06
Geography Explorer	Ed	Instant S/W	16K		AB	£40.95
Ghost Town	Gm	Adventure Int.	16K	C	IS	£12.50
					AB	£18.99
					OH	£15.25
Golfers Challenge	Gm	MA	16K	C	IS	£10.50
Golden Voyage	Gm	Adventure Int.	16K	C	IS	£15.00
					AB	£18.99
					OH	£15.25
Golf	Gm	Molimerx	32K	D	AJ	£17.19
Good Games No.I	Gm	Cload Inc	16K	C	IA	£11.95
Guns of Fort Defiance	Gm	Avalon Hill	32K	C	IA	£12.95
Guns of Fort Defiance	Gm	Avalon Hill	32K	D	IA	14.95
Gun World	Gm	Ogre	16K	C	IC	£10.00
Graphics Assembler	Ut	Kansas	16K	C	IH	£7;50
Graphs	Ed	Bryants S/W	16K	C	HW	£3.75
Graphit	Ut	Molimerx	48K	D	AJ	£19.55
Gridref	Ed	Bryants S/W	16K	C	HW	£3.75
Hellfire Warrior	Gm	MA	16K	C	IS	£21.00
Highway Code	Do	Bryants S/W	16K	C	HW	£3.75
HMS Impetuous	Gm	Adventure Int.	32K	D	AB	£18.99
Home Budget Planner	Bs	Emjay	16K	C	IG	£8.00
House of 30 Gables	Gm	Instant S/W	16K	C	AB	£8.50
Hyperlight Patrol	Gm	Fantastic S/W	16K	D	GB	£11.25
ID Mailing List	Ut	Instant S/W	16K	D	GB	£19.50
					AB	£20.50
IRV	Ut	Instant S/W	16K	D	GB	£23.75
IRV	Ut	Instant S/W	16K	C	IS	£16.50
					AB	£20.50
Infinite Startrek	Gm	Kansas	16K	C	IH	£9.50
Inventory Control	Bs	MA	16K	C	IS	£11.00
Inventory Control	Bs	MA	32K	D	IS	£50.00
Inventory'S'	Bs	MA	16K	C	IS	£17.00
Invoicing	Bs	Tridata	48K	D	ID	£140.00
Instant Calculator	Do	MA	16K	C	IS	£8.50
Interlude	Gm	Syntonic S/W		C	GB	£20.25
Interlude	Gm	MA	16K	C	IS	£14.00
Invaders From Space	Gm	MA	16K	C	IS	£10.50
Invasion Orion	Gm	Automated Simulation		C	IS	£18.50
Invasion Orion	Gm	Automated Simulation		D	GB	£15.75
Investor's Paradise	Gm	Instant S/W	16K	C	AB	£8.50
Jabbertalky	Gm	Automated Simulation	16K	D	GB	£18.95
Jet Fighter Pilot	Gm	Instant S/W	16K	C	GB,AD	£12.50
Jumbo	Gm	Molimerx	16K	C	AJ	£17.25
Jumbo	Gm	Molimerx	32K	D	AJ	£19.55
Jumbles	Ed	Bryants S/W	16K	C	HW	£3.75
KFS-80	Ut	MA	32K	D	IS	£75.00
Keyboard 80	Do	MA	16K	C	IS	£7.50
KVP	Ut	MA	16K	C	IS	£10.50

Spelling Builder	Ed	MA	32K	C	IS	£15.00
Spellsort	Ed	Bryants S/W	16K	C	HW	£3.75
Spelltest	Ed	Bryants S/W	16K	C	HW	£3.75
Spider Mountain	Gm	MA	16K	C	IS	£9.50
Starfighter	Gm	MA	16K	C	IS	£18.50
Starfleet Orion	Gm	MA	16K	C	IS	£18.50
Starflite	Gm	MA	16K	C	IS	£10.50
Starlord	Gm	Bryants S/W	16K	C	IC	£10.00
Statistics	Ed	MA	16K	C	IS	£6.50
S.T.A.D	Ut	MA	16K	C	IS	£15.00
Storybuilder	Ed	Bryants S/W	16K	C	IS	£12.50
					HW	£3.75
Stock	Bs	Tridata	48K	D	ID	£170.00
Stock or Bonds	Gm	Avalon Hill	16K	C	IA	£12.95
Stone of Sisythus	Gm	Adventure Int.	32K	D	AB	£33.70
ST-80	Ut	MA	16K	C	IS	£34.00
ST-80D	Ut	MA	16K	D	IS	£45.00
Strange Odyssey	Gm	Adventure Int.	16K	C	IS	£12.50
					AB	£18.99
					OH	£15.25
Structural Analysis	Bs	Casual Computing	16K	C	OF	£60
Sunday Golf	Gm	Adventure Int.	16K	D	AB	£18.99
Surveyor's Apprentice	Ed	Instant S/W	16K	C	AB	£8.50
Super Breakthrough	Gm	Kansas	16K	C	IH	£8.50
Super Copy	Ut	Kansas	16K	C	IH	£9.50
Super Nova	Gm	Big 5	16K	C	IS	£11.50
Super Pims Data Base	Do	MA	16K	C	IS	£10.50
Super T-Legs	Ut	MA	16K	C	IS	£7.50
Superscript	Ut	MA	32K	D	IS	£21.00
Superkeys	Ut	MA	32K	D	IS	£35.00
Super Simon	Ut	MA	16K	C	IS	£7.50
Swamp War	Gm	Instant S/W	16K	C	AB	£12.50
Swords and Socery	Gm	Kansas	16K	C	IH	£9.50
Sword Of Zedek	Gm	MA	16K	C	IS	£11.50
System Copy Mod I	Ut	MA	16K	C	IS	£9.50
System Master Monitor	Ut	Kansas	16K	C	IH	£17.00
System to Basic	Ut	Kansas	16K	C	IH	£8.50
Tcopy III Mod III	Ut	MA	16K	C	IS	£10.50
T-Step	Ut	MA	16K	C	IS	£7.50
TL DIS	Ut	Instant S/W	16K	C	AB	£12.50
TRS Test	Ut	Instant S/W	16K	C	AB	£8.50
TRS-80 Opera	Do	MA	16K	C	IS	£7.50
Tables Test	Ed	Bryants S/W	16K	C	HW	£3.75
Tanktics	Gm	Avalon Hill	16K	C	IA	£15.95
					OH	£15.50
Targetsounds	Ed	Bryants S/W	16K	C	HW	£3.75
Teachers Assistant 1	Ed	MA	16K	C	IS	£9.50
Teachers Assistant 2	Ed	MA	16K	C	IS	£9.50
Teachers Assistant 3	Ed	MA	16K	C	IS	£9.50
Teacher's Aid	Ed	Instant S/W	16K		AB	£34.50
Temple Of Apshai	Gm	MA	16K	C	IS	£18.50
Temple of the Sun	Gm	Instant S/W	16K		AB	£16.50
The Count	Gm	Adventure Int.	16K	C	IS	£12.50
					AB	£18.99
					OH	£15.25
The Mean Checkers Machine	Gm	Adventure Int.	16K	C	AB	£14.35
The Congo Mission	Gm	Bryants S/W	16K	C	IC	£10.00
The Quest	Gm	Bryants S/W	16K	C	IC	£12.00
The Secret Wilderness	Gm	Ogre	16K	C	IC	£6.50
Toolkit	Ut	Kansas	16K	C	IC	£12.00
Time Traveller	Gm	MA	16K	C	IS	£10.50
Time Trek	Gm	MA	16K	C	IS	£10.50
Tiny Pascal	Ut	MA	16K	C	IS	£17.00
Timser	Ed	MA	16K	C	IS	£10.50
Tiny Compiler	Ut	MA	16K	C	IS	£12.50
Towers	Ed	Bryants S/W	16K	C	HW	£3.75
Treasure Quest	Gm	Adventure Int.	16K	C	AB	£14.35
Tunnels of Fahad	Gm	Adventure Int.	16K	C	AB	£12.50
Turret & Track	Gm	MA	16K	C	IS	£7.50
Two Heads of the Coin	Gm	Adventure Int.	32K	D	AB	£18.99
Typing Teacher	Ed	Instant S/W	16K	C	AB	£8.50
Typing Tutor	Do	MA	16K	C	IS	£12.00
Typewriter	Ut	Kansas	16K	C	IH	£6.25
Ultra Man	Ut	Instant S/W	16K		AB	£21.50
Ultra Mon	Ut	MA	16K	C	IS	£17.00
Up Periscope	Gm	MA	16K	C	IS	£10.50
Utility II	Ut	Instant S/W	16K	C	IA	£8.50
Valley	Gm	CT Software	16K	C	PF	£9.95
Valley of the Kings	Gm	Instant S/W	16K	C	IC	£10.00
Venture	Gm	Horizon S/W	16K	C	IA	£11.95
Vocabulary Builder 1	Ed	MA	16K	C	IS	£12.50
Vocabulary Builder 2	Ed	MA	16K	C	IS	£12.50
Voodoo Castle	Gm	Adventure Int.	16K	C	IS	£12.50
					AB	£18.99
					OH	£15.25
Voyager	Gm	Avalon Hill	16K	C	IA	£12.95
What Next?	Gm	Baust Computing	4K	D	HA	£6.95
What Next?	Gm	Baust Computing	4K	C	HA	£3.95
Word Processor	Bs	Kansas	16K	C	IH	£19.50
Word Watch	Ed	Instant S/W	16K	C	IA	£6.50
X-Wing Fighter II	Gm	MA	16K	C	IS	£7.50
X-Ref	Ut	MA	16K	C	IS	£7.50
Y-Y Bar	Bs	MA	16K	C	IS	£10.50
Yahtzee	Gm	Kansas	16K	C	IH	£8.50
Your Life Score	Do	Bryants S/W	16K	C	HW	£3.75
Zossed in Space	Gm	Adventure Int.	16K	D	AB	£14.35
Zossed in Space	Gm	Adventure Int.	16K	C	CH	£10.00
9 Kid's Games	Ed	MA	16K	C	IS	£12.50
76 Basic Programs	Do	MA	16K	C	IS	£23.00

TANDY TRS-80

MINI AMPLIFIER SPEAKER

This functional grey box is designed to give your Tandy a voice. Well not so much a voice, but you can program the Tandy to give you sound effects from the cassette port. By plugging this unit into the cassette lead from your computer you can amplify the signals leaving your computer.



The unit features an inbuilt speaker, volume control, input jack and external speaker/ear piece jack. It is powered by a nine volt battery which lasts for a reasonably long time, except if you wind the volume up to the ear cracking levels. Doing this will considerably shorten the battery life.

Operationally, this amplifier did its job adequately but not brilliantly.

The Archer amplifier/speaker costs £7-49 and is available from Tandy Computer Centres.

TANDY 2000

Glyn Moody tries out the newest addition to Tandy's flock and finds that despite its ugly duckling appearance the machine runs like a beauty.

TANDY is following Commodore in hedging all its bets in the micro game. Instead of opting for one processor or operating system, Tandy has launched systems based on 6809s, Z-80s, 8085s and 68000s, running variously under CP/M, TRS-DOS and Xenix. Latest addition to the stable is the Model 2000, an MS-DOS machine sporting the new and elusive 80186 Intel processor — see Chip-Chat page 37.

The rest of the specification is undramatic: 128K RAM expandable to 768K, two half-height 720K 5.25in. floppies, keyboard and screen, for a cost of about £2,000. A 10Mbyte hard disc version is available for £3,678.

No recesses

The review system had high-resolution colour graphics, for which an extra board is required. The 14in. colour screen is very large and bulky and lacks any helpful recesses to grapple with, which makes it even more unmanageable. The main processor unit is also surprisingly large, and its general appearance is not helped by Tandy's ugly duckling styling. Apart from the disc drives, the power and reset buttons are positioned sensibly at the front instead of being placed awkwardly at the rear. Unfortunately the design of the keyboard socket verges on the wilfully perverse: a DIN-type plug connects inside a deep recess that is inaccessible with the unit in normal use.

At the rear, from left to right, are sockets for a monochrome TV, parallel interface and RS-232 port. To the left of the cooling fan and kettle-plug power socket are four removable slots used for additional boards. Inside is the corresponding card cage and the discs and power supply take up most of the remaining space. The motherboard is slotted underneath, and is accessed by removing four screws, although Tandy discourages users from tampering directly with the innards. Similarly two screws release the main cover.

On the review machine evidence of last-minute patching was apparent since a couple of EPROMs and a few straggling wires were visible on the main circuit board. The overall design of the internal circuitry is robust and the review system worked first time when reassembled.

The keyboard shares the same unlovely rounded styling as the processor unit, and uses novel roll-up legs for tilting. Apart

(continued on next page)



The main processor unit and colour monitor are large and bulky, and awkward to handle.

Benchmarks

The table shows the time in seconds to run eight standard Microsoft Basic routines. As the table shows, the Model 2000 is one of the fastest machines we have reviewed.

	BM1	BM2	BM3	BM4	BM5	BM6	BM7	BM8	Av.
Tandy Model 2000 — 80186	0.5	2.0	4.3	4.7	5.2	9.5	13.8	13.7	6.7
HP Model 16 — 68000	0.2	0.6	1.4	1.6	1.7	2.8	4.3	15.0	3.5
OEM Orion — 8086	0.6	2.1	4.8	4.9	5.8	10.5	16.7	13.0	7.3
IBM PC — 8088	1.2	4.8	11.7	12.2	13.4	23.3	37.4	30.0	16.8

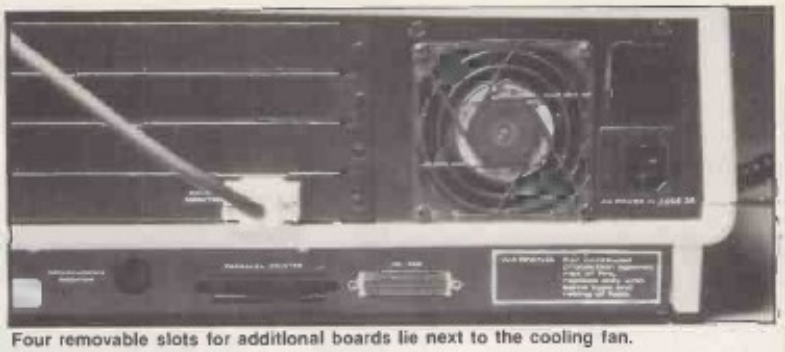
TANDY

(continued from previous page)

from standard Qwerty, there are 12 function keys, and a numeric pad which provides additional characters \ and – in default mode. With the number lock On, the numeric pad functions normally. The keys themselves are light and slightly shallow but should present no problems for word-processing applications.

The main point of interest with the Model 2000 is clearly its processor, which is one of the new generation of chips on the 8086 but with an extended instruction set and running at higher speeds — in this case 8MHz. The speed certainly shows up in performance: the Benchmarks are the second fastest of any machine we have tested, beaten only by those of the 68000-based HP 16. Unfortunately this power is rather marred in Basic by a curious hobbling vertical scroll which soon becomes tiring on the eyes. Microsoft Basic and MS-DOS come bundled as standard.

Additional cards slot in at the rear of the machine and mate internally via edge-connectors. Current options available are high-resolution monochrome or colour graphics; memory up to 786K, the first 128K of which is plugged directly on to the motherboard, and a mouse peripheral.



Four removable slots for additional boards lie next to the cooling fan.

Specification

CPU: Intel 80186, running at 8MHz
Memory: 128K RAM expandable to 768K; 128K, bootstrap only
Display: 12in. monochrome or 14in. colour; 25 lines by 80 characters; 640 by 400 pixels; eight colours from a palette of 16
Keyboard: 12 function keys; numeric keypad
Interfaces: RS-232, parallel port
Discs: one or two half-height 720K 5.25in. floppies
Software in price: MS-DOS and Basic
U.K. prices: twin floppy-disc £1,999 ex. VAT, hard disc £3,678 ex. VAT.
U.K. distribution: Tandy Corporation, Tameway Tower, Bridge Street, Walsall, West Midlands, WS1 1LA. Tel: (0922) 648181. Available now.

Conclusions

● The 80186 is a neat chip but it is also in very short supply and there is a backlog of orders for the Model 2000 in the U.S. So apart from the initial launch batch in the U.K., machines will be very hard to come by.

● The Tandy will never win any beauty contests, despite its new styling. But if it is speed rather than looks that you are after, the 2000 offers good value for money.

● The standard system specification provides a usable business machine, though the footprint verges on the excessive. In particular, the colour monitor is far too large so unless colour facilities are vital, the monochrome option is probably a better choice.

SWAP MICRO TO MICRO FILE TRANSFER SYSTEM



"Get your microcomputers talking to each other!"

SWAP allows you to transfer any programs and data between 2 computers of different manufacture. SWAP consists of 2 floppy disks and a cable configured for your 2 chosen computers. Any combination is available from:

IBM PC	Apple (CP/M)	Tandy II (CP/M)
Sirius	DEC Rainbow	Comart Communicator
Apricot	ICL PC	Epson QX10
Superbrain	Rair	ECS 4500
Televideo	North Star Advantage	

The price of SWAP is £158 (£135 plus VAT and postage and packing). Please specify your computers when ordering.

MERCATOR COMPUTER SYSTEMS LTD
3 Whitchurch Road, Clifton, Bristol BS8 1NU.
Telephone: (0272) 731079
Telex 44220 Comcel Ref 247

MERCATOR COMPUTER SYSTEMS

SWAP-SWAP-SWAP-SWAP-SWAP

● Circle No. 133

CONVERT YOUR 80-BUS SYSTEM TO 16 BIT WITH A COSTGOLD ADD-ON PROCESSOR BOARD.

Run CP/M-86. Read/Write IBM PC compatible discs.

CA802 £525
8MHz 8088 Processor
256K Bytes RAM
serial port and real time clock
EPROM based monitor
Supports CP/M-86, concurrent CP/M-86 and MP/M-86

CA801 £375
8MHz 8088 Processor
128K Bytes RAM
EPROM based monitor
Supports CP/M-86
Can be expanded later to CA802 spec

CP/M-86 for CA801/CA802, including utilities £199

SPECIAL OFFER.... CA802 and CP/M-86 £695

Either 80-bus format board enables retention of the Z80 card to run your existing software. Both cards will read or write IBM PC compatible discs & all Gemini formats. A 8088 bus expansion connector is also provided.

All prices include p&p but exclude VAT at 15%.

CA801 & CA802 are available now. Orders to Costgold Research, The Old School, Stretham, Cambridge CB6 3LD. Tel (035 389) 282/3.

Coming soon Concurrent CP/M-86.

COSTGOLD RESEARCH, THE OLD SCHOOL, STRETHAM, CAMBRIDGE CB6 3LG.

Tel: (035 389) 600

● Circle No. 134

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

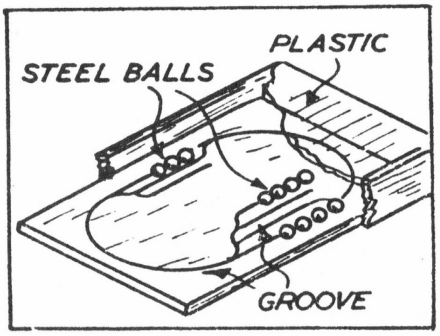


HANDY HINTS

HERE'S SOMETHING JUST A BIT DIFFERENT. IT'S A 'HANDY HINT' FROM THE 1955 'GADGETS ANNUAL'. ANOTHER GREAT IDEA!

A DELIGHT FOR THE YOUNGSTERS

AN INEXPENSIVE ELECTRIC "RAILWAY"



THIS is just about the simplest type of "electric railway" you could have. Draw out a plan of the rail system on a suitable wooden base about twelve to eighteen inches long and not more than ten inches wide. Now cut a groove in the wood to represent the "rails". The "trains" are simply sets of steel balls running in the grooves. You can incorporate any number of sidings, points and crossovers. Complete the assembly, if you prefer, by adding sides and covering in the top with a sheet of clear plastic.

To power the trains, hold a strong magnet under the base and move it along in front of the steel balls. In quite a short while the balls will become lightly magnestised and tend to stick together in groups. Movement of the magnet guides the various "trains" over the respective tracks. With practice, very good control of the "rolling stock" is possible, and the youngsters will be delighted with their novel toy.

Here's a super article from Peter Phillips correcting bugs in David Ahl's original maze creating program. There's a downloadable .zip file 'amazing.zip', containing cassette, disk and floppy-tape images. It's from the downloads page at www.trs-80.org.uk. ED.

Amazing Bug

By Peter Phillips

David Ahl's incredible book *101 TRS-80 Computer Games* demonstrated what was possible with BASIC. One of my favourites was *Amazing*, a program that draws rectangular mazes with just one solution. At the time it was difficult to understand how it worked, but I was impressed.

I typed in *Amazing* again and was surprised to find it had a couple of bugs. The problems are more frequent on small maze sizes. Here's an example of a correct 3,3 maze.

```
:  :--:--:
I        I
:  :--:  :
I  I     I
:  :  :--:
I  I     I
:--:--:  :
```

All rooms are reachable and there is one unique solution from the entrance at top to the exit at the bottom. But on some runs, you get a maze without an exit at the bottom. (Bug #1)

```
:  :--:--:
I     I  I
:--:  :  :
I     I  I
:  :  :  :
I  I     I
:--:--:--:
```

And, in some runs, you end up with some rooms not opened and (sometimes) a missing exit as in this example (Bug #2)

```
:  :--:--:
I  I     I
:  :--:  :
I  I     I
:  :  :--:
I  I  I  I
:--:--:--:
```

I was surprised but compelled to fix it.

Theory

In broad strokes, the *Amazing* algorithm works by starting with all rooms closed, creating an entrance, and then randomly extending the path until it encounters a dead end–where no unopened room is found at the head of the path. At that point, it searches the maze for an open room and tries to extend it. Once all of the maze rooms are opened, the maze is complete and is printed.

A few variables keep track of the drawing:

H,V	maze width and height
R,S	current position in the maze
Z	flag indicating an exit has been created
W(H,V)	room open (0 means closed)
V(H,V)	left and bottom walls of the given room
Q	temporary flag, indicating that choosing the down direction creates an exit

Comment [1]: open? 0 means closed?

The top entrance and starting point for drawing a path is chosen randomly. The top edge, with entrance, is immediately printed.

V(H,V) has 4 possible values to indicate which walls are open. Zero means both are closed. 2 or 3 means the right wall is open and 1 or 3 indicate the bottom wall is open.

The Z flag creates a lot of extra code. When it is zero, a move from the bottom row is allowed to create an exit. The Z flag is the key to Bug #1. It is possible for the random walk through the maze to open every room without creating an exit.

Bug #1 Fix

The fix is to add an exit if one does not exist. Change line 1200 to do this and move the start of the print loop to 1205:

```
1200 IF Z=0 THEN X=RND(H):V(X,V)=V(X,V)+1
1205 FOR J=1 TO V
```

Bug #2

The second bug is not nearly as obvious. After several runs and adding debug PRINT statements to the code I finally noticed the problem was the GOTO 990 at line 830. When drawing the path, the four possible directions are at lines 940, 980, 1020 and 1090 (left, up, right, down). So a jump to 990 is a move UP but with line 980 skipped. It needs to be changed, but what is the correct fix?

Decision Tree

I call lines 270 to 930 the *Decision Tree*. The code determines which direction to move (if possible) from the current R,S position. It is done with a lot of IF/THEN/GOTO statements and ultimately leads to 15 possible outcomes.

Each exit outcome is a random choice based on the number of possible directions which range from 0 to 3. Four is not possible because the path always moves from an already open room (so there is always one unavailable backwards direction).

#Dir.	Cases	Directions
0	1	(NO PATH)
1	4	(UP) (DN) (LF) (RT)
2	6	(RT,DN) (UP,DN) (UP,RT) (LF,DN) (LF,RT) (LF,UP)
3	4	(UP,RT,DN) (LF,RT,DN) (LF,RT,UP) (LF,UP,DN)
Total	15	

The decision tree figure visually shows how conditions accumulate through each IF/GOTO test. The entry line, 270, chooses between the LEFT direction being open or the LEFT direction being blocked. Blocked is one of two cases: either R,S is on the LEFT most room of the maze or because the room to the left of R,S is not open.

The conditions for each direction LEFT, RIGHT, UP, DOWN are abbreviated as LF, RT, UP, DN. Exclamation points mark a direction that is *not* available (e.g. !LF)

When a rounded outcome node is reached it shows the list of random direction choices.

Lower in the tree additional conditions appear. BOTTOM refers to R,S being on the last row of the maze. Z=1 or Z=0 are checks for the exit created flag (Z=0 means no exit has yet been created).

The Q←1 labels indicate the Q flag is set. Q tells the wall opening code to create an exit if the direction is DN. If an exit is created, Z is set to 1.

Bug #2 Fix

We can now trace through the tree to see what conditions exist when we hit the (UP BUG) node.

Line	Condition	Next
270 Start	!LF	600
600	!UP	790
790	RT	810
810	BOTTOM	820
820	Z=0	830
830	Q←1	(UP BUG)

All of the conditions together yields:

!LF !UP RT BOTTOM Z=0

Clearly, (UP BUG) is wrong as UP is not an available direction.

LEFT and UP are excluded leaving RIGHT and DOWN as the only valid directions. DOWN is allowed by the combination of BOTTOM and Z=0 (exit not created). The correct direction list (with Q set to 1) is (RT,DN).

To fix, line 830 must GOTO 850 instead of the incorrect 990:

```
830 Q=1: GOTO 850
```

Original Edition

For the original *101 BASIC Computer Games Microcomputer Edition* the same bugs are in *Amazing* but the fixes require different line numbers.

For Bug #1, the original edition fix changes to line 1010:

```
1010 IF Z=0 THEN X=RND(H):V(X,V)=V(X,V)+1
1011 FOR J=1 TO V:PRINT "I";
```

For Bug #2, change line 695 to GOTO 710 instead of 830:

```
695 Q=1:GOTO 710
```

Conclusion

Figuring out how *Amazing* really worked was about as difficult when I first saw it. Better tools, computers and more experience helped but not as much as I'd hoped.

I'm alway impressed at the complexity of these relatively small programs. If you run into some old source code, take a closer look, you may be amazed at what you'll find out.

***Amazing* Listing**

The original code but with a few comments added and some blank lines.

```
5 REM splash screen, input maze
dimensions H,V = width,length

10 CLS: PRINT @ 412,"AMAZING"
20 PRINT @ 538,"COPYRIGHT BY"
30 PRINT @ 587,"CREATIVE COMPUTING
MORRISTOWN, NEW JERSEY"
40 FOR II=1 TO 1500: NEXT
50 CLS: INPUT"WHAT ARE YOUR WIDTH AND
LENGTH"; H,V
60 IF H<>1 AND V<>1 THEN 90
70 PRINT "MEANINGLESS DIMENSIONS. TRY
AGAIN"
80 FOR A=1 TO 500: NEXT A: GOTO 50
90 PRINT @ 522,"PRINTOUT IS IN
PROGRESS, PLEASE BE PATIENT"
100 FOR II=1 TO 1500: NEXT

105 rem initialization setting Q,Z and
choosing entrance X
```

```
106 rem W(H,V) is the open room
marker, V(H,V) is the right wall (1) +
bottom wall (2) marker
110 DIM W(H,V), V(H,V)
120 Q=0: Z=0: X=RND(H)

125 rem print top line of maze with
hole at entrance X

130 FOR I=1 TO H
140 IF I=X THEN 160
150 LPRINT ":--";: GOTO 170
160 LPRINT ":  ";
170 NEXT I
180 LPRINT ":"

185 rem set open count C to 1, mark
entrance as open, increase open count
187 rem then use R,S = X,1 as the
current (open) room
188 rem so R,S = X,Y or column,row
190 C=1: W(X,1)=C: C=C+1
200 R=X: S=1: GOTO 270

210 IF R<>H THEN 250
220 IF S<>V THEN 240
230 R=1: S=1: GOTO 260
240 R=1: S=S+1: GOTO 260
250 R=R+1
260 IF W(R,S)=0 THEN 210

265 rem find path from open room

270 IF R-1=0 THEN 600
280 IF W(R-1,S) <>0 THEN 600
290 IF S-1=0 THEN 430
300 IF W(R,S-1)<>0 THEN 430
310 IF R=H THEN 350
320 IF W(R+1,S)<>0 THEN 350
330 X=RND(3)
340 ON X GOTO 940,980,1020

350 IF S<>V THEN 380
360 IF Z=1 THEN 410
370 Q=1: GOTO 390

380 IF W(R,S+1)<>0 THEN 410
390 X=RND(3)
400 ON X GOTO 940,980,1090

410 X=RND(2)
420 ON X GOTO 940,980

430 IF R=H THEN 530
440 IF W(R+1,S)<>0 THEN 530
450 IF S<>V THEN 480
460 IF Z=1 THEN 510
470 Q=1: GOTO 490

480 IF W(R,S+1)<>0 THEN 510
490 X=RND(3)
500 ON X GOTO 940,1020,1090

510 X=RND(2)
520 ON X GOTO 940,1020

530 IF S<>V THEN 560
540 IF Z=1 THEN 590
550 Q=1: GOTO 570

560 IF W(R,S+1)<>0 THEN 590
570 X=RND(2)
580 ON X GOTO 940,1090

590 GOTO 940

600 IF S-1=0 THEN 790
610 IF W(R,S-1)<>0 THEN 790
620 IF R=H THEN 720
630 IF W(R+1,S)<>0 THEN 720
640 IF S<>V THEN 670
650 IF Z=1 THEN 700
660 Q=1: GOTO 680

670 IF W(R,S+1)<>0 THEN 700
680 X=RND(3)
690 ON X GOTO 980,1020,1090

700 X=RND(2)
710 ON X GOTO 980,1020
720 IF S<>V THEN 750
730 IF Z=1 THEN 780
740 Q=1: GOTO 760

750 IF W(R,S+1)<>0 THEN 780
760 X=RND(2)
770 ON X GOTO 980,1090

780 GOTO 980

790 IF R=H THEN 880
800 IF W(R+1,S)<>0 THEN 880
810 IF S<>V THEN 840
820 IF Z=1 THEN 870
830 Q=1: GOTO 990

840 IF W(R,S+1)<>0 THEN 870
850 X=RND(2)
860 ON X GOTO 1020,1090

870 GOTO 1020

880 IF S<>V THEN 910
890 IF Z=1 THEN 930
900 Q=1: GOTO 920

910 IF W(R,S+1)<>0 THEN 930
920 GOTO 1090
```

```
930 GOTO 1190

935 rem move left
940 W(R-1,S)=C
950 C=C+1: V(R-1,S)=2: R=R-1
960 IF C=H*V+1 THEN 1200
970 Q=0: GOTO 270

975 rem move up
980 W(R,S-1)=C
990 C=C+1
1000 V(R,S-1)=1: S=S-1: IF C=H*V+1
THEN 1200
1010 Q=0: GOTO 270

1015 rem move right
1020 W(R+1,S)=C
1030 C=C+1: IF V(R,S)=0 THEN 1050
1040 V(R,S)=3: GOTO 1060

1050 V(R,S)=2
1060 R=R+1
1070 IF C=H*V+1 THEN 1200
1080 GOTO 600

1085 rem move down
1090 IF Q=1 THEN 1150
1100 W(R,S+1)=C: C=C+1:IF V(R,S)=0
THEN 1120
1110 V(R,S)=3: GOTO 1130

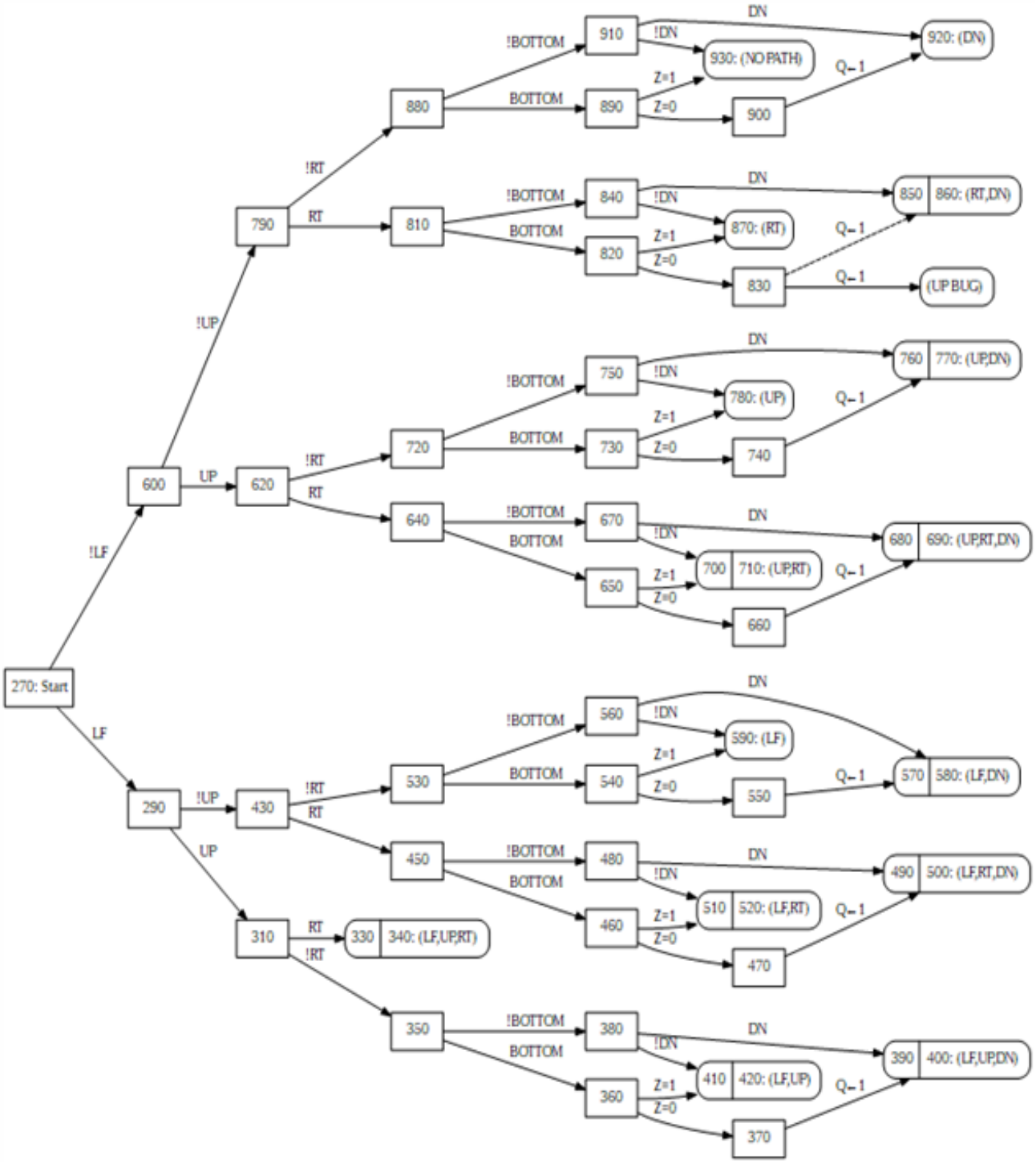
1120 V(R,S)=1
1130 S=S+1: IF C=V*H+1 THEN 1200
1140 GOTO 270

1145 rem move down making exit
1150 Z=1
1160 IF V(R,S)=0 THEN 1180
1170 V(R,S)=3: Q=0: GOTO 1190

1180 V(R,S)=1: Q=0: R=1: S=1: GOTO 260
1190 GOTO 210

1195 rem print the resulting maze
1200 FOR J=1 TO V
1210 LPRINT "I";
1220 FOR I=1 TO H
1230 IF V(I,J)<2 THEN 1260
1240 LPRINT "   ";
1250 GOTO 1270
1260 LPRINT "  I";
1270 NEXT I
1280 LPRINT " "
1290 FOR I=1 TO H
1300 IF V(I,J)=0 THEN 1340
1310 IF V(I,J)=2 THEN 1340
1320 LPRINT ":  ";
1330 GOTO 1350
1340 LPRINT ":--";
1350 NEXT I
1360 LPRINT ":"
1370 NEXT J
1380 END
```

Figure 1: Direction Decision Tree



Each box is a decision with the labeled outcomes on arcs.
Terminal rounded boxes show random directions chosen:
LF Left
RT Right
UP Up
DN Down

MODEL 1 PROBLEMS, AND THE MISPRINT DIDN'T HELP!

WELLSWANG
WITH HELP FROM FORUM MEMBERS, LARRY & EUDIMORPHODON

I got a MODEL I (26-1001) recently, but it could not start.
When I push the power button, there is only a scanline on the monitor, just like lost h-sync.
First I checked the power output, +5v, +12v and -5v are all normal.
I tried to replace Z80 chip and all RAM chips, but it still display the scan-lines.



I use oscilloscope to check the CLK (Z80 Pin6) signal, it's normal and at 1.77Mhz. But PIN 16,17,18, 23, 24, 25, 26 are all in HIGH state, the Z80 is not working, it's so strange.
I replaced 3 Z80 chips, still same.

I checked following signal frequency. All signals are normal except Z50 Pin9 is about 50Khz,and Z43 Pin7 in LOW state (I think this is normal, system is not start-up still in 32 Char mode.)

The address lines are outputs from the Z-80 used to specify memory locations. If any address line is not active (stuck on one state), there will be garbage' (random characters) on the screen at boot. To check the address lines, remove the DIP Shunt at position Z3 and power-up the system. If the screen fills with a pattern of "@9@9@9@9...", then all the address lines A0 – A9 are good. The remaining address lines A10 – A15 can be checked for activity with a scope.

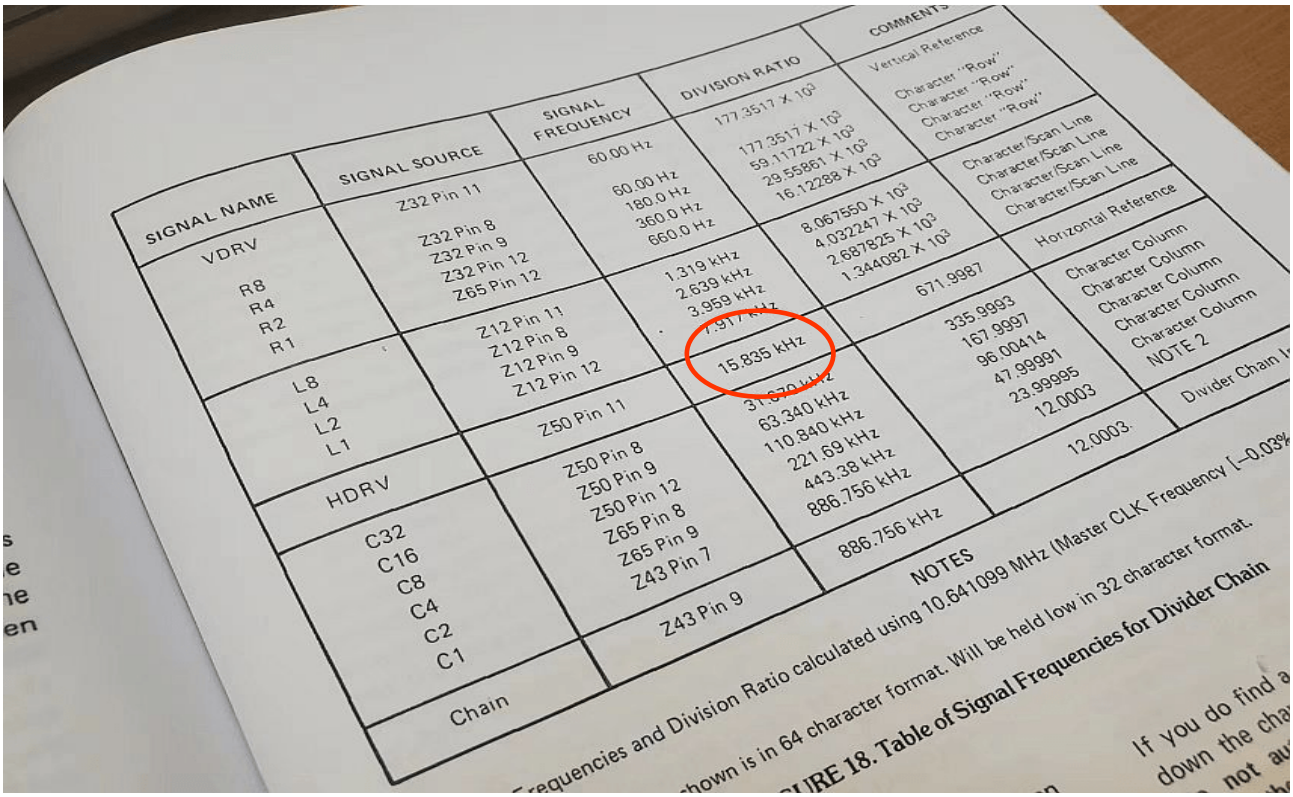
Here's why you get @9s. With the ROMs removed, the CPU reads all 1's (that is, hex FFH) whenever it reads from the area the ROMs are supposed to be in, because an open connection tends to float to a 1 with this type of logic. Hex FFH is the RST 38H instruction. So when the processor is turned on and fetches an instruction from 0000, it gets a RST 38H (FFH). This makes it push 0001 on the stack and jump to address 0038. At that address it reads another FFH, so it pushes 0039 on the stack and jumps to 0038 again. Now it's in an infinite loop. As the loop continues, the stack grows and grows, going backward through memory. When it reaches the frame buffer between 3C00 and 3FFF, you can see it on the screen. It looks like a string of @9's because the TRS-80 displays a 00 byte as an @ character, and 39 is an ASCII 9.

OK. So the fact that you get @9's means that the CPU is working. The data bus appears to be working correctly too, as if a data line were stuck at 0 the CPU wouldn't be reading FF's, and if it were stuck at 1 the CPU couldn't be writing both @'s and 9's.

There could still be a problem with the address bus or the ROMs. Note that a problem with the ROMs might not be in the ROM chips themselves but in the sockets or circuits on the motherboard that connect them to the rest of the system. But trying a different set of ROMs does seem like a good idea.

pc 0000 gets FF pushes 0001 to stack jumps to 0038
pc 0038 gets FF pushes 0039 to stack jumps to 0038
pc 0038 gets FF pushes 0039 to stack jumps to 0038

Also Check the RAM IC's to see if any of them are RED Hot. The -5 VDC Supply gets shorted Internally on those RAM IC's making them HOT.
(Larry)

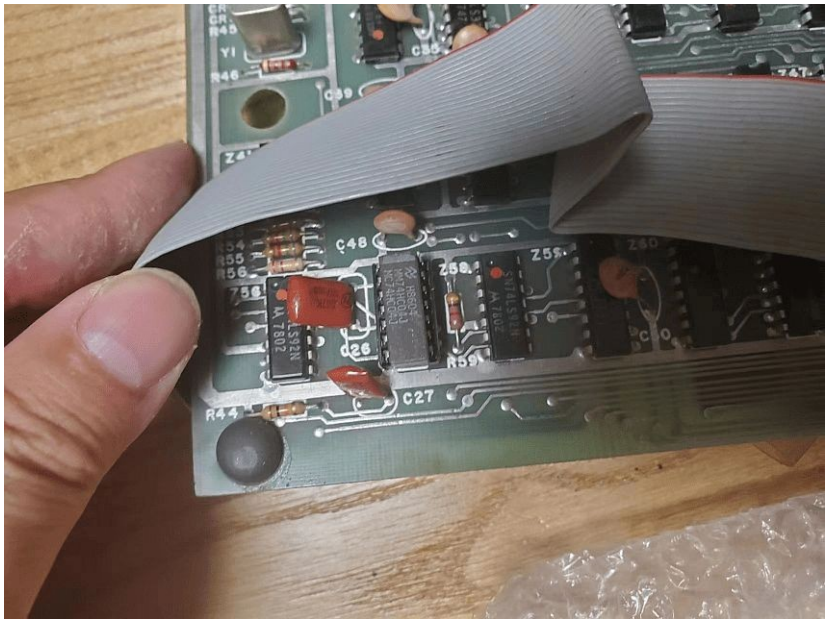


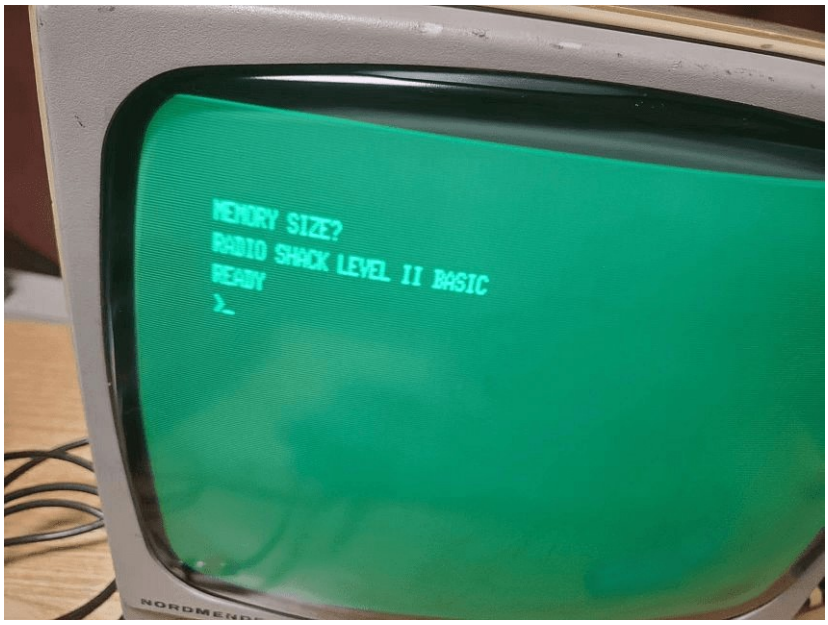
Where did you get your manual/table? Because parts of it seem wrong. I'm looking at a copy of the original 1978 service manual, and the *important* signal on Z50 is on pin 11, IE, the horizontal drive frequency, and that should be 15.840khz, not 11.835khz. If that is actually what you're reading that's way, way, off.
(Eudimorphodon)

Oh dear! It seems I have an error in my manual! Quickly sorted, but after further tests, I found the Z80 CPU is also failed. I replaced it too.

Now the TRS-80 Model I works fine, as you can see.

Thanks everyone for your kindly help!





LINE	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0001	0	H	0	H	0	H	G	H	0	H	0	H	0	V
0002	1	L	1	L	1	L	G	L	1	L	1	L	1	V

EBAY.CO.UK 'BARGAINS'

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



£
£363



£207



£200



£48



£41



£13



£276



£117



£113



£30

EBAY.CO.UK 'BARGAINS'

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



£

£71



£132



£332



£79



£502



£70



£82



£92



£28



£45

RESTORING THE CHEAPEST TRS-80 AT THE SWAP MEET

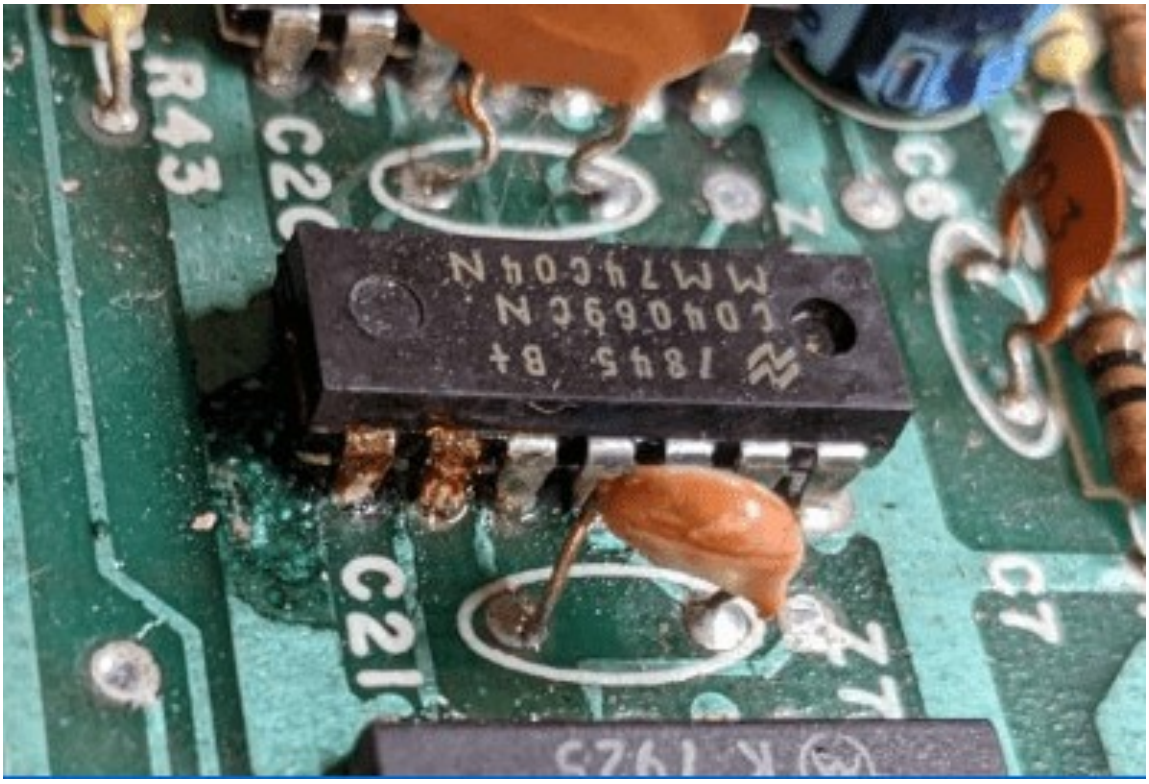
Tom Nardi



We don't know if you've looked into it recently, but the prices for vintage computers are through the roof right now. These classic machines are going through something of a renaissance at the moment, with even relatively commonplace computers commanding several hundred dollars if they're in good condition. For those looking to start a collection without breaking the bank, you may need to accept some specimens that have seen better days.

That's the situation [Vlado Vince] recently found himself in — he wanted to get his hands on a TRS-80 Model I, but wasn't willing to spend eBay prices. So he waited until the Vintage Computer Federation's swap meet in June and was able to snag a "fully functional" example for $95.

Unfortunately the seller must have been using some form of that phrase which we were previously unaware of, as it took a considerable amount of work to get it back online.



Getting the machine home [Vlado] was only able to get a very faint video signal out of it, and even then, it had no horizontal sync. Upon opening it up he found considerable corrosion, in addition to signs of previous repairs and bodges. Doing some research he found that one of the corroded ICs was indeed part of the video circuit, so it was carefully extracted and replaced with a socket to take the new chip. He also removed a hacked together memory upgrade that was likely installed in the 1980s, figuring it being there would only complicate the repair.

This still didn't give him a useful video signal, so he started a closer examination of the board. It turns out there were subtle modifications all over the place, with traces cut and whole components removed for reasons unknown. With the board schematics in hand, [Vlado] started undoing the changes and returning the circuits to their stock arrangement. He also repaired the badly damaged keyboard connector, as the previous owner's bodge wires didn't seem to be up to the task. These efforts paid off as he was finally able to get the machine to boot, though the video was erratic.

At a loss for the continuing display problems, [Vlado] started replacing components in the video circuit hoping to luck out. He swapped out the electrolytic capacitors and variable resistors, and even tried replacing components in the power supply, but the issue remained.



He eventually got the oscilloscope out and started poking around the board, looking for anything that seemed out of the ordinary. He found that the signal coming from that very first video IC he replaced was out of spec; as it turned out he had replaced the original 74C04 with a 74**LS**04, but what he actually needed was a 74**HC**04.

With the correct chip in place the image stabilized, and the machine was finally in the "fully functional" condition that he was promised in the first place.

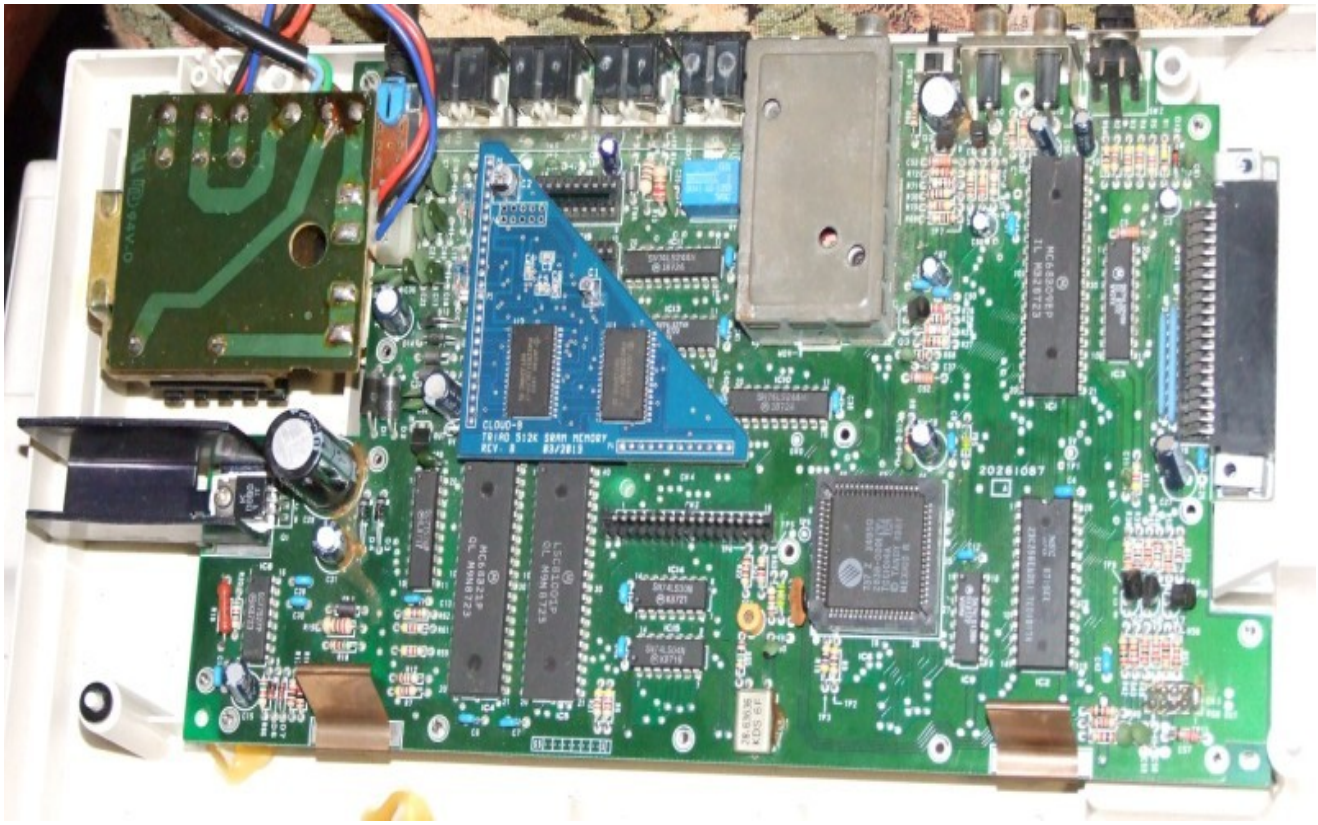
Ultimately [Vlado] had set out to get his hands on a TRS-80 so he could get a better understanding of how it inspired Voja Antonić's design of the Galaksija. Considering the amount of effort he put into this repair, we're willing to bet he got all the insight he was looking for. Especially when combined with the experience of building his own Galaksija replica.

Incidentally, if you take a look at our video from the VCF swap meet, you can actually see the TRS-80 [Vlado] ended up buying at around the 0:50 mark. If you happen to have grabbed one of the other computers covered in that video and got it back up and running, we'd love to hear about it **

*(** How about it everyone? If you haven't much time to spare, just email me a couple of photos and/or a link to find the info. Don't keep you adventures hidden!! ED.)*

IN MAV'S WORKSHOP
Upgrading the Coco 3
Re-cap and CPU upgrade

by Ian Mavric



Last time we looked at my American Color Computer 3 which found it's way to me in Australia and to use it here I converted the 120V power supply to 240V. This month I am completing the upgrade by replacing the problem capacitors and upgrading the CPU with a Hitachi 63C09 "Super-6809".

CAPACITORS.... TARGETED OR 'SHOTGUN' ATTACK?

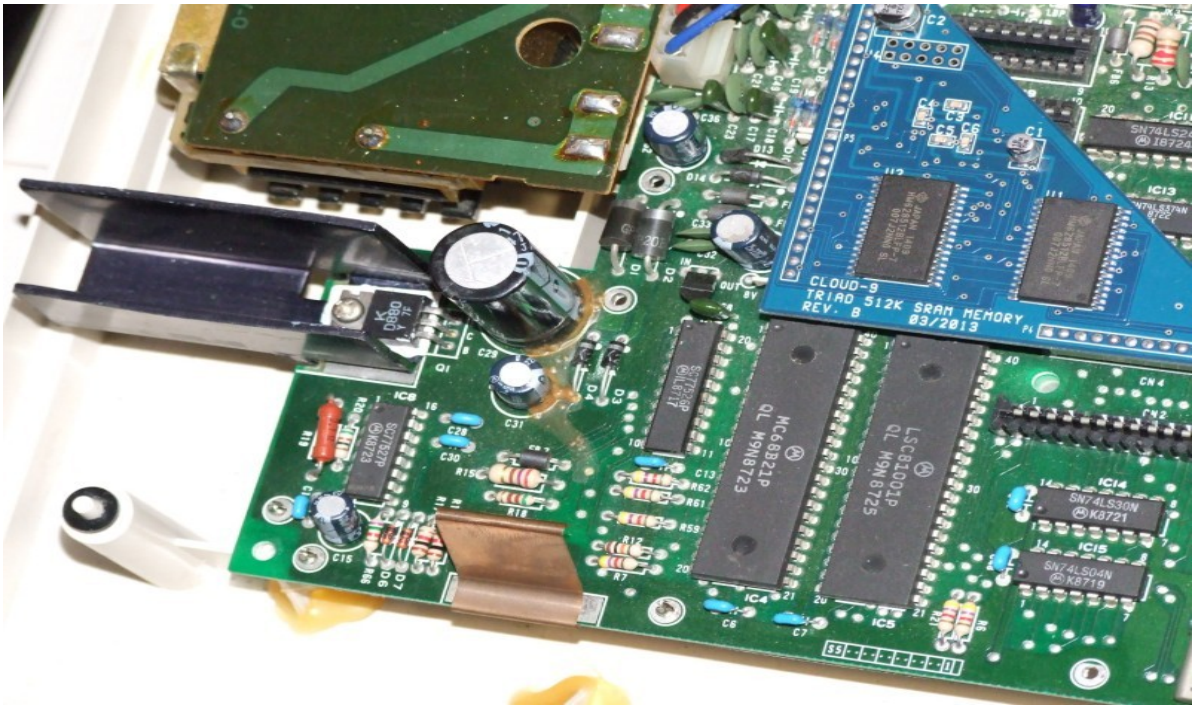
Re-capping computers is a hot topic in vintage computer circles. Basically capacitors, especially the electrolytic types (e-caps), tend to degrade over years and decades and drift away from their designed specification.
Once they move far enough away they don't do the job they are designed to do which makes the computer produce unexpected results. Cheaper e -caps also can buldge, crack and leak out, causing damage over time. As computers pass 30 years old these should be checked and any problems not ignored, and rectified in a timely manner.

Shot gunning is so-called because in this technique every capacitor is just changed out for new ones without checking the old ones. My opinion is that this is overkill, time-consuming and expensive. Some motherboards like those on the TRS-80 Model 4P and Tandy 2000 are enormous and historically don't have any e-caps of high-concern. One the plus side you do end up with a computer full of new e-caps and you don't have to worry about it again for another 2 or 3 decades.

In a targeted attack fewer e-caps are replaced. Usually those that are known anecdotally or historically to be problematic, but also those which are tested to have started drifting away from their proper values. Often you can replace a smaller number e-caps and end up with a machine as reliable as one that has been shot-gunned.

RE-CAPPING COLOR COMPUTERS

Rather interestingly, e-cap problems rarely show up on the Color Computer 1 and 2, in fact especially on the Coco 1. I've had Cocos in my workshop to repair for more than 30 decades and never had a Coco 1 need a capacitor replaced. Although I have no proof or way to prove it, I have a theory that since the Coco 1 and 2 were basically designed by engineers which worked on major audio designs at Radio Shack, those computers were fitted with very high quality e-caps. It wasn't until the Color Computer 3 that Radio Shack realised they could save more money per unit sold by putting in cheaper capacitors. The upshot of this is that the Coco 3 is the one most likely to suffer problems with e-caps bulging, leaking, or their values just drifting over time. This makes the Coco 3 the ideal candidate for the targeted re-cap procedure.



Picture: shows all the e-caps of concern on the Coco 3

In the picture above you can see the e-caps in the power supply area of the Coco 3. These, especially the two large ones, are the main culprits for a Coco 3 with unreliable power to either not start up or if it does, do funky things. My experience of repairing Coco 3s over the last 25 years has enabled me to make a list of e-caps of concern in the machine. Replacing these will bring the power-provision circuit back to peak performance:

C29: 4700uf/16V - can be installed upright
C62: 1000uf/16V - must be bent over to clear the keyboard
C63: 1000uF/25V - must be bent over to clear the keyboard
C31: 220uF/35V - can be installed upright
C15, C42: 100uF/16V - must be bent over to clear the keyboard, and 512K Ram expansion
C54: 470uF/16V - can be installed upright
C27: 100uf/16V - can be installed upright

As you can see I also have some notes I refer to in order to make life easier. Some e-caps are better mounted bent over so they can clear things like the keyboard and 512K Ram expansion. As with all re-caps, buy the highest quality that you can afford, and if the e-cap is not available in the voltage rating of the original part, it's safe to go to the next higher voltage. Eg, C54 which is a 470uF/16V... if you can only find a 25V version you can use that. It will very likely be physically bigger but bend it over so everything fits and it will still work. When it is finished it should look like this:

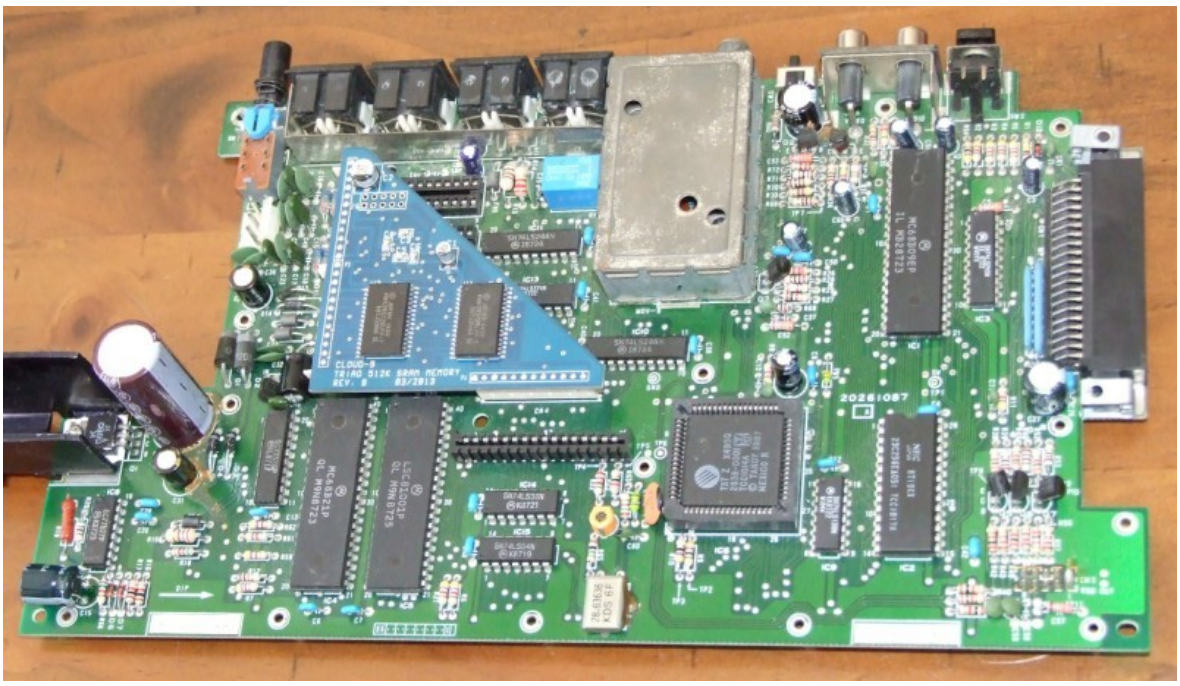


Photo: re-capped, compare it to the photo at the top of the article

IMPORTANT NOTE: Interestingly there are two different Coco 3 motherboard, the US NTSC one and a re-designed one for PAL countries, which have a little PAL decoder added underneath.
The differences in the boards seem to be mainly the component layout has changed slightly in the PAL version (CPU in a different place, for example), but comparing the parts list of both boards most of the parts carry the same part number. C66, which is removed when the system is upgraded to 512K, is up near the joystick ports on the NTSC Coco 3, but on the PAL one is down near the GIME chip, but carries the same part number and does the same job.

ALSO IMPORTANT to remember: that list above hold true for most Coco 3s I have seen but it's always wise to double check the e-caps you are pulling out are not different to what you would expect.
Once I found a system with a 220uF at C63, which I would normally expect to be a 1000uF. Rather than question it I just put a new 220uF in, and suggest you do the same if you encounter this anomaly.

CPU UPGRADE - HITACHI 63C09E - THE SUPER 6809

For a lot of vintage computers a feature called processor upgradeability existed. You may recall upgrading your 386DX-25 to a 386DX-33 or -40 for a modest speed improvement, or your 486DX-33 to a DX2-66 or DX4-100 for a much more noticeable speed improvement.
Back in the PC days you could replace your Intel 8088 with an NEC V20 for a bit of a speed improvement. In the TRS-80 world products like the XLR8er for the Model 4 and 286 Express for the Tandy 1000/A/SX. For the Coco 3 the main and only processor upgrade is removing the 68B09E and replacing it with a Hitachi 63C09E.

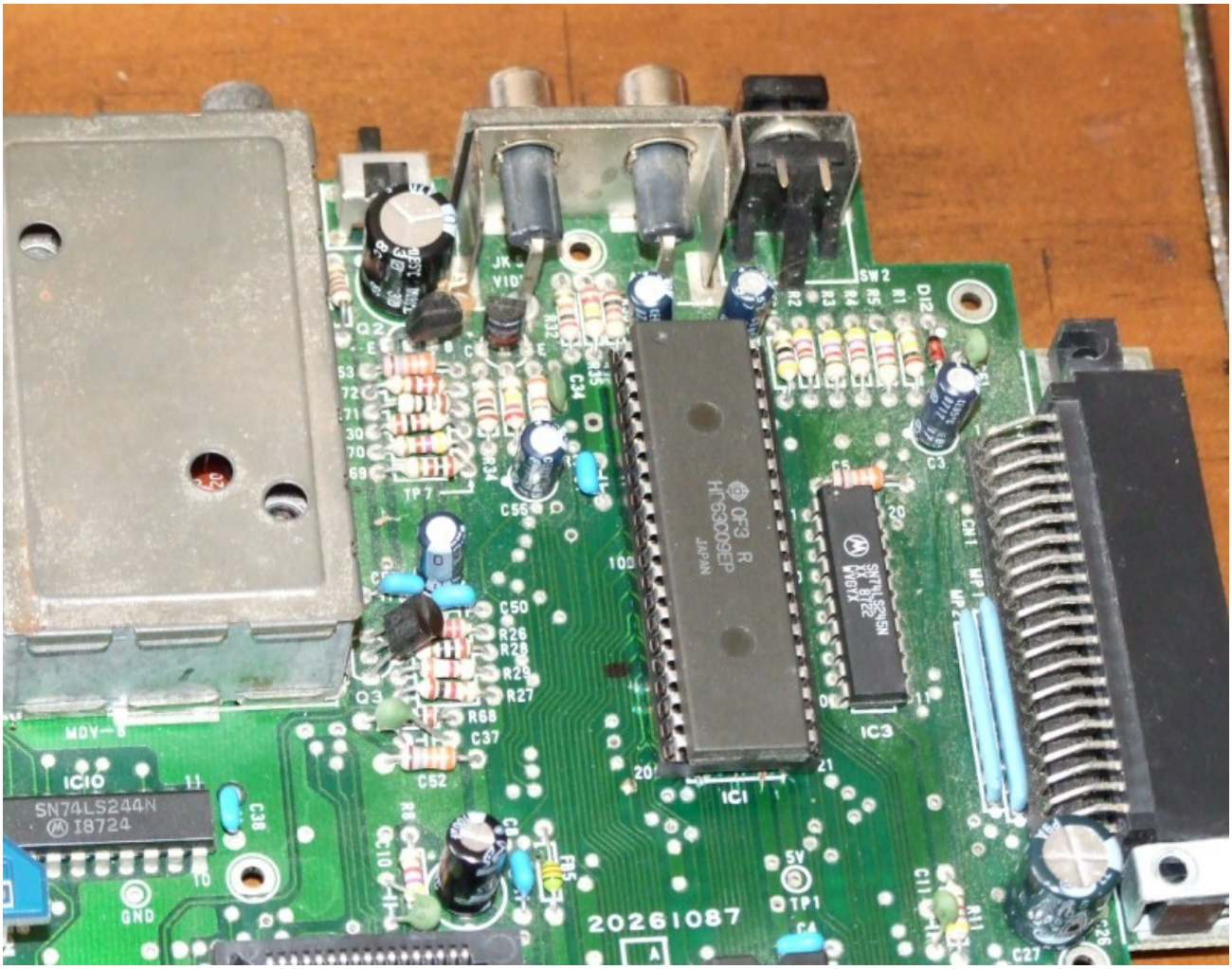
You can google for more information but basically this CPU is considered a super-6809, with a few additional instructions, faster execution of some functions, and it runs cooler than the CPU it replaced.

All Coco 3s have their CPU soldered in, so this must be removed and unfortunately it's a pig of a job.

If you don't have the specialised de-soldering station/soldering re-work station which is temperature controlled you are best to send it to an expert to do.

Overheating and lifting pads are common, and don't even think about trying this with a manual suction pump and cheap 25W soldering iron. You have been warned!

People who have attempted it have sent me their mess of a Coco 3 motherboard and in all cases I have been able to fix it, but the resulting bodge-wire rectification is ugly. The upgrade pictured below took me about 20 minutes with a Hakko temperature-controlled soldering rework station:



Picture: CPU upgraded to a socketed Hitachi HD63C09EP

I should have stated earlier, but better late than never.. the purpose of this upgrade is that more and more new software for the Coco 3 takes advantage of the 6309, but also the popular upgrade to OS/9 Level II called NitrOS-9 takes advantage of the CPU.

FINISHING OFF

Our Coco 3 is now ready for its new life as a Super Coco 3, and with the upgrades will probably not need any internal attention for another couple of decades.

As a final touch on these systems I like to put in an updated name badge signifying the system is now 512K.

I get mine from Cloud-9 but for people in Australia I have a few spares. A shiny new badge really lifts the appearance of the computer, and long-time Coco 3 people may point out that in fact Radio Shack did not include an updated badge as part of the official 512K upgrade, which I thought was a bit of on oversight on their part. It looks great:



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.

VIDEO GENIE and TRS80

COMPUTERS
Video Genie EG3003 12K Basic 16K RAM
£295.00 + VAT inc. fitted Sound Unit.

EXPANSION
Expansion Box for above £185 + VAT or £225 inc. RS232C 32K S100 Memory Board for above £130 + VAT.

DISK DRIVES
5¼" TEAC 40 Track Disk Drive £225 Single or £400 Dual + VAT.

PRINTERS
EPSON MX80T — FABULOUS PRINTER — used by Sharp, Commodore etc. other Printers on request — from £350 (MX80T) + VAT.

BOOKS
"Disassembled Handbook for TRS80" — A Self Teach — Series of Books — Written by Bob Richardson of New York. Very well received by U.S. Reviewers, **Essential** Reading for TRS80, Video Genie & Radio Hams.

Vol. 1 Decoding the ROM & Calls, etc.	£6.45
Vol. 2 Using Calls, to shorten programs, etc.	£8.75
Vol. 3 D/A — AD, Spooling, etc.	£10.50
Vol. 4 Teletype, Morse Techniques	£14.50
Vol. 5 Voice Systems	TBA

+ Much else in each volume.
Various Books — Lists on application.

SPECIAL OFFER
CREED 7E TELEPRINTER — **INCLUDING INTERFACE** for Video Genie users (TRS80 also available)
VERY FEW LEFT — £65 (£5 fitting) + VAT

THE ROM
Are you fed up of loading a lower case driver every time you switch on? — Want your name inside your computer — Better loading (especially TRS80) — Sick of Bouncy keyboards-s-s! You need our new ROM— Simply remove old — plug in new.
1) Firmware Driver for lower case (little letters)
2) Security Code (Name & Post Code?) — up to 21 characters
3) Improved Tape Loading
4) Alleviates Repeating Characters (Keybounce)
5) Checks for feature ROMS
6) For V.G. owners it is possible to re-address your Printer Routine — A LA - TRS80!

£14.95 + VAT

FEATURE ROMS
A new Series of ROMS starting with "Feature ROM A"
1) Single key command entry e.g. Shift A = AUTO
2) Flashing Cursor (can be toggled on/off)
3) Repeating Characters — with delay — (Toggle)
4) Machine Code Monitor & Editor
5) System Load & Save for backing up those system programs — uses no RAM so can deal with a **FULL** 16K PROGRAM.

£18.95 + VAT

LOWER CASE HARDWARE
Unplug two ICs Plug in two Modules. Connect three wires — Easy!

£19.00 + VAT

NEW!
PLUG ON UNIT FOR FEATURE ROMS
Now you can simply plug on the back of your computer, insert ROM & AWAY YOU GO.
V.G. £29.95 (inc. ROM A) + VAT. TRS80 £34.50 (inc. ROM A) + VAT
HI-SPEED HS80 CASSETTE
Plug on to 2400 BAUD "GALAXY" in 40 seconds! Full Basic & System save features, Data stores very **MUCH** FASTER!

£42.50 complete + VAT

48K RAM INSIDE FOR V.Genie
Plug in our modules. Connect three wires. **No Piggy-Backing** Keep your old RAM (or sell it!)
Compatible with expansion.

£70.00 + VAT (fitting £10)

General Northern Microcomputers [GNOMIC]
46 Middle Street, Blackhall, Hartlepool, Cleveland. Tel. Peterlee (0783) 863871

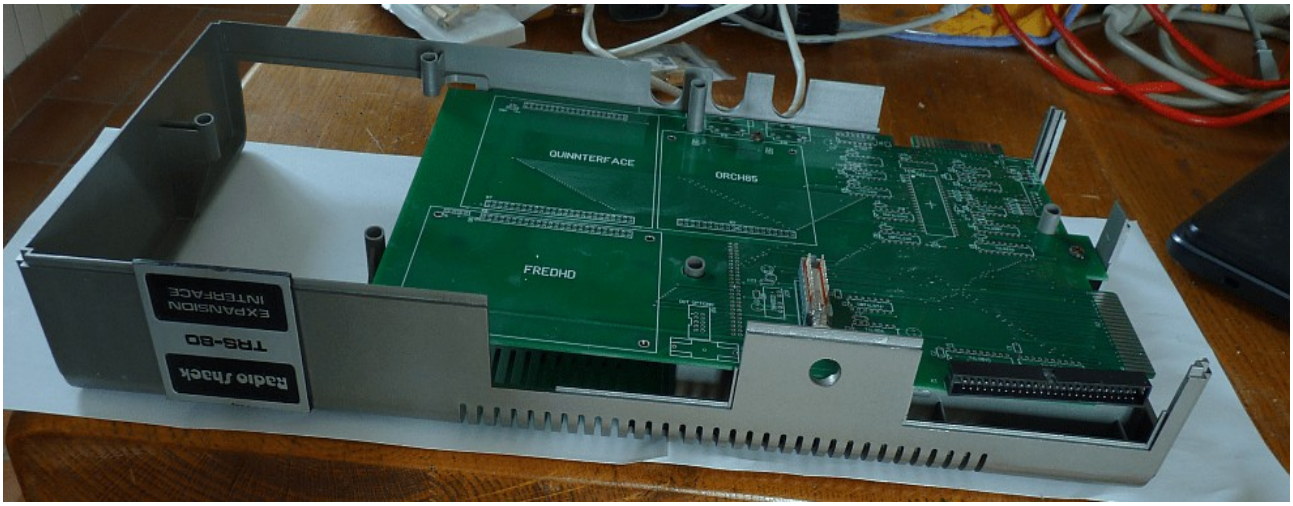
NEWS FROM FRANCE

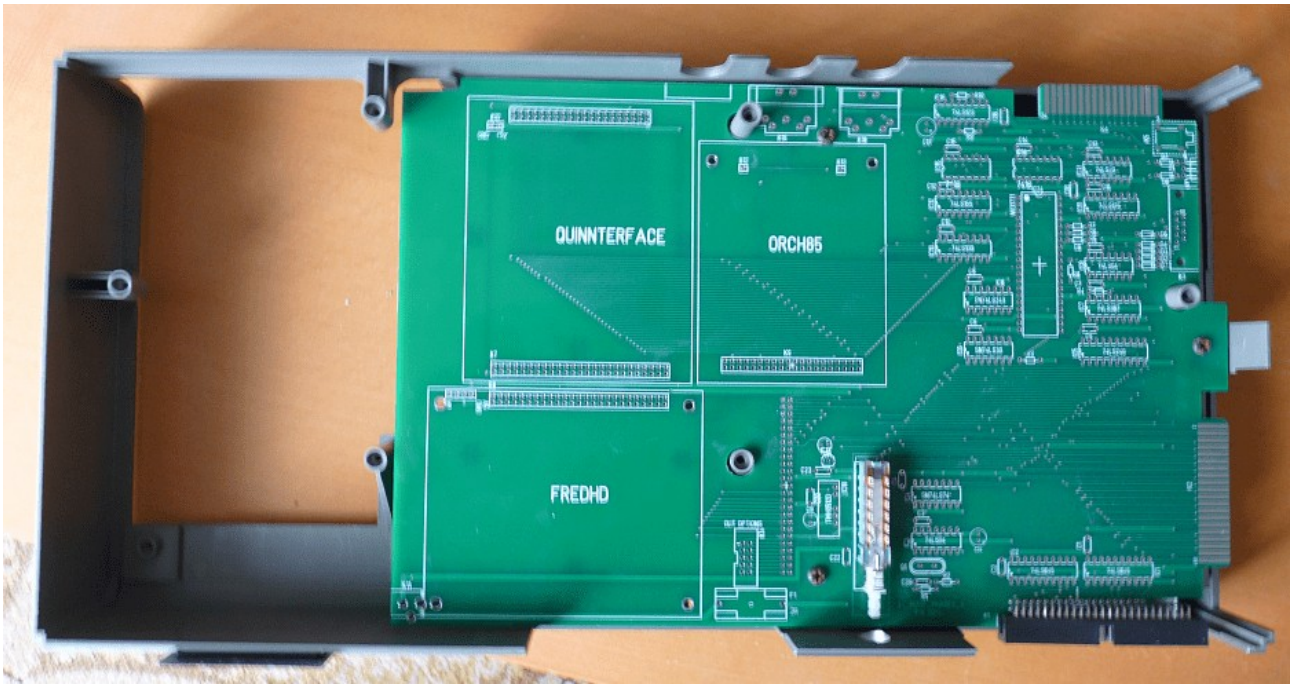
Liberté • Égalité • Fraternité

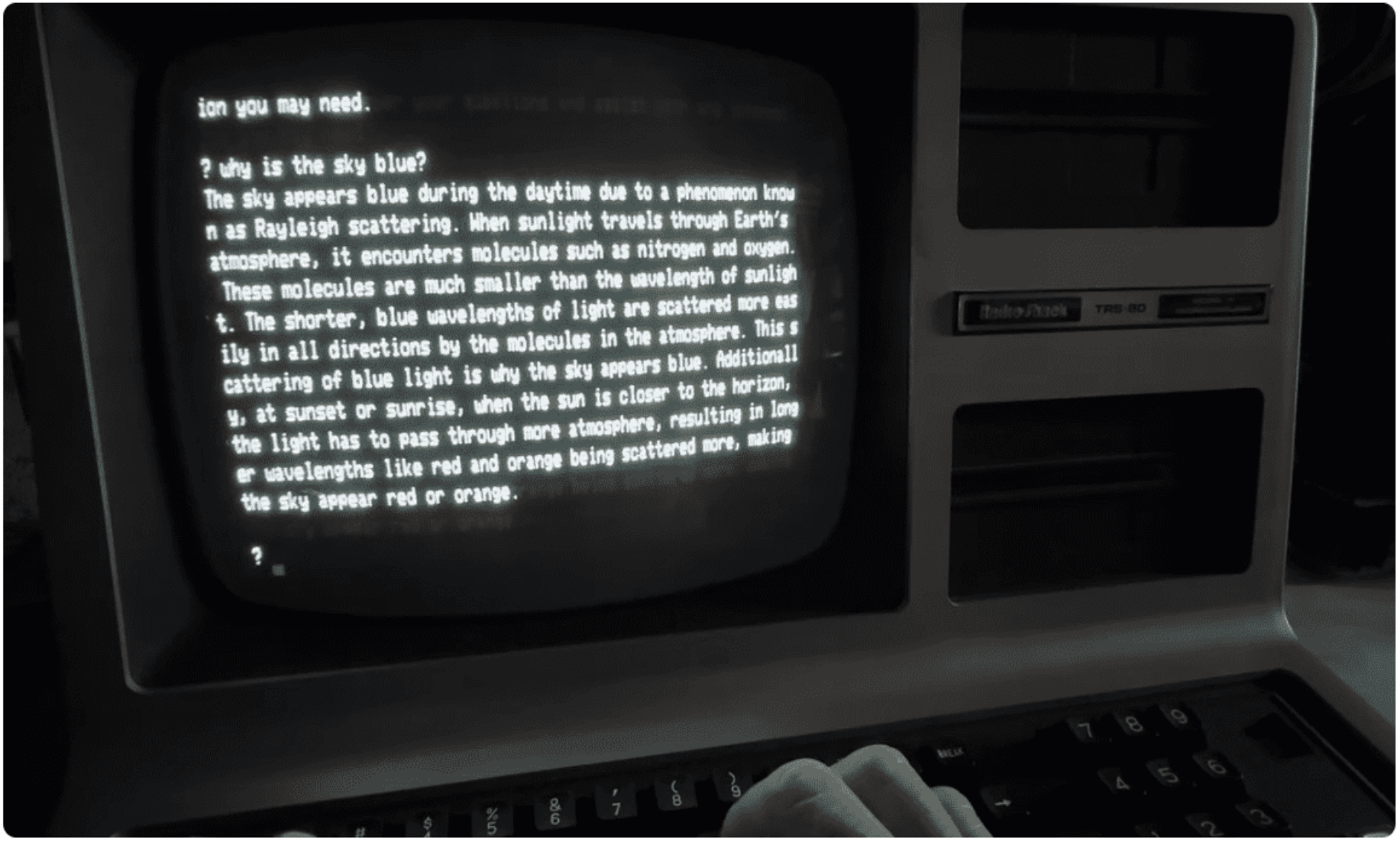
PASCAL HOLDRY

I'm currently on a project that Ian asked me to do. This card will include a Quinnterface + FredHD + ORCHU85 + Floppy + 1 joystick with an extension connector and will be placed in place of the original extension PCB.
PLEASE REMEMBER, this BETA release is 'work in progress' as I've just received the PCB and have already found errors!!!

What a fantastic amount of effort you're putting in Pascal, truly outstanding. ED







Interacting with ChatGPT from a TRS-80 Model III computer

I couldn't resist this image, check it out on YouTube ED.

TRS8BIT 2023 Competition

0K Z-80 Disassembler

A disassembler is a great addition to a debugger. It can show actual instructions instead of hexadecimal opcodes which, if you're like me, are quite tedious to disassemble by hand. Since a debugger must coexist with the program being debugged it should be as small as possible lest you get caught without enough space for the program and the debugger. Waste not, want not, as the old saying goes.

A Z-80 disassembler literally fitting in zero kilobytes is a tad unlikely. Nonetheless, there's every hope it can be squeezed into something less than 1024 bytes and will round down to zero kilobytes.

The smallest entry based on code size and run-time data space used will be the winner. You can use Z-80, 6809, 6803 or 68000 assembly to program your disassembler subroutine. All entries will be in source code form and will be published at the end of the competition for all to learn from and admire.

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

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

For details on exact requirements, other rules and a running leader board please consult:

http://48k.ca/comp2023.html

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

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



The *BEST* in TRS-80s

Call The Right Stuff

Ask for Ian
The number is +61 416 184 893

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

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





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

WHERE 8 BITS AND 16K STILL RULE OK

TRS8BIT

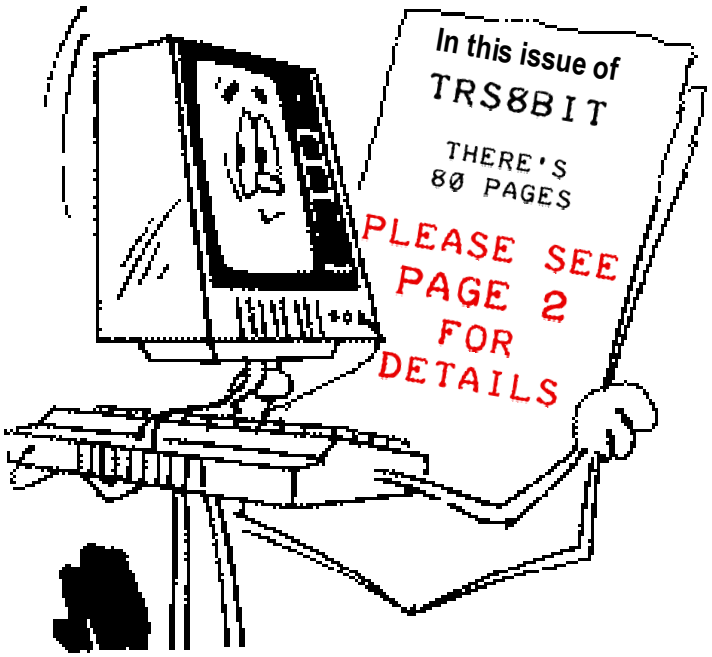
PRODUCED AT WWW.TRS-80.ORG.UK



WOW, ANOTHER XMAS, ANOTHER YEAR AND THE FINAL ISSUE OF VOLUME 17. I CAN'T BELIEVE HOW QUICKLY TIME IS FLYING.
ANYWAY, AS ALWAYS, BOTH DEE AND MYSELF WISH ALL YOU TANDY FANS OUT THERE A V E R Y M E R R Y CHRISTMAS AND A HAPPY 'TANDYING' NEW YEAR.

MANY CONGRATULATION GO TO
MATT BOYTIM
ON WINNING THE 2023 TRS8BIT COMPETITION. HE MANAGED TO CRAM A DISASSEMBLER INTO ONLY 887 BYTES. IF YOU WOULD LIKE TO CONTACT MAV DIRECTLY, MATT, TO CHOOSE YOUR PRIZE.

GEORGE PHILLIPS HAS WRITTEN AN ARTICLE OUTLINING JUST HOW THE ENTRIES STACKED UP THROUGHOUT THE YEAR AND MATT HAS WRITTEN A SUPER ARTICLE SHOWING US ALL JUST HOW HE MANAGED TO ACCOMPLISH THE TASK.



TALKING OF ARTICLES FOR THIS EDITION, THERE'S SOME REAL 'CHRISTMAS CRACKERS' FOR YOU TO GET YOUR TEETH INTO OVER THE FESTIVE PERIOD.

ONE OF THE MOST UNUSUAL ITEMS IS NEWS OF A MAJOR UPGRADE TO LEVEL 2.5 BASIC. (TWOHAF). IT STARTED OUT WITH YOUR EDITOR 'PLAYING' WITH A.I., 'CHATGPT', AS YOU CAN READ ON PAGE 34.
THEN, WITH GEORGE PHILLIPS'S HELP AND ENCOURAGEMENT IT LED TO THE UPGRADE OF TWOHAF. IT'S A VERY USEFUL UPGRADE TO THE STANDARD LEVEL 2 ALLOWING CASSETTE USERS TO ENJOY MANY OF THE BENEFITS OF DISK BASIC. IF LIKE ME, YOU'RE A LEWIS ROSENFELDER FAN, JUST THINK OF ALL THOSE 'DEF FN' THAT ARE NOW AVAILABLE!
GEORGE EVEN UPGRADED THE ACULAB FLOPPY TAPE VERSION, FOR USE ON TRS80GP!

WE WOULD BOTH BE INTERESTED TO HEAR IF YOU FIND IT USEFUL. ARE THERE ANY LEVEL2 CASSETTE USERS OUT THERE?

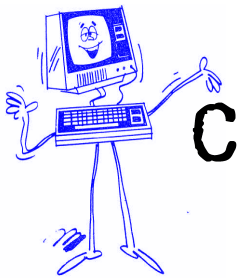
ON THE HARDWARE SIDE OF THINGS, THERE'S NEWS OF VARIOUS YOUTUBE VIDEOS OF INTEREST AND MAV SORTS THE PROBLEM OF RIFA CAPS. ON THE MODEL 2. HE HAS ALSO SENT IN AN ARTICLE SHOWING JUST WHAT A SUPER TIME YOU ALL HAD AT THE RECENT 2023 T.A.

WELL, I THINK THAT CLOSES OUR 17TH YEAR OF TRS8BIT. WILL WE BE HERE FOR THE 18TH? YOU-BET-YA!



TAKE CARE

DUSTY



CONTENTS

PAGE 04 2023 COMPETITION RESULTS
MATT BOYTIM WINS!

PAGE 05 COMPETITION ENTRY DETAILS
GEORGE PHILLIPS

PAGE 07 DETAILS OF THE WINNING ENTRY
MATT BOYTIM

PAGE 25 NEW AND IMPROVED 'STAR-TREK'
AL WALLACE

PAGE 27 IN MAV'S WORKSHOP
M2 RIFA CAPS - HOW TO SORT THEM

PAGE 32 MINIATURE TRS-80'S
TREVOR FLOWERS

PAGE 34 USING A.I. (CHATGPT)

PAGE 38 FOR SALE & WANTED

PAGE 39 THE CGP115 - KEEPING IT ALIVE!
BRIAN K WHITE

PAGE 47 BASIC LEVEL 2.5 (TWOHAF)
ENHANCED - NOW A MUST HAVE UPGRADE TO L2!
GEORGE PHILLIPS

PAGE 48 MIMIC MISTER MEMORY
BASIC ONE-LINER JUST FOR CHRISTMAS FUN

PAGE 51 T.A. 2023 REPORT
THE FULL LOW-DOWN WITH PHOTOS
IAN MAVRIC

PAGE 63 'SURFIN THE NET'
SOME INTERESTING VIDEOS ON YOUTUBE

PAGE 66 THE WAY WE WERE
A LOOK BACK TO THE DAY

PAGE 72 HANDY HINTS
FROM THE 1950'S. LAST OF THE SERIES

PAGE 74 U.K. EBAY BARGAINS
WERE YOU ONE OF THE LUCKY ONES?

PAGE 76 IN MAV'S WORKSHOP
CO-CO REPAIRS - PIA CHIP MIX-UP
IAN MAVRIC

LATEST UPGRADE AVAILABLE
VERSION 2.5.2

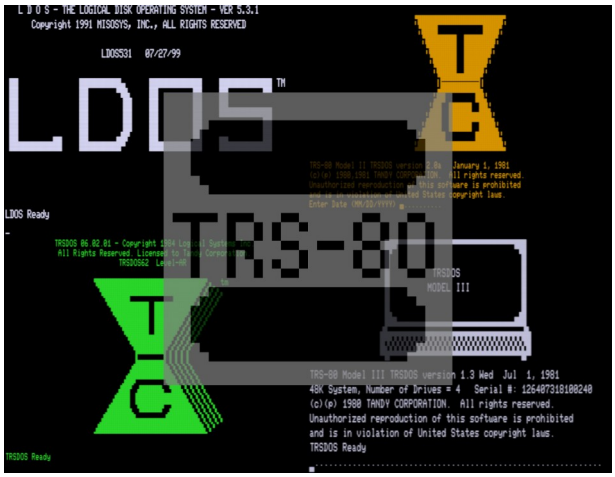
DON'T HAVE A TRS-80?

CRAZY PRICES ON EBAY?

TIRED OF RIFA CAPS BLOWING?

OF FLOPPIES SHEDDING OXIDE?

OR JUST PLAIN DEAD CHIPS?





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



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

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

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

NEW VERSION 2.5.2 IS OUT **NOW** FEATURING:
HARD DRIVE SUPPORT IN NEWDOS/80 2.5
Z-180 CPU FOR INITIAL XLR8ER SUPPORT.

NUMEROUS BUG FIXES AND TWEAKS.

```
Current bank status: (1=Available, 0=Unavailable)

Bank # ==> 0  1  2  3  4  5  6  7  8  9 10
          ================================
Status ==> 1  1  1  0  0  0  0  0  0  0  0

TRSDOS Ready
fixbank

FIXBANK - 4.0.0 - Copyright (C) 1986 by Michael B. Carter.
Licensed to M.A.N. Systems.  All rights reserved.

TRSDOS Ready
bankstat

Current bank status: (1=Available, 0=Unavailable)

Bank # ==> 0  1  2  3  4  5  6  7  8  9 10
          ================================
Status ==> 1  1  1  1  1  1  1  1  1  1  1

TRSDOS Ready
```

```
FORMAT PROGRAM FOR APPARAT HARD DISK DRIVES

HARD DISK DRIVE RELATIVE NUMBER?  (0 - 3)  0
RELATIVE NUMBER OF FIRST HEAD TO FORMAT?  (0 - 7)  0
NUMBER OF HEADS TO FORMAT?  (1 - 8)  4
NUMBER OF CYLINDERS THIS DRIVE HAS?  (1 - 1024)  153
REL NUMBER OF FIRST CYLINDER TO FORMAT?  (0 - CC-1)  0
NUMBER OF CYLINDERS TO FORMAT?  (1 - CC)  153
TRACK STEPPING RATE CODE?  (0 - 15)  0
MAXIMUM ALLOWED 256 BYTE SECTORS PER TRACK IS 33
DATA SECTORS PER TRACK?  32
1  MAXIMUM ERROR SECTORS PER TRACK.
SECTOR INTERLEAVE COUNT?  (0 TO DSPT-1)  3
READY TO FORMAT? (Y OR N)  Y
FORMATTING
```

The 2023 TRS8BIT Competition



Is...

Matt Boytim

WELL DONE MATT, MANAGING TO PRODUCE A DISASSEMBLER IN ONLY 887 BYTES.
FURTHER DETAILS ON PAGE 7

2023 Competition Results
George Phillips

The challenge for 2023 was to write the smallest possible Z-80 disassembler. I'd hoped that the competitors would make one less than 1024 bytes in size which, after rounding down, could be cheekily referred to as a “0K” disassembler. I was not disappointed.

The race started off slowly with no entries coming in until Peter Phillips submitted one in mid October. Shortly thereafter Matt Boytim put in his first attempt and after a bit of warmup the lead between them flipped back and forth right up until the very last hours. Luckily for them Ven Reddy joined in with only a week left and just didn't have enough time to challenge the lead. The winner was:

Matt Boytim with a stunning 887 byte entry.

Well done, Matt. All the entries are up for everyone's reading pleasure on github:

https://github.com/gp48k/z80-dis-tiny

See Matt's article in this newsletter for an in-depth explanation of his program.

In setting up the rules I tried to strike a balance between letting the programmers exercise maximum cleverness and keeping the results at least within earshot of practicality. Limiting the program to having only two contiguous areas of memory was a bit part of that. Scattering the code all over memory can simplify dispatching but is hardly saving memory.
I also decided to count the stack space used as part of the program to avoid egregious use of the stack to store uncompressed data or other tricks. The hope was that any especially onerous memory saving techniques could be expanded a bit to obtain a still small but practical Z-80 disassembler. I'd say that is the case with the contest entries.

The following table shows all the entries in the order they were submitted. The size of an entry is at least the sum of the size of the code, data and stack usage. In some cases it is a bit more as empty space between regions was included to satisfy the “two contiguous regions” rule. Cycles/Line gives a rough average of how long it takes the disassembler to process a single instruction.

It is, of course, irrelevant to the competition which only takes size into account but gives some insight into how the programs function and evolve. You can see how the first few of Peter's entries hit that sweet spot of size optimization where reducing the size of the program goes hand-in-hand with a reduction in running time. At least for a while, less code and data means less to do. That changes with a sudden big increase where clearly some big trade-off of speed for size happened.

Entrant	Size	Code	Data	Stack	Cycles/Line
Peter	1638	696	932	10	18232
Matt	1889	1294	587	8	3288
Peter	1232	460	762	10	18555
Matt	1349	816	515	18	4752
Matt	1326	793	515	18	4671
Matt	1200	667	515	18	4885
Peter	1166	471	685	10	13616
Peter	1040	472	558	10	12500
Matt	1028	507	501	20	5702
Peter	1018	450	558	10	12038
Matt	999	490	489	20	5715
Peter	997	434	553	10	19283
Matt	964	462	482	20	5428
Peter	938	477	451	10	18110
Matt	929	389	522	18	4638
Peter	920	480	430	10	17956
Ven	2103	1763	301	14	3583
Ven	2035	1699	299	12	3640
Ven	1933	1593	297	14	3717
Peter	900	468	420	12	20577
Peter	898	466	420	12	20576
Matt	887	349	520	18	4828

What is also evident is the reduction in size slowed down over time. It was quite clear that ever more effort was needed finding bytes to save. It would be tempting to conclude that the entries were converging asymptotically to some optimal size. I don't think that's true. It may be true that the particular approaches were running out of steam. Or maybe that they simple ran out of time.

So while the competition itself has ended I hope this isn't the end of finding ever smaller Z-80 disassemblers. I welcome anyone out there to give it a try and submit their smaller versions to the github repository.

Though be warned; shaving bytes off code is like eating potato chips – once you start it is very hard to stop.

Matt Boytim

Shows how it was achieved!

This article is about my entry to the 2023 trs8bit competition "0k Z-80 Disassembler". Actually I didn't intend to participate this year - not from lack of interest but like many others lack of time. But one lazy Sunday I thought well let's see what I can knock out in a day and submit that just for a participation medal - which was my initial 1881 byte entry. But of course with that was a lot of low hanging optimization fruit so I knew I could do better which lead to another submission - and another, and... After a few iterations of this it became addictive and I found myself fully immersed in the competition.

To assist George Phillips with his trs8bit article following the competition I put together this explanation of my submission. But it came out like an article and George suggested I just submit it as an article - so here it is.

Before going further though I want thank George for his work in facilitating the competition. Sometimes it's hard to know just how to get started on something but he provided some examples and simple test framework that really lowered the barriers to getting started and participating. This was also the case for last year's Wordle competition. If you didn't look at George's downloads for these competitions you should. And of course thanks to Dusty for hosting the competition, and Ian for supplying the fabulous prizes.

Included with this article is the source code for my final submission - with comments added. Comments starting with ;; were added after the competition for the purpose of this article. Comments starting with just ; were comments in the actual submission.

I don't regard this is a good learning example for z80 assembly. There is a lot of good programming practice that was completely ignored to save space and time. Good progamming is usually a compromise between size, speed, readability, maintainabilty, etc - all of which but the first are absent here. So just keep that in mind. And so without further ado...

A disassembler is essentially a number (machine code) to string (assembly code) converter so it should be no surprise that the workhorse of my disassembler is a printf-like function called out_str (output string) plus two variants out_str_arg (with argument) and out_str_ind (indexed). The 'format' specifiers are single character codes that are otherwise unused in the strings and replaced with either the formatted input argument or formatted data taken directly from machine code input. A sequential set of 'format' specifiers was desired for decoding convenience and a somewhat large number was needed. The character range '3' through '@' is used which allows for up to 14 of which only 13 are used. The mapping is:

'3' - out_reg
'4' - hexb_at_hl
'5' - hexw_at_hl
'6' - out_hlixiy
'7' - out_rp0
'8' - out_rp

'9' - hexrel_at_hl
':' - hex_a
';' - out_cond_hi
'<' - out_bit
'=' - out_reg_hi
'>' - hexofs_at_hl
'?' - out_cond

The ones suffixed with '_at_hl' take their value directly from the machine code input (next byte at the PC which for the disassembler is at HL), otherwise they take their value from the argument passed to out_str_arg:

out_reg:
Output register A, B, C, D, E, H, L, (HL).

hexb_at_hl:
Output hex byte value $NN at the PC.

hexw_at_hl:
Similar to hexb_at_hl already described - output hex WORD value $NNNN at the PC and PC+1.

out_hlixiy:
Output the register pair HL, IX, or IY. IX and IY and as HL in many instructions so this generates 'HL' unless IX or IY is acting as HL then it outputs 'IX'or'IY'. HL/IX/IY is discussed in more detail below.

out_rp0:
Output register pair AF, BC, DE, HL. Some instructions use SP instead of AF - for example 'INC SP'.

out_rp:
Output register pair SP, BC, DE, HL. Most instructions use AF not SP - for example 'PUSH AF'.

hexrel_at_hl:
Output relative branch target of JR or DJNZ. This is a two byte value computed as the PC plus the signed byte at the PC, printed as $NNNN.

hex_a:
Output the hex byte value of the argument $NN.

out_cond_hi:
Output the condition flags for conditional instructions NZ, Z, NC, C, NV, V, P, M. In some instructions the condition field is in upper bits and in others lower bits - this is for the ones which use the upper bits.

out_bit:
Output a bit position value as single digit 0 to 7 with no $ prefix.

out_reg_hi:
Similar to out_reg. Output register A, B, C, D, E, H, L, (HL) but specified by the upper bits.

hexofs_at_hl:
Output a signed hex byte offset value at the PC. This is used for indexed addressing where the IX/IY index offset ranges from -128 decimal to +127 decimal. Printed as '-' or '+' followed by $NN value ranging from $00 (0) to $80 (128).

out_cond:
Similar to out_cond_hi. Output the condition flags for conditional instructions - for the instructions which use the lower bits.

The strings with the format specifiers are then 'template strings' for the z80 assembly instructions. A few examples follow.

In the simplest case the template string is the full assembly instruction. Examples are:

"NOP"
"HALT"
"EXX"
"IM 0"
"IM 1"
"IM 2"

However most instructions use registers and/or data. Examples are:

The entire "LD r,s" 8-bit load group:
"LD =,3"

The entire "LD r,$nn" 8-bit load immediate group:
"LD =,4"

The entire "LD rp,$nnnn" 16-bit load immediate group:
"LD 8,5"

The DJNZ instruction:
"DJNZ 9"

The "JR cond,$nnnn" conditional relative branch instructions:
"JR ?,9"

From these few examples and the complete list of format codes above it should be apparent that template strings can be constructed for all of the z80 instructions.

It is worth noting that the choice of template strings isn't unique. For example the three "IM 0", "IM 1", "IM 2" instructions could be represented with a single template "IM <" but decoding the three to a common template and constructing the necessary argument would take more code than would be saved by eliminating the near duplicate strings. But in most cases it's more efficient to decode a group of instructions to a common template which should be apparent from the LD group examples.

So ideally a z80 instruction could be disassembled by reading the opcode, selecting the corresponding template string, constructing the out_str_arg argument, and calling either out_str or out_str_arg to generate the assembly output. And for the most part that is how the disassembler works.

Including the format codes, less than 64 characters are used in the strings (primarily because both upper and lower case aren't used) so the characters can be represented with only 6 bits which leaves room for two flags per character. Many instructions include a space so one flag is used for space compression - if set then a space follows the character. And since the strings are variable length the other flag is used to indicate the last character of the string. This precludes expressing the strings as natural string literals in the disassembler code so a less natural format is used where each character is expressed as a separate byte. Here are a few examples of how the strings are represented in the code:

```
"LD =,4"
defb    'L'-32,'D'-32+64,'='-32,','-32','4'-32+128

"DJNZ 9"
defb    'D'-32,'J'-32,'N'-32,'Z'-32+64,'9'-32+128

"ADC 6,7"
defb    'A'-32,'D'-32,'C'-32+64,'6'-32,','-32,'7'-32+128
```

But for the purpose of discussion I will continue to express the strings as natural strings with the understanding that they are stored this way in the code.

out_str_ind that was already mentioned selects the string at the specified index from a list of strings. This can be used as a decoding 'helper' - rather than fully decode a set of instructions to their corresponding individual templates the entire group can be decoded to a single string list. For example the entire 8-bit arithmetic and logic group can be decoded from a single indexed string list:

```
"ADD A,3"
"ADC A,3"
"SUB 3"
"SBC A,3"
"AND 3"
"XOR 3"
"OR 3"
"CP 3"
```

where the index value corresponds to the arithmetic/logical operation. out_str_ind in turn calls out_str with the selected string.

Since the goal of calling an out_str function is to output the complete assembly instruction these is seldom need to return to the caller of out_str - it can just return to the caller of the disassembler. This is convenient because the string argument to the out_str functions is passed on the stack. This allows the string literal to immediately follow the RST (CALL) instruction and no explicit stacking of the string argument is required. For example the call or the "LD r,s" example is:

```
isld
        rst     out_str_arg
        defb    'L'-32,'D'-32+64,'='-32,','-32,'3'-32+128
```

out_str_arg does not return here - the RST (CALL) is effectively a branch (JP) with argument on the stack.

However, in cases where a return is needed it is a little tricky because the return address has to be stacked *before* the string argument to out_str functions. There are a few ways this can be accomplished - here is one way:

```
        ld      bc,ret_here
        push    bc
        rst     out_str_arg
        defb    'L'-32,'D'-32+64,'='-32,','-32+128
ret_here
        ...
```

One interesting detail is that some of the format character replacement routines which are called from out_str themselves use out_str - hence out_str is recursive. For example the '?' condition replacement calls out_str_ind with the following list:

```
"NZ"
"Z"
"NC"
"C"
"NV"
"V"
"P"
"M"
```

One of the more interesting ones is the '3' and '=' register replacement:

```
"A"
"B"
"C"
"D"
"E"
"H"
"L"
"(HL)"
"(IX>)"
"(IY>)"
```

where the out_str_ind index is adjusted when IX or IY is in use instead of HL, and the subsequent out_str call replaces the '>' with the IX/IY offset.

The opcode decoding which leads to the out_str calls takes advantage of the z80 opcode format. The upper two bits divide the instruction set into four groups, and the remaining six bits organize the instructions into rows and columns. That is, the opcodes are of the form:

ggrrrccc

Within the rows and columns there are varying amounts of regularity. In one case (the 8-bit load group) the entire group has a regular structure (except for the HALT instruction which is a special case). In some cases an entire column has regularity, and in other cases there is partial regularity, and in some cases not much recognizable regularity at all. Consequently the opcode decoding varies depending on this regularity. In general the opcodes are decoded to the point that out_str_ind can finish the job.

The workhorse of the opcode decoding is a dispatch routine which jumps to the corresponding handler for the opcode. Initially I used a dispatch routine that used a DJNZ construct that branched around the handlers rather than to them which resulted in a two byte construct instead of three (DEC, JR Z). A jump table might have seemed more efficient but the groups being decoded were small enough that the jump table overhead seemed to dominate.

What I failed to recognize though was that the jump table would replace all of the dispatch routines not just one. But when amortizing the overhead across all dispatch routines the jump table approach was more favorable. Once I realized that I adopted a jump table approach.

The jump table consists of single byte jump offsets - similar to the JR instruction except for simplicity the jump offsets are positive only so jumps can be forward not backward, but with a 'reach' of 255.

For further efficiency, the branch offsets in the table are relative to the table entry itself. This allows the branch target to be easily computed by adding the offset read from the table to the address it was read from. Essentially like:

```
jrtbl:  ; a = table index
        ld      b,0
        ld      hl,table
        ld      c,a
        add     hl,bc
        ld      c,(hl)
        add     hl,bc
        jmp     (hl)'
```

This also allows the table to be placed anywhere (for example doesn't require strategic placement at a 256 byte boundary).

A jump table is also used in decoding of the out_str 'format' characters.

The disassembler is allowed to be ORG'ed at 0 to allow use of the z80 RST instructions. RST's are 1-byte calls instead of 3-bytes. This obviously can save a lot of memory for frequently called subroutines. But less obvious this can also be used for frequently occurring constructs too short for ordinary call's. For example, if even the 2-byte construct

```
        ld      a,(hl)
        inc     a
```

is replaced with a RST it saves a byte for each occurrence.

There are 8 RST's - 'RST $0', 'RST $8', ..., 'RST $38' Use of the RST's is a bit tricky though because as they are spaced 8 bytes apart which leaves wasted bytes for very short uses such as the 'LD A,(HL)' example above (which including the RET not shown is 3 bytes), and is too small for any substantial subroutine (so a jump must be added before the next RST). Use of the RST's essentially have a fixed cost which must be offset by corresponding savings which consists of tucking useful code into the RST region and of course through use of the RST routines themselves.

Because the holes between RST's can be too small to fill with useful code and the cost is fixed (those bytes count whether you use them or not) I included some small functionality in some of the RST's that save a few bytes elsewhere. For example, in the 'LD A,(HL)' RST I load B with $40 and C $80 which are convenient constants to have available.

```
rst20:
ld_a_hl
        ld      a,(hl)
        inc     hl
        ld      bc,$4080
        ret
```

So a side effect of reading a byte from the input buffer is that B and C are loaded with these constants.

Because of this difficulty of well utilizing the RST's I used only 6 of the available 8:

```
rst0:
z80dis:

rst10:
rrca3
        rrca
        rrca
        rrca
        and     7
        bit     2,a
        ret

rst18:
jrtbl8  ld      b,a
        and     7
jrtbl

rst20:
ld_a_hl

rst28:
out_str_ind

rst38:
out_str
out_str_arg
```

The function of most of the RST's should be clear from the above discussion. The ones which aren't are at least partially shown to provide some additional context. rrca3 is used to shift the 'rrr' row field of the opcode to the lsb position which is done somewhat frequently, with the side effect of testing bit 2 of the result.

'RST $0' isn't actually used in the disassembler but the disassembler can be called with a 'RST $0' saving the caller 2 bytes - but I didn't get a 2-byte credit toward the competition for that.

out_str and out_str_arg are the same function because if the template string doesn't reference an argument then the argument value doesn't matter and it doesn't hurt for call out_str_arg when out_str will do. Both function names are used though to emphasize whether an argument is being passed or not.

Just when you think you're done, there are four extended opcodes in the z80 instruction set. Two of the extended opcodes specify the use of use IX or IY instead of HL. That is, where HL would be used IX or IY (or IX+/-$nn or IY+/-$nn) is used instead. These opcodes are referred to as IX and IY prefixes. These are disassembled by flagging the use of IX or IY and calling the disassembler. That is, for these cases the whole disassembler is itself recursive. This can be seen at the beginning of the disassembler code:

```
z80dis:
        xor     a               ;; normal entry (A=0)
        ld      iyl,a           ;IYL=isixiy ;; prefixed IX/IY entry (A=1,2)
```

IYL is an undocumented feature of the z80 - it is the low part of the IY register. IYL is used by the disassembler to store the HL/IX/IY flag. On normal entry A is zeroed and saved to IYL so HL is used, but when called from the IX/IY prefix handler A contains either 1 for IX or 2 for IY and the 'XOR A' is skipped.

The other two extended opcodes correspond to additional extended instructions which are specified by the second byte. Consequently these are decoded similar to the standard opcodes but using the second byte. Fortunately there is a limited number of extended instructions.

The behavior of the disassembler for illegal opcodes isn't fully specified by the contest rules - only that it doesn't do anything too bad - crash, output unprintable characters, overflow the output buffer, etc. So for the most part illegal opcodes are a don't care. Initially for completeness I made some effort to handle illegal opcodes gracefully but to save memory I later resigned to the natural behavior. The biggest problem with illegal opcodes in my implementation is that the out_str_ind routine could index past the end of the string list leading to unpredictable behavior. To address this a string list terminator was appended to the string lists where this is possible. In some cases the terminator can get appended to the disassembler output so the terminator must correspond to a printable character or characters - and of course must be a character that otherwise doesn't occur in the strings. My preferred terminator is " " (space followed by flagged space):

```
        defb    ' '-32+64+128  ; = $c0
```

But for more compact detection a value of $ff is used which corresponds to "_ " (_ followed by flagged space). $ff can be detected with an 'INC' which saves a byte compared to 'CP $c0'.

To further reduce the size some careful arrangement of the code was done to avoid branching altogether (relying heavily on fall-through) and where branching is necessary to use relative jumps (2-byte JR's) instead of absolute jumps (3-byte JP's).

I used very few registers. The entire disassembler uses almost just A, B, and C. DE and HL are used but almost exclusively as the input/output pointers not for use by the disassembler itself. There are a few places that HL is used, primarily because HL is the 16-bit accumulator so it is handy when 16-bit math is required (you can see it used in the jrtbl routine above). Very few actual z80 instructions are used. To be honest before this contest I didn't realize 'NEG' was an extended (2-byte) z80 instruction which I use in the disassembler and I worked unsuccessfully to replace with a 1-byte instruction (such as SUB).

A few additional registers would sure be helpful though which is why I resorted to IYL for the HL/IX/IY flag as already mentioned. And IYH for saving the out_str_arg argument (not previously mentioned). I could not find a way to effectively use the alternate primed register set (AF', BC', DE', HL'). The main challenge with use of the primed registers is that the non-primed registers become hidden when the primed set is selected and their contents inaccessible - so when the value is needed you have to switch back and forth which counters the potential savings.

I used very few of what I would call programming 'tricks'. Probably the only 'trick' is following a CALL immediately with the subroutine data which as already explained is used with the out_str functions. This 'trick' is also used with the jrtbl function where the jump table immediately follows the RST (CALL).

And that's it. I hope you found this at least somewhat interesting and helpful - and like me curious of what the 2024 competition will be...

I THINK THIS IS A REAL MASTER-CLASS IN ASSEMBLY PROGRAMING! MATT HAS EVEN SUBMITTED FOR PUBLICATION A SOURCE LISTING, WHICH IS SHOWN ON THE NEXT PAGES. SUBJECT TO MATT'S APPROVAL, I WILL HAPPILY SENT A .TXT FILE FOR YOU TO ASSEMBLE. PLEASE EMAIL ME AT THE USUAL ADDRESS. *ED*

```
Z80DIS:
        XOR     A               ;; NORMAL DISASSEMBLER ENTRY POINT A=0
        LD      IYL,A           ;IYL=ISIXIY ;; RE-RENTRY POINT FOR IX/IY PREFIX
A=1,2
        RST     LD_A_HL         ;; GET A=OPCODE, B=$40 SIDE EFFECT
        SUB     B               ;B=$40 ;; TEST FOR OPCODES LESS THAN $40
        JP      C,GRP_00        ;; GO IF OPCODE $00-$3F
        SUB     B               ;; TEST FOR OPCODES LESS THAN $80
        JR      C,GRP_40        ;; GO IF OPCODE $40-$7F
        SUB     B               ;; TEST FOR OPCODES LESS THAN $C0
        JR      C,GRP_80        ;; GO IF OPCODE $80-$BF
        JR      GRP_C0          ;; GO IF OPCODE $C0-$FF

RST10:
RRCA3                           ;; SHIFT RIGHT 3 BITS AND MASK 3 LSB'S
        RRCA
        RRCA
        RRCA
        AND     7               ;; TEST BIT 2 OF RESULT
        BIT     2,A             ;; RETURN TO CALLER
        RET

RST18:                          ;; JUMP TO 8 ENTRY TABLE ON STACK
JRTBL8  LD      B,A             ;; SAVE A TO BE PASSED AS PARAMETER
        AND     7               ;; MASK 3 LSB'S
JRTBL   EX      (SP),HL         ;; GET TABLE ADDRESS, SAVE HL
        LD      C,A             ;; GET TABLE INDEX IN C
        LD      A,B             ;; GET PARAMETER
        JR      JT8Z            ;; JUMP PAST RST REGION AND CONTINUE

RST20:
LD_A_HL                         ;; GET NEXT INPUT BYTE FROM INPUT AT HL
        LD      A,(HL)          ;; A=(HL)
        INC     HL              ;; BUMP INPUT POINTER TO NEXT
        LD      BC,$4080        ;; LOAD B=$40, C=$80 AS SIDE EFFECT
        RET                     ;; RETURN TO CALLER

OSIZ    DEC     A               ;; BUMP STRING INDEX
        PUSH    BC              ;; PUSH BC SO CAN FALL THROUGH TO POP

RST28:
OUT_STR_IND                     ;; OUTPUT STRING FROM TABLE INDEXED BY A
        POP     BC              ;; GET STRING TABLE ADDRESS FROM STACK
OUT_STR_IND_P1
        OR      A               ;; TEST INDEX FOR 0
        JR      Z,OTS0_P1       ;; DONE IF 0 - GO TO OUTPUT STRING
        PUSH    AF              ;; SAVE STRING INDEX TO STACK
OSI0    LD      A,(BC)          ;; SEARCH FOR END STRING - GET NEXT CHAR
        INC     BC              ;; BUMP POINTER
        RLCA                    ;; TEST MSB=1 WHICH MARKS STRING END
        JR      NC,OSI0         ;; LOOP UNTIL END OF STRING
        ;; TEST IF END OF TABLE INCASE INDEX IS GREATER THAN TABLE SIZE
        INC     A ;CP       ('_'-32+64)*2+1 ; DONE IF LIST TERMINATOR
        JR      Z,OTS2          ;; EXIT END OF STRING TABLE
        POP     AF              ;; RESTORE INDEX FROM SACK
        JR      OSIZ            ;; KEEP SEARCHING FOR STRING AT INDEX

RST38:
OUT_STR                         ;; OUTPUT STRING POINTED TO ON STACK
OUT_STR_ARG                     ;;    ...WITH PARAMETER IN A
        LD      IYH,A           ;IYH=OTSARG ;; SAVE PARAMETER IN IYH
OTS0    POP     BC              ;; GET STRING POINTER
OTS0_P1 LD      A,(BC)          ;; GET NEXT CHAR OF STRING PLUS FLAGS
        INC     BC              ;; BUMP POINTER
        PUSH    BC              ;; SAVE POINTER
        PUSH    AF              ;; SAVE FLAGS
        AND     $3F             ;; MASK OFF FLAGS
        ADD     A,32            ;; CONVERT CHAR TO ASCII
        LD      (DE),A          ;; SAVE TO OUTPUT BUFFER (MAY BE OVERWRITTEN)
        INC     DE              ;; BUMP OUTPUT POINTER
        SUB     A,'3'           ;; SUBTRACT FIRST REPLACEMENT CHAR '3'
        CP      '?'-'3'+1       ;; TEST IF IN REPLACEMENT RANGE '3'-'?'
        CALL    C,REPLACE       ;; REPLACE CHAR WITH EXPANSION IF IN RANGE
        POP     AF              ;; RESTORE CHAR WITH FLAGS
        ADD     A,A             ;; TEST FLAGS BIT7=EOS, BIT6=APPEND ' '
        JP      P,OTS1          ;; GO IF NO ' ' TO ADD
        LD      A,' '           ;; APPEND ' ' TO OUTPUT
```

```
        LD      (DE),A          ;; PUT ' ' IN OUTPUT BUFFER
        INC     DE              ;; BUMP OUTPUT POINTER
OTS1    JR      NC,OTSØ         ;; LOOP IF NOT END OF STRING
OTS2    POP     BC              ;; CLEAR STACK
LD_DE_Ø XOR     A               ;; APPEND Ø TO OUTPUT BUFFER
        LD      (DE),A          ;; PUT NULL IN OUTPUT BUFFER
        ;; NOTE THE RETURN ADDRESS WAS USED AS THE STRING POINTER
        RET                     ;; RETURN TO PREVIOUS CALLER

GRP_4Ø                          ;; HANDLE OPCODES $4Ø-$7F (LD GROUP)
        CP      $76-$4Ø-$4Ø     ;; TEST FOR HALT (SPECIAL CASE)
        JR      NZ,ISLD         ;; OTHERWISE LD
ISHALT
        RST     OUT_STR         ;; OUTPUT STRING AND RETURN
        DEFB    'H'-32,'A'-32,'L'-32,'T'-32+128  ;; HALT
; MOVE ISLD DOWN SO JR'S WILL REACH
;ISLD
;        RST     OUT_STR_ARG
;        DEFB    'L'-32,'D'-32+64,'='-32,','-32,'3'-32+128

GRP_CØ                          ;; HANDLE OPCODES $CØ-$FF
        RST     JRTBL8          ;; DECODE USING 3 LSB'S AS SUBGROUP
        DEFB    GCØØ-$          ;; OPCODE IS PASSED FOR FURTHER DECODE
        DEFB    GCØ1-$
        DEFB    GCØ2-$
        DEFB    GCØ3-$
        DEFB    GCØ4-$
        DEFB    GCØ5-$
        DEFB    GCØ6-$
        DEFB    GCØ7-$

GCØ5                            ;; HANDLE OPCDES BINARY 11XXY1Ø1
        RST     RRCA3           ;; SHIFT XXY TO LSB'S
        RRA                     ;; SHIFT XX TO LSB'S, TEST Y
        JR      NC,GCØ5Y        ;; GO IF Y=Ø
GCØ5X                           ;; ;; HANDLE OPCODES BINARY 11XX11Ø1
        RST     JRTBL8          ;INSTEAD OF JRTBL SO A IS PASSED
        DEFB    GCØ5Ø-$         ;; JUMP TABLE
        DEFB    GCØ51-$
        DEFB    GRP_ED-$
        DEFB    GCØ53-$

GCØ53   DEC     A               ;FD ;; IY PREFIX
GCØ51   JR      Z8ØDIS+1        ;DD ;; IX PREFIX

GCØ5Ø
ISCALL
        RST     OUT_STR         ;; OUTPUT STRING AND RETURN
        DEFB    'C'-32,'A'-32,'L'-32,'L'-32+64,'5'-32+128

;GCØ52                          ;; NOT NEEDED, JUMP DIRECT FROM TABLE
;        JR      GRP_ED          ;ED ;; $ED PREFIX


; MOVE GCØ5Y DOWN SO JR'S WILL REACH
;GCØ5Y                          ;; HANDLE OPCODES BINARY 11XXØØØ1
;ISPUSH
;        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG AND RETURN
;        DEFB    'P'-32,'U'-32,'S'-32,'H'-32+64,'8'-32+128

; TUCKED JRTBL8 TAIL IN HERE SO JR'S WILL REACH
JT8Z    LD      B,Ø             ;; BC=OFFSET INTO TABLE
        ADD     HL,BC           ;; ADD OFFSET TO TABLE START
        LD      C,(HL)          ;; GET OFFSET FROM HERE TO JUMP TO
        ADD     HL,BC           ;; ADD OFFSET TO CURRENT TABLE ENTRY
        EX      (SP),HL         ;; RESTORE HL, JUMP ADDRESS TO STACK
        RET                     ;; JUMP TO ADDRESS CROM TABLE

GCØ6                            ;C IS CLR ;; CARRY IS USED AS FLAG
GRP_8Ø                          ;C IS SET ;; CLR OR SET DEPENDING ON HOW GOT HERE
        PUSH    AF              ;SAVE C ;; SAVE CARRY
        CALL    ISALUX          ;; TRICK TO USE OUT_STR BUT RETURN TO US
        DEFB    'A'-32,'D'-32,'D'-32+64,'A'-32,','-32+128 ;; STRING TABLE
        DEFB    'A'-32,'D'-32,'C'-32+64,'A'-32,','-32+128
        DEFB    'S'-32,'U'-32,'B'-32+64+128
        DEFB    'S'-32,'B'-32,'C'-32+64,'A'-32,','-32+128
        DEFB    'A'-32,'N'-32,'D'-32+64+128
```

```
        DEFB    'X'-32,'O'-32,'R'-32+64+128
        DEFB    'O'-32,'R'-32+64+128
        DEFB    'C'-32,'P'-32+64+128

; TUCKED GCØ5Y IN HERE SO JR'S WILL REACH
GCØ5Y                           ;; HANDLE OPCODES BINARY 11XX1ØØ1
ISPUSH
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG AND RETURN
        DEFB    'P'-32,'U'-32,'S'-32,'H'-32+64,'8'-32+128

; TUCKED ISLD IN HERE SO JR'SWILL REACH
ISLD
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG AND RETURN
        DEFB    'L'-32,'D'-32+64,'='-32,','-32,'3'-32+128

GCØØ                            ;; HANDLE OPCODE BINARY 11XXX000
ISRETC
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG AND RETURN
        DEFB    'R'-32,'E'-32,'T'-32+64,';'-32+128

GCØ1                            ;; HANDLE OPCODES BINARY $11XXY001
        RST     RRCA3           ;; SHIFT XXY TO LSB'S
        RRA                     ;; SHIFT XX TO LSB'S, TEST Y
        JR      C,GCØ1Ø         ;; GO IF Y=1
ISPOP                           ;; HANDLE OPCODES BINARY 11XX0001
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG AND RETURN
        DEFB    'P'-32,'O'-32,'P'-32+64,'8'-32+128

GCØ1Ø   RST     OUT_STR_IND     ;; HANDLE OPCODES BINARY 11XX1ØØ1
        DEFB    'R'-32,'E'-32,'T'-32+128
        DEFB    'E'-32,'X'-32,'X'-32+128
        DEFB    'J'-32,'P'-32+64,'('-32,'6'-32,')'-32+128
        DEFB    'L'-32,'D'-32+64,'S'-32,'P'-32,','-32,'6'-32+128

GCØ2
ISJPC
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG AND RETURN
        DEFB    'J'-32,'P'-32+64,';'-32,','-32,'5'-32+128

GCØ3                            ;; HANDLE OPCODES BINARY 11XXX011
        RST     RRCA3           ;; SHIFT XXX TO LSB'S
        DEC     A               ;; TEST FOR XXX=ØØ1
        JR      NZ,NOT_CB       ;; GO IF XXX!=ØØ1 (NOT $CB)
GRP_CB                          ;CB ;; $CB PREFIX
        CP      A,IYL           ;IYL=ISIXIY, A=Ø ;; A=Ø FROM DEC ABOVE
        RST     LD_A_HL         ;; GET EXT OPCODE FROM INPUT
        JR      Z,GRP_CBX       ;; GO IF NOT IX/IY PREFIXED
        RST     LD_A_HL         ;; ELSE PREFIXED, SKIP OVER OFFSET GET EXT OPCODE
        PUSH    HL              ;; SAVE CURRENT POSITION
        DEC     HL              ;; ADJUST INPUT POINTER TO POINT TO OFFSET
        DEC     HL
        CALL    GRP_CBX         ;; DECODE INSTRUCTION AS PREFIXED
        POP     HL              ;; RESTORE INPUT POINTER
        RET                     ;; RETURN TO CALLER

GRP_CBX
        SUB     B ;$4Ø          ;; B=$4Ø, TEST FOR EXT OPCODES LESS THAN $4Ø
        PUSH    AF              ;SAVE A AND C ;; SAVE OPCODE AND CARRY
        JR      C,GRP_CBØØ      ;; GO IF EXT OPCODE $ØØ-$3F, ELSE BIT INSTR'S
GRP_CBCØ                        ;; BIT INSTRUCTION BIT
GRP_CB8Ø                        ;; BIT INSTRUCTION RST
GRP_CB4Ø                        ;; BIT INSTRUCTION SET
        RLCA                    ;; GET BIT INSTRUCTION IN LSB'S
        RLCA
        AND     3               ;A=Ø,1,2 NOT 3
        CALL    BIT_RES_SET     ;; USE TRICK TO RETURN TO HERE AFTER OUT_STR
        POP     AF              ;; RESTORE OPCODE
        RST     OUT_STR_ARG     ;; OUTPUT REST OR STRING WITH ARG AND RET TO
CALLER
        DEFB    '<'-32,','-32,'3'-32+128

BIT_RES_SET                     ;; TRICK TO RETURN FROM OUT_STR
        RST     OUT_STR_IND     ;; OUTPUT STING AT INDEX
        DEFB    'B'-32,'I'-32,'T'-32+64+128 ;; STRING TABLE
        DEFB    'R'-32,'E'-32,'S'-32+64+128
        DEFB    'S'-32,'E'-32,'T'-32+64+128
```

```
GRP_CB00                        ;C IS SET IN AF ON STACK
        CALL    ISCBALUX        ;; TRICK TO RETURN FROM OUT_STR
        DEFB    'R'-32,'L'-32,'C'-32+64+128
        DEFB    'R'-32,'R'-32,'C'-32+64+128
        DEFB    'R'-32,'L'-32+64+128
        DEFB    'R'-32,'R'-32+64+128
        DEFB    'S'-32,'L'-32,'A'-32+64+128
        DEFB    'S'-32,'R'-32,'A'-32+64+128
        DEFB    128             ;; UNUSED
        DEFB    'S'-32,'R'-32,'L'-32+64+128

NOT_CB                          ;; HANDLE OPCODES 11XXX011 FOR XXX!=001
        INC     A               ;; RESTORE A (WAS DEC'ED TO TEST XXX=001)
        RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX AND RETURN
        DEFB    'J'-32,'P'-32+64,'5'-32+128 ;; STRING TABLE
        DEFB    128             ;; NOT USED (IS THE XXX=001 CASE)
        DEFB    'O'-32,'U'-32,'T'-32+64,'('-32,'4'-32,')'-32,','-32,'A'-32+128
        DEFB    'I'-32,'N'-32+64,'A'-32,','-32,'('-32,'4'-32,')'-32+128
        DEFB    'E'-32,'X'-32+64,'('-32,'S'-32,'P'-32,')'-32,','-32,'6'-32+128
        DEFB    'E'-32,'X'-32+64,'D'-32,'E'-32,','-32,'H'-32,'L'-32+128
        DEFB    'D'-32,'I'-32+128
        DEFB    'E'-32,'I'-32+128

GC04
ISCALLC
        RST     OUT_STR_ARG     ;; OUTPUT STRING AND RETURN
        DEFB    'C'-32,'A'-32,'L'-32,'L'-32+64,';'-32,','-32,'5'-32+128

;GC06                           ;; HANDLE OPCODES BINARY 11XXX110
;ISALUN
;        JR      GRP_80

GC07                            ;; HANDLE OPCODES BINARY 11XXX111
ISRST                           ;; RST GROUP
        AND     $38             ;; MASK RST ADDRESS FROM OPCODE
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG AND RETURN
        DEFB    'R'-32,'S'-32,'T'-32+64,':'-32+128

GRP_ED                          ;; $ED PREFIX
        RST     LD_A_HL         ;; GET EXT OPCODE FROM INPUT
;        ADD     $40            ;; NOTE REMOVED SOME CHECKING HERE TO SAVE BYTES
TO
                                ;;    TAKE ADVATAGE OF ILLEGAL INSTR'S ARE DON'T-
CARES
        CP      C ;$40+$40=$80  ;; C=$80 AS SIDE EFFECT FROM ABOVE
;        JP      C,LD_DE_0      ;GRP_ED00, GRP_EDC0
;        CP      $80+$40
        JR      C,GRP_ED40
GRP_ED80
;        XOR     $A0+$40        ;10X..Y.. XIT IF X=0 OR Y=1
        AND     $1F             ;ONLY NEEDED IF NOT ONLY GRP_ED80 CASE!
        LD      C,A
;        AND     $24            ;BOUND TO VALID RANGE 0-15
;        JP      NZ,LD_DE_0     ;XIT80
;        LD      A,C
        AND     $03             ;; OPCODE IS 101XX0YY MASKED TO 000XX0YY
        ADD     A,C             ;; FORM TABLE INDEX AS XXYY
        RRCA
        RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX AND RETURN
        DEFB    'L'-32,'D'-32,'I'-32+128
        DEFB    'C'-32,'P'-32,'I'-32+128
        DEFB    'I'-32,'N'-32,'I'-32+128
        DEFB    'O'-32,'U'-32,'T'-32,'I'-32+128
        DEFB    'L'-32,'D'-32,'D'-32+128
        DEFB    'C'-32,'P'-32,'D'-32+128
        DEFB    'I'-32,'N'-32,'D'-32+128
        DEFB    'O'-32,'U'-32,'T'-32,'D'-32+128
        DEFB    'L'-32,'D'-32,'I'-32,'R'-32+128
        DEFB    'C'-32,'P'-32,'I'-32,'R'-32+128
        DEFB    'I'-32,'N'-32,'I'-32,'R'-32+128
        DEFB    'O'-32,'T'-32,'I'-32,'R'-32+128
        DEFB    'L'-32,'D'-32,'D'-32,'R'-32+128
        DEFB    'C'-32,'P'-32,'D'-32,'R'-32+128
        DEFB    'I'-32,'N'-32,'D'-32,'R'-32+128
        DEFB    'O'-32,'T'-32,'D'-32,'R'-32+128
; TERMINATOR ONLY NEEDED IF INDEX IS NOT BOUNDED ABOVE!
        DEFB    '_'-32+64+128 ; LIST TERMINATOR
```

```
GRP_ED40                        ;; HANDLE EXT OPCODES $ED,$40-$7F
        RST     JRTBL8          ;; DECODE USING 3 LSB'S AS SUBGROUP
        DEFB    GED400-$        ;; OPCODE IS PASSED FOR FURTHER DECODING
        DEFB    GED401-$
        DEFB    GED402-$
        DEFB    GED403-$
        DEFB    GED404-$
        DEFB    GED405-$
        DEFB    GED406-$
        DEFB    GED407-$
GED400                          ;; HANDLE EXT OPCODES $ED, BINARY $01XXX000
ISINPC
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG XXX AND RETURN
        DEFB    'I'-32,'N'-32+64,'='-32,','-32,'('-32,'C'-32,')'-32+128

GED401                          ;; HANDLE EXT OPCODES $ED, BINARY $01XXX001
ISOUTC
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG XXX AND RETURN
        DEFB    'O'-32,'U'-32,'T'-32+64,'('-32,'C'-32,')'-32,','-32,'='-32+128

GED402                          ;; HANDLE EXT OPCODES $ED, BINARY $01XXY010
        RST     RRCA3           ;; SHIFT XXY TO LSB'S
        RRA                     ;; SHIFT XX TO LSB'S, TEST Y
        JR      C,ISADCW        ;; GO IF Y=1
ISSBCW                          ;; Y=0
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG XX AND RETURN
        DEFB    'S'-32,'B'-32,'C'-32+64,'6'-32,','-32,'7'-32+128
ISADCW
        RST     OUT_STR_ARG
        DEFB    'A'-32,'D'-32,'C'-32+64,'6'-32,','-32,'7'-32+128

GED403                          ;; HANDLE EXT OPCODES $ED, BINARY $01XXX011
        RST     RRCA3           ;; SHIFT XXY TO LSB'S
        RRA                     ;; SHIFT XX TO LSB'S, TEST Y
        JR      C,ISLDRPATNN    ;; GO IF Y=1
ISLDATNNRP                      ;; Y=0
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG XX AND RETURN
        DEFB    'L'-32,'D'-32+64,'('-32,'5'-32,')'-32,','-32,'7'-32+128
ISLDRPATNN
        RST     OUT_STR_ARG
        DEFB    'L'-32,'D'-32+64,'7'-32,','-32,'('-32,'5'-32,')'-32+128

GED404                          ;; HANDLE EXT OPCODES $ED, BINARY $01XXX100
        AND     $38
;        JR      NZ,XITED       ;; REMOVED TO SAVE BYTES - DON'T CARE
ISNEG
        RST     OUT_STR_IND     ;; UGLY HACK TO SAVE 1 BYTE - USE OUT_SR_IND
        DEFB    'N'-32,'E'-32,'G'-32+128
        DEFB    '_'-32+64+128 ; LIST TERMINATOR ;; NEEDED IF NOT NEG

GED405                          ;; HANDLE EXT OPCODES $ED, BINARY $01XXX101
        RST     RRCA3           ;; SHIFT XXX TO LSB'S
        RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX XXX AND RETURN
        DEFB    'R'-32,'E'-32,'T'-32,'N'-32+128
        DEFB    'R'-32,'E'-32,'T'-32,'I'-32+128
        DEFB    '_'-32+64+128 ; LIST TERMINATOR

GED406                          ;; HANDLE EXT OPCODES $ED, BINARY $01XXX110
        RST     RRCA3           ;; SHIFT XXX TO LSB'S
        RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX XXX AND RETURN
        DEFB    'I'-32,'M'-32+64,'0'-32+128
        DEFB    128             ;; NOT USED
        DEFB    'I'-32,'M'-32+64,'1'-32+128
        DEFB    'I'-32,'M'-32+64,'2'-32+128
        DEFB    '_'-32+64+128 ; LIST TERMINATOR

GED407                          ;; HANDLE EXT OPCODES $ED, BINARY $01XXX111
        RST     RRCA3           ;; SHIFT XXX TO LSB'S
        RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX XXX AND RETURN
        DEFB    'L'-32,'D'-32+64,'I'-32,','-32,'A'-32+128
        DEFB    'L'-32,'D'-32+64,'R'-32,','-32,'A'-32+128
        DEFB    'L'-32,'D'-32+64,'A'-32,','-32,'I'-32+128
        DEFB    'L'-32,'D'-32+64,'A'-32,','-32,'R'-32+128
        DEFB    'R'-32,'R'-32,'D'-32+128
        DEFB    'R'-32,'L'-32,'D'-32+128
        DEFB    '_'-32+64+128 ; LIST TERMINATOR
```

```
GRP_00                          ;; HANDLE OPCODES $00-$3F
        RST     JRTBL8          ;; DECODE USING 3 LSB'S AS SUBGROUP
        DEFB    G000-$          ;; OPCODE IS PASSED FOR FURTHER DECODE
        DEFB    G001-$
        DEFB    G002-$
        DEFB    G003-$
        DEFB    G004-$
        DEFB    G005-$
        DEFB    G006-$
        DEFB    G007-$

G000                            ;; HANDLE OPCODES BINARY 000XXX000
        RST     RRCA3
        JR      NZ,G0004        ;GO IF BIT2=1
        RST     OUT_STR_IND
        DEFB    'N'-32,'O'-32,'P'-32+128
        DEFB    'E'-32,'X'-32+64,'A'-32,'F'-32,','-32,'A'-32,'F'-32,''''-32+128
        DEFB    'D'-32,'J'-32,'N'-32,'Z'-32+64,'9'-32+128
        DEFB    'J'-32,'R'-32+64,'9'-32+128

G0004                           ;; HANDLE OPCODES BINARY 000XXX000
ISJRC
        AND     3
        RST     OUT_STR_ARG
        DEFB    'J'-32,'R'-32+64,'?'-32,','-32,'9'-32+128

G001                            ;; HANDLE OPCODES BINARY 000XXX001
        RST     RRCA3
        RRA
        JR      C,ISADDW
ISLDW
        RST     OUT_STR_ARG
        DEFB    'L'-32,'D'-32+64,'7'-32,','-32,'5'-32+128
ISADDW
        RST     OUT_STR_ARG
        DEFB    'A'-32,'D'-32,'D'-32+64,'6'-32,','-32,'7'-32+128

G002                            ;; HANDLE OPCODES BINARY 000XXX010
        RST     RRCA3
        JR      NZ,G0022        ;GO IF BIT2=1
        RRCA
        JR      C,G0021
        RST     OUT_STR_ARG
        DEFB    'L'-32,'D'-32+64,'('-32,'7'-32,')'-32,','-32,'A'-32+128
G0021   RST     OUT_STR_ARG
        DEFB    'L'-32,'D'-32+64,'A'-32,','-32,'('-32,'7'-32,')'-32+128
G0022   AND     3
        RST     OUT_STR_IND
        DEFB    'L'-32,'D'-32+64,'('-32,'5'-32,')'-32,','-32,'6'-32+128
        DEFB    'L'-32,'D'-32+64,'6'-32,','-32,'('-32,'5'-32,')'-32+128
        DEFB    'L'-32,'D'-32+64,'('-32,'5'-32,')'-32,','-32,'A'-32+128
        DEFB    'L'-32,'D'-32+64,'A'-32,','-32,'('-32,'5'-32,')'-32+128

G003                            ;; HANDLE OPCODES BINARY 100XXX011
ISINCDECW
        RST     RRCA3           ;; SHIFT XXX=ZZY TO LSB'S
        RRA                     ;; SHIFT ZZ TO LSB'S, TEST Y
        JR      C,ISDECW        ;; GO IF Y=1
ISINCW                          ;; Y=0
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH AGR ZZ
        DEFB    'I'-32,'N'-32,'C'-32+64,'7'-32+128
ISDECW
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH AGR ZZ
        DEFB    'D'-32,'E'-32,'C'-32+64,'7'-32+128

G004                            ;; HANDLE OPCODES BINARY 000XXX100
ISINC
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH AGR XXX
        DEFB    'I'-32,'N'-32,'C'-32+64,'='-32+128

G005                            ;; HANDLE OPCODES BINARY 000XXX101
ISDEC
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG XXX
        DEFB    'D'-32,'E'-32,'C'-32+64,'='-32+128

G006                            ;; HANDLE OPCODES BINARY 000XXX110
```

```
ISLDN
        RST     OUT_STR_ARG     ;; OUTPUT STRING WITH ARG XXX
        DEFB    'L'-32,'D'-32+64,'='-32,','-32,'4'-32+128

G007                            ;; HANDLE OPCODES BINARY 000XXX111
        RST     RRCA3           ;; SHIFT XXX TO LSB'S
        RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX XXX
        DEFB    'R'-32,'L'-32,'C'-32,'A'-32+128
        DEFB    'R'-32,'R'-32,'C'-32,'A'-32+128
        DEFB    'R'-32,'L'-32,'A'-32+128
        DEFB    'R'-32,'R'-32,'A'-32+128
        DEFB    'D'-32,'A'-32,'A'-32+128
        DEFB    'C'-32,'P'-32,'L'-32+128
        DEFB    'S'-32,'C'-32,'F'-32+128
        DEFB    'C'-32,'C'-32,'F'-32+128

;================================

                                ;; REPLACE CODE WITH CORRESPONDING STRING
REPLACE DEC     DE              ;; RESTORE OUTPUT POINTER (WAS INCREMENTED)
        LD      B,IYH           ;IYH=OTSARG ;; OUT_STR ARG WAS SAVED TO IYH
        CALL    JRTBL           ;; JUMP TO INDEXED OFFSET AND RETURN TO CALLER
        DEFB    OUT_REG-$       ; '3' ;; REGISTER A,B,C,D,E,H,L,(HL),(IX+/-$NN),
(IY+/-$NN)
        DEFB    HEXB_AT_HL-$    ; '4' ;; $NN AT PC
        DEFB    HEXW_AT_HL-$    ; '5' ;; $NNNN AT PC
        DEFB    OUT_HLIXIY-$    ; '6' ;; HL,IX,IY
        DEFB    OUT_RP0-$       ; '7' ;; REGISTER PAIR SP,BC,DE,HL
        DEFB    OUT_RP-$        ; '8' ;; REGISTER PAIR AF,BC,DE,HL
        DEFB    HEXREL_AT_HL-$  ; '9' ;; $NNNN = PC PLUS BYTE OFFSET AT PC
        DEFB    HEX_A-$         ; ':' ;; $NN IN ARG
        DEFB    OUT_COND_HI-$   ; ';' ;; SAME AS OUT_COND BUT USING HIGH BIT
FIELD
        DEFB    OUT_BIT-$       ; '<' ;; VALUE 0-7 IN ARG
        DEFB    OUT_REG_HI-$    ; '=' ;; SAME AS OUT_REG BUT USING HIGH BIT FIELD
        DEFB    HEXOFS_AT_HL-$  ; '>' ;; +/-$NN AT PC
        DEFB    OUT_COND-$      ; '?' ;; CONDITION NZ,Z,NC,C,NV,V,P,M IN ARG

OUT_COND_HI                     ;; OUTPUT CONDITION CODE IN HIGH BIT FIELD OF A
        RST     RRCA3           ;; SHIFT HIGH BIT FIELD TO LOW
OUT_COND                        ;; OUTPUT CONDITION CODE IN REG A
        RST     OUT_STR_IND     ;; OUTPUT STRING AT UNDEX AND RETURN TO CALLER
        DEFB    'N'-32,'Z'-32+128
        DEFB    'Z'-32+128
        DEFB    'N'-32,'C'-32+128
        DEFB    'C'-32+128
        DEFB    'N'-32,'V'-32+128
        DEFB    'V'-32+128
        DEFB    'P'-32+128
        DEFB    'M'-32+128

OUT_RP                          ;; OUTPUT REGISTER PAIR IN REG A
        INC     A               ;; TRANSLATE HL/IX/IY TO 0
RP0     INC     A
        AND     3
        JR      NZ,NOTHLIXIY    ;; GO IF NOT HL,IX,IY, ELSE FALL THROUGH
OUT_HLIXIY
        LD      A,IYL           ;IYL=ISIXIY ;; GET IX/IY PREFIXED FLAG
        RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX AND RETURN TO CALLER
        DEFB    'H'-32,'L'-32+128
        DEFB    'I'-32,'X'-32+128
        DEFB    'I'-32,'Y'-32+128

OUT_RP0                         ;; OUTPUT REGISTER PAIR IN REG A
        INC     A               ;; TRANSLATE SP TO 0
        AND     3
        JR      NZ,RP0          ;; GO IF NOT SP, ELSE FALL THROUGH TO SP CASE
NOTHLIXIY
        RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX AND RETURN
        DEFB    'S'-32,'P'-32+128
        DEFB    'A'-32,'F'-32+128
        DEFB    'B'-32,'C'-32+128
        DEFB    'D'-32,'E'-32+128

ISCBALUX                        ;; PLACED HERE TO GET FALL THROUGH TO HEXB_AT_HL
ISALUX                          ;;    AND SHORT BRANCH TO OUT_REG
        RST     RRCA3           ;; SHIFT ..XXX... TO LSB'S
```

```
        POP     BC              ;; GET POINTER TO STRING TABLE FROM STACK
        CALL    OUT_STR_IND_P1  ;; OUTPUT STRING AT INDEX - POINTER IN BC NOT ON
STACK
        POP     AF              ;; RESTORE CARRY FLAG
        JR      C,OUT_REG       ;; IMMEDIATE VALUE IF CARRY=0, ELSE REGISTER
(CARRY=1)
HEXB_AT_HL
        RST     LD_A_HL         ;; GET IMMEDIATE BYTE AT PC
HEX_A                           ;; OUTPUT $NN IN A
        LD      B,A             ;; SAVE VALUE IN B
HEX_B                           ;; OUTPUT $NN IN B
        LD      A,'$'           ;; OUTPUT '$' HEX PREFIX
        LD      (DE),A
        INC     DE              ;; BUMP
HB0     LD      A,B             ;; RESTORE VALUE FROM B
        RRCA                    ;; SHIFT MS NIBBLE TO LS NIBBLE
        RRCA
        RRCA
        RRCA
        CALL    HB1             ;; OUTPUT MS NIBBLE
        LD      A,B             ;; RESTORE VALUE FROM B
HB1     AND     $0F             ;; OUTPUT NIBBLE
        CP      10              ;; HEX TO ASCII TRICK USING Z80 DAA
        SBC     A,$69
        DAA
        LD      (DE),A
        INC     DE
;        JP      LD_DE_0        ;; JUST INLINE HERE SINCE FAR BRANCH IS SAME 3
BYTES
        XOR     A               ;; APPEND 0 INCASE THIS IS THE END OF OUTPUT
        LD      (DE),A          ;; MAY BE OVERWRITTEN
        RET                     ;; RETURN TO CALLER

HEXOFS_AT_HL                    ;; OUTPUT INDEX OFFSET +/-$NN
        RST     LD_A_HL         ;; GET OFFSET VALUE AT PC, C=$80 AS SIDE EFFECT
        CP      C ;$80          ;; TEST IF NEGATIVE
        LD      C,'+'           ;; GET '+' INCASE POSITIVE
        JR      C,OFSPLUS       ;; GO IF POSITIVE
        NEG                     ;; CONVER TO POSITIVE
        LD      C,'-'           ;; GET '-' SINCE NEGATIVE
OFSPLUS
        LD      B,A             ;; SAVE VALUE
        LD      A,C             ;; OUTPUT THE +/-
        LD      (DE),A
        INC     DE              ;; BUMP
        JR      HEX_B           ;; OUTPUT POSITIVE OFFSET IN B

HEXREL_AT_HL                    ;; OUTPUT RELATIVE BRANCH TARGET $NNNN
        RST     LD_A_HL         ;; GET BRANCH OFFET VALUE AT PC
        LD      C,A             ;; COPY OFFSET TO C
        RLCA                    ;; TRICK TO SIGN EXTEND BYTE
        SBC     A,A             ;; CARRY=SIGN, AFTER SUB A=$00 OR $FF
        LD      B,A             ;; BC=SIGN EXTENDED BYTE OFFSET
        PUSH    HL              ;; SAVE PC
        ADD     HL,BC           ;; COPMUTE BRANCH TARGET
        EX      (SP),HL         ;; RESTORE PC
        POP     BC              ;; BRANCH TARGET TO BC
        JR      HEX_BC          ;; OUTPUT $NNNN IN BC

HEXW_AT_HL                      ;; OUTPUT $NNNN AT PC AND PC+1
        LD      C,(HL)          ;; GET LSB IN C
        INC     HL              ;; BUMP
        LD      B,(HL)          ;; GET MSB IN B
        INC     HL              ;; BUMP, FALL THROUGH TO HEX_BC
HEX_BC                          ;; OUTPUT $NNNN IN BC
        CALL    HEX_B           ;; OUTPUT $NN IN B
        LD      B,C             ;; GET LS BYTE
        JR      HB0             ;; OUTPUT NN IN B W/O $ PREFIX

OUT_BIT                         ;; OUTPUT BIT POSITION VALUE 0-7
        RST     RRCA3           ;; SHIFT ..XXX... TO LSB'S
        JR      HB1             ;; OUTPUT USING HEX NIBBLE TAIL

OUT_REG_HI                      ;; OUTPUT REGISTER IN BIT FIELD OF A
        RST     RRCA3           ;; SHIFT HIGH BIT FIELD TO LOW
OUT_REG                         ;; OUTPUT REGISTER IN A
        INC     A               ;; TRANSLATE (HL),(IX+/-$NN),(IY+/-$NN) TO 7
```

```
        AND     7
        CP      7               ;; CHECK FOR (HL),(IX+/-$NN),(IY+/-$NN)
        JR      NZ,NOTHL        ;; GO IF NOT
        ADD     A,IYL           ;; ADD IYL=ISIXIY FLAG TO TRANSLATE (HL) TO HL/
IX/IY
NOTHL   RST     OUT_STR_IND     ;; OUTPUT STRING AT INDEX AND RETURN
        DEFB    'A'-32+128,'B'-32+128,'C'-32+128,'D'-32+128,'E'-32+128,'H'-
32+128,'L'-32+128
        DEFB    '('-32,'H'-32,'L'-32,')'-32+128
        DEFB    '('-32,'I'-32,'X'-32,'>'-32,')'-32+128
        DEFB    '('-32,'I'-32,'Y'-32,'>'-32,')'-32+128
```

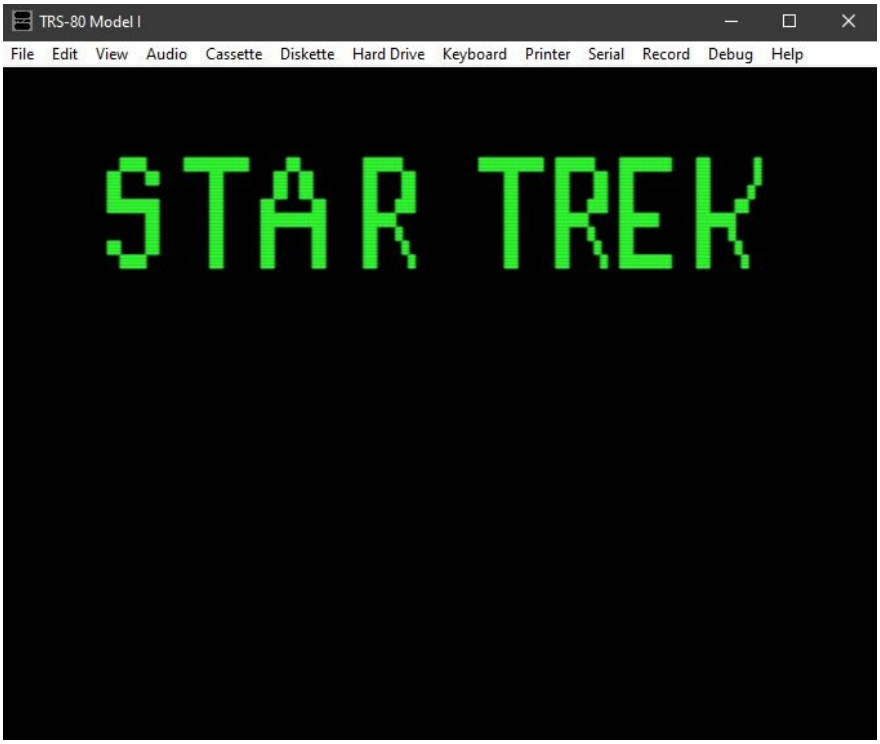


TRS-80

Repairs & Upgrades

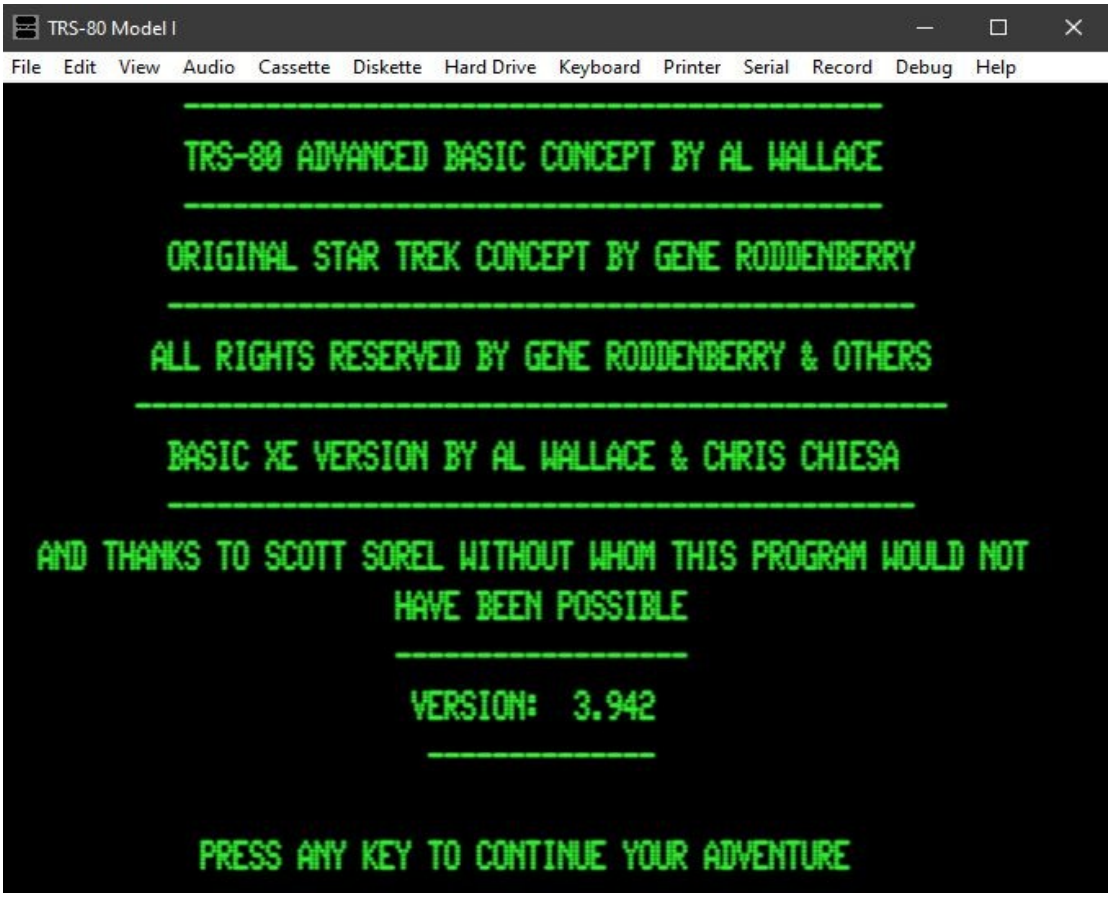
HNC qualified electronics engineer
with 40 years TRS-80 experience.
Enquiries to: trs80@shanefoster.co.uk





Al Wallace is the driving force behind this upgraded and much improved game of 'STARTREK'. He's VERY keen to have any feedback from users. Please try it out let him know what you think. Contact him direct at al.wallace1964@yahoo.com

There's a copy of the program from the 'downloads' page, at www.trs-80.org.uk


TRS-80 Model I
File Edit View Audio Cassette Diskette Hard Drive Keyboard Printer Serial Record Debug Help
TRS-80 ADVANCED BASIC CONCEPT BY AL WALLACE
ORIGINAL STAR TREK CONCEPT BY GENE RODDENBERRY
ALL RIGHTS RESERVED BY GENE RODDENBERRY & OTHERS
BASIC XE VERSION BY AL WALLACE & CHRIS CHIESA
AND THANKS TO SCOTT SOREL WITHOUT WHOM THIS PROGRAM WOULD NOT HAVE BEEN POSSIBLE
VERSION: 3.942
PRESS ANY KEY TO CONTINUE YOUR ADVENTURE


TRS-80 Model I
File Edit View Audio Cassette Diskette Hard Drive Keyboard Printer Serial Record Debug Help
YOU ARE IN SECTOR - 6 , 5
AT COORDINATES ..... 4 , 4
ENERGY LEVEL - - - - 15000
SHIELDS ARE AT 100%
COMPUTER SYSTEMS - - GREEN
WARP DRIVE SYSTEM ....GREEN
IMPULSE DRIVE - - - - GREEN
PHASER BANK SYSTEM ...GREEN
TORPEDO SYSTEM - - - GREEN
PHOTON TORPEDOES 10
STARDATE - - - - 0.00
* ALERT CONDITION ...RED
MAIN MENU:
NAVIGATION - - - 1
COMPUTER 2
SHIELDS/CHAFF - 3
WEAPONS 4
CHEAT - - - - - - - 5
ABANDON SHIP 6
STATS - - - - - - - 7
COORDINATE SYS 8
SKIP TURN ... 0

IN MAV'S WORKSHOP
Nip Rifa Madness In The Bud On The TRS-80 Model II

Ian Mavric



This is a short article with a tip on how to get your vintage computer acquisition up and running quickly. This about the TRS-80 Model II, but it might relate to the Model 16, and other computers as well.

WE ALL HATE RIFA MAINS FILTER CAPS

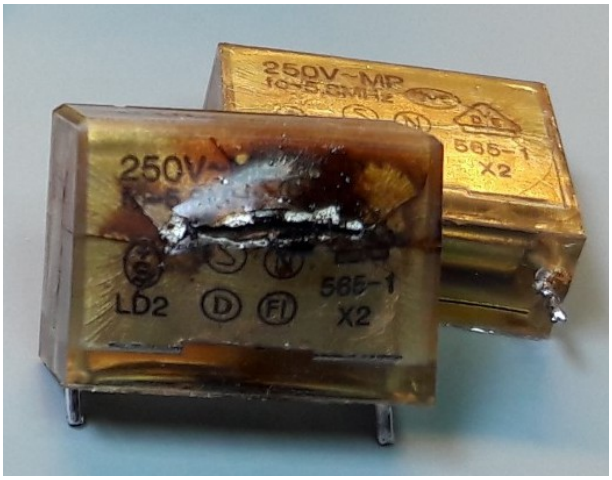


Photo: Rifa caps... blown (front) and about 5 seconds before (behind)

Once smelled, never forgotten. These paper and foil in oil mains filter capacitors designed in the 70s and used in all manner of electronics (not just computers) do a good job at what they were designed to do. The trouble is the plastic gets brittle with age, cracks, and then shorts internally and the oil smokes. It's a mess to clean up, and makes a very acrid smell.
That said, when they fail they rarely damage the equipment. Computers often still keep running without crashing as the Rifa cap lets go and smoke exists the computer through the cooling vents and disk drives. Removing them before they blow makes sense, it's also time consuming.



Photo: a Model III which has just popped a Rifa mains filter cap

THIS BLOKE CAME UP TO ME

I was exhibiting some vintage computers at a computer market in Melbourne in April 2023. My usual exhibits include a Model III, 4P, Coco 3, C64, Apple IIe, and Mac SE/30. A gentleman walked up to me saying he had an old TRS-80 sitting under his billiards table for, oh, about 18 years and asked if I was interested. Keen to know what model he had I asked for a description, and he said it was bigger than the Model III, with a bigger disk drive which was mounted vertically. Ah, that's a Model II. A deal was reached and an hour later he returned with a quite well preserved Model II in the trunk of his car.

Eager to see if it worked I wanted to get it home, plug it in and see if I'd bought a rough diamond or a dud.

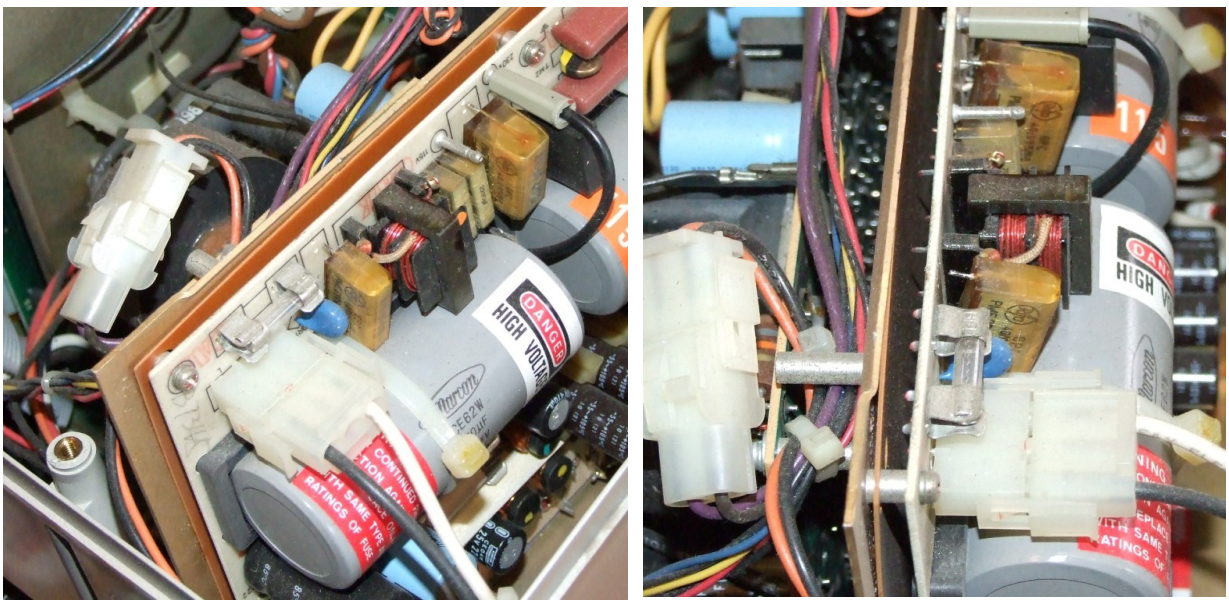
THE MODEL II POWER SUPPLY

The power supply (PSU) in Australian Model IIs is made by Astec, who manufactured robust well designed PSUs not just for Radio Shack but any other computer maker who needed one. The PSU in the Model II is about twice the size of the Model 4s Astec PSU, and twice as complicated. It's also a pig to remove and re-fit. Whilst the Model III and 4 Astec PSUs have small 0.1uF Rifa mains filter caps, the Model II PSU has a positively enormous 0.22uF which has more than double the oil inside it. When it blows, it's like a Model 4 Rifa madness smoke out on steroids. Cheech & Chong would even be impressed if not for the absurdity of it.

LET'S DISABLE THE RIFA CAPS AND TEST THE MACHINE

What if we were to just remove them? Actually the system will run fine without them. Mains filtering will be absent and it is a good idea to replace them at some stage shortly after the rest of the computer has been found to be in good working condition, or repaired. I prefer to use Suntan brand yellow MKT X2 types myself.

On the Model II PSU the 0.22uF and 0.1uF capacitors are mounted nicely at the top in easy reach. To disable them all you need to do is heat up one leg of the capacitor while prizing one side of the cap out. The other leg can stay soldered in. There is about 5mm of space behind the PSU to get the soldering iron tip in and it takes about 5 seconds per capacitor to liberate a leg. You can undo the top three screws securing the PSU to its mounting plate and lean it over to get the soldering iron in if space is at a premium.



Photos: Rifa caps disabled by prying out one leg each

WAS IT A SUCCESS?

So it turned out the Model II was a good buy. It was a standard system, with a Rev C processor board. Everything worked after I re-seated the 4 cards, and I was greeted with the friendly INSERT DISKETTE message. As usual the keyboard was unhappy due to the usual foam and foil design of the key switches where the foam turns to dust and the keyboard doesn't work. I connected a Type-II HDA and F48 and it wouldn't boot, but after I replaced the U11 Prom on the processor board with the last version designed to boot a hard drive, it loaded first time and I have been using it all weekend.

CONCLUSION

For those keen to see one of these Rifa caps blow in real time I suggest people check this video on YouTube uploaded by my old mate Peter Cetinski: https://youtu.be/njbwdbcfXjc

Ian Mavric
ianm@trs-80.com



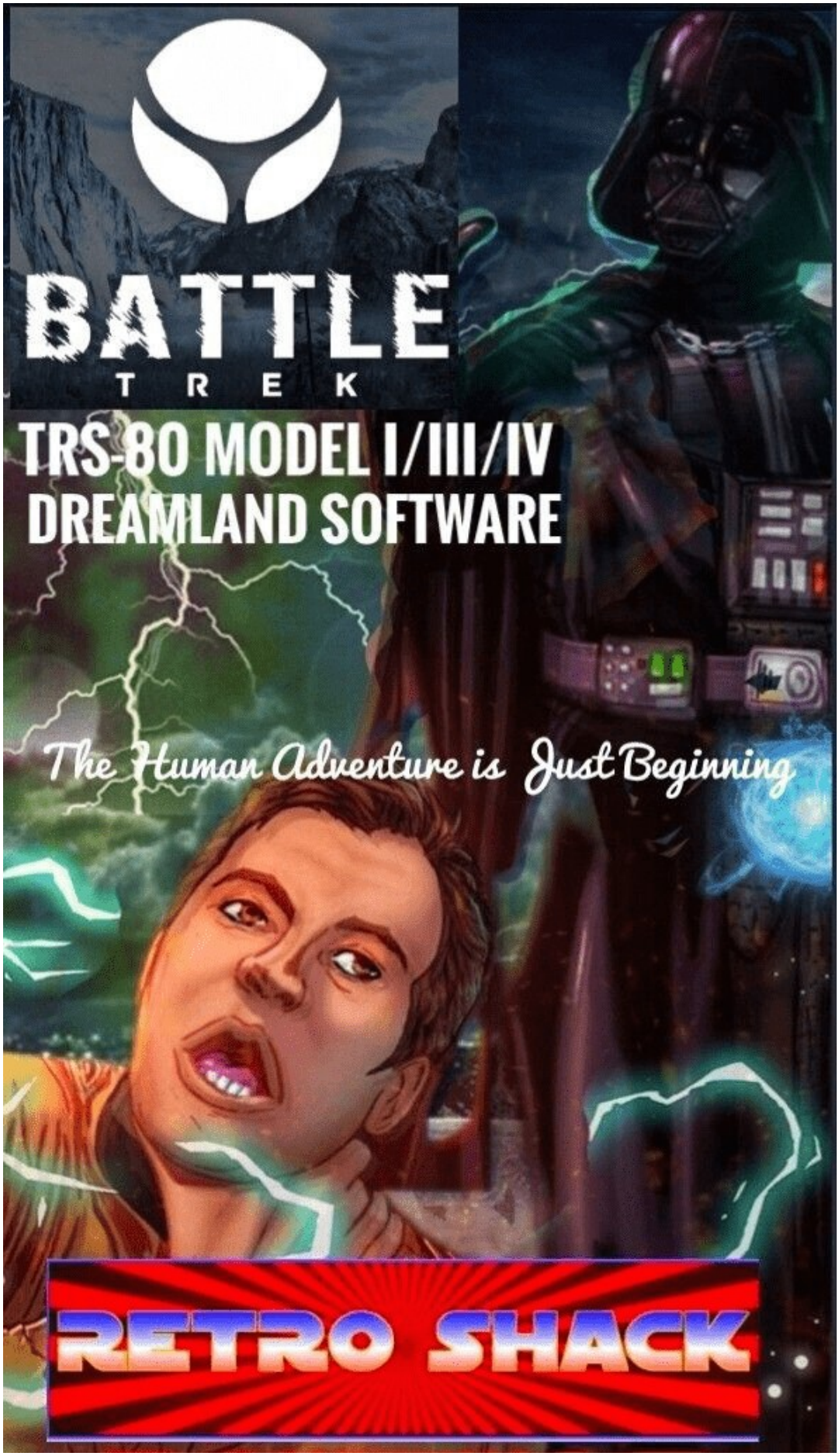
Ian Mavric is an IT Specialist who also restores and collects TRS-80's and classic cars. He lives with his wife and kids in Melbourne, Australia.

DAI SOFTWARE
FROM IQ SERVICES

* ANIMAX
A general purpose 3D graphics generator for use in all modes providing easy to use drawing and animation facilities. £7.50

* ANIGRAM
As ANIMAX but designed specifically for the easy production of diagrams in education and including annotation facilities. £8.50

Full instructions supplied with each cassette.
Prices include VAT & pp. CANAL HOUSE,
ARDISHAIG, ARGYLL, SCOTLAND
Tel: 0546 3212


BATTLE
TREK
TRS-80 MODEL I/III/IV
DREAMLAND SOFTWARE
The Human Adventure is Just Beginning
RETRO SHACK

Trevor Flowers' Miniature TRS-80 Microcomputer Houses an Adafruit QT Py for a Working Display

Built around an Adafruit QT Py, this 1:6-scale TRS-80 Model III boasts considerably higher specifications than the original.

Maker and vintage computing enthusiast Trevor Flowers has put together an ultra-compact clone of the Tandy-Radio Shack TRS-80 microcomputer — powered by an Adafruit QT Py board running CircuitPython.

Released in 1977 by Tandy through its Radio Shack store fronts, the original TRS-80 — later rebadged to the Model I — kicked off a family of microcomputing devices which would run the gamut from all-in-one systems with floppy drives and CRT displays to the popular TRS-80 Model 100 portable. The TRS-80 Model II, released in 1980, came with the aforementioned display, up to two 180kB floppy drives, up to 48kB of RAM, and a Zilog Z80A processor running at 2MHz.

Flowers' miniature recreation, though, has very different specifications. Inside the 3D-printed chassis lies an Adafruit QT Py S3 microcontroller development board, which despite its diminutive size boasts an Espressif ESP32-S3 chip with two Tensilica Xtensa LX7 cores running at up to 240MHz, 512kB of static RAM (SRAM), 2MB of pseudo-static RAM (PSRAM), and 4MB of flash storage — all upgrades on the TRS-80 by multiple orders of magnitude.

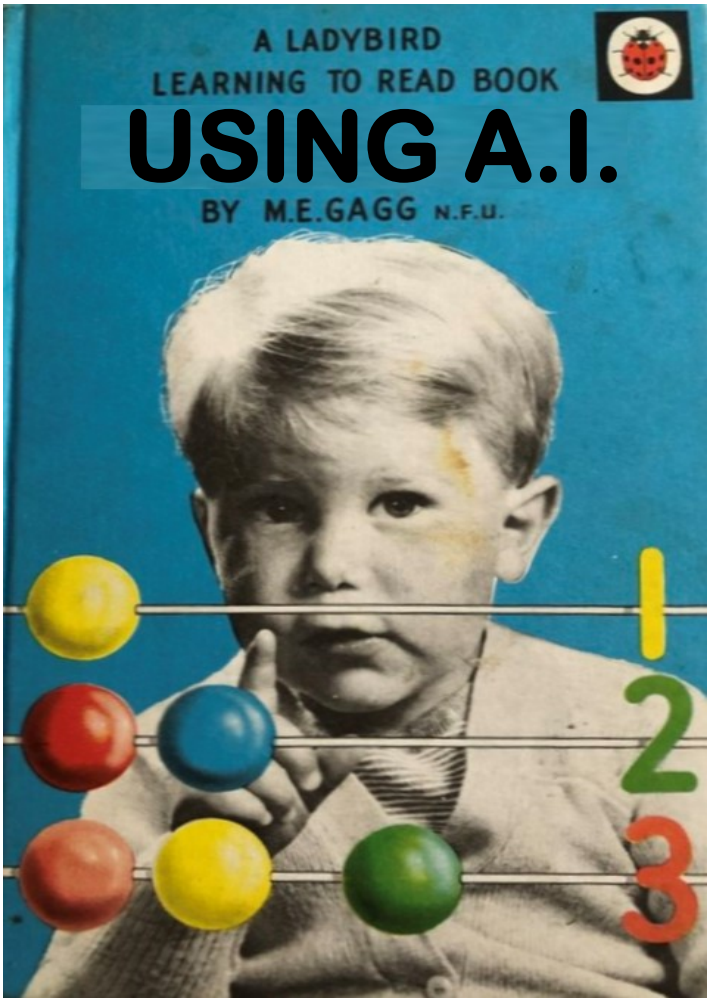
"This [miniature] is driven by this QT Py running CircuitPython or Arduino," Flowers explains. "So, there's not an accurate TRSDOS or

VDOS emulator for it. On the other hand it has Wi-Fi, BLE [Bluetooth Low Energy], and more memory and pixels than the original."

The project builds on Flowers early display-only TRS-80 miniature model, which lacked electronics and instead used swappable cards to simulate a static display. "As shipped," Flowers explained of the original, which was created to a 1:6 scale, "it will show a short BASIC program." The new variant, though, can display animations — or even double as a functional Internet of Things (IoT) connected display system in a retro-themed chassis.
More information on the project is available on Flowers' Mastodon thread, while the non-functional variant is listed for sale on his Transmutable store at $42.



This tiny TRS-80 packs an Adafruit QT Py S3 microcontroller board, giving it specifications far in excess of full-size originals.
© Trevor Flowers



Many thanks to George Phillips for his programing expertise with this article

Having been totally amazed with the photo of the M3 accessing the web and using ChatGPT, I thought, I really must find out a bit more about all the fuss AI was making. It coincided with a big pow-wow with many head of governments arriving in the U.K. to discuss fears and concerns with the use of AI.

The first idea I had was, would it be able to compose a limerick? So I asked 'please write a limerick using the name 'trs8bit'. (I know, no prizes or originality, but it is quite a tricky name to rhyme!). In under 1 second it came back with ...

There once was a coder named trs8bit,
In circuits and code, he really was most fit.
With ones and with zeros,
His programs were heroes,
A digital wizard, no one, could outwit!

I was quite impressed with that. Not much use, I know, so I wondered if it could write a BASIC program. Not just any BASIC, as the Model 1 uses Microsoft BASIC. So I asked the following question ...

Please code a program, in Microsoft BASIC, to input a string variable and print the string in reverse order. Again, in less then a second if came back with ...

Sure, no problem. Here's a simple Microsoft BASIC program that inputs a string variable and prints the string in reverse order:

```
10 INPUT "Enter a string: ", A$
20 FOR I = LEN(A$) TO 1 STEP -1
30   PRINT MID$(A$, I, 1);
40 NEXT I
50 END
```

This program will prompt you to enter a string, and then it will use a FOR loop to print the characters of the string in reverse order. The MID$ function is used to extract each character of the string starting from the last character (character at position LEN(A$)) and moving backward to the first character (character at position 1). The loop goes from the length of the string down to 1, with a step of -1 to iterate through the characters in reverse order.

I thought that was really good little program. Easy to follow and much the same thing I would have produced. Then I started thinking, I wonder if ChatGPT could cope with Z80 assembler? How about an attempt at an 'INSTR' function? Who knows, it might even get into Level 2.5 BASIC (TWOHAF). Anyway, half expecting a message "sorry mate, no idea what you're on about!" I asked the following question.

Please code a program in Z80 assembler the find the position of a substring within another string.

No Problem !! (I was most impressed with the TWO exclamation marks!)

```
ORG 1000h   ; Set the origin address to 1000h

; Define the main routine
MAIN:
  LD HL, String     ; Load the address of the main string into HL
  LD DE, Substring  ; Load the address of the substring to search for into DE
  CALL FindSubstring ; Call the subroutine to find the substring

  ; Check the result in register A
  CP 0              ; Check if A is zero (not found)
  JP Z, NotFound    ; Jump to NotFound label if the result is zero
  ; If found, the position is in register HL

Found:
  ; Handle the found case here
  ; HL contains the position of the substring in the main string

NotFound:
  ; Handle the not found case here
  ; A is zero

  HALT         ; Halt the processor

; Subroutine to find a substring within a string
FindSubstring:
  PUSH BC       ; Preserve BC register pair
  PUSH HL       ; Preserve HL register pair

SearchLoop:
  LD B, 0       ; Reset B to zero for substring index

  ; Check if the substring is empty
  LD A, (DE)
  CP 0
  JP Z, FoundSubstring

  ; Start comparing characters
CompareLoop:
  LD A, (HL)    ; Load a character from the main string
  LD C, (DE)    ; Load a character from the substring

  ; Compare characters
  CP C
  JP NZ, NotEqual

  INC HL        ; Move to the next character in the main string
  INC DE        ; Move to the next character in the substring
  INC B         ; Increment the match counter
```

```
  ; Check if the entire substring is found
  LD A, (DE)
  CP 0
  JP Z, FoundSubstring

  ; Continue comparing characters
  JP CompareLoop

NotEqual:
  ; Reset the index to the beginning of the substring
  POP HL        ; Restore HL register pair
  LD DE, Substring
  INC HL        ; Move to the next character in the main string
  JP SearchLoop

FoundSubstring:
  POP HL        ; Restore HL register pair
  POP BC        ; Restore BC register pair
  LD A, B       ; Copy the match counter (position) to A
  RET

; Data section
String:     DB "This is a test string with a substring", 0
Substring:  DB "substring", 0
```

This took, at the most, 3 seconds!
Not having a great deal of assembler experience, I emailed George Phillips and asked what he thought of the AI generated code. Here's his reply.

"The code ChatGPT produced is pretty good. There are some mistakes. "ld c,(de)" is not a legal instruction and at one point it checks for length == 0 but only checks half the BC register.

The routines wouldn't be directly usable in TWOHAF as they're set up to handle NUL terminated strings. BASIC strings use a byte to keep track of their length and so don't have a terminating character."

I then decided to experimented with the 'MID$=' function. Once again I typed - Please code a Z80 assembler program to replace part of a string with a substring. Once again, ChatGPT responded in under 2 seconds

OK, here we go

```
; Z80 Assembly program to replace part of a string with a substring

; Input:
;   DE - Pointer to the source string
;   HL - Pointer to the substring to be inserted
;   BC - Length of the substring
; Output:
;   DE - Pointer to the modified string

org 32768 ; Start address of your program

main:
    ld a, (de)          ; Load the current character from the source string
    cp 0               ; Check if it's the null terminator
    ret z              ; If it is, we're done

    ld (hl), a          ; Replace the character in HL with the one from DE
    inc de              ; Move to the next character in DE
    inc hl              ; Move to the next character in HL
    dec bc              ; Decrement the substring length

    ld a, c             ; Load the substring length into A
    cp 0               ; Check if we've replaced all characters
```

```
    jr z, done          ; If we have, we're done

    jr main             ; Continue replacing characters

done:
    ret                 ; Return with DE pointing to the modified string

; End of program
```

I thought about pushing my luck and 'going for gold' and asking for a Level 2 BASIC compatible program next.

Can you write z80 assembler code which is compatible with Tandy's Level 2 BASIC ?

Certainly! If you want to write Z80 assembly code that is compatible with Tandy's Level 2 BASIC (commonly found on TRS-80 computers), you'll need to follow a specific memory layout and call conventions. Below is a simple example of Z80 assembly code that prints a message and waits for a keypress, which should be compatible with Tandy's Level 2 BASIC:

```
; Tandy's Level 2 BASIC-compatible Z80 assembly code
.org $1800   ; Starting address in TRS-80 memory

; Define constants for system routines
OUTCHR  equ $FFF8   ; Address of the character output routine
RDKEY   equ $FFCF   ; Address of the key input routine

; Define a message to be printed
message db "Hello, World!", 13, 10, 0 ; The message followed by CR and LF

; Entry point
start:
    ld hl, message      ; Load address of the message
    call print_message  ; Call the print_message routine

wait_keypress:
    call RDKEY          ; Call the key input routine
    cp 0                ; Check if a key has been pressed (0 means no key pressed)
    jr z, wait_keypress ; If no key, keep waiting

    ret                 ; Return to Tandy's Level 2 BASIC

; Subroutine to print a null-terminated string
print_message:
    ld a, (hl)          ; Load the character from the address in HL
    or a                ; Check if it's the null terminator
    retz                ; If null terminator, return
    call OUTCHR         ; Call the character output routine
    inc hl              ; Move to the next character
    jr print_message    ; Repeat for the next character

.end
```

In this code:

We set the starting address to $1800, which is a common starting address for Tandy's Level 2 BASIC on TRS-80 computers.

We define two constants, OUTCHR and RDKEY, which represent the addresses of system routines used by Tandy's Level 2 BASIC. These addresses are typical for TRS-80 systems.

We define a message to be printed. Make sure to adjust the message to your needs.
(I liked the use of 'WE' there)



As always, of course, I'm on the lookout for any software for the U.K.'s **Aculab 'Floppy Tape'**. There were versions specifically 'zapped', for use on the AFT, of Electric Pencil, Scripsit, Zen, Level III BASIC and VisiCalc. There was also the enhanced BASIC, XBAS. It's so disappointing to think that they have all been lost!

Please email me at the usual address if you can help.

The code loads the address of the message and calls the print-message routine to print it. It then enters a loop waiting for a keypress using the RDKEY routine.

When a key is pressed (non-zero result from RDKEY), it exits the loop and returns control to Tandy's Level 2 BASIC.

Please note that the specific addresses of system routines and memory layout can vary slightly between different models and versions of TRS-80 computers, so you may need to adjust the code accordingly for your specific system.

Even I could see that there were problems here so George quickly came to the rescue and explained what had gone wrong.

"*Looks like ChatGPT here misunderstood a bit.*

It took "compatible with Level 2 BASIC" as "use ROM routines". Looks pretty correct though "retz" should be "ret z". Funny how it uses "or a" in one spot but the bigger and less efficient "cp 0" in another.

You were right in that it missed using a ROM routine which does all the work for it in waiting for a key press. And for that matter there is a ROM routine to print a string. Unlike the BASIC strings it does want the strings terminated with either carriage return (character 13) or character 3.

I will upgrade BASIC 2HAF with the functions, and as you mentioned CVI and MKI$ as they are quite short routines it should be easy enough. I'll throw in CVS, CVD, MKS$ and MKD$ since they are trivial variants of CVI and MKI$.

MKI$ etc. just changes the argument another way and copies the binary representation of the number into the string. A BASIC integer variable uses two bytes so you get a two byte string. This is much more compact that what is looks like in ASCII which may be as much as 6 characters (for -32000 and other large negative integers).

Here's where BASIC's way of doing strings has an advantage. MKI$(0) will return a length 2 string consisting of two CHR$(0)'s. That wouldn't work with strings that use a 0 byte as a terminator as it'd be an empty string.

-- George"

TANDY CGP115 PRINTER



One of my (if not the) favourite printers. It's small, light-weight, good supply of normal paper rolls still available, has 80 columns when needed and it was inexpensive to buy, back in the day. The two main draw-backs were, internal cogs were plastic, and the small ink cartridges didn't last long and were very costly to replace.

Here, Brian K. White (bw.aljex@gmail.com), gives us fully detailed information about the inner working and fixes for the main headaches.

The Plotter Models were know as many different models from various manufacturers, viz.

Tandy CGP-115, Tandy 26-3605, Sharp CE-150, Commodore 1520, Atari 1020, Mattel Aquarius 4615, Texas Instruments HX-1000, Oric MCP40, Silver Reed EB50, ALPS DPG-1302

PENS:

lass-shop.de
Search box: "stifte" ("pen" in German)

http://www.lass-shop.de/epages/63686166.sf/de_DE/?ObjectID=30124112&ViewAction=FacetedSearchProducts&SearchString=stifte

US 016.01 - 4 black
US 017.01 - 4 color, black blue green red

Web shop does not handle US shipping address. Order by email to:
Peter at info@lass-hno.de

Peter speaks English, you can write the email in English.

They will make a PayPal invoice / request for payment. You have to manually check your PayPal in a desktop browser to see it and pay it. Doesn't appear in the Android PayPal app or even in the Android browser, but does appear in FireFox in desktop mode.

GEARS:
(motor spindle gears)

Specs:
13 tooth
1.5mm bore
0.25 modulus

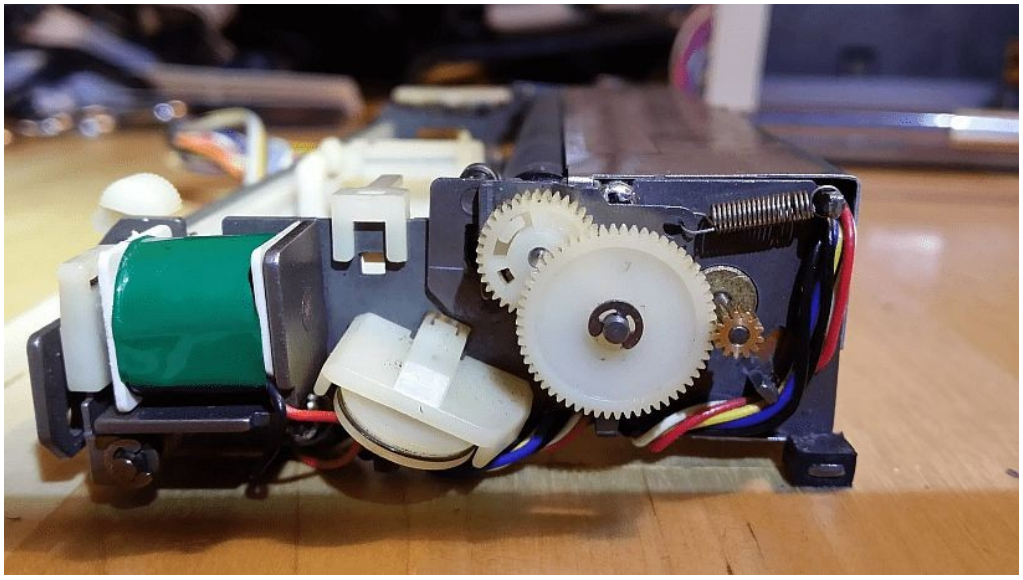


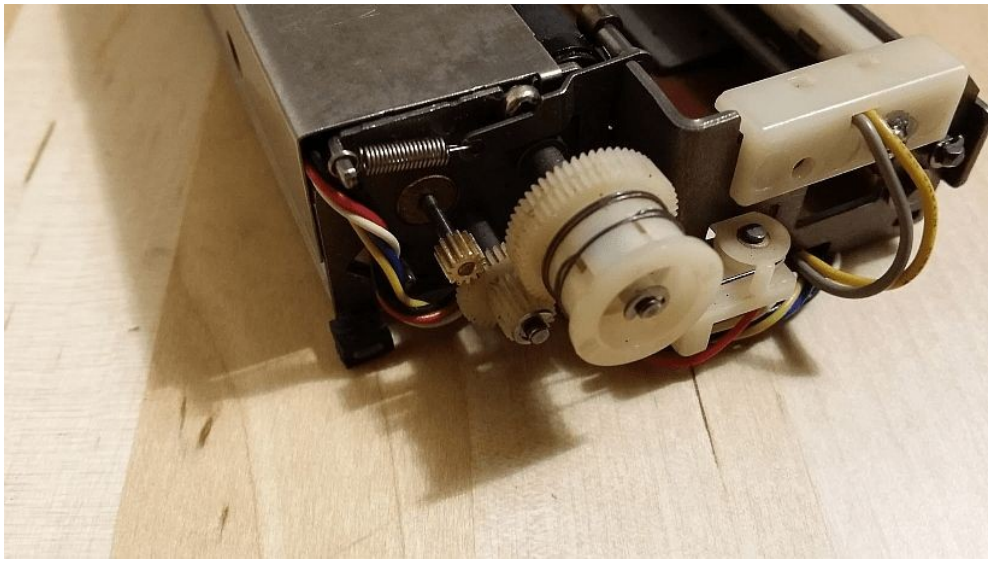
Xtreme Production

Model: EA-055-13
Motor Pinion 13T (1.5mm hole, 0.25M)
http://www.xtreme-production.com/xtreme/index.php?main_page=product_info&cPath=159_307&products_id=3013

Many resellers list the same part. Some have stock while others do not. I had success at rcparty.com

I used a small hobbyist table top vise to press the gears onto the motor shafts. Panavise 301, with plastic jaws. You can remove a thin metal sheet that covers the motors on the back, and that exposes the motors enough to get the corners of the vise jaws on both ends of the motor shaft.
The jaws on that model are plastic, so they don't harm the motor shaft on the back of the motors. If you use a steel jaw vice, be sure to protect the rear motor shaft from direct metal to metal contact.





PAPER:

Specs:
plain bond
4.5" wide
7/16" core
up to 3" diameter
(typically 140 to 160 feet, original Radio Shack item was 180 feet but that's not a common size)

Typical search: 4.5" bond roll

http://www.itplanet.com/parts/Star__87999850.htm

https://www.cibowares.com/4-5-x-160-white-bond-1-ply-register-roll-package-of-20/

http://www.sunshinesupport.com/prod/receipt-paper-rolls/1-ply-45-inch-x-165-paper-25-rolls-45165.htm

---

REFILLING PENS:

The new pens from the German medical supplies company are pretty expensive considering that they are so small and don't last very long.

The new pens can be refilled fairly easily with insulin syringes available from local drug stores. Example: Walgreens item 677232
10-pack, 29 gauge, 0.3mL
super thin syringes for insulin

When looking for alternative inks, you need to know that the original ink is water-based. Use water based drafting/plotter pen ink.
Example:
Chartpak/Koh-I-Noor models
3080-F-BLA black
3080-F-BLU blue
3080-F-RED red
3080-F-GRE green

Clean up with a calligraphy pen cleaning solution, like:
http://artid.com/members/calligraphy/blog/post/1529-pen-and-brush-cleaner-recipe

or Koh-I-Noor Rapido-Eze.
It works to clean up old dried pens too if you soak the disassembled parts in a jar for a day or so.

Fill a syringe to about the 20 mark. (0.2ml)

Around the tip of the pen there are 3 vent holes.
Insert the syringe into one of those. Angle towards the centre a little to minimize how much the vent passage gets opened by the needle.

Slowly (giving the ink time to soak into the fibre filler inside) push the plunger from about 20 to about 10. When you see ink in the other vent holes, you've actually gone slightly too far.

Remove the needle, wipe it down with a damp paper towel and cap it.

Wipe the pen tip with a dry or slightly damp paper towel to clean the excess and wick out a little from the vent holes too.

Repeat for all 4 colors.

Put the printer in 80 column mode (jumper 2 = off) and LLIST a basic program until the writing looks clean. It may splatter a little at first.

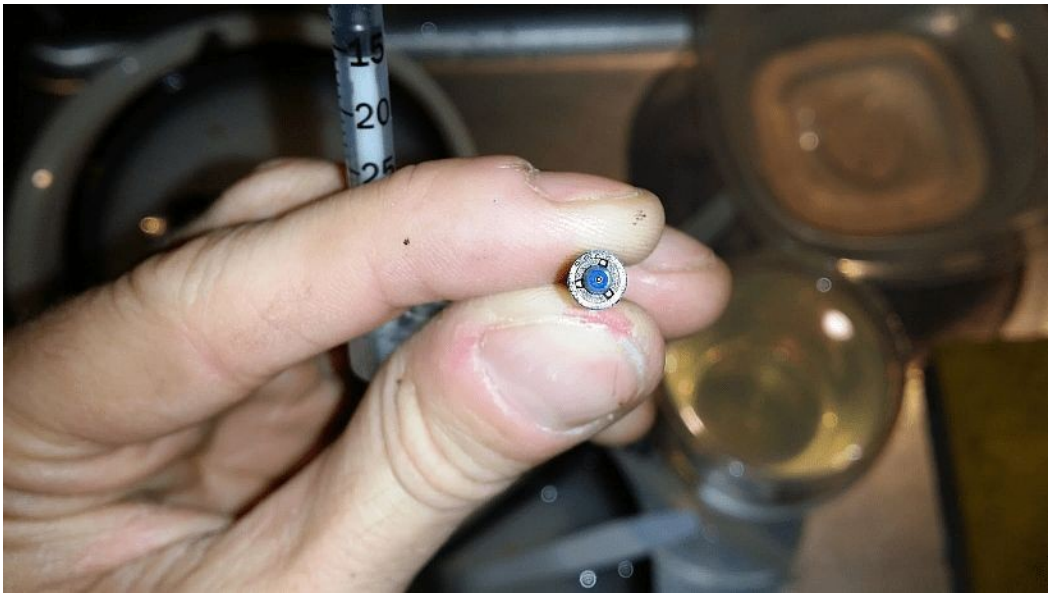
I have no idea how many times a pen can be refilled before the ball point wears out, but even once is worth it, considering how expensive they are and how quickly they run out. I have only refilled each color about twice so far.

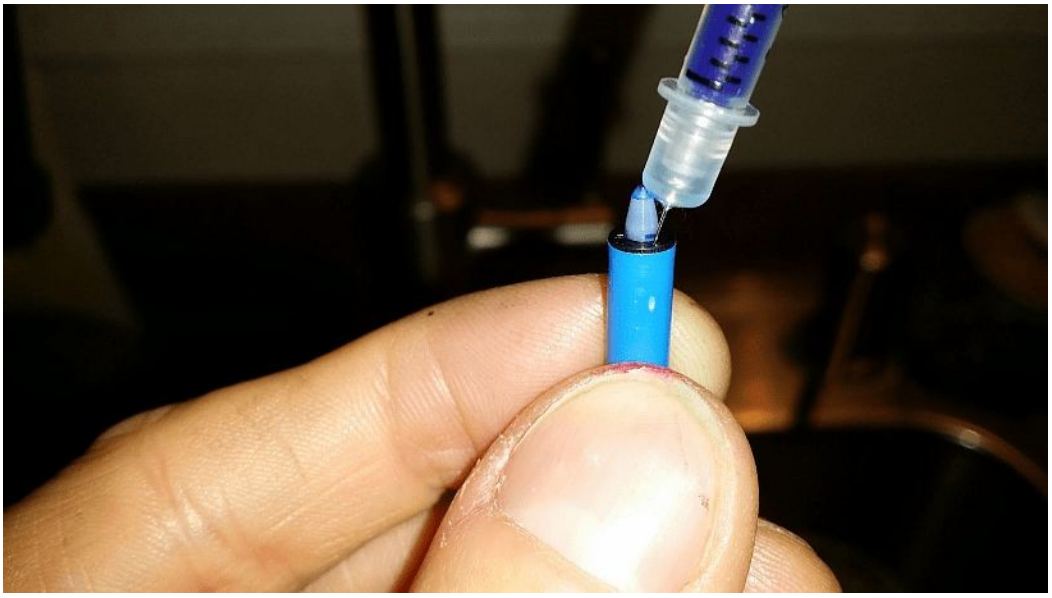
For the old steel barrel pens:
You have to completely disassemble the pens and soak the parts in the cleaning solution for a few hours or days.
You may need to touch the metal ball point tip briefly into a flame to free up the ball from being stuck one time. You have to inject ink with the pen tip off. You can't go it through a vent hole like you can for the new all plastic pens. I've generally had less than 50% success rate refilling and reviving old pens. Even after completely disassembling and cleaning all parts and freeing the stuck ball points. But sometimes it does work.

PICS / VIDEO:

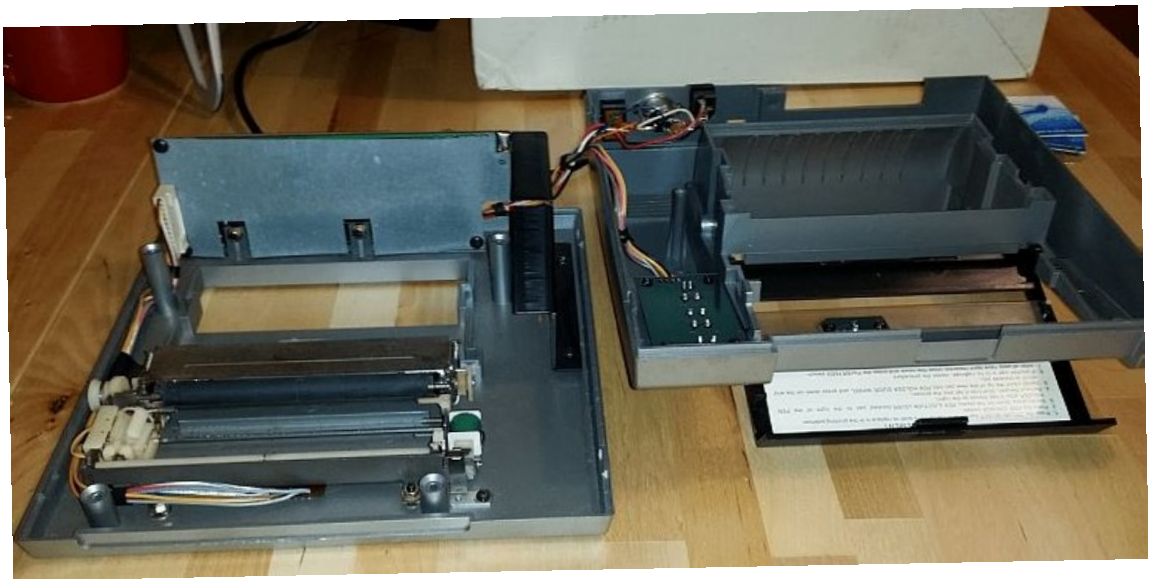
https://goo.gl/photos/8HNKcnyMKBJupYnq9

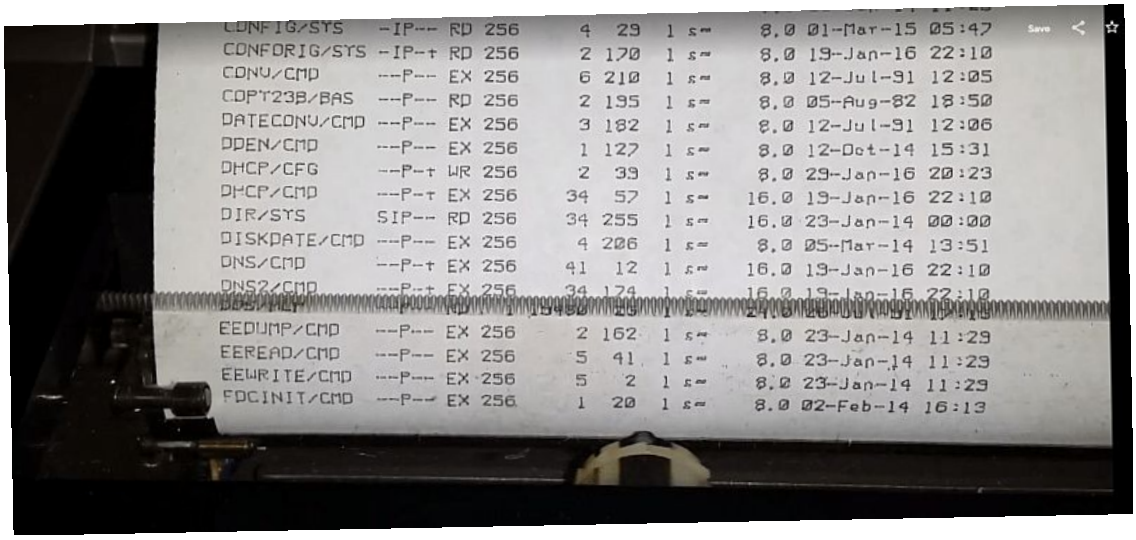

Walgreens
SUPER THIN
SYRINGES
for insulin
Comfort Assured™
1/2 Unit Scale Markings
Latex Free
0.3mL Volume
29 Gauge
1/2" (12.7mm) Needle Length
10 Single-use Syringes









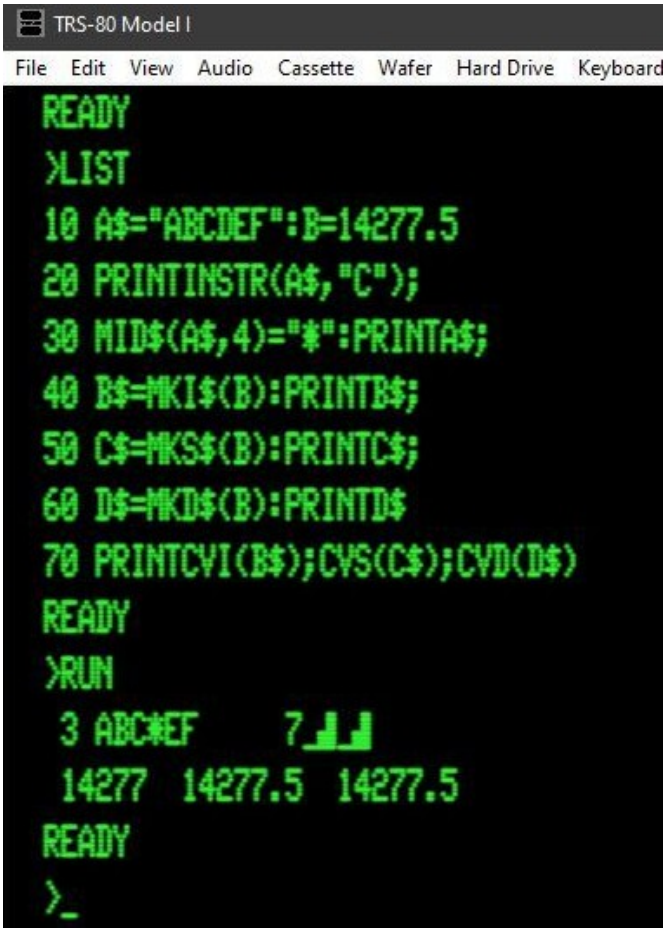


```
CONFIG/SYS    -IP--  RD 256    4   29  1 s=    8.0 01-Mar-15 05:47
CONFORIG/SYS  -IP-+  RD 256    2  170  1 s=    8.0 19-Jan-16 22:10
CONV/CMD      ---P--  EX 256    6  210  1 s=    8.0 12-Jul-91 12:05
COPY23B/BAS   ---P--  RD 256    2  195  1 s=    8.0 05-Aug-82 18:50
DATECONV/CMD  ---P--  EX 256    3  182  1 s=    8.0 12-Jul-91 12:06
DDEN/CMD      ---P--  EX 256    1  127  1 s=    8.0 12-Oct-14 15:31
DHCP/CFG      ---P-+  WR 256    2   39  1 s=    8.0 29-Jan-16 20:23
DHCP/CMD      ---P-+  EX 256   34   57  1 s=   16.0 19-Jan-16 22:10
DIR/SYS       SIP--  RD 256   34  255  1 s=   16.0 23-Jan-14 00:00
DISKDATE/CMD  ---P--  EX 256    4  206  1 s=    8.0 05-Mar-14 13:51
DNS/CMD       ---P-+  EX 256   41   12  1 s=   16.0 19-Jan-16 22:10
DNS2/CMD      ---P-+  EX 256   34  174  1 s=   16.0 19-Jan-16 22:10
EEDUMP/CMD    ---P--  EX 256    2  162  1 s=    8.0 23-Jan-14 11:29
EEREAD/CMD    ---P--  EX 256    5   41  1 s=    8.0 23-Jan-14 11:29
EEWRITE/CMD   ---P--  EX 256    5    2  1 s=    8.0 23-Jan-14 11:29
FDCINIT/CMD   ---P--  EX 256    1   20  1 s=    8.0 02-Feb-14 16:13
```





BASIC, LEVEL 2.5 (TWOHAF)



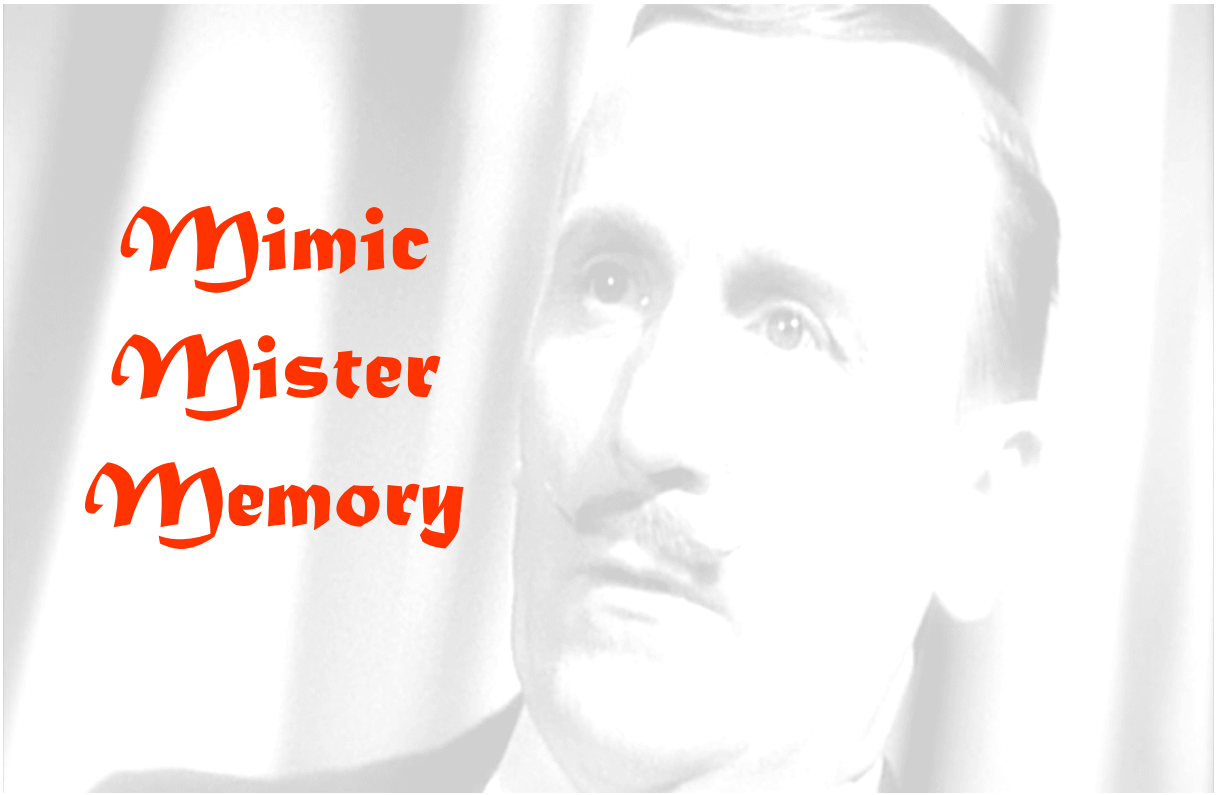
FOR THOSE WHO HAVE READ THE ARTICLE ON A.I., THANKS TO GEORGE PHILLIPS, HERE IS A SCREEN SHOT OF THE UPGRADE TO LEVEL 2.5 BASIC (TWOHAF).

MORE DISK-LIKE FUNCTIONS HAVE NOW BEEN ADDED, AS YOU CAN SEE OPPOSITE.

WITH THE ORIGINAL THREE FUNCTIONS, '&F', 'DEF FN' AND 'DEF USR' THIS BECOMES A MUCH ENHANCED VERSION OF LEVEL 2.

THE UPGRADED CASSETTE AND ACULAB FLOPPY TAPE VERSIONS ARE READY FOR YOU TO USE AND ENJOY AND ARE AVAILABLE FROM THE DOWNLOADS PAGE AT

WWW.TRS-80.ORG.UK



E.T. FONEHUME

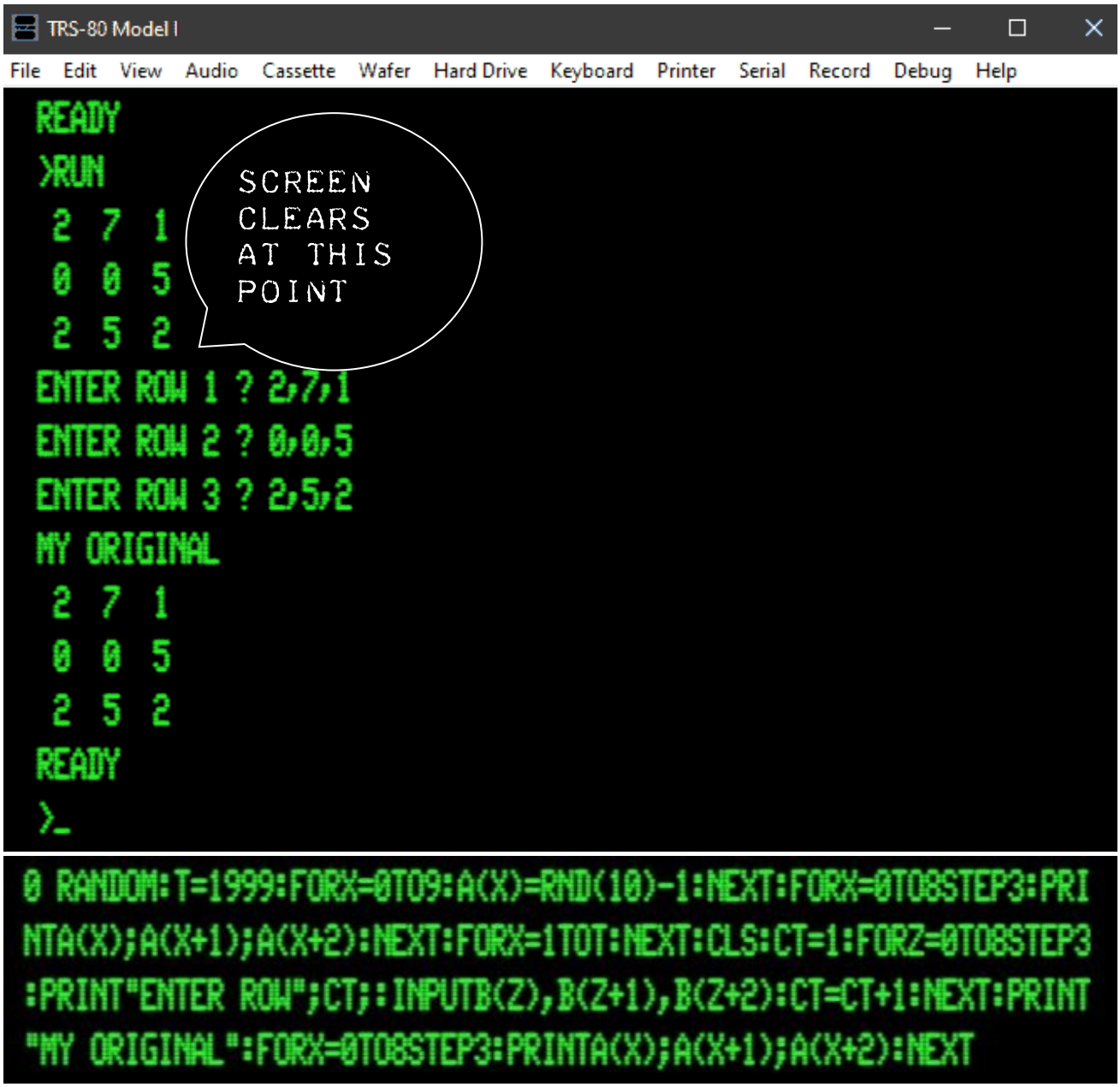
HI DUSTY, HERE'S WISHING YOU AND DEE A VERY MERRY CHRISTMAS AND A HAPPY NEW YEAR.

I THOUGHT I WOULD RESURRECT AN OLD GAME JUST FOR THE XMAS SEASON. THE ORIGINAL TITLE WAS, I BELIEVE, 'MIMIC'. SINCE IT'S OBJECT IS TO TEST YOUR MEMORY, I FOUND THE PICTURE OF THE ORIGINAL 'MR MEMORY' FROM THE 1930'S FILM 'THE 39 STEPS'. A BIT BEFORE EVEN OUR TIME EH!

ANYWAY, AS A ONE-LINER, THE 'FRILLS' HAVE BEEN CUT OUT SO HERE'S THE RULES OK PLAY.

AN ORIGINAL 3X3 GRID OF NUMBER, FROM 0-9 IS SHOWN ON THE SCREEN FOR A LIMITED TIME. YOU NEED TO REMEMBER THE NUMBERS AND THEIR POSITIONS BEFORE THE SCREEN CLEARS.
YOU'RE THEN REQUIRED TO ENTER THE NUMBERS FROM THE THREE ROWS IN THE CORRECT ORDER. IT'S BEST TO ENTER THE THREE NUMBER TOGETHER VIZ '1,2,3' AS SHOWN. WHEN YOU'VE COMPLETED THAT, THE ORIGINAL GRID IS SHOWN AND YOU CAN COMPARE YOUR ANSWERS! QUITE TRICKY AT FIRST UNTIL YOU GET THE HANG OF IT. JUST REDUCE THE VALUE OF THE VARIABLE 'T' TO INCREASE THE DIFFICULTY LEVEL. UNDER 500 IS VERY TRICKY!

TO MAKE IT INTO A GAME, KEEP SCORE OF YOUR ERRORS AND, SAY, AFTER THREE GOES, THE PERSON WITH THE LOWEST NUMBER WINS! LOSERS (OR DO I HAVE TO CALL THEM RUNNERS-UP NOWADAYS?) DO THE WASHING UP, OR PERHAPS, THIS YEAR, THEY COULD CLEAN GRANDDAD'S CAR!




```
0 RANDOM:T=1999:FORX=0TO9:A(X)=RND(10)-1:NEXT:FORX=0TO8STEP3:PRI
NTA(X);A(X+1);A(X+2):NEXT:FORX=1TOT:NEXT:CLS:CT=1:FORZ=0TO8STEP3
:PRINT"ENTER ROW";CT;:INPUTB(Z),B(Z+1),B(Z+2):CT=CT+1:NEXT:PRINT
"MY ORIGINAL":FORX=0TO8STEP3:PRINTA(X);A(X+1);A(X+2):NEXT
```

HERE'S A LITTLE ITEM THAT CAUGHT MY EYE RECENTLY, ALTHOUGH NOT REALLY TANDY RELATED I JUST THOUGHT I'D SNEAK IT IN! I CURRENTLY HAVE MY HEAD IN A BOOK BY MAX HASTINGS ENTITLED "THE SECRET WAR". IT'S SUB-TITLED "SPIES, CODES AND GUERRILLAS". (ISBN NUMBER 978-0-00-750374-2). IT'S A FASCINATING READ. THOUGH SOMEWHAT HARD GOING AT TIMES. I FIND IT ASTOUNDING TO HEAR ABOUT THE TRICKS THAT BOTH THE ALLIES AND THE AXIS FORCES 'TRIED ON'.

THE BIT THAT MADE ME SMILE WAS, WHEN THINGS WERE QUIET AT BLETCHLEY PARK, THIS WAS REFERRED TO AS 'DOODLE TIME' AND ONE OF THE 'BOFFINS', PETER HILTON, ONCE USED SUCH TIME TO COMPOSE A PALINDROME: *"DOC NOTE, I DISSENT, A FAST NEVER PREVENTS A FATNESS, I DIET ON COD"*

I WAS TEMPTED TO ASKED MY 'NEW BEST FRIEND' CHATGPT, IF IT COULD DO ANY BETTER. IT STRUGGLED A BIT AND COULD ONLY ADD A NUMBER OF SMALL PALINDROMES TOGETHER TO BEAT 14 WORDS.
THE BEST SINGLE ONE IT CAME UP WITH, THAT'S NEW TO ME, WAS *"EVA, CAN I SEE BEES IN A CAVE?"*

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



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

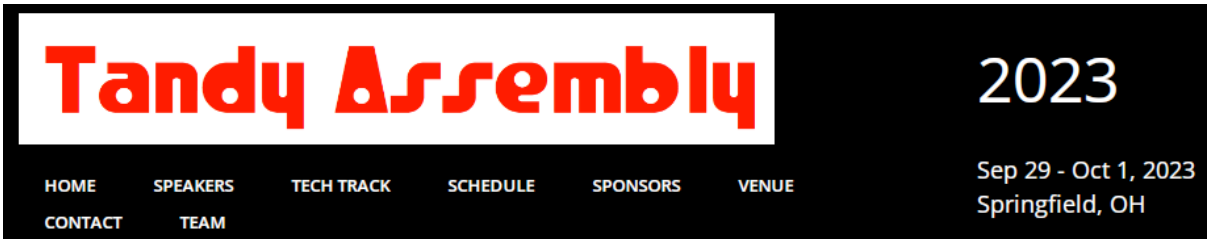
newSOFT

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

IN MAV'S TRAVELS

TA23 Report

by Ian Mavric



Tandy Assembly 2023 was held from September 29-October 1. Here is my yearly report on the great event. Once again the show was held in Springfield Ohio, at the Marriott Courtyard hotel. This year, as in most years, we had fantastic weather and the show ran perfectly without a hitch.

THURSDAY 28th SEPTEMBER - THE DAY BEFORE THE SHOW

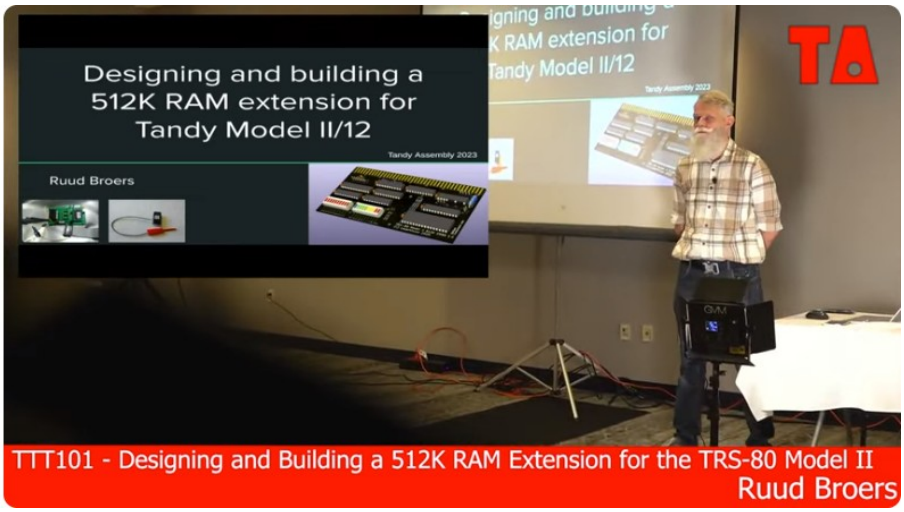
A group of us went to the usual USAF museum on Thursday, mainly those who arrived Wednesday night. While at the museum we also ran into Ruud Broers, a TA regular, and his wife who had been travelling in the USA and Canada for a couple of weeks before TA.

Afterwards we headed to the usual Chinese Buffet for late lunch which was not far away from the USAF museum and a bargain. Six of us went and I don't think one of us paid more than $15 including drinks! The great thing about this place is that as well as the usual Chinese staples you would expect, there is a good number of typical western foods to choose from if Asian food is not your thing. After this we headed back to the Marriott, went to the bar and thought about dinner. Typically our function room for the event would be handed over in the afternoon but this time it would be still in use and not available until mid-morning on Friday, so most of us had an early night and made plans to set up our tables before the Tech Track talks started at 1pm.

FRIDAY 29th: THE TANDY TECH TRACK (TTT) CLASSES

The day was always going to be a very busy one so we started with a hearty breakfast and started on setting up our tables once the rooms was handed over to us. The five TTT classes for the day were:

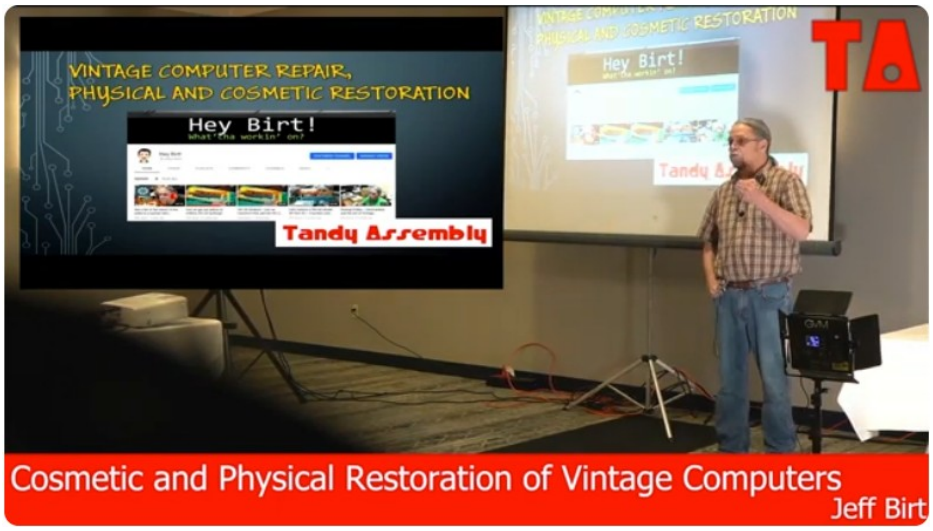
TTT101: DESIGNING A 512K RAM EXTENSION FOR THE MODEL II



https://www.youtube.com/watch?v=_I01NHzUzNk

Ruud talked about how he designed a 512K Ram extension for the TRS-80 Model II. An example of how the computer was designed to be able to use large amounts of memory, much more than the 64K the Model II came with, but it was physically impossible in 1979 to accomplish that using 16Kx1bit chips.

TTT102: COSMETIC AND PHYSICAL RESTORATION



https://www.youtube.com/watch?v=cMDg_5pAbC0

Jeff Birt was back again, this year talking about a very important topic, the physical and cosmetic restoration of vintage computers. Many of us can repair the electronics of a computer, but the cosmetics often are not attempted or attempted with the wrong method. Jeff provides a great talk everyone should watch.

TTT103: TRS-80 ACCESSORIES WHICH FELL SHORT OF THE MARK



https://www.youtube.com/watch?v=oVBCiEPTtxY

Once again this year I teamed up with Eric Dittman to talk about some of the TRS-80 items which, for various reasons, turned out to be rather underwhelming. Printers, computers, software, as well as accessories all cop a serve in this fun and at times irreverent talk.

TTT104: TRS-IO++: ONE CARD TO RULE THEM ALL



https://www.youtube.com/watch?v=LUctEBIcoBQ

Arno Puder teams up with Matt Boytim to discuss the latest iteration of the TRS-IO card which has been around for a while. The new card works on the Model I, as well as the III and 4, and is a fun project for the DIY types who like to build their own devices.

TTT105: USING MULTIDOS: AN INTRODUCTION



https://www.youtube.com/watch?v=3Ws1bQWp534

Tim Halloran returned this year, with a continuation of Vernon's talk from last year about MultiDOS, this time how to drive the operating system. With the addition of MultiDOS now available for the FreHD this is a must-see talk.

After the TTT sessions a dinner is supplied to everyone who attended the sessions, either as a member of the viewing audience or presenter. The meal was great and the room filled with conversation for a couple of hours afterwards.

SATURDAY 30th SEPTEMBER: IT'S SHOW TIME!

The first day where the general public come to the show is Saturday and we were all ready and they started coming in at 10am. It's also the busiest day of the show.

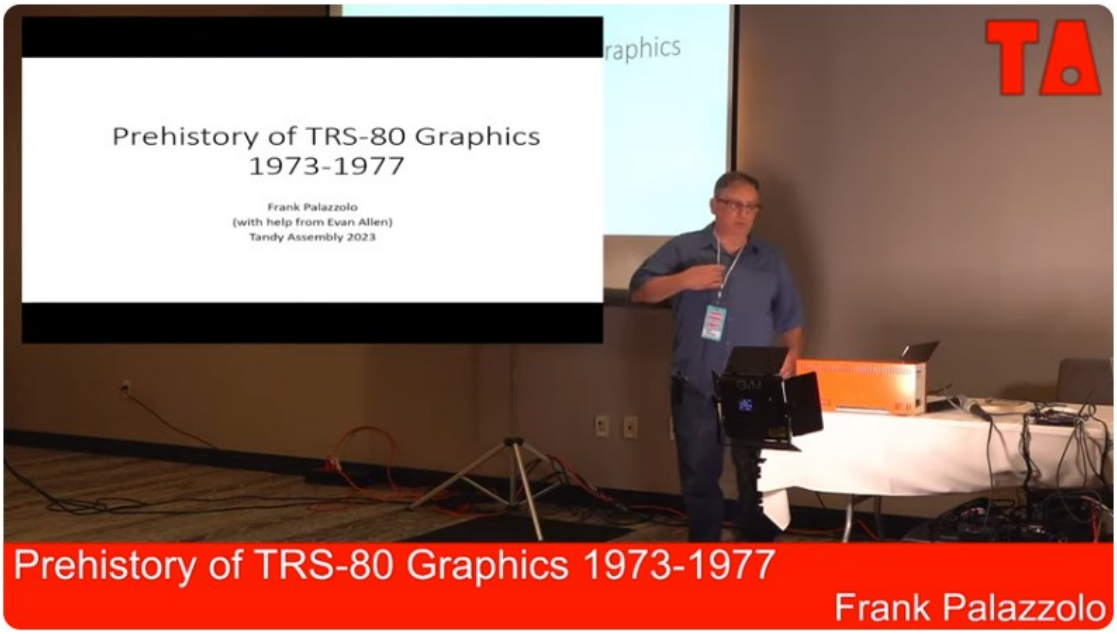
After breakfast, those if us with stands (or tables, or displays , or exhibits) finish getting them ready. Talks continue throughout the day, the first one for Saturday was Lawrence Kesteloot:



https://www.youtube.com/watch?v=UmPeaTayUtc

Lawrence gave a fascinating talk about the tools he developed and used for working with the TRS-80 in the current day and age.

At 1pm the next talk was from Frank Palazzolo:



https://www.youtube.com/watch?v=0qts0onGJJo

Frank gives us a great history lesson which covers the timeline up to the introduction of the Model I. A deep dive into how the display and basic graphics the computer was capable of (64x16 character and 128x48 graphics) for those interested in learning why the Model I and III have the graphics they have.

AUCTION TIME!



https://www.youtube.com/watch?v=TohQQqbgLqQ

The popular no-reserve auction was the next feature of the day, starting at 2:30pm and running for approximately two hours. I think everyone who was after a bargain got one, and I had a great time being auctioneer again. The rapid rate at which I try to go through the auction lots (about 1 per minute for 100+ items) makes for lively bidding and a few laughs along the way. Some of the more interesting lots included a highly-upgraded Coco 3, Tandy CM-8 monitor, a gorgeous restored Model II system, mountains of software, books and magazines, modems and test gear.

TANDY TRS-80 TRIVIA GAME



https://www.youtube.com/watch?v=CSdPqO6pbXg

After dinner we had the ever popular trivia game with three experts up on the front panel. Eric Dittman won up against good efforts by Eric Canales and Ron Lauzon. The audience were also involved after last year I concurrently ran an audience game (similar to pub trivia on a smaller scale) and this time we had some 50 entrants!

A countdown style finish was employed to deduce who won the audience game (there are only 10 points max which could be obtained) so I was pleased to find 5 winners who got 9 points. No one got the maximum, which is 10 points. Since we are onto a winner with the new trivia game format it will continue this way next year.

Evening socialising concluded the day and most people were ready to turn in for the night around 10pm after a very busy and tiring first day.

SUNDAY 1st OCTOBER: THE REST OF THE SHOW

One last talk concludes the show was Stephen Frey who runs the Dayton Microcomputer Association Computer Museum and talks about the TRS-80s he used back in the 80s and the machines he has in his museum.



https://www.youtube.com/watch?v=joBNIZ60F-M

And before you knew it the show was over. At around 3pm we start packing up and saying our goodbyes. Final orders for products are placed and talk about the long drives and flights home are in abundance. After the show a large group of us headed to a Texas themed steakhouse for a well earned dinner before heading back to our hotel rooms to pack, or those leaving that night to begin the long drive home.

HUNGRY? ME TO...

For the first time a stall with cakes, cookies, cotton-candy run by Pastries&Pearls girls was a big hit with everyone with a sweet tooth. We hope they will return again in 2024.



ANYONE IN THE SOUTHERN STATES

For those who can't get enough Tandy Assembly and live in Texas or neighbouring states, a miniature TA will be held in June 2024 in Dallas as part of the VCF-SW event. An area has been set aside for Tandy and TRS-80 related exhibits, which will carry the Tandy Assembly name. Being close to where the computers were designed and built, as the show comes together people should keep their eyes on the VCF-SW web-site to see which TRS-80 old-timers and features the show will comprise of.



https://www.vcfsw.org/

I HAVE GRADUATED TO TANDY ASSEMBLY COMITTEE

I was pleased to find I've been added to the Tandy Assembly committee. The first meeting of the TA committee had an interesting agenda item to discuss which was my addition to the organisation team due to my increasing roles within the show in recent times. The team sent me an invite and I eagerly accepted.

My role for the moment is to keep TA in people's minds by posting weekly on the Tandy Assembly Facebook page. As TA24 draws closer and announcements are made I'll update the news there and on the discord server. Dusty also mentioned a quarterly TA column might work for the newsletter so I will see you all again next issue.



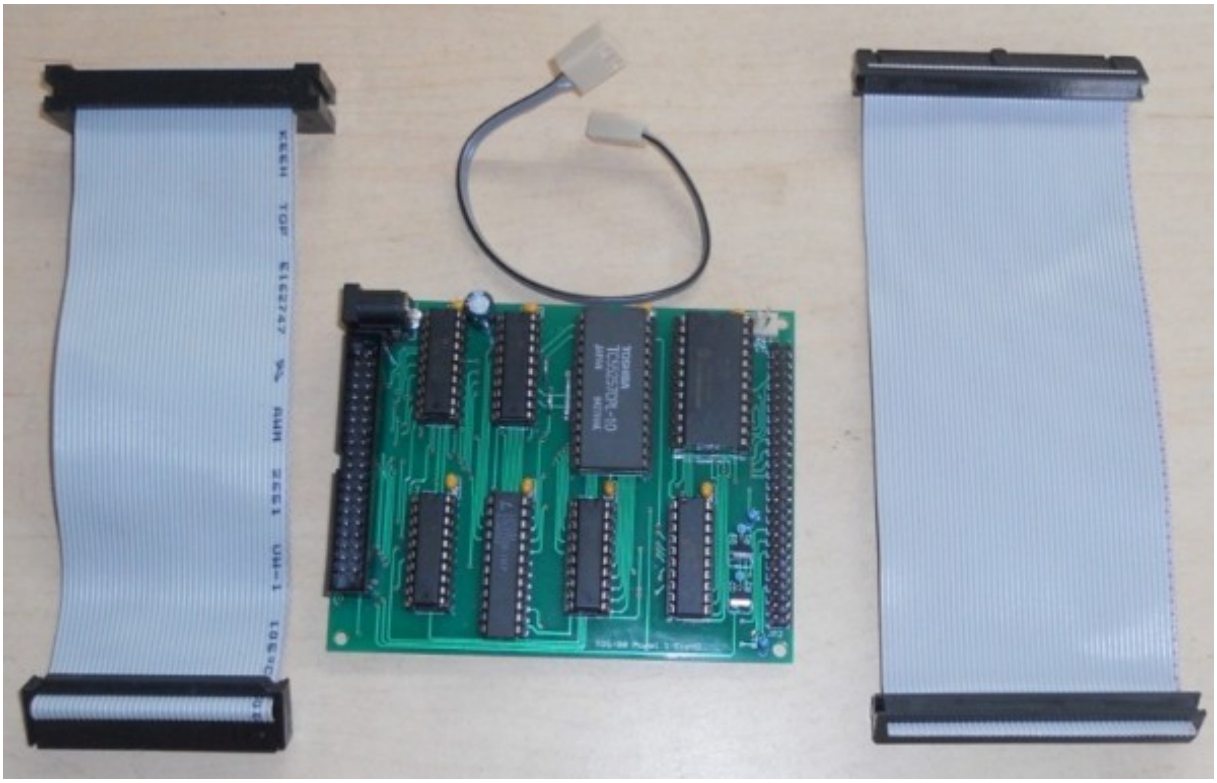
Ian Mavric
ianm@trs-80.com



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.

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

- COMPLETLY ASSEMBLED AND TESTED -



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

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

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

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

The amazing

"FreHD"



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

Order yours today from
http://www.trs-80.com.au/

Thirsting for adventure?
Brass lantern getting dusty?



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

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



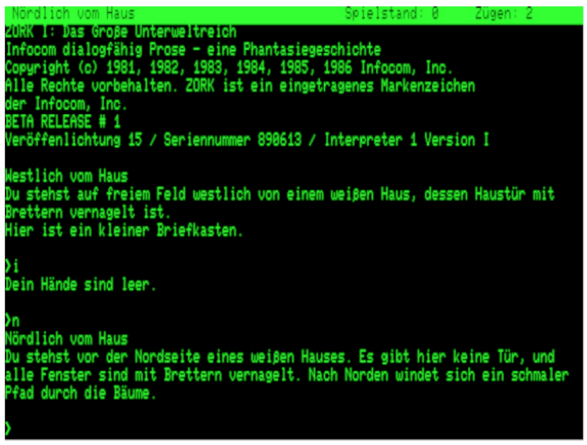
M4ZVM is available for download from https://gitlab.com/sijnstral/m4zvm (includes feature details and source)
Enhanced versions ready for download!

*128K recommended

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



Beyond Zork



Zork 1 (German)

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

Surfin' The Net



Scott's RETRO TECH Workshop

Internal audio for M1

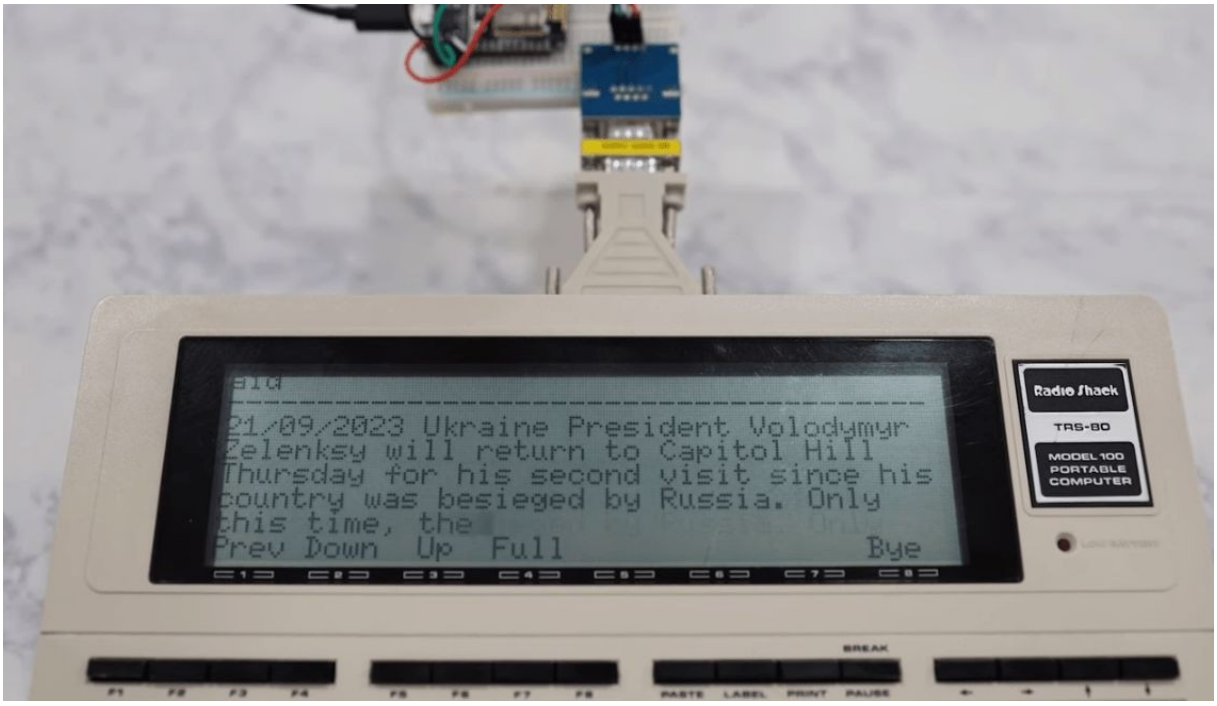
Add internal audio to the TRS-80 Model 1. With the parts from Amazon and a little soldering and drilling one inconspicuous hole you can have sound without an external powered amplifier.

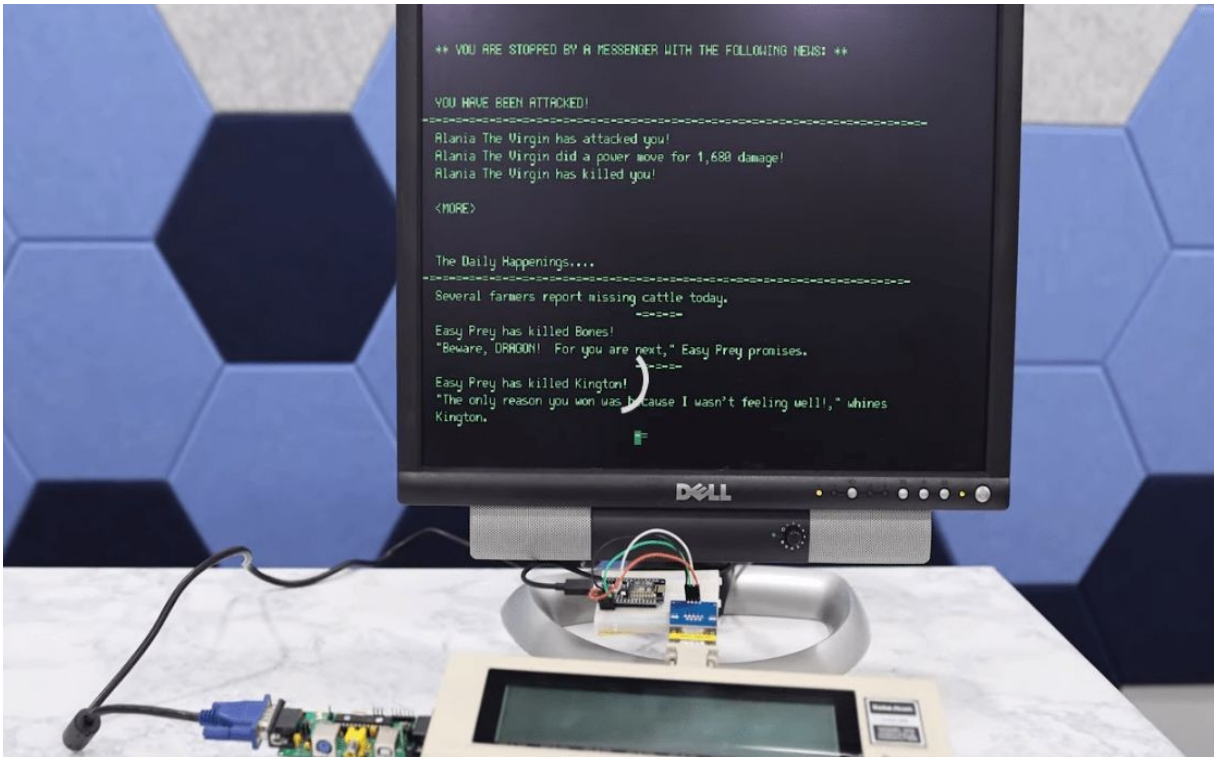
https://www.youtube.com/watch?v=CNKTvUUDiAM




Cosmic
Fighter

What's Ken making





https://www.youtube.com/watch?v=tCyuzZlxT3w



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



STOP! PAYING HIGH SOFTWARE PRICES, IF YOU OWN A TANDY OR GENIE

CLOAD MAGAZINE IS FOR YOU

OVER 10,000 COPIES A MONTH SOLD IN AMERICA THIS MONTHLY CASSETTE WITH AT LEAST 6 PROGRAMS IS NOW AVAILABLE IN EUROPE.

ISSUE 44 (LAST MONTH) INCLUDED: CROLON DIVERSION (ARCADE GAME WITH SOUND) * TROLLS TREASURE (FULL LENGTH ADVENTURE) * RPN CALCULATOR * PLUS A FULL SCREEN EDITOR AND BANNER (PRINTS 2" LETTERS ON SCREEN)

ISSUE 45 (THIS MONTH) INCLUDES: * TENPIN BOWLING (FOR 2 PLAYERS WITH SOUND) * FRANKENSTEIN ADVENTURE * HEAT/ENERGY AUDIT * CALENDAR PROGRAM

* HUNDREDS OF BACK ISSUE PROGRAMS AVAILABLE. SEND FOR DETAILS
* TANDY COLOUR COMPUTER OWNERS SEND FOR DETAILS OF CHROMASETTE MAGAZINE

CLOAD IS AVAILABLE £4.95 SINGLE COPY (STATE WHICH)
£25.00 SIX MONTHS SUBSCRIPTION

(STATE WHICH ISSUE TO START)
£47.50 12 MONTHS SUBSCRIPTION
(STATE WHICH ISSUE TO START)

ONLY FROM: MICRODEAL

DEAL HOUSE, LUXULYAN, BODMIN
CORNWALL. Tel 0726 850821

Peter A. Stark
PO Box 209
Mt. Kisco NY 10549

Solving Keyboard Interface Problems

... would you believe a UART ?

A common need is for a device to convert the parallel ASCII output of a typical cheap keyboard into a serial Teletype-compatible output. A new UART (Universal Asynchronous Receiver-Transmitter) from Intersil makes the job very easy, and this article describes a simple circuit which easily adapts to a wide variety of uses.

The UART is a 40-pin integrated circuit made by several manufacturers under different numbers. It consists of two parts: The Receiver receives serial teleprinter data (in ASCII, Baudot, or any other code), one bit at a time, and converts it into a parallel form, eight bits at once. The Transmitter does the exact opposite, converting parallel data into serial format. Since UARTs are generally used for sending teleprinter characters, the parallel group of bits sent at once generally represents one character. Eight bits would be used in ASCII, five bits in the older Baudot code, and UARTs also can send six or seven bits at a time.

The word *asynchronous* means that these characters can appear at random intervals, either one right after another, or perhaps separated by unknown times. To indicate to the receiver when the next character is arriving, each serial group of bits is preceded by a start bit and followed by one or two stop bits. The UART transmitter automatically generates these bits, and the receiver also checks for them and uses them for timing. In addition, the transmitter can generate an optional parity bit, which is used by the receiver to check whether a possible error has occurred in character transmission. Although the UART can be used both as a transmitter and as a receiver (at the same time), in this circuit we use it only as a transmitter, converting parallel data such as might come out of a keyboard or computer output port into serial data which might be needed by a teleprinter.

The first UARTs needed two operating voltages, -12 volts and +5 volts, and consumed appreciable amounts of power. Later versions eliminated the need for the -12 volt supply and could therefore run on a single +5 volt supply such as might be needed in typical digital systems. Intersil's UART operates from a single +5 volt supply and has the further advantage that it requires less than one half milliampere of supply current. Four versions of the unit are available. The IM6402 is TTL compatible and operates on +5 volts. The IM6402A is CMOS compatible and operates on any voltage between +4 and +11 volts. The IM6403 and IM6403A are similar but have a built-in oscillator and divider circuit which makes them very easy to use. All of these ICs are second sourced by Harris under the numbers HM6402, etc.

In order to set the precise operating speed of a UART, a frequency oscillator is needed to generate a frequency exactly sixteen times the bit rate of the serial signal. For example, standard ASCII teleprinter signals send at 110 bits per second, and thus a frequency of 11 x 16 or 1760 Hz is needed. This frequency is often generated with a separate 555 timer, or more exactly with an external crystal oscillator and a fre-

quency divider IC. The IM6403 and IM6403A are especially useful because they have that oscillator and divider built-in. Only one component is needed to generate a clock: an external crystal. For standard 110 bit-per-second speed a 3.58 MHz color TV crystal is needed. The crystal is simply connected between pins 17 and 40 of the UART as shown in Fig. 1. A built-in divider divides that frequency by 8192 to produce almost exactly 1760 Hz. This scheme tremendously simplifies any design and has only one disadvantage: Whereas in a standard UART the receiver and the transmitter can run at different speeds, in the IM6403 they must run at the same speed since they share the same crystal.

Fig. 1 shows the diagram of a simple circuit using the IM6403. The parallel ASCII input at Bit 1 through Bit 7 comes to the UART at the upper left; the serial output exits at the right. A "character present" strobe signal applied to either the A or the B input of a 74121 monostable provides the start pulse for the UART which tells the transmitter to start working.

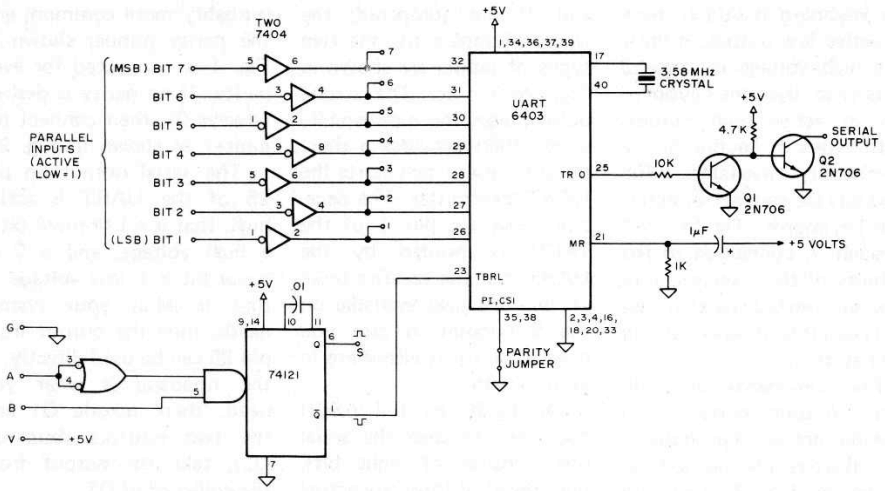


Fig. 1. Schematic diagram.

The only other components required are an RC network connected to pin 21 which provides a Master Reset when power is first applied. The following paragraphs describe some of the input and output characteristics; Fig. 2 shows some alternative wiring arrangements.

As shown in Fig. 1, the inputs are designed for a keyboard which provides a low voltage for a 1 and a high voltage for a 0. Two 7404 ICs are used to invert these logic voltages to a high for a 1, as required by the UART. If the 1 is a low voltage near 0 volts,

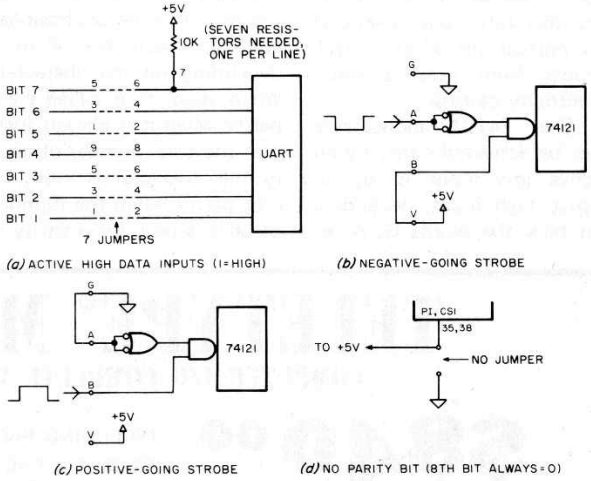


Fig. 2. Optional connections.

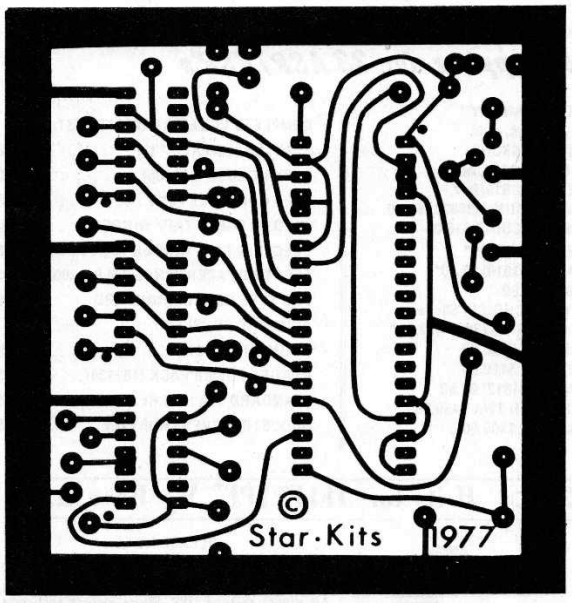


Fig. 3. Printed circuit board (full size).

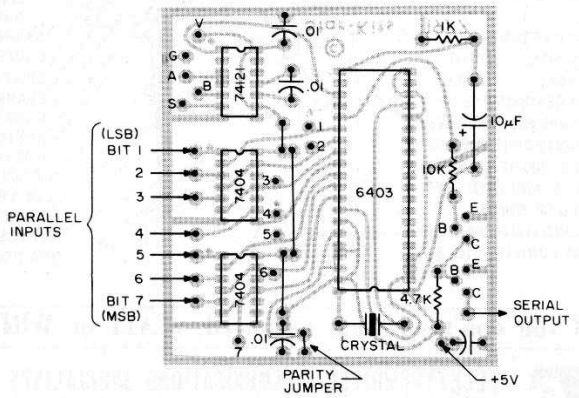


Fig. 4. Parts placement.

the keyboard is said to have an active low output; if the 1 is a high voltage of some 3 volts or so, then the keyboard has an active high output. Both types of keyboards are commonly available. The inverters are needed for active low keyboards. Tie points 1 through 7, connected to the outputs of the inverters, provide an inverted signal in case you need it somewhere else in your system.

For keyboards or computer output ports which provide active high outputs no inverters are needed; as shown in Fig. 2a they are simply left out and bypassed. But in this case the tiepoints should be used to provide seven 10k pullup protection resistors (only one is shown) to protect the seven UART inputs from possible static electricity damage.

The 74121 monostable can be activated either by an active low input or by an active high input, depending on how the points G, A, B, and V are jumpered; the correct jumpers for the two types of inputs are shown in Fig. 2b or 2c. The strobe pulse pulses the monostable, which then provides a short output pulse which starts the UART transmitter. The negative spike on pin 1 of the 74121 is needed by the UART, but the positive spike on pin 6 is also available on the S tiepoint in case you have a use for it elsewhere in your system.

Although the full ASCII code as sent over the serial line consists of eight bits, only seven of those are actual data. The eighth is generally used for parity. In many home-type systems this bit is intentionally made a 0, whereas in more professional systems it may be a 0 or 1 depending on the character. When used, it is called *even* parity when it is chosen such that the total number of ones in the character is even, or *odd* parity when the number of ones is odd. Even parity is probably more common, and the parity jumper shown in Fig. 1 is connected for even parity. If no parity is desired (always 0), then connect the jumper as shown in Fig. 2d.

The serial output on pin 25 of the UART is active high; that is a 1 or *mark* bit is a high voltage, and a 0 or *space* bit is a low voltage. If that is what your system needs, then the output from pin 25 can be used directly. If the opposite is what you need, then include Q1 and the two resistors (but not Q2); take the output from the collector of Q1.

Q2 is peculiar to my system because all serial lines in my system use an unusual convention: an open circuit is a 1 (mark), whereas a grounded circuit is a 0 (space). The collector of Q2 does just that – it is open for a mark and is grounded for a space. My Teletype machines have built-in control circuits which translate from this open-grounded configuration to a loop current basis. The advantage of such a setup is that this allows multiple inputs and outputs to be paralleled, as can a keyboard and an amateur radio-Teletype converter. Whichever input is active at the moment, generating spaces, provides an output. In the same way, this signal can be fed in parallel to a teleprinter, to my video display, and to my acoustic coupler. Simple phono plugs or phone plugs can be used for switching inputs and outputs around in any configuration.

The circuit is easy to build; since no very high pulse rates are involved, almost any construction method can be used. If you wish, you can use the printed circuit board layout of Fig. 3, with the parts locations as shown in Fig. 4. Use the indicated layout to make your own board; alternatively, drilled and plated boards will be available from the author for $7 each. ■

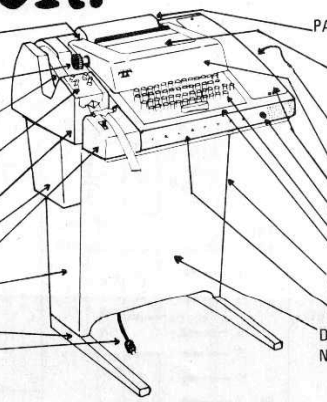
TELETYPE© MODEL 33 ASR

COMPUTER I/O COMPLETE WITH:

- Tape Punch
- Tape Reader
- Ready to Go
- Line/Local Wired
- Guaranteed 30 Days

$840.00 INCLUDING PACKING
F.O.B. - N.J. FACTORY

LOOK! *Hard to Get Parts & Supplies for 33 ASR/KSR's*



PAPER SPINDLE (181043) $1.25
TAPE SPINDLE (182918) $1.25
PLATEN KNOB (180039) $1.00
PAPER TAPE
28 ROLLS 1 CARTON $32.00
7 ROLLS 1 BOX $10.00
MYLAR 1 ROLL $23.00
REPERF COVERS (182109) $3.50*
Replacement UPE-800 PUNCH $75.00
CHAD BOX (182965) $3.50
Replacement UX800 READER $50.00
REPAINTED STAND WITH FEET (183246) $30.00
FEET ALONE (183243) $12.00/Pr.
LINE CORD (182510) $3.25

PAPER – WHITE or CANARY $22.00 / 12 - 5" diam. rolls.
BLACK NYLON RIBBONS
SINGLE SPOOL $11/Doz.
DOUBLE SPOOL $15/Doz.
ANSWER BACK DRUM (180827) $7.00
CODED (SPECIFY CODE) $10.00
DOME LID (181137) $5.75*
BLANK PLATE (181910) $7.00*
KNOB (181824) $1.00
KEYTOPS (YOUR CHOICE) 60c ea.
ONE PIECE COVER (187300) $22.00*
UCC-6 LINE-LOCAL $150.00
UCC-3 FOR TWX $150.00
FRONT PLATE (181812) $2.50
DATA SETS 101C FOR TWX $450.00
NEW 105A FOR TWX $300.00

COMPLETE COVER READY TO INSTALL INCLUDING ALL *'s	$ 35.00
COPY HOLDER (182036)	$ 14.00
TAPE UNWINDER (12")	$ 32.00
WIND UP PAPER TAPE WINDER	$ 22.00
ELECTRIC TAPE WINDER	$ 55.00
ELECTRIC PAPER WINDER (LPW-300)	$ 50.00
DEC TYPE READER RUNCARD	$ 45.00
ACOUSTIC COUPLER OMNITEK 701A	$160.00
READER POWER PACK (182134)	$ 35.00
STANDARD EIA INTERFACE	$ 55.00
ACOUSTIC COVERS with FAN	$258.00

If you don't see what you need, CALL or WRITE! ○ If its for TELETYPE© We Have It.

TELETYPEWRITER COMMUNICATIONS SPECIALISTS
550 Springfield Avenue • Berkeley Heights, N. J. 07922
(201) 464-5310 • TWX: 710-986-3016 • TELEX: 13-6479
Deaf Line TTY: 201-464-5314
SUBSIDIARY OF VAN'T SLOT ENTERPRISES, INC.

PLEASE NOTE
Due to increased costs of processing orders minimum orders: $10.00 cash or check, $25.00 charge. All orders shipped UPS. Please include sufficient postage.

T13

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



The Ghosts of Blackwood Manor is an award-winning interactive horror from Stefan Vogt, the acclaimed author of Hibernated and The Curse of Rabenstein.

Take on the role of novelist Thomas King and his wife Cora as they move into a remote Scottish manor house, just days before Christmas 1986. Seeking inspiration for his next book, Thomas begins to explore the history of Blackwood Manor, only to uncover a dark secret that dates back centuries to the days of the Great Scottish Witch Hunts.

The Ghosts of Blackwood Manor is a complex interactive fiction piece written in Infocom's advanced Z-machine format XZIP. The game has three different endings, a bad, a neutral and a good ending, depending on the player's choices. It is very likely that the player, on a first playthrough, will not reach the good ending, even though it is possible. Reaching the bad or neutral ending first is part of the experience and helps the player understanding the true intentions of some of the NPCs and the impact of decisions made, so they may be reconsidered.



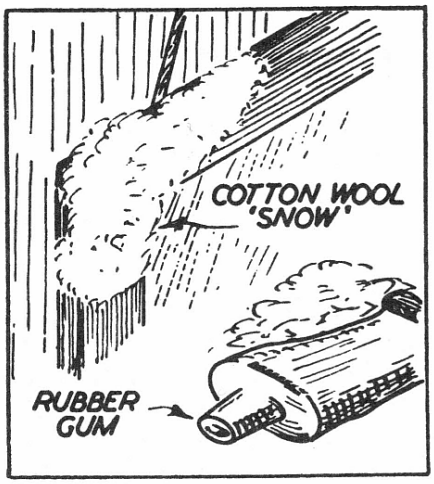
You can order or download the game at https://8bitgames.itch.io/ghosts

HANDY HINTS



HERE'S SOMETHING JUST A BIT DIFFERENT. IT'S A 'HANDY HINT' FROM THE 1955 'GADGETS ANNUAL'. ANOTHER GREAT IDEA! YOU CAN SUBSTITUTE 'RUBBER GUM' WITH 'BLU-TACK' NOW-A-DAYS.

1955 GADGETS ANNUAL



HOW TO DECORATE WITH

COTTON-WOOL SNOW

SCRAPS of cotton wool on pictures and other ornaments, make excellent seasonal decorations at Christmas time, coupled with sprigs of holly and mistletoe. To get the best effect, stick the "snow" in place, wherever required, with just a touch of rubber gum. You can also attach "snow" to vertical surfaces in this manner.

When the time comes to remove the decorations, the cotton wool can be pulled off and all traces of gum can be rubbed off with the fingers.

SOME SIMPLE TOYS MADE FROM ODDS AND ENDS

IT'S fun making toys, especially when there's a kiddy in the house ready and eager to play with them. The following suggestions are for simple toys that can be knocked together from odds and ends at no expense.

Every kiddy likes a toy that can be pulled along on a piece of string, so here is something both simple and suitable. Take a large cotton-reel, then cut a couple of discs from plywood and nail them at each end of the reel. The discs should be about 4in. diameter, each having a small hole at the centre. If a long loop of cord is threaded through the reel, the toy can be pulled along. But this toy can be greatly improved by fixing a little bell around the reel with a short length of string, so that as the reel revolves the bell will ring. A further improvement would be to paint bright bands of colour on the discs, so that they would merge prettily when the toy was in motion. Many children will appreciate this simple plaything far more than an elaborate toy costing a great deal of money.

Another simple toy is a miniature doll's house consisting of several small boxes and a roof, all separate, that can be assembled in various ways. The boxes need no description beyond that they should be open-fronted and not too deep. Any size will do so long as they are all alike. The roof is merely another box, but of triangular shape. The house can be made in many forms. Plenty of tiny furniture can be constructed from scraps of wood, cardboard, matchboxes, and so on. Of course the house will need to be finished off with a coat or two of paint.

Quite a good constructional set can be made from odds and ends of wood cut to various shapes. For example, a length of ½in. square stuff can can be cut up into bricks for building walls. Strips of plywood make useful beams and girders for bridge building. Window frames and doors can also be made as useful accessories. Both boys and girls like building forts, houses, bridges, and so on. The various accssories should all be painted in natural colours and the resultant buildings will look very real.

I THOUGHT THAT AS WE'RE ALL GETTING IN THE CHRISTMAS SPIRIT AND AS THIS IS THE FINAL CONTRIBUTION OF THE 1950'S HANDY-HINTS, I WOULD INCLUDE A SECOND 'TREAT' FOR YOU ALL!

HERE'S HOW TO MAKE SIMPLE TOYS WITH VERY LITTLE MATERIAL TO WORK WITH AND EVEN LESS MONEY TO BUY STUFF WITH. BEARING IN MIND THAT AT THIS TIME, MOST FOOD AND CLOTHES WERE STILL RESTRICTED AS RATIONING WAS STILL PREVALENT WITHIN THE U.K.

I CAN'T IMAGINE THAT KIDS TODAY WOULD BE OVER-JOY'D WITH THE RESULTING PRESENTS, BUT AT THE TIME I'M SURE, THEY WOULD HAVE BEEN MOST WELCOME.

EBAY.CO.UK 'BARGAINS'

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



£
£NS



£200



£16



£25



£92



£160



£81



£103



£235



£4

IN MAV'S WORKSHOP

Interesting Coco Repair
PIA Chip Mixup Anomaly

by Ian Mavric



This may be common knowledge among the Coco repairers out there, but I couldn't find any references to it anywhere (not that I tried really hard) so I thought I'd write a short article which may help those diagnosing and repairing a Coco 1 or 2.

IT WORKS - SORT OF.....

Years ago I picked up a Coco 2 for a song because it sort of worked. It would work in text mode but graphics tried to work but didn't It would display mixed semigraphics modes with text when you ran a graphical game (see that picture, at the top). In text mode the computer worked fine, it just didn't do graphics.
I figured this would be easy to work out so I dropped in a replacement 6847 VDG (the chip responsible for the text and graphics), and no change. I did the same with all the socketed chips in the computer and the problem persisted.



Photo: the Colour Computer 2 in question

I even changed the CPU, Ram chips, and Rom chips even though I was 100% convinced the devices were fine - and they were. Time to break out the scope and see what signals were telling the VDG to display text or graphics mode and they were all screwed up.
The VDG talks to one of the Peripheral Interface Adapter (PIA) chips and that communication was broken. I tried another PIA chip, still not working. Tested the traces between the VDG and PIA - perfect.

At this point I pull at an old fashioned paper schematic, which in this instance was easier to find it on www.colorcomputerarchive.com and I noticed that the two PIAs in the Coco, an MC6821 and MC6822 had switched sockets.
I guess an easy mistake to make if you are not paying attention, and the system will run in text mode with the chips in the wrong sockets. I prized them out one last time and switched them back to the sockets the schematic said they should be in, powered it up with Mega-Bug cartridge in and the computer was fixed!

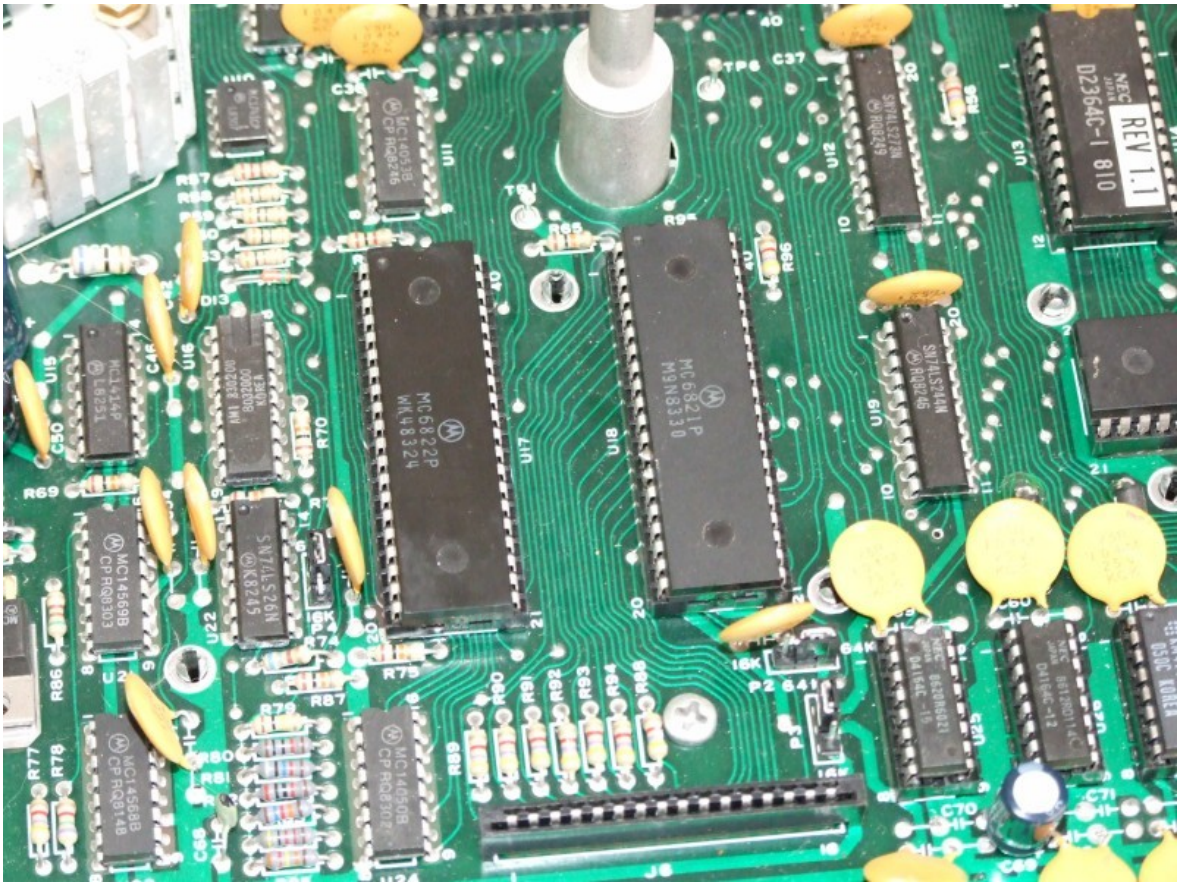


Photo: The two PIA chips in the wrong sockets

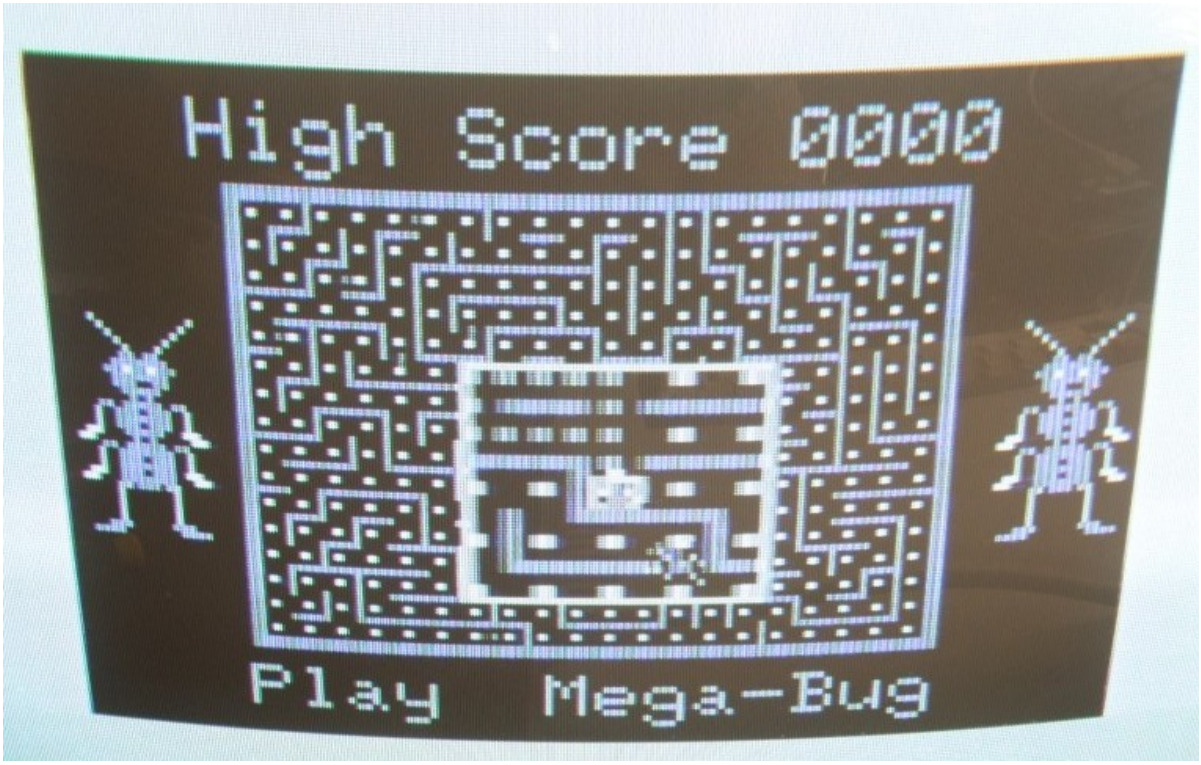


Photo: Mega-Bug up and running properly on the CC2.

DOES THE COCO 1 DO THIS AS WELL?

Curious to see if this problem occurred on the Coco 1, so I pulled one off the shelf, the re-designed version of the Coco 1 and switched the PIA chips, plugged in Mega-Bug and held my breath. It did the same thing! Removed the cart and messed around with Basic in text mode and the computer worked fine. .

CONCLUSION

So there you have it, if you are pulling out chips on an old Coco 1 or 2 during repairs be sure to pay particular attention to which sockets you put the PIA chips back into, as switching them will make your computer seem to be working perfectly fine, until you go to use a graphics mode... and let's face it, the graphics modes are why you bought the Coco in the first place.

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.

GENIE OWNERS – LOAD AND EVEN COPY SYSTEM TAPES ONTO YOUR SECOND CASSETTE WITH THE

SYSTEM LOADER

Though it is possible to both load and save Basic programs onto Genie's internal and external cassette systems, the inherent fault is that machine language programs can only be loaded via the internal cassette, which sometimes is not so reliable.

Mike Chalk has altered all that! For with his System Loader not only can you actually load machine language program through the second cassette port, but can even copy them as well!!

Yes, you can actually make copies of system tapes, and have the choice of either the second or internal cassette to do it. It's all made very easy too, just by pressing a single key, with no need for a file name.

Still more goodies in the program, for Mike has devised it so that it prints out the name, the entry, start and end addresses in the bargain. And there's a facility to jump to any address during the procedure.

It's a Basic program which can therefore be loaded from any source and it creates a machine language program to do the work. It's even useful for the TRS-80 too!

It's from Kansas and only from Kansas – **£8.50**

DISASSEMBLER

This program is the ideal introduction to machine language, for it allows you to 'list' system tapes and even the resident Microsoft Basic in ROM, allowing the start and stop addresses to be specified with each line displayed slowly and a facility to halt the display.

There is the option to either disassemble parts or whole machine language programs from memory, or strings of machine code can be types which will be translated into standard assembly language.

It's another from Mike Chalk, so has a bonus – of allowing conversion of hexadecimal to decimal and decimal to hexadecimal.

Suitable for both the Genie and Tandy TRS-80 and only from Kansas – **£8.50**

Kansas

All prices VAT paid and post free. First class return post service. Barclaycard, Visa orders dispatched same day if phoned in by 3pm. Answering machine ordering service after 6pm and Sundays to allow use of cheap rate. Catalogue available on request.

Kansas City Systems, Unit 3, Sutton Springs Wood, Chesterfield, Derbys. Tel 0246 850357



The *BEST* in TRS-80s

Call The Right Stuff

Ask for Ian
The number is +61 416 184 893

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

http://www.trs-80.com.au/







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

WHERE 8 BITS AND 16K STILL RULE OK